GÖTEBORGS UNIVERSITET

Sju heuristiker för utvärdering av webb-API:er

En studie i användarcentrerade utvärderingskriterier för webb-API:er

Seven heuristics for evaluation of web APIs

A study of user-centered evaluation criteria for web APIs

KRISTOFER HOLMGREN
PER-OLOF P. EDQVIST

Kandidatuppsats i Informatik

Rapport nr. 2013:035
ISSN: 1651-4769

Göteborgs universitet
Institutionen för tillämpad informationsteknologi
Göteborg, Sverige, Maj 2013

Abstrakt

Antalet webb-API:er som är öppna för allmänheten ökar snabbt. Detta gäller både webb-API:er som tar betalt för åtkomst eller som är gratis att använda. Såväl större som mindre organisationer, professionella- som hobbyutvecklare använder dessa webb-API:er för att addera värde till sina system eller skapa helt nya applikationer. I denna spännande utveckling saknas det användarcentrerade utvärderingsmetoder för webb-API:er. Tidigare forskning efterfrågar en anpassning av MDI-metoder för utvärdering av webb-API:er. Syftet med denna uppsats är att anpassa heuristisk utvärdering som kommer från området människa-datorinteraktion (MDI). Anpassningen innebär att ta fram kriterier som passar för utvärdering av webb-API:er. I vår fallstudie utför vi ett experiment där vi samlar in data om de användningsproblem vi stöter på. Experimentet består av att vi skapar tester mot ett webb-API, MobileResponse, som fortfarande är under utveckling. Därefter använder vi oss av en enkät för att undersöka vilka problem andra systemutvecklare haft vid användning av MobileResponse. Resultatet från dessa två datainsamlingar, tillsammans med det teoretiska ramverk vi skapat, ligger som grund för vår analys då vi försöker komma till roten av vad dessa problem är. Vi formulerar därefter kriterier (heuristiker) som fångar upp dessa typer av problem. Slutsatsen blir en rekommendation i form av en lista med heuristiker som kan användas vid heuristisk utvärdering av webb-API:er.

Nyckelord: Utvärdering, heuristiker, webb-API

Abstract

The amount of web APIs that are open to the public is increasing fast. Both the free web APIs as well as the ones that cost money to access. Both bigger and smaller organizations, professional and non-professional developers use these web APIs to add value to their systems or use it to create new applications. This is an area study where alot of research has been done in the last few years. In spite of this there are still no user-centered evaluation methods for web APIs. Previous research calls for an adjustment of human computer interaction-methods (HCI) for the purpose of evaluating web APIs. The purpose of this essay is to adjust heuristic evaluation wich is a method originating from HCI. The adjustment will result in a presentation of evaluation criteria, suitable for evaluating web APIs. In our case study we perform an experiment where we collect usability problems that we come across. The experiment includes us programming tests towards a web API, called MobileResponse, that is still under development. After that we will use a questionaire to investigate what problems is experienced by other system developers when interacting with MobileResponse. Theese two data-gathering activities will, together with the theorethical framework we've created, act as a base for the analysis. The analysis will lead us to identify underlying causes of the problems found. After that we formulate heuristics that should encapsulate these types of problems. The conclusion is a recommendation in shape of a list of heuristics that can be used for heuristic evaluation of web APIs.

Keywords: Evaluation, heuristics, web API

Tack

Vi vill tacka Bosbec AB för att vi fick tillgång till MobileResponse under får fallstudie. Vi vill även tacka deltagarna som besvarat vår enkät.

Vi tacka vår handledare Henrik Sandklef som uppmanade oss att inte ge upp när det kändes hopplöst.

Vi vill även tacka alla de personer som hjälpt oss med feedback på uppsatsen under arbetet.

1. Introduktion

I detta kapitel presenterar vi bakgrunden och det större sammanhang som vi placerar vår forskning inom. Under rubriken bakgrund motiverar vi varför det finns ett intresse för det problemområde vi forskat inom. Vi går sedan vidare till en problemdiskussion där vi preciserar vad vi anser är problematiskt i dagens läge. Detta mynnar ut i ett tydligt definierat syfte för uppsatsen.

1.1 Bakgrund

Webb-API:er (Application Programming Interface) blir allt vanligare och viktigare inom utveckling av nya tjänster och system. Flera större företag som Facebook och Google öppnar av olika anledningar upp sina webb-API:er för allmänheten att använda sig av (Bodle 2011). Exempel på detta är Facebook- och Googles inloggningstjänster som används av tredje part för både hemsidor och mobila applikationer.

Det finns även webb-API:er som är tillgängliga för allmänheten men tar betalt för att man ska kunna använda det. I vår fallstudie har vi fått möjligheten av företaget Bosbec att undersöka deras webb-API MobileResponse som är av denna typ.

Med tillgång till dessa typer av webb-API:er öppnar sig nya möjligheter för utvecklare att skapa ny mjukvara. Mashups är ett begrepp som beskriver tjänster som kombinerar funktioner från befintliga webb-API:er och datakällor för skapa ett nytt system (Zang et al. 2008). En metafor för denna typ av utvecklingsmöjlighet är hur DJ:s blandar ihop musikstycken av andra artister för att skapa ett eget stycke.

Utvecklingen av webben till det som idag benämns Web 2.0, kan till stor del beskrivas som att mer och mer av innehållet på webben är användargenererat. Allteftersom fler företag, organisationer och myndigheter möjliggör åtkomst till deras data och webb-API:er så skapas nya förutsättningar för exempelvis mashups. Kopplingar mellan webb-API:er och hur webben utvecklas beskrivs i flertalet artiklar (Zang et al. 2008; Bodle 2011; Parnin & Treude 2011; Maleshkova et al. 2010; Letia & Chifu, 2011).

De systemutvecklare som använder webb-API:er för att programmera nya system kan ses som slutanvändarna av webb-API:et (traditionellt sett hade användaren av det nya systemet var slutanvändaren). Interaktionen för systemutvecklare med webb-API:er är inte alltid smärtfri, enkel eller självklar. Vi tror att interaktionsdesignmetoder kan ge svaret på hur man förenklar

användandet, då målet med interaktionsdesign är att ta bort eller reducera de negativa och förstärka de positiva aspekterna vid interaktionen (Turban et al. 2011; Rogers et al. 2011).

1.2 Problemdiskussion

Det har publicerats forskning som pekar på användbarheten av metoder från området människa-datorinteraktion vid utvärdering och design av webb-API:er (Beaton et al., 2008). Denna forskning pekade på att det krävs mer undersökning kring hur man ska förändra dessa MDI-metoder så de passar vid utvärdering av webb-API:er. Det är denna utforskningsmöjlighet som vår uppsats behandlar. Under 1990-talet presenterade och förfinade Jakob Nielsen sin heuristiska utvärderingsmetod. Den består av tio heuristiker (kriterier, tumregler) om vad man ska tänka på vid design av grafiska gränssnitt. Vi vill anpassa Nielsens heuristiker så att de är användbara för utvärdering av webb-API:er. Vi anser att det kommer vara ett värdefullt bidrag för utvecklare av webb-API:er då de får en enkel metod för att identifiera användarrelaterade problem.

1.3 Syfte/Problemformulering

Tidigare forskning efterfrågar en anpassning av MDI-metoder för utvärdering av webb-API:er (Beaton et al. 2008). Syftet med denna uppsats är att anpassa heuristisk utvärdering som kommer från området människa-datorinteraktion (MDI). Anpassningen innebär att ta fram kriterier som passar för utvärdering av webb-API:er.

Vilka heuristiker är lämpliga för utvärdering av webb-API:er?

1.4 Definitioner

Under uppsatsen kommer vi återanvända ett antal begrepp. Det är viktigt för förståelsen av uppsatsen att läsaren uppfattar begrepp på samma vis som vi gör.

Vi har valt att benämna personer som skapar webb-API:er för utvecklare av webb-API:er. Personer som utvecklar något med hjälp av ett webb-API kallar vi systemutvecklare. Detta är för att inte läsaren ska blanda ihop utvecklare av API:et med utvecklare som använder API:et.

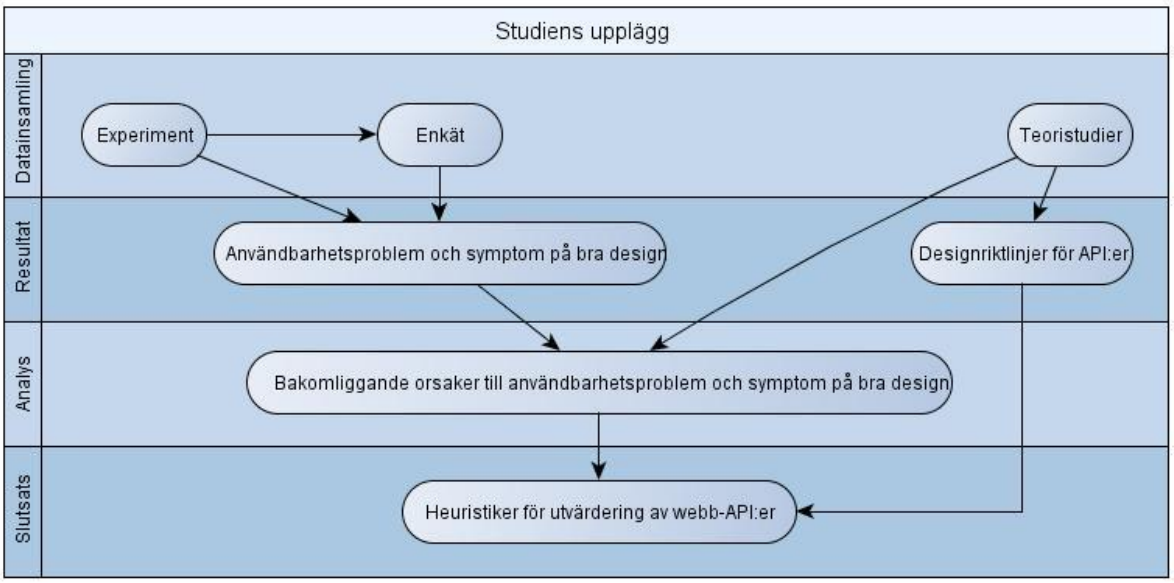
När vi skriver om API:er som är webbaserade skriver vi alltid webb-API:er. Detta är för att inte förvirra läsaren så att denne tror att vi pratar om andra typer av API:er.

Bosbecs webb-API heter MobileResponse och är en del av ett större system. När vi pratar om detta webb-API skriver vi MobileResponse.

Det finns skillnader mellan vad interaktionsdesign och människa-datorinteraktion betyder, men vi har förenklat och generaliserat och använder MDI som ett paraplybegrepp för användarcentrerade metoder.

1.5 Studiens upplägg

Figur 1.1 visar en övergripande bild av studiens upplägg. Vi har valt att inte beskriva upplägget i mer detalj då vi tydligt vill visa vilka aktiviteter som ledde fram till våra heuristiker. Exempelvis teoristudierna innehöll mer utöver designriktlinjer för API:er. All teoristudie beskrivs under teorikapitlet. Under varje huvudrubrik i uppsatsen ges en förklaring till vad som presenteras under kapitlet.



Figur 1.1 En övergripande bild av studiens upplägg

2. Teori

I detta kapitel beskrivs de teoretiska begrepp som bygger grunden för vårt arbete. Under första rubriken *Utvärdering och kriterier* introducerar vi begrepp från MDI-området som handlar om utvärdering och kriterier. Specifikt presenterar vi heuristisk utvärdering som är den utvärderingsmetod som vi vill applicera på webb-API:er. Vidare under rubriken *Relaterad forskning* redogör vi för den forskning som hittills skett mellan ämnesområdena MDI och API-utvärdering. Därefter under rubriken *Designriktlinjer för API:er* introducerar vi ett ramverk med kognitiva dimensioner som hjälper vid design av API:er. Slutligen under rubriken *API:er och webb-API:er* förklarar vi skillnader mellan vanliga API:er och webb-API:er.

2.1 Utvärdering och kriterier

Utvärderingmetoder används i syfte att förbättra en produkts design (Rogers et al. 2011). Inom MDI-området är syftet med utvärderingen inte endast att utvärdera användbarheten utan även användarens upplevelse av interaktionen. En enkel och billig metod för utvärdering av användbarheten hos grafiska gränssnitt kallas heuristisk utvärdering. Denna utvärderings-metod bygger på tio heuristiker som är skapade för att hjälpa till att identifiera en större mängd användningsrelaterade problem.

2.1.1 Heuristisk utvärdering

Heuristisk utvärdering är en metod som togs fram för att experter ska kunna utvärdera design av grafiska gränssnitt. En heuristik kan liknas vid en princip eller kriterium som säger på en högre nivå hur något bör vara designat. Personer med kunskap kring MDI och som har kännedom om tilltänkta användarna av ett system är det vi menar med experter. I heuristisk utvärdering går experten igenom en del av ett gränssnitt flera gånger och har heuristikerna i åtanke för att avgöra om exempelvis en knapp är optimalt placerad eller om rätt information finns tillgänglig för användaren. Genom att utföra dessa avgränsade inspektioner hittar experten problemsymptom eller förfinar problemdiagnosen (Rogers et al. 2011).

Som nämnts ovan togs de ursprungliga tio heuristikerna fram för utvärdering av grafiska gränssnitt. För att heuristikerna ska vara användbara för andra typer av gränssnitt krävs att de anpassas för det den typ av gränssnitt som är aktuellt. Det krävs mellan fem och tio heuristiker för att de ska vara användbara, då fem inte är tillräckligt för att fånga upp alla problem och tio är för många för utvärderaren att hålla reda på (Rogers et al. 2011).

Det finns ett starkt samband mellan designriktlinjer och heuristiker. Ett sätt att skapa heuristiker för ett nytt problemområde är att se om det finns några generella designriktlinjer för området. Dessa kan översättas till frågor som blir heuristiker i heuristisk utvärdering (Rogers et al. 2011).

Det är viktigt att notera att heuristisk utvärdering inte utvärderar ett system i dess helhet. Utvärderingen kan inte säga om systemet har rätt funktioner för en specifik användares behov, det kan bara säga om produkten går smidigt och enkelt att använda. Liknande utvärderingsmetoder missar användbarhetsproblem då man endast utvärderar produkten under en kort stund (Löwgren & Stolterman 2004).

2.2 API:er och webb-API:er

Ett API är gränssnitt som tillåter ett system att använda ett annat systems funktioner. Operativsystemet Windows har exempelvis ett API som tillåter programmerare att använda dess funktionalitet. Ett annat känt API är Swing för Java. Swing är ett API som gör det möjligt att skapa grafiska gränssnitt för Java program.

Skillnaden mellan ovan nämna API:er och webb-API:er är att den sistnämnda exponerar sin funktionalitet genom internet. Det kan tyckas att webb-API:er kan vara samma konstruktion som webbtjänster men det finns skillnader som gör dem olika. En skillnad är att Webb-API:er ofta bygger på en designarkitektur som kallas RESTful[1] (Representational State Transfer). Webb-API:er som konstruerats efter denna arkitektur är ofta enkla att använda och använder HTTP (Hypertext Transfer Protocol) för att skicka och ta emot meddelanden (Maleshkova et al. 2010).

Den skillnad som beskrivs ovan får som effekt att vid användning av webb-API:ets funktioner måste systemutvecklaren göra HTTP-anrop och skicka med relevant data exempelvis i form av JSON.

Om systemutvecklaren använder sig av en utvecklingsmiljö vid användning av ett vanligt API så har denne lättare att se metoder, parametrar direkt vid programmeringen. För en systemutvecklare som använder sig av ett webb-API så måste denne få ovan nämnda information genom dokumentation eller feedback från webb-API:et.

2.3 Historik över forskning om MDI-utvärdering av API:er

Forskning kring användbarhetsutvärdering av API:er har ökat under den senare delen av 00-talet. Forskningen på senare år har haft olika fokus. En del forskning har fokuserat på detaljer i själva koden (Ellis et al. 2007) medan annan forskning handlat om vilka utvärderingsmetoder man skall kombinera för att få reda på så mycket användbarhetsproblem som möjligt (Beaton et al. 2008).

Grill et al. (2012) tar vid där Beaton slutar och presenterar en metodik för hur man använder och kombinerar MDI-metoder i syfte att utvärdera användbarheten hos API:er. En av metoderna som föreslås är heuristisk utvärdering och författarna presenterar 16 heuristiker som de härlett från designriktlinjer för API:er (Zibran 2008).

[1] http://www.ibm.com/developerworks/webservices/library/ws-restful/

Robillard (2009) genomförde en undersökning med utvecklare hos Microsoft om vilka hinder de möter när de ska lära sig använda ett API. Den överhängande slutsatsen som drogs är att de resurser som finns tillgängliga för lärande till stor del avgör hur enkelt lärandet blir. Resurser delades in i underkategorier som: **exempel** (för få eller otillräckliga exempel), **generella hinder** (ospecificerade problem med dokumentationen), **innehåll** (en specifik del av innehåll saknas eller presenteras inte tillräckligt), **uppgift** (ingen information hur man ska använda API:et för att utföra en uppgift), **format** (resurser finns inte i det önskade formatet) och **design** (för lite eller dåligt dokumenterat om de högre nivåerna av design).

DeLine & Robillard (2011) följde upp sin studie från 2009 med en större fältstudie där syftet fortfarande var att undersöka vad som gör ett API svårt att lära sig. Denna gång var det ett större fokus på hinder för lärande som har med dokumentation att göra. För de utvecklarna som var med i undersökningen var svårigheterna vid inlärning inte relaterade till hur de ska använda API:er. Det var istället förståelsen för hur API:et relaterar uppåt till dess problemdomän och hur det relaterar neråt i dess implementation som var det svåra att förstå.

Stylos & Clarke (2007) drar slutsatsen att API:er med objekt vars konstruktor kräver andra parametrar än defaultvärden drar ner användbarheten hos ett API. Denna slutsats drogs genom att analysera deltagares beteende med det kognitiva ramverket. Detta visade att konstruktorer som kräver parametrar inte är kompatibelt med de vanligaste sätten vi lär oss på.

2.4 Designriktlinjer för API:er

Henning (2009) presenterar en lista med riktlinjer för hur man bör designa API:er och vilka konsekvenser ett dåligt designat API kan få. En av riktlinjerna som föreslås är att API:er bör dokumenteras innan de implementeras. En annan intressant riktlinje som Henning presenterar är att bra API:er är ergonomiska. Med denna riktlinje menas bland annat konventioner för namngivning, kodlayout och dokumenteringsstil. Henning menar förövrig att en stor del av det som är ergonomiskt har med struktur att göra, exempelvis att funktioner alltid placerar parametrar av ett särskilt slag i samma ordning varje gång.

Bloch (2008) sammanfattade kärnan från sina föreläsningar om hur man bäst designar API:er genom ett antal kortfattade visdomsord, exempelvis att namn och dokumentation spelar stor roll. En del av dessa kan ses som designriktlinjer av högre nivå medan andra visdomsord påminner om enskilda punkter på en checklista att pricka av.

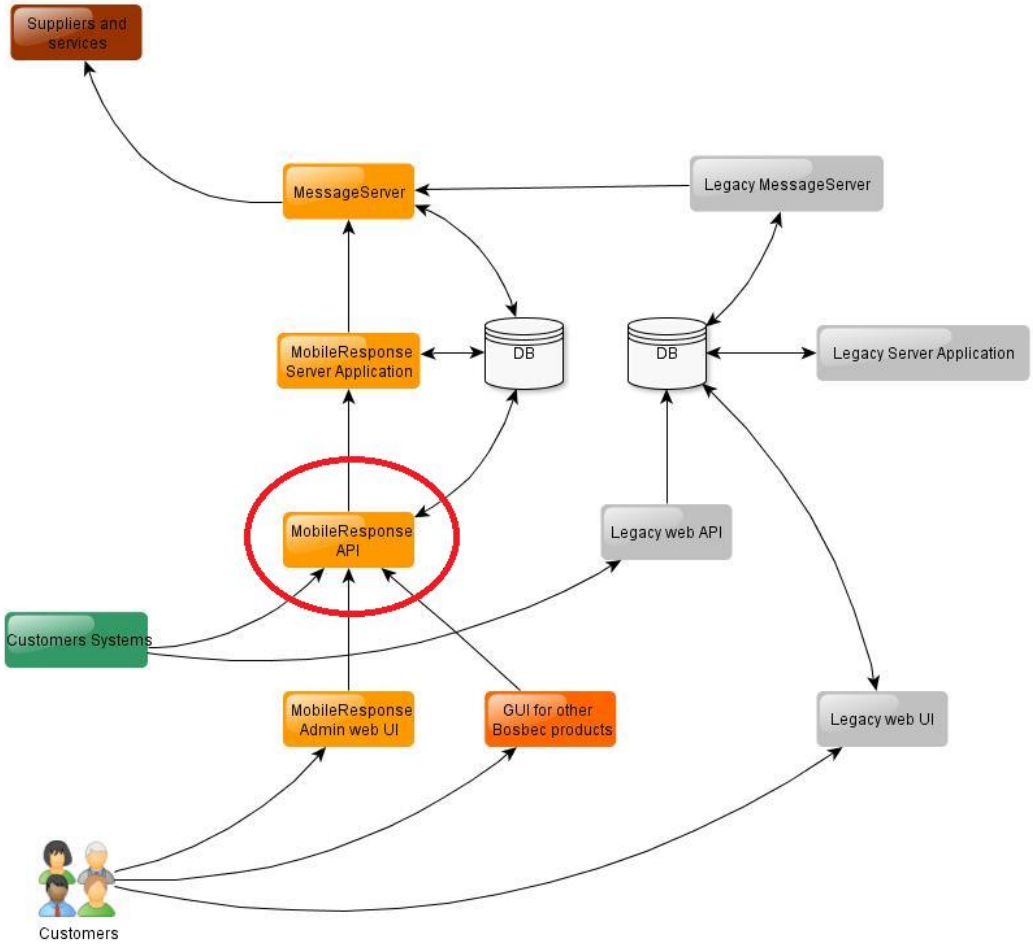
Clarke (2004) beskriver i sin artikel om hur de använder olika tekniker för att utvärdera användbarheten hos de API:er som skickas med dotNET. Författaren presenterar ett ramverk med kognitiva dimensioner som på något sätt påverkar hur systemutvecklare använder och förväntar sig att ett API ska fungera. Några exempel på dimensioner är **abstraktionsnivå**, **inlärningsstil**, **fasthet** och **överrensstämmelse med problemområdet**. Dimensionerna kan användas för att analysera resultaten av en användbarhetsstudie eller vid skapandet av kod som inte implementerats.

3. Fallstudieobjekt: MobileResponse

Under följande kapitel kommer vi förklara mer om MobileResponse, som varit i fokus för vår fallstudie. Under första rubriken ges en generell bild av vad MobileResponse är för typ av system. Frågor som vi besvarar är: vem är det till för, i vilket utvecklings-stadie befinner det sig i. Vidare under rubriken *Administratörsgränssnittet för MobileResponse* presenterar vi ett grafiskt gränssnitt att använda under tiden man utvecklar system mot webb-API:et eller administrerar sitt konto.

3.1 Vad är MobileResponse?

MobileResponse är ett system som agerar knutpunkt för, framförallt textbaserad kommunikation via sms-, email- eller app-meddelanden. I MobileResponse kan användare skapa flöden eller processer som startas av, skapar eller på annat sätt använder informationen som flödar som meddelanden i systemet.



Figur 3.1 Komponenter eller delsystem som är relaterade till MobileResponse. Vår avgränsning i denna uppsats är markerat i den röda cirkeln.

Systemet är i en period där det håller på att lanseras samtidigt som det utvecklas. Systemet består av ett antal komponenter. En komponent är det webb-API som vi valt som fallstudieobjekt. Detta webb-API är ett gränssnitt som exponeras ut till utvecklare som använder det underligande systemet. Gränssnittet exponeras därmed såväl för företagets egna system och kundernas system.

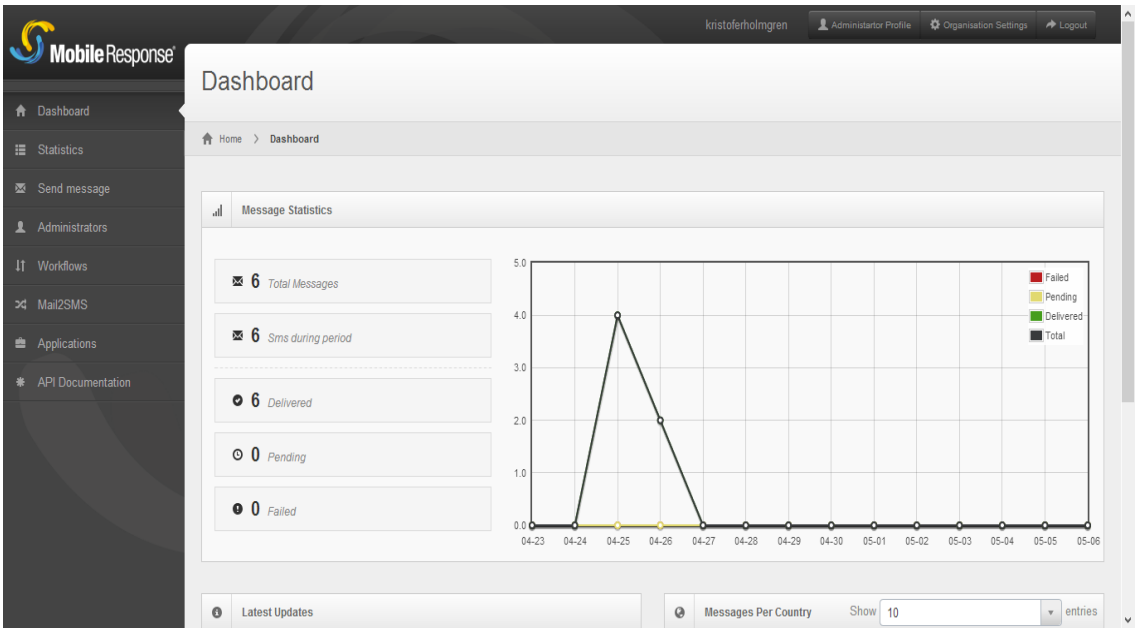
När Bosbec presenterar systemet för potentiella kunder sägs det att det är en kommunikations-plattform. Deras kunder tycker att detta sätt att hantera kommunikation tillför värde i flera steg. Det vanligaste användningsområdet, där de flesta kunderna börjar använda systemet, är processen att skicka meddelanden till en eller flera mottagare organiserade i olika grupper.

3.1.1 Administratörsgränssnittet för MobileResponse

MobileResponse tillhandahåller ett webbgränssnitt där användare kommer åt olika sorters funktionalitet. Administratörsgränssnittet är ett enkelt visuellt lager ovanpå webb-API:et som

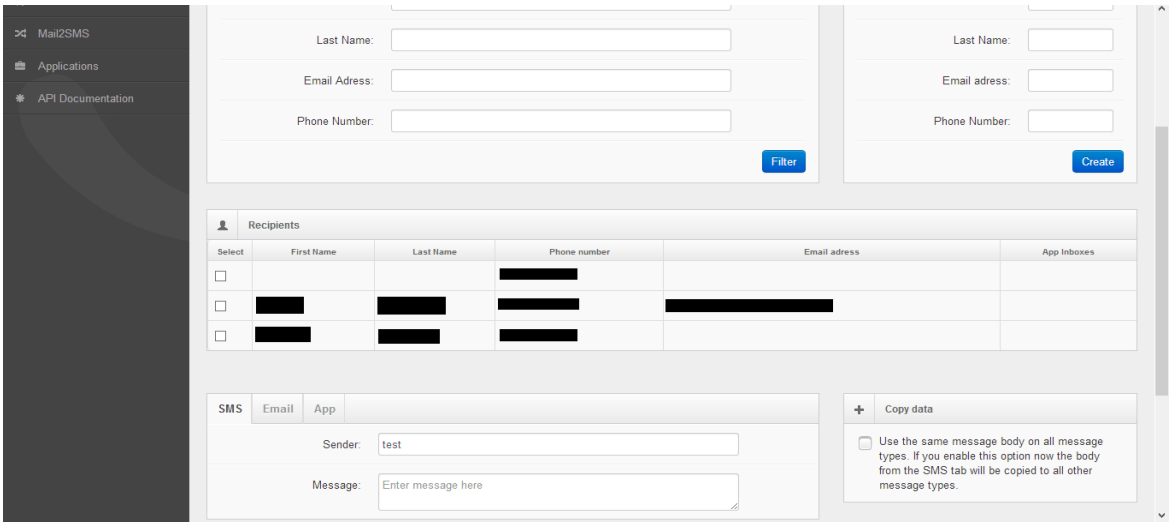
fungerar som en hjälp för användare att administrera sina konton och MobileResponse. Användaren kan inte göra något mer än det som redan finns i webb-API:et. Nedan följer en genomgång av de grafiska gränssnitt vi stötte på under experimentet.

Kontrollpanel



Figur 3.2 Kontrollpanel för en användare med administratörskonto. Flikarna till vänster ger möjlighet att se statistik, skicka meddelanden, hantera administratörer, hantera workflows, mail2sms, applikationer samt API-dokumentation.

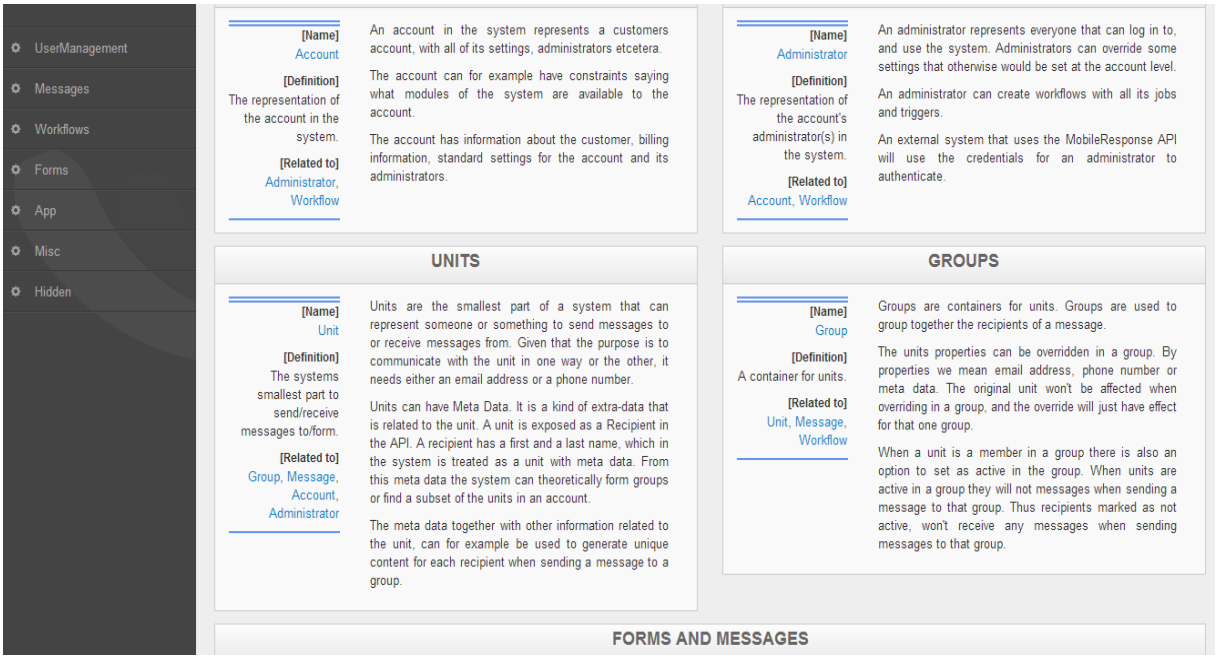
Skicka meddelande



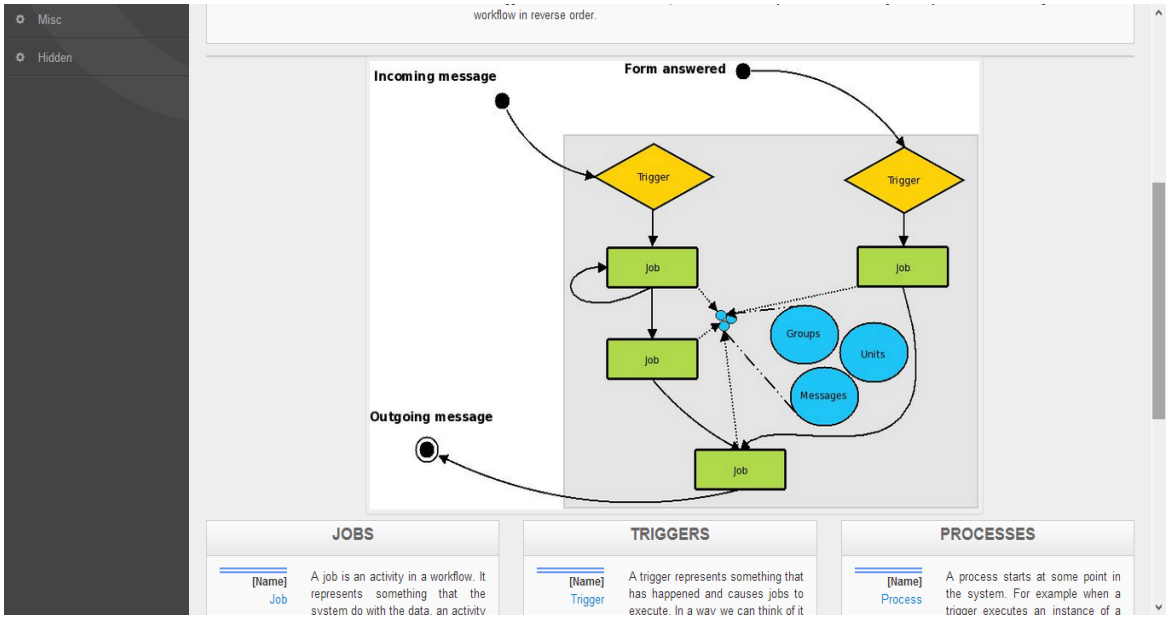
Figur 3.3 Vyn där användaren kan skicka meddelanden till mottagare.

3.1.2 API-dokumentation

Dokumentationen för MobileResponse är åtkomligt från administratörsgränssittet. Menyn är kategoriserad utefter övergripande rubriker som *System overview*, *Workflow overview* samt kategorier av klasser. Nedan visar vi exempel på dessa sidor.



Figur 3.4 Översikt av klasserna. Varje klass har ett namn, en definition samt en lista med relaterade klasser. Förutom detta beskrivs klassen med en längre brödtext som utförligt förklarar vad klassen representerar.



Figur 3.5 Översikt av vad ett workflow är samt vilka klasser som är inblandade.

4. Metod

Som en del i vår metod räknar vi litteraturinsamling som gjordes kontinuerligt under arbetet. Som utgångspunkt för litteraturinsamling har vi använt oss av sökfunktionen hos biblioteken hos Chalmers och Göteborgs Universitet. Vi använde sökord som: API, usability, heuristic evalutation, API evaluation, web API, usability problems, HCI och sökord som var vanligt förekommande i de problemområdena. Utöver detta har vi även använt litteratur från vår utbildning som referenser och även som källor vari vi funnit bra sökord att gå vidare med i sökandet efter litteratur.

Som tidigare nämnt är syftet med den här fallstudien att ta fram heuristiker för utvärdering av webb-API:er. För att kunna formulera så bra heuristiker som möjligt behövde vi samla in data som pekade oss i rätt riktning och som kunde ligga som grund för våra heuristiker. Då vårt undersökningsområde är tämligen outforskat anser vi att en kombination av experiment & enkät tillsammans med relaterad teori var det bästa tillvägagångsättet för att skapa och motivera heuristiker.

Vi genomförde två datainsamlingar under vår studie. Den första datainsamlingen var ett experiment där vi programmerade tester av funktionaliteten i MobileResponse i syfte att upptäcka användbarhetsproblem och bra design som förhindrade användbarhetsproblem. Den andra aktiviteten var en enkätundersökning riktad mot andra systemutvecklare som har utvecklat något system som i sin tur använder MobileResponse. Denna metodologiska triangulering har hjälpt oss att stärka giltigheten i vårt resultat (Rogers et al. 2011). Vi ville hitta så många användbarhetsproblem som möjligt samt bekräfta dem hos andra.

4.1 Experiment

Patel & Davidson (2011) förklarar att experiment är ett upplägg på undersökning där man försöker ha kontroll på de flesta variabler och undersöker en eller ett fåtal av dessa. De menar att resultatet som framkommer främst är giltigt för den grupp som studerats under experimentet. Ett sätt att göra resultatet mer generaliserbart är att ha en noggrann process där en del av en population slumpvis valts ut och blir representativt för hela populationen.

Det var svårt att få tag på individer som kunde medverka i vår undersökning, då fallstudieobjekt inte är fullständigt i drift utan fortfarande är under utveckling. Ett problem var att det inte finns några kunder som var i startgroparna med att börja använda MobileResponse. Vår undersökning avser att hitta problem från det att man börjar använda MobileResponse. Det fanns inga systemutvecklare hos Bosbec som skulle påbörja sin implementation under

den tidsperiod där vi kunde observerat. Med den korta tiden som finns till förfogande för själva genomförandet av experimentet i vårt projekt så var vi tvungna att själva ta rollerna som systemutvecklare mot MobileResponse.

Syftet med att använda experiment som metod är således inte att studera en enskild variabel utan att använda metoden på ett sådant sätt som gör att vi kontroll på de variabler som skiljer experimentdeltagarna åt. Syftet med kontrollen var att det endast skulle vara deltagarnas förkunskap av MobileResponse och programmeringskunskap som varierade. Deltagarna presenteras under rubriken *Deltagare*, under rubriken *Miljö för experimentet* presenteras det vi gjort för att säkerställa att vi har kontroll på andra variabler.

4.1.1 Deltagare

Deltagarna i experimentet fick själva välja geografisk plats att arbeta från. Kravet var att under genomförandet skulle deltagaren kunna komma i kontakt med support om behovet uppstod. Deltagarna för experimentet förväntades ha olika förkunskaper och i denna undersökning hade vi två deltagare med följande likheter och skillnader.

Deltagare1: arbetar deltid som systemutvecklare och har förkunskap både i programmerings-språket C#, dotNET och fallstudieobjektet MobileResponse.

Deltagare2: arbetar inte med programmering men programmerar som hobby. Har begränsad förkunskap i programmeringsspråket C#, dotNET och ingen förkunskap kring fallstudieobjektet MobileResponse.

Experimentets båda deltagare läser samma utbildning och är med och skriver denna uppsats. I och med detta förväntar vi oss att följande saker skall underlättas:

- Språket i dagböcker – Vi har samma förståelse av terminologi som används.
- Genomförandet underlättades av att vi båda varit med och anpassat metoden.

Följande faktor kan innebära svårigheter eller osäkerheter som måste förklaras hur vi hanterat för att säkerställa att undersökningen blir korrekt genomförd.

- Vi har liknande teoretisk referensram vilket kan innebära att vi missar perspektiv på användbarhetsproblem som hade funnits hos en person från en annan utbildning.

4.1.2 Miljö för experimentet

Om vi formulerar experimentet att motsvara en situation i verkliga världen skulle det låta som följande: En systemutvecklare har i uppdrag att ta fram en del i ett system där man behöver använda sig av en exponerad tjänst hos ett webb-API. För att komma igång behövs en övergripande plan med moment som ska genomföras för att uppgiften skall anses vara löst. Under programmeringsmomenten arbetar systemutvecklaren med testdriven utvecklings-metodik (Beck 2002).

Anledningen till att vi valt att använda testdriven utveckling är då tillvägagångssättet är metodiskt. Lösningarna som deltagarna skapar bör se ganska lika ut även om det finns olika förutsättningar. Man måste formulera, för sig själv och i kod, hur man avser att testa att varje steg man modellerat och tänkt ut. Testen skrivs först, följt av implementering och sedan omstrukturering så kallat refaktorisering.

En viktig poäng med testdriven utveckling är att minimera risken för att hoppa över steg i utvecklingen. Att hoppa över steg i utvecklingen kan ske på grund av att man upplever sig som erfaren systemutvecklare som inte gör den typen av misstag (Martin 2008).

För att säkerställa att de moment som skall klaras av blir likadana för varje deltagare, hade vi på förhand skapat ett användningsfall som återfinns hos majoriteten av alla kunder som använder MobileResponse. Detta användningsfall beskrivs under rubriken *Förberedelse*.

För att kontrollera förutsättningarna i så stor utsträckning som möjligt såg vi till att båda experimentets deltagare använder samma utvecklingsmiljö samt samma version av utvecklingsmiljön. Inledningsvis hade vi diskussioner i att genomföra experimentet med utvecklingsmiljön Mono[2]. Detta gjordes så experimentet kunnat genomföras oavsett vilket operativsystem som kördes på datorerna. För att bättre ha kontroll på experimentets miljö bestämde vi oss för att hålla oss till Windows-miljö.

I detta fall kunde den integrerade utvecklingsmiljön (IDE) *Microsoft Visual Studio 2012 Ultimate* användas eftersom det funnits tillgängligt för båda deltagarna via *MSDNAA*[3]. Utöver

[2] http://www.mono-project.com/Main_Page
[3] https://www.dreamspark.com/

utvecklingsmiljön använde båda deltagarna produktivitetsverktyget *ReSharper*[4] och testramverket *nUnit*[5] för att köra tester.

Med experiment som en metod för undersökning behövde vi en struktur för hur data skulle skrivas i dagböckerna. Vi skapade en mall för dagböckerna som tvingades oss att tänka till kring vad som är bra eller dåligt med webb-API:et före, under och efter programmeringen (se bilaga 1). För dagböcker använde vi Googles ordbehandlare[6] (i Google Drive). Vi såg före, under och efter programmeringen som olika moment i utvecklingen.

4.1.3 Förberedelse

Vi förberedde experimentet genom att skapa ett övergripande användingsfall med syftet att testa funktioner som MobileResponse exponerar. Anledningen till att vi valde just det användningsfall är för att det är vanligaste användningsfallet bland kunderna till vårt fallstudieobjekt.

Förklaring till användningsfallet:

I de flesta fall har kunden något system varifrån kunden kan få ut information om vilka mottagarna är och vilka deras telefonnummer är. Ett exempel är där kunden har ett kundregister och vill undersöka vilka av dessa som är intresserade av en kommande produktlansering. Kunden vill då använda MobileResponse för att skapa meddelanden. Dessa meddelanden skall skickas som sms, till en eller flera mottagare som man organiserat i en grupp.

Stegvis beskrivning som beskriver användningsfallet:

1. Logga in i systemet
2. Skapa mottagare och organisera i grupper
3. Skapa och skicka meddelande till vald(a) grupper.

Experimentet genomfördes under fem dagar och vi följde vår experimentmall (se bilaga 1) för datainsamling.

[4] http://www.jetbrains.com/resharper/
[5] http://www.nunit.org/
[6] https://drive.google.com/

4.2 Enkät

Efter experimentet blivit genomfört skapade vi en enkät med syftet att samla in data från andra systemutvecklare som programmerat mot MobileResponse. Under nästa rubrik beskriver vi de mål som vi ville uppnå med enkäten. Därefter finns rubriken *Urval* där vi anger antal respondenterna och vilken bakgrund de har. Sedan kommer rubriken *Frågor* som motiverar de typer av frågor vi ställde.

När enkäten besvarades fanns en av författarna till denna uppsats tillgänglig via Skype och på samma plats som två av enkätens svarande. Ett syfte med detta, tillsammans med formuleringen av frågorna, var att just kunna förtydliga och säkerställa att de svarande fick hjälp att tolka frågeställningarna och vid behov resonera om svaren.

4.2.1 Mål

Det huvudsakliga målet med enkäten var bekräfta de problem och tecken på bra design som vi upptäckt genom experimentet. Utöver det huvudsakliga målet fanns en förhoppning att de svarande kunde ge egna exempel på problem de stött eller tecken på bra design. Genom att intervjua systemutvecklare med annan bakgrund än vår egen hoppades vi fånga upp problem och tankar som vi kunde missat under vårt experiment.

4.2.2 Urval

Vi skickade enkäten till tre stycken anställda på Bosbec. Alla deltagare hade utvecklat mot MobileResponse i någon form av webbgränssnitt eller mobil applikation.

Då ett av huvudmålet med enkäten var att bekräfta problem vi hade upptäckt, försökte vi få så hög svarsfrekvens som möjligt på enkäten. Av den anledningen gjorde vi urvalet så att alla som har utvecklat någon typ av system som är färdigt och används i någon utsträckning skall vara tillfrågade och ombedda att svara på enkäten. Med avgränsningen att endast välja de som utvecklat något system som bygger på MobileResponse kom vi fram till att det finns endast tre kandidater. De tre som besvarat enkäten har utvecklat någon typ av webbgränssnitt mot MobileResponse, två av dem har även utvecklat mobilappar som jobbar mot MobileResponse.

4.2.3 Frågor

De frågor vi ställt genom enkäten består både av öppna frågor samt frågor mer riktade mot kortare svar där respondenten håller med eller motsätter sig. Rogers et al. (2011) anser att demografiska frågor kan vara bra för att hjälpa till att placera svaranden i rätt kontext.

De demografiska frågor vi ställt handlar om erfarenhet som är relevant för att jämföra med deltagare i experimentet. Detta eftersom vi på förhand visste att de svarande är män i åldersspannet 30-40 år, som alla arbetar för samma företag men sitter inte alltid på samma kontor.

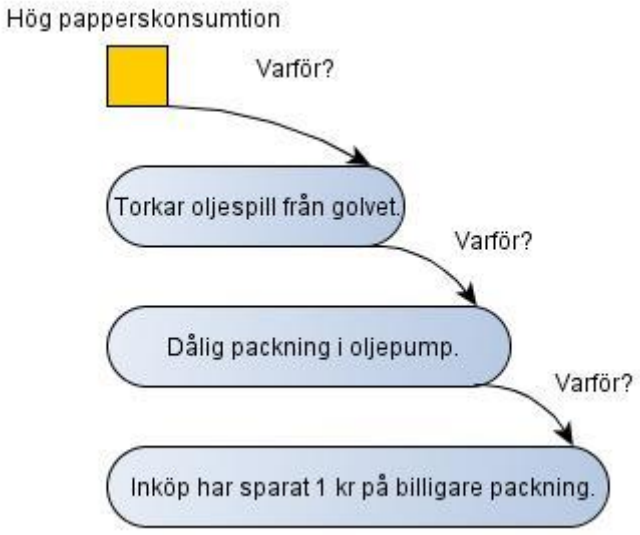
Då tiden som kunde avsättas till att besvara enkäten var mycket begränsad var vi tvungna att utforma frågorna på ett tillräckligt omfattande vis. Det gjorde att vi tvingades prioritera vilka frågor och åsikter vi ville ha svar på för att på bästa sätt stärka upp vår datainsamling.

Patel & Davidson (2011) tar upp hur man genom olika typer av frågor och svar uppnår olika grad av standardisering och strukturering. Vår enkät har hög grad av standardisering och något lägre grad strukturering. Det betyder att vi har valt frågor som inte endast har fasta svarsalternativ. Standardiseringen syftar till att vi ställer samma frågor till alla respondenter. I situationen där vi hjälper svarande att tolka frågorna avser vi att ge så konsekvent överens-stämmande förklaringar som möjligt till alla svarande. Vidare förklarar hur man brukar inleda med neutrala frågor som rör bakgrund, men även avslutningsvis ge en chans för de svarande att ta upp något som missats eller saknas i enkäten (Patel & Davidson 2011; Rogers et al. 2011).

4.3 Varför-Varför-Varför? som analysmetod.

Löwgren & Stolterman (2004) beskriver en metod som de kallar Varför-Varför-Varför?. De använder metoden som en undersökningsmetod för att behålla ett brett perspektiv och ifrågasätta och överskrida uppfattningar om problem ifråga.

Metoden kan ses som en dialog där ett problem uttrycks av en person och en kedja av varför frågor ska ifrågasätta den rådande uppfattningen om orsakerna till problemet. Under analysen tar vi de upptäckta problemen från experimentet och bearbetar problemformuleringen för att utforska bakomliggande orsaker.



Figur 4.1 En illustration hur Varför-Varför-Varför?-kedja kan se ut. Ett företag har hög papperskonsumption och ställer sig varför frågor tills de kommit fram grundorsaken. Grundorsaken var att inköp sparade en 1 per packning men det kanske kostade mer i papperskonsumption.

4.4 Analysmetod för enkätdata

Eftersom enkäten till hälften bestod av fritextfrågor och andra hälften bestod av graderingsfrågor har vi bearbetat graderingsfrågorna kvantitativt och fritextfrågorna kvalitativt.

För den kvalitativa bearbetningen har vi inspirerats av Patel och Davidsons (2011) exempel på grundad teori. Under bearbetningen läste vi våra anteckningar från dagböckerna och jämförde med svaren vi fick på fritextfrågorna för att undersöka om det fanns några gemensamma teman. Detta gjordes flera gånger för att bekräfta de problem och symptom på bra design som vi identifierat.

Patel & Davidson (2011) beskiver att statistik är en vetenskap för att kvantitativt bearbeta data. För graderingsfrågorna använde vi en enkel metod där vi tittade på genomsnittet för att avgöra om respondenterna höll med oss, var neutala eller hade motsatt åsikt i våra beskrivningar av problem och symptom på bra design.

4.5 Kritisk granskning av metoden

Vi är medvetna om att vi genom vår datainsamling inte kommer hitta alla användbarhets-problem. Dels beror detta på den begränsade tid vi har haft till förfogande, dels för att MobileResponse fortarande är under utveckling.

Resultatet som vi får fram är de problem som vi upplevde fanns under den perioden där vi tog rollen som systemutvecklare som utvecklade mot MobileResponse. Resultatet kan inte generaliseras i större utsträckning än det stöd vi hittat under litteraturinsamling.

I vår ursprungliga planering hade vi för avsikt att genomföra intervjuer med andra systemutvecklare som kommit i kontakt med MobileResponse och genom transkribering få mer kvalitativ data att komplettera experimentet med. Men då det inte fanns möjlighet för deltagarna att avsätta tid för intervjuer omstrukturerade vi vår planering och skapade en enkät som skulle motsvara de huvudsakliga frågor vi önskat ställa vid intervjuer.

Patel & Davidson (2011) beskriver att enkät och intervju har många likheter, framförallt i de fall där man använder vad de kallar för enkät under ledning. De menar att man gör sig tillgänglig för att hjälpa de svarande att genomföra och tolka frågorna i enkäten när den genomförs.

Ett eventuellt problem med experimentet är att vi tolkar oss själva och gör ett urval i de tankar vi skriver ned i dagboken. Diskussion uppstod om vi fått bättre data genom att observera varandra vid experimentet. Vi såg ett stort hinder med att observera varandra då de utvecklingslösningar som den observerade skapade kunde färga framtida lösningar som observanten ska skapa. Därför ansåg vi att det var bättre att utföra experimentet själva så vi tog tillvara på vår egna problemlösningsmetodik samt följde vår egen systemutvecklingsprocess.

Vid reflektion över vårt val av analysmetod skall det nämnas att metoden aldrig är bättre än den personens förmåga att genomföra metoden (Löwgren & Stolterman 2004). Beroende på hur man formulerar sina varför-frågor riskerar man komma fram till olika bakomliggande orsaker. Det finns inget som garanterar att man kommer fram till ”rätt” bakomliggande orsaker och inte heller något som hur många steg kedjan måste bestå av för att vara giltig. Personens erfarenhet och bakgrund påverkar i stor grad vilka varför-frågor denne formulerar.

5. Resultat

I detta kapitel presenterar vi resultatet från de två datainsamlingar som genomfördes.

Under rubriken *Experiment* presenterar vi våra två dagböcker med formuleringar kring hur deltagarna har tänkt inför de situationer och uppgifter som uppstått. Dagböckerna innehöll även stegvisa förklaringar till hur uppgifterna har lösts. Resultatet av experimentet är en stor

mängd kvalitativ data där vi plockat ut citat eller summerat ihop kvalitativ data för att beskriva ett användbarhetsproblem.

Resultatet av enkätundersökningen presenteras under rubriken *Enkät*. Data som samlades in från enkäten hjälpte till att metodologiskt triangulera betydelsen av problem som identifierats under experimentet.

5.1 Experiment

Då metoden hade två återkommande aktiviteter, programmering och modellering, har vi valt att presentera resultaten under två rubriker. Vad som skiljer modellering från programmering kan vara svårt att avgöra och har i första hand varit upp till experimentdeltagaren att förklara vid genomgång och bearbetning av resultatet.

Som generell riktlinje har vi tänkt att modelleringstegen blir de steg där det varit frågan om avbildning och förståelse för problemområdet som MobileResponse hanterar. De andra modelleringssteg som har med språkspecifika detaljer för implementering har istället räknats till programmering.

5.1.1 Upptäckta problem

Under insamlingen av data strävade vi efter att skriva ner så mycket som möjligt. Syftet med detta var att vi inte ska göra någon egen tolkning av vilken data som är relevant eller vilken betydelse den kunde få. Vi upptäckte under arbetet med att analysera och presentera resultatet att vissa designbeslut resulterat i lösningar där problem har undvikts. Av denna anledning har vi omformulerat dessa lösningar till problem-formuleringar.

Under kommande rubriker presenterar vi resultatet. Vi ger en bakgrund, ett namn på problemet samt citat från dagböcker för att betona kärnan i problemet.

5.1.1.1 Modellering

Som vi tidigare beskrivit handlar modellering om avbildning och förståelse för problemområdet.

I situationen där man som systemutvecklare står med ett antal punkter som skall lösas har man en ofta en idé eller bild av vart man måste börja. Utöver att veta vilken utgångspunkt det egna systemet skall ha behöver man förståelse för om webb-API:et kräver att man angriper situationen på ett visst vis. Vi kallar problemet: **Svårt att veta vart man ska börja**.

> *”Även om jag hade läst på om de olika klasserna som finns så var det svårt att veta exakt i vilken sekvens de skulle användas. Jag visste inte heller att jag var tvungen att autentiseras innan användning sker. Detta var svårt tills jag hittade exempel.”* (deltagare 2)

> *”Även om jag på förhand vet hur jag ska börja inser jag att det är svårt att hitta informationen då man söker igenom dokumentationen”* (deltagare 1)

Om man inte har fått någon förkunskap så måste lösningen vara lätt att hitta i dokumentation eller motsvarande.

Vi formulerade ett problem som: **Svårt att få dynamisk förståelse.** Utdrag ur dagbok:

> *”Även om jag visste vilka metoder och attribut som klasserna har var det svårt att förstå hur de ska användas tillsammans.”* (deltagare 2)

> *”Det gavs en bra statisk förståelse men mindre dynamisk förståelse som jag läst i UML-boken av Larman. Som alternativ till, eller som komplement till deras exempel hade fler typer av visualisering, exempelvis diagram som ger förståelse för det dynamiska varit hjälpfulla.”* (deltagare 2)

Larman (2004) menar att man kan uttrycka statiska och dynamiska egenskaper hos ett system genom olika typer av modellering. Exempelvis tillståndsdiagram är ett sätt att ge systemutvecklare förståelse för hur ett system är uppbyggt och fungerar innan denne försöker använda det.

Till skillnad från grafiska användargränssnitt har webb-API:er större behov av dokumentationen och många problem kan troligtvis hanteras med olika typer av dokumentation. Vi identifierade problem under modellering som vi summerat som problemet: **Dokumentation**

> *“Innan jag hittade exempel på hur man använder funktioner så var det svårt att göra något.”* (deltagare 2)

> *“Dokumentationen var enkel att hitta i då den var lättnavigerad, det fanns ingen sökfunktion men jag kände inget behov av det då dokumentationen var relativt liten.”* (deltagare 2)

En viktig del i resultatet var de saker som kunde utläsas ur vår kvalitativa data. En del saker som man lyckats med i detta webb-API skulle kunna inneburit problem om de inte designat på detta vis. Vi kallar detta: **Bra exempel** och det visas i följande citat.

> *"Om det inte funnits bra exempel på hur man använder funktionerna tillsammans för att uppnå vad man vill göra hade det varit svårt att få förståelse för webb-API:et. "* (deltagare 2)

> *"I dokumentationen fanns en tutorial som delat upp processen i lagom stora steg och som introducerade klasser i lagom stora steg. Det fanns även detaljerad beskrivning av JSON struktur för både svar och anrop vilket ökade förståelsen för hur man kommunicerar med webb-API:et."* (deltagare 2)

En intressant del i kommunikation handlar om att förstå innebörden av orden i de sammanhang de befinner sig i, det som brukar kallas semantik. Problem som har med semantik att göra kan ha symptom på olika nivåer. Ett par av dessa har vi funnit i **Namngivning som är inkonsekvent och tvetydig**. Det är två separata problem, men kan tyckas vara relaterade.

> *"När man får svar från de anrop man gör finns det ett attribut som heter 'Success', jag tolkade det som att det jag ville genomföra lyckades. Men efter diskussioner med utvecklarna till webb-API:et fick jag veta att det endast betyder att anropet lyckades, inte operationen som systemet gör när anropet behandlats."* (deltagare 2)

> *"En annan osäkerhet relaterad till namn på attribut och objekt i webb-API:et är att man anropar funktioner för en 'Unit', men det handlar om 'Recipients'."* (båda deltagarna i diskussion)

Flera av de situationer där vi identifierat problem eller där det existerade bra lösningar återfinns även i andra delar än vad vi i experimentet har testat. Exempel på andra delar där kommit i kontakt med i samband med utforskandet av webb-API:et och dess dokumentation. Däribland namngivning som är inkonsekvent och tvetydig som beskrivits ovan.

Valfrihet hos användaren kan vara flera saker, i vår fallstudie har vi noterat att det skulle kunna innebära flera positiva aspekter att inte låsa in användaren i samma miljö och till

samma programmeringsspråk som webb-API:et är skrivit i. Vi hade dock bestämt för experimentets skull att vi skulle kontrollera så många variabler som möjligt för att bara förkunskaper skulle skilja. Som alternativ hade vi kunnat genomföra experimentet så att samma person testade webb-API:et från olika programmeringsspråk. I loggboken för en deltagare läser vi:

> *"Någonting som jag upplevde som positivt i detta fallet, men som lika väl kunde varit ett hinder eller problem är att webb-API:et tillåter viss valfrihet hos användaren. Den valfrihet jag menar är till exempel att anropen görs via HTTP och med JSON-format på data, vilket gör att jag som användare inte blir låst till att använda samma programmeringsspråk som webb-API:et är utvecklat i. Vilket kunde varit ett problem om man måste använda en viss typ av programmeringsspråk eller format på data som man inte är van vid."* (deltagare 1)

5.1.1.2 Programmering

Vikten av att vara konsekvent har vi tagit upp tidigare och i programmeringsmomentet visar det sig i form av hur strukturen för anrop och svar är designad. Vi gav problemet namnet: **Konsekvent struktur** och baserar det bland annat på följande utdrag:

> *"När jag väl genomfört och skapat förutsättningar för att hantera det första anropet och svaret förstod jag direkt vad som krävs för att använda funktioner."* (deltagare 1)

> *"Strukturen på anrop och svar var återkommande och uppbyggnaden av url:en gjorde att man snabbt kunde förstå vilken funktion man ville komma åt och vilken typ av objekt man hanterade."* (deltagare 2)

En artikel som varit underlag till vårt teoriavsnitt beskriver hur Microsoft har skapat olika personas baserat på olika beteenden hos systemutvecklare som använder deras utvecklingsprodukter (Clarke 2004). Med tanken på att utvecklare är olika personer med olika preferenser insåg vi att data vi samlat in tyder på att man vill kunna se **Status för skapade objekt** på olika vis när man debuggar system som jobbar mot webb-API:et.

> *"När jag hade skapat ett objekt behövde jag dess id för att genomföra nästa steg. Dock så hade jag inte sparat det lokalt och kunde inte se det genom webbgränssnittet. Under detta tidiga skede hade det underlättat att*

> *kunna se alla attribut som ett objekt har genom ett webbgränssnitt då jag inte skapat något testfall för att hämta ut id från recipients."* (deltagare 2)

Dokumentationen har gjort sig påmind flera gånger i vår insamlade data. **Bristfällig dokumentation** beskriver exempel på hur man i dokumentationen antyder något, men inte motsvarar förväntningarna.

> *"Det fanns beskriven struktur på JSON i exemplen men det fanns inte beskrivet när jag sökte på klasserna i dokumentationen. Däremot finns det en rubrik som heter JSON så jag antar att det kommer att skrivas i framtiden. Utan JSON-strukturen dokumenterad går det fortfarande att lösa problemen men det underlättar väldigt mycket om man kan se JSON-strukturen i webb-API-dokumentationen."* (deltagare 2)

Med begreppet **Masterdata** avser vi vilken data man på systemnivå bestämt samlas ihop för att motsvara en entitet.

> *"Under modelleringen var masterdatan tillräcklig, däremot kändes den otillräcklig när man kommit längre med programmeringen och ville se exakt struktur på t.ex. JSON. "* (deltagare 2)

Förståelse för begrepp i domänen handlar om mer än vad någonting heter, det är relaterat till semantik som tagits upp under tidigare problem men även en större betydelse än i det specifika sammanhanget.

> *"När jag hade skickat ett sms via systemet så fick jag inget sms i mobilen. Genom att logga in på webbgränssnittet såg jag att mina meddelanden hade "status: pending". Det fanns ingen förklaring för vad "pending" betyder eller vad som bör göras. Jag visste inte om jag skulle göra någonting, skulle någon annan göra något, var det MobileResponse det var fel på eller något annat system? Jag kunde heller inte söka på pending i dokumentationen."* (deltagare 2)

I exemplet ovan stöter vi på ett begrepp som vi tidigare fått viss kunskap om från systemet: Status. I detta fall har osäkerheten i att inte veta om man tolkar status rätt i dess sammanhang en stor roll. Anropet i sig fick en status som antyder att det var lyckat, men nu finns det en ny typ av användning för samma ord och ett nytt begrepp, att någonting är *pending*. Detta hade

undvikits om man lyckats förmedla förståelsen för begreppen i olika situationer i systemet. När denna process var *pending* behövde man förstå innebörden för begreppen relaterade till sms-meddelanden.

Om man behöver oroa sig för **Konsekvenser vid felanvändning** påverkas tryggheten i vad man vågar testa. Exempelvis ordbehandlingsprogram finns knappar för att ångra en handling vilket skapar trygghet. I vår data kunde vi se deltagare 2 hade funderingar kring det.

> *"När jag kommit en bra bit med programmeringen och började använda mer funktioner i webb-API:et uppstod ibland frågor om de saker jag gjorde eventuellt kunde bli fel och skapa problem i webb-API:et. Om man vet med sig att det inte kan få så stora konsekvenser om man skriver något fel så får man större självförtroende och vågar testa mer vilket är positivt."* (deltagare 2)

Användare behöver skapa klasser som motsvarar systemets klasser för att hantera dem i sitt system. Det är inte nödvändigt i alla situationer men är ett mer objektorienterat sätt att hantera det.

> *"Eftersom jag förstår att systemet har klasserna och använder dem tycker jag, som användare att det skulle tillföra mycket värde om systemet kunde ge mig samma klasser i någon form av bibliotek eller paket. Det finns funktionalitet för att översätta klasser mellan olika programmeringsspråk och om jag fick det direkt när jag började jobba mot webb-API:et så hade det sparat tid och underlättat förståelsen för hur systemet ser på världen."* (deltagare 1)

Som deltagare 1 tar upp finns det vissa tillvägagångssätt som skulle möjliggöra att systemet exponerar ut klasser som används inom webb-API:et.

Pålitligheten i systemet var en typ av problem som togs upp. Pålitlighet diskuterade vi och sa att det kunde innebära många olika saker. I detta fall så handlade det inte nödvändigtvis om driftsäkerhet, utan att man som användare kan lita på att systemet inte kommer att förändra de anrop som man skapar beroenden till. Symptomet visade sig enligt:

> *"Då jag jobbade mot webb-API:et i dess testmiljö (som motsvarar det som i det närmaste skall komma ut i produktionsmiljön) uppstod ett problem i*

> *att miljön jag jobbade mot, det vill säga hela webb-API:et uppdaterades. Det fick mig att inse att det finns ett problem som man kanske inte alltid tänker på om de inte visar sig. Men att systemet/webb-API:et är pålitligt, att man kan vara säker på att systemet/webb-API:et inte förändras när man väl gjort en implementation mot det."* (deltagare 1)

När man väl byggt ett system så att det är beroende av en viss del av ett webb-API, så är det inte rimligt att hantera alla eventuella framtida förändringar.

Det kan uppstå **Bieffekter av val som webb-API-utvecklarna tagit vid design av webb-API:et**, vilket sedan måste hanteras av en användare av webb-API:et.

> *"Då mitt arbetssätt var fokuserat på att göra minsta möjliga implementering, insåg jag att en användare av webb-API:et kommer att behöva hantera bieffekter av webb-API-designen. I detta fallet handlar det om att man valt att designa API:et som ett webb-API, vilket betyder att användare måste hantera felkoder som inte kommer från själva webb-API:et utan exempelvis även från webbservern. Om webb-API:et kraschar på servern skulle man kunna få ett fel från webbservern som säger att någonting kraschat. Helt enkelt felkoder i HTTP-svaret. Designvalet gör att användare av webb-API:et måste hantera HTTP-svar på olika sätt för att hålla sina system stabila vid händelse av webb-API-system-krasch."* (deltagare 1)

Det kan diskuteras om webbservern behöver betraktas som en del av API:et när det handlar om ett webb-API. Men oavsett hur man ser på det så blir den indirekta konsekvensen att en användare av webb-API:et måste hantera något som inte paketerats ihop med webb-API:et. Om webb-API:et följer gängse standarder så bör det finnas konsekventa och återkommande strukturer och betydelse.

5.2 Resultat från enkät

I detta avsnitt går vi igenom resultaten av enkäten men utan att lägga för stor vikt vid de faktiska procentuella antal svar på specifika frågor. Vi har istället som mål att summera upp svaren på frågorna i en mer berättande form. I resultatanalysen kopplas sedan följande resultat ihop med andra delar av datainsamlingsaktiviteter för uppsatsen.

Bakgrundsinformation för att sätta de svarande i en kontext

Samtliga svarande har angivit cirka 8 års erfarenhet av programmeraryrket, vi visste på förhand att de svarande är män i 30-40 års ålder som arbetar på samma företag och har viss geografisk spridning sett till arbetsplats. De svarande ansåg att de inte hade stor förståelse eller förkunskap om MobileResponse, däremot var de väl införstådda med och vana vid det programmeringsspråk och utvecklingsmiljö de använde vid tillfället. På frågan om modellering kunde ingen uttryckligen säga att de yrkesmässigt modellerar lösningen till problemet innan det angrips.

Upplevelsen av MobileResponse

Svarande är med liknande ordval överens om att deras förväntningar har mötts av MobileResponse. Dock har de angivet ett par saker som vi även identifierat som problem, exempelvis inkonsekvens och stabilitet.

De positiva aspekterna som nämns kan summeras till: Enkelt, genomtänkt och återkommande funktioner för anrop. Som kontrast till de positiva har man i andra frågor gett svar att de saker som orsakar problem kan summeras till: Avsaknad av viss funktion och dokumentation, vissa delar av systemet kan vara förvirrande. Likaså vissa begrepp och koncept. Delar av MobileResponse har stundtals slutat fungera.

En svarande ansåg att kontakt med de som utvecklat webb-API:et underlättat arbetet och gav möjlighet för användaren att påverka systemet under tiden det utvecklats. Dokumentation uppgavs underlätta, likaså underlättade det att komma åt bakomliggande databas. Felkoder som inte kunde peka på vad som användaren gjort fel var ett vanligt hinder.

Resultaten av gradering

Resultaten av graderingen hoppades vi skulle ge oss en mer tydlig bild om vår uppfattning delades. Sammanfattningsvis har de 15 graderingsfrågor fått svar som varit precis samma eller pekat åt samma håll i alla fall utom två. De två frågorna gällde dokumentation och dynamisk förståelse. Det uppkom inte några önskemål om att vi skulle förklara hur frågorna skulle tolkas, därmed skulle de svarandes egna tolkningar kunna förklara spridningen i de frågorna.

De frågorna där enigheten bland svarande var störst handlar om Bristfällig dokumentation, Pålitlighet i systemet, samt Bieffekter av val som API-utvecklarna tagit vid design av API:et. Alla de tre frågorna ansågs därmed vara på kritisk eller näst intill kritisk nivå.

6 Resultatanalys

I detta kapitel analyserar vi de problem som presenterades under förra kapitlet. Målet med vår analys är att hitta bakomliggande orsaker till problemen. Vi vill identifiera de bakomliggande orsakerna till problemen genom att använda Varför-Varför-Varför?-metoden som beskrivits under metodkapitlet.

6.1 Analys av användbarhetsproblem

Våra upptäckter från experimentet gav oss exempel på både konkreta problem men även symptom på god design som vi anser undanröjde problem för användaren.

Nedan är en lista med de namn vi gav på de problem och symptom vi presenterade under resultatkapitlet. Genom att ställa oss varför frågor har vi kunnat precisera mer exakt vad de bakomliggande orsakerna kan vara. Genom att genomföra varför-metoden får vi tydligare problemformulering och bakomliggande orsaker. En tydlig problemformulering kan även peka på lösningar.

Problemsymptom: Svårt att veta vart man ska börja

- *Varför var det svårt att veta var man ska börja?*

Eftersom jag inte hittade något avsnitt i dokumentationen om hur man kommer igång.

- *Varför hittade du inte det?*

Antingen lyckades jag inte söka igenom dokumentationen tillräckligt eller så fanns det inte.

I detta läge har vi kommit fram till ett svar med två potentiella orsaker till varför man inte enkelt kan komma igång: Det är svårt att söka i dokumentationen eller det saknas information i dokumentationen.

Forskning kring vad som gör ett API svårt att lära sig har visat att de flesta lär sig främst genom att läsa dokumentation (Robillard 2009). Det näst vanligaste inlärningssättet för systemutvecklare är att titta på kod-exempel. För att lösa problemet med att användaren har svårt att veta vart denne ska börja bör man erbjuda ett introduktionsexempel då detta ger användaren en utgångspunkt för att bygga förståelse.

Problemsymptom: Svårt att få dynamisk förståelse

- *Varför är det svårt att få dynamisk förståelse?*

Eftersom diagram och modeller som är beskrivna för webb-API:et är beskrivna på ett sådant sätt att de ger mer statisk information, alltså vad någonting är, istället för vad det gör.

- *Varför behöver du veta vad det gör?*

Jag vill veta hur objekten ska samarbeta för att uppnå det mål jag har.

- *Varför vill du veta hur objekten skall samarbeta?*

Jag behöver veta vilka klasser och objekt jag behöver använda, samt i vilken ordning?

- *Varför behöver du veta det?*

Då vet jag veta vilka anrop jag ska göra, samt i vilken ordning de skall göras för att jag ska uppnå målet med användningsfallet.

Enligt en undersökning om hur man beskriver och mäter ett API:s användbarhet med kognitiva dimensioner, visade det sig att systemutvecklare ofta kunde hitta rätt klass för sin uppgift i dokumentationen men fortsatte att leta ändå (Clarke 2005). Forskarna trodde dock inte att detta berodde på att dokumentationen behövde göra sambandet mellan uppgiften och de behövda klasserna tydligare, utan att abstraktionsnivån på klasserna i dokumentationen var för låg. Genom att skapa klasser med högre abstraktionsnivå som var tydligt relaterade till uppgiften programmerarna skulle utföra, minskades tiden det tog för att lyckas.

Statisk och dynamisk modellering av klasser har två olika syften. Medan de statiska modellerna visar klasspaket, klassnamn, attribut och metodsignaturer så visar dynamisk modellering logiken, koden (metodernas) beteende, vilka objekt som behövs och hur de samarbetar via meddelanden och metoder (Larman 2004). Problemet som uppstår när en utvecklare inte får en dynamisk förståelse för objekt påpekas även av Robillard (2009) när han intervjuar systemutvecklare mot API:er. Han undersökte vad deras största frustrationer är när det gäller anpassning av kod från Microsofts tutorials.

Symptom på bra design: Design av dokumentation hjälper

- *Varför hjälper det när dokumentationen är designad som i MobileResponse dokumentation?*

Bra indelad dokumentation gör att man hittar det man behöver med liten ansträngning.

- *Varför är det bra det att dokumentationen är indelad?*

Eftersom det gör det lättare att avgränsa och sortera ut det som är intressant för situationen.

Detta symptom på bra design förtydligas av nästa identifierade symptom på bra design. Motiveringen till båda symptomen ges efter nästa varför-analys.

Symptom på bra design: Exempel i dokumentationen med lagom stora steg

- *Varför är det lagom stora steg?*

Om den hade varit indelad i för stora steg så hade eventuella problem varit att man får för många steg, klasser, metoder att komma ihåg. Om den varit indelat i för små steg så hade eventuella problem varit att man tappar fokus för den övergripande logiken i processen.

Som utvecklare vill man oftast inte lära sig varenda klass, metod och hur de hänger ihop i ett enda svep. Liknande tankar formuleras av Clarke (2004), vars kognitiva dimensioner har en dimension som heter "Working Framework" och definieras som:

> *"The size of the conceptual chunk (developer working set) needed to work effectively."*
> *(Clarke 2004, s. 7)*

För att skriva bra exempel bör man alltså avväga hur pass många klasser man introducerar per steg.

Problemsymptom: Tester gick igenom även när det inte borde det

- *Varför gick tester igenom när det inte borde gjort det?*

Eftersom jag misstolkade JSON svaret och trodde att den operation jag försökte utföra hade lyckats.

- *Varför trodde du att den hade lyckats?*

Eftersom det stod "status : success" och ingen övrig information om vad för typ av status det var frågan om.

Problemet här kan inte härledas till endast en orsak. Hade systemutvecklaren haft mer tid, varit mer metodisk hade problemet kanske inte uppstått. Men eftersom felet ändå begicks så tyder det på att andra systemutvecklare skulle kunna göra samma sak. Man skulle kunna definiera problemet som tvetydighet i feedback. Om man namngav på följande vis:

”requestStatus : success” så kanske man kunde undvikt missförståndet. Nielsens heuristik *Error Prevention* skulle kunna vara relevant för detta problem då den menar att designern ska designa så att situationer där fel kan uppstå, inte uppstår (Rogers et al. 2011).

Symptom på bra design: Valfrihet i programmeringsspråk

- *Varför är det bra med frihet i val av programmeringsspråk?*

Användaren blir inte låst till att använda samma programmeringsspråk som de som utvecklar webb-API:et och det är bra.

- *Varför är det bra?*

Det gör att man som användare kan använda det språk som man är van vid och behärskar bäst.

Fördelarna med webb-API:er som gör anrop via HTTP och använder JSON som dataformat gäller både för systemutvecklaren som utvecklarna av webb-API:et. Att kunna använda det språk man känner sig mest bekväm med skulle kunna ses som att designen uppfyller Nielsens heuristik *Flexibility and Efficiency of use* om man modiferar den ursprungliga betydelsen till denna kontext.

Symptom på bra design: Konsekvent struktur är viktigt när man gör anrop

- *Varför är det viktigt med konsekvent struktur när man gör anrop?*

Programmeringsmässigt är det bra för det gör det lättare att skapa bra kod samt om man har gjort processen en gång så kan man förutsäga hur det ska se ut sen.

- *Varför är det viktigt att skapa bra kod och kunna förutsäga hur det ska se ut senare?*

Eftersom koden blir lättare att underhålla.

Fördelarna med konsekvent struktur är flera. Efter att ha fått förståelse för hur man gör det första anropet minskar inlärningsbelastningen för att lära sig använda nya funktioner (Henning 2009). Det minskar tiden man behöver lägga ner på framtida lärande då man enkelt kommer ihåg proceduren.

Problemsymptom: Kan inte se status för skapade objekt

- *Varför kunde jag inte se status?*

Det fanns inget verktyg för att se status.

- *Varför fanns det inget verktyg för att se status?*

För utvecklarna av webb-API:et har inte haft resurser eller prioriterat att skapa ett sådant verktyg. De kanske tycker att webb-API:et redan erbjuder sätt att se status för objekt.

Nielsens heuristik *Flexibility and efficiency of use* säger att acceleratorer (som är dolda för ovana användare) kan påskynda interaktionen för expertanvändare. I detta fall är det dock för ovana användare som en lösning i form av grafisk gränssnitt som ges av webb-API:et skulle påskynda interaktionen. Under ett tidigt utvecklingsskede när man inte har utvecklat något eget grafiskt gränssnitt för att se status på de objekt man skapat underlättar tillgång till ett enkelt standardgränssnitt som tillhandahålls av webb-API:et. För en del skulle det kunna vara användbart att se status på objekt som ett led i debuggningsstrategin.

Vi argumenterar för att detta problem även passar in under Nielsens heuristik *Visibility of system status* men då synligheten kan lösas genom att använda andra funktioner i webb-API:et för att få feedback (antingen i JSON eller i eget grafisk gränssnitt) så anser vi att det passar in bättre på förstnämnda heuristiken.

Problemsymptom: Bristfällig dokumentation

- *Varför är dokumentationen bristfällig?*

Dokumentation var påbörjad på vissa ställen men inte fullständig.

Även om problemet gick att lösa utan dokumentering så var det en mer tidskrävande process att behöva göra anrop för att få JSON-strukturen. Bristfällig eller icke befintlig dokumentation är ett vanligt problem vid inlärning och användning av API:er (Robillard 2009; Bloch 2008; DeLine & Robillard 2011). Bristfällig dokumentation kan som i det här fallet beröra en låg nivå av dokumentation eller i andra fall högre mer konceptuell nivå (DeLine & Robillard 2011).

Problemsymptom: Bristfälliga definitioner av koncept/masterdata

Under modelleringen var masterdatan tillräcklig, däremot kändes den otillräcklig när man kommit längre med programmeringen och ville se exakt struktur på t.ex. JSON.

- *Varför är definitionerna av koncepten bristfälliga?*

Eftersom man inte kan vara säker på vilken information som skall skickas med i olika anrop bara för att man har förstått grundkonceptet för objektet i fråga.

Likt problemet ovan saknas här dokumentation av lägre nivå (statisk modellering, datastruktur). Bloch (2008) anser att varje klass, metod och parameter bör dokumenteras. En annan tanke kring dokumentation på lägre nivå i samband med webb-API:er är att man bör ange vilken typ av attribut det är på tal om för klasser. Är attributet ålder en int eller string i webb-API:et?

Problemsymptom: Förståelse för problemområdet

- *Varför behöver du förstå problemområdet?*

För att förstå den feedback jag fick av systemet när jag försökte uppnå mitt mål.

- *Varför kunde du inte förstå feedbacken?*

Även om jag har en generell förståelse för ord och begrepp från feedbacken jag mottog, så hjälpte det mig inte i hur jag skulle gå tillväga för att kunna lösa det.

Svaret ovan ledde till två möjliga vägar att gå vidare med varför-kedjan.

- *Varför kunde du inte skaffa den specifika förståelsen för vad feedbacken betydde?*

Det fanns ingen dokumentation.

- *Varför visste du inte hur du skulle gå vidare?*

Feedbacken gav mig bara en status men inga ledtrådar hur jag ta mig ur det eller vart jag skulle vända mig.

Även om systemutvecklaren det i det här fallet fick feedback och förstod att meddelande hade status *pending* så gav det inte systemutvecklaren någon förståelse. Detta skulle kunna göras till ett problem med sin lösning i dokumentation, men om man istället stannar kvar vid just feedback så passar Nielsens heuristik *Help users recognize, diagnose, and recover from errors*.

Jag hade redan blivit varse om problemet, däremot så hade jag inte fått någon diagnos eller möjliga sätt att lösa problemet. Om feedbacken gett mer information än "status" pending som exempelvis att "tjänsten x är nere för tillfället" och "kontakta support" så hade jag fått diagnos

och ett möjligt sätt att gå vidare. Man skulle kunna se det som en form av dokumentationslösning men det är inte den traditionella typen av API-dokumentation i så fall.

Problemsymptom: Vågar inte testa funktioner

- *Varför vågar man inte testa funktioner i systemet?*

Eftersom jag inte vet vad som kan hända.

- *Varför vet du inte vad som kommer att hända?*

Eftersom jag gjorde ett anrop men kunde inte avgöra om den förväntade händelsen inträffade.

Bra system bör uppmuntra och tillåta utforskning genom att låta användaren experimentera (Tidwell 2011). Bloch (2008) presenterar en designriktlinje som går ut på att API:er ska vara enkla att använda och svåra att använda på fel sätt och nästan omöjliga att göra skada med.

Problemsymptom: Användare behöver skapa klasser som motsvarar systemets klasser

- *Varför behöver användaren skapa klasser som motsvarar klasserna i webb-API:et?*

För att det är den mest objektorienterade lösningen.

- *Varför måste det vara en objektorienterad lösning?*

Det måste inte vara en objektorienterad lösning men det underlättar för systemutvecklaren att dela samma definition av klasser som utvecklarna webb-API:et.

Genom att ge utvecklarna exakt samma klasser som används i webb-API:et skulle man även överföra webb-API-utvecklarnas syn på vad objekten i problemområdet. Det skulle kunna ses som ett steg att överbrygga eventuella tvetydigheter om vad en klass egentligen representerar.

Problemsymptom: Pålitligheten och oföränderlighet i systemet

- *Varför behöver systemet vara pålitligt och oföränderligt?*

Eftersom jag som systemutvecklare har byggt mitt system utifrån de rådande överenskommelser och omständigheter som råder vid situationen för systemutvecklingen.

- *Varför har du byggt det så?*

Eftersom det inte är rationellt och rimligt att ta med alla eventuella framtida förändringar som kan ske. Ingen har de resurserna att tillverka sådana system.

Detta är svårt att relatera till en heuristik vid ett tillfälle som utvärderas då det är ett problem som uppstår över en längre tid. Om en heuristik formuleras utefter problem av denna typ är en förutsättning för att upptäcka denna typ av problem att webb-API:et förändrats över tid. Den skulle alltså inte vara relevant vid en första heuristisk utvärdering av ett webb-API.

Problemsymptom: Får hantera bieffekter av val som webb-API-utvecklarna tagit vid design av webb-API:et

- *Varför måste du hantera bieffekter av webb-API-utvecklarnas beslut?*

Eftersom mitt system måste kunna klara av tillfälligheter och fel som orsakas av system som det beror på.

- *Varför måste det klara av fel i andra system?*

Eftersom annars kraschar mitt system utan att det kan förklara för användaren vad denne gjort fel.

- *Varför måste du förklara det för användaren?*

Eftersom annars har jag bara låtit användbarhetsproblemet passera mitt system och hamnar i knäet på användaren.

Det vi kommit fram till med den här varför-kedjan är viktigt men kan inte härledas till en enskild heuristik. Det vi identifierar är istället att man bör designa så att man inte låter andras problem skickas obemärkt vidare till nästa led av användare.

6.2 Analys av enkätsvar

Rogers et al. (2011) förklarar att man kan behöva rensa upp i enkätsvar då man misstänker att någon av de svarande inte förstått frågan och att svaret som givits inte tycks höra hemma bland de andra svaren. I resultaten som presenterats under rubriker ovan fann vi inget som tyder på missförstånd. I enkäten hade vi valt korta formuleringar på frågor och i vissa fall endast ett ord att ta ställning till. Det var en osäkerhet som vi funderade över när vi tog fram enkäten men valde att hantera genom att vara tillgängliga för samtliga svarande då enkäten besvarades.

Faktorer som var relevanta att ha i åtanke när vi genomförde analysen var att två tredjedelar av de som svarat på enkäten befunnit sig fysiskt på samma plats som utvecklarna av MobileResponse. I och med att de har befunnit sig på samma plats som utvecklarna av MobileResponse så skulle kunna innebära de påverkat utvecklingen av MobileResponse. En annan aspekt är att de genom sin närvaro och den informella kommunikation som sker på en arbetsplats fått en djupare förtåelse för problemområdet och MobileResponse som helhet.

Vi gjorde valet att presentera resultaten i löpande text då vi ansåg det bättre uttryckte resultatet än diagram och grafiska representationer när vi endast haft tre svarande. Vad vi kom fram till genom att bearbeta resultaten till en form av löpande text, är att de svarande har själva använt uttryck som direkt kan kopplas till de resultat som vi funnit genom experiment och litteratursökningar. Exempel på det är stabilitet, konsekvent/återkommande, dokumentation och kod-exempel.

Vidare hade vi förstått av Patel & Davidson (2011) och Rogers et al. (2011) att man kan välja hur man graderar frågor, genom att vända på betydelsen av högsta graderingen för att ringa in åsikt och liknande. Vi använde denna teknik för graderingsfrågorna kring dokumentation. Vi har två frågor där svarande skall ta ställning till dokumentation, den ena frågan handlade om dokumentation och den andra om bristfällig eller avsaknad av dokumentation. Vi såg att resultatet blev att de svarande gav olika graderingar beroende på hur frågan ställdes. Anledningen till att vi valt att ställa den frågan på två sätt är eftersom vi upptäckt och identifierat frågan på flera olika ställen i litteratur och genom experimentet.

De problem och symptom på bra design som vi genom datainsamling kommit fram till före enkäten valde vi att presentera som 15 problem som de svarande skulle gradera. Vår tolkning av resultatet från graderingarna var: Där resultatet var 4 eller 5 vilket motsvarar kritiskt och nära på det, så håller de svarande med det vi kommit fram till. Om svaret är 3, det vill säga koncentrerat till mitten, så är tolkningen att vi inte får bekräftat medhåll, men inte heller något tydligt ställningstagande mot vår föreslagna problem eller exempel på bra design. Om svaret är 1 eller 2 har vi tolkat det som att svarande anser problemet vi presenterat inte stämmer eller är viktigt.

Problem och symptom på bra design där vi fick medhåll

- *Svårt att veta vart man ska börja*
- *Dokumentation*

- *Bra exempel*
- *Namngivning som är inkonsekvent och tvetydighet*
- *Konsekvent struktur*
- *Bristfällig dokumentation*
- *Konsekvenser vid felanvändning*
- *Pålitlighet i systemet*
- *Bieffekter av val som API-utvecklarna tagit vid design av API:et*

Problem och symptom på bra design där respondenterna var neutrala

- *Svårt dynamisk förståelse*
- *Valfrihet för användaren*
- *Masterdata*
- *Användare behöver skapa klasser som motsvarar systemets klasser*
- *Förståelse för begrepp i domänen*

Problem och symptom på bra design där vi inte fick medhåll från respondenterna

- *Status för skapade objekt*

Sammanfattningsvis anser vi att vi har träffat rätt med identifierade problem och symptom på bra design. Något som tydligt utmärker detta är hur samtliga respondenter använder olika programmeringsspråk men anger att det inte spelar någon roll om man har den friheten. Respondenterna har angivit att det är deras huvudspråk eller att det är i det språket de har störst erfarenhet. Dålig design märks tydligt medan bra design utmärker sig genom att inte synas. I resultatet av enkäten kan vi utläsa att systemutvecklarna har dragit fördel av friheten att jobba med sitt huvudprogrammeringsspråk. Samtidigt har de inte lagt märkte till att detta möjliggjordes genom god design.

6.3 Heuristiker för metoden heuristisk utvärdering av webb-API:er

För att göra våra heuristiker användbara vid utvärdering av användbarhet hos webb-API:er begränsar vi oss till sju stycken heuristiker då detta nummer är tillräckligt högt för att täcka in resonabelt många problem, men även tillräckligt lågt för att utvärderaren ska kunna ha dem i huvudet under en utvärderingssession. Vi motiverar vårt val av följande sju heuristiker med förklaringen att de är mest frekvent förekommande både i undersökningens insamlade data och insamlad litteratur.

Erbjud användaren ett fotfäste att gå vidare ifrån

Denna heuristik innebär att när användaren inte vet hur eller vad som ska göras, ska denne erbjudas en tydlig utgångspunkt i form av exempel som är uppdelad i lagom stora steg. Syftet med exemplet är dels att introducera klasser, men även den högre konceptuella logiken för hur de är menade att användas.

Motivering: Vi identifierade i experimentet svårigheter i att veta hur man ska börja använda webb-API:et. Detta problem bekräftades i vår enkät, men i den undersökta litteraturen saknas fokus på detta moment. Detta är relaterat till inlärningsproblematik vilket tas upp av (Robillard 2009; DeLine & Robillard 2011; Clarke 2004).

Ge både statisk och dynamisk förståelse för systemet

Hjälp användaren att inte bara förstå datastrukturer, utan även hur objekten ska användas tillsammans. Förståelsen kan ges genom exempel, UML-tillståndsdiagram.

Motivering: Denna heuristik identifierades under experimentet och var framförallt viktigt under modelleringsfaser. I enkäten fick vi bekräftelse då deras arbetssätt inte inkluderade modellering. Larman (2004) menar att dynamisk förståelse behövs för att förstå vad som händer och hur man ska använda klasserna tillsammans.

Anpassa abstraktionsnivå efter användningsscenario

Även om användaren har hittat rätt objekt för att klara sin uppgift är det inte alltid säkert att de inser detta. Genom att försöka skapa klasser med hög abstraktionsnivå kan detta problem undvikas.

Motivering: Denna heuristik kan vid en första anblick påminna om förra heuristiken. Skillnaden är att även om det finns UML-diagram för hur man använder olika klasser i ett specifikt syfte, så är det inte säkert att man ens hittar rätt klasser då namnen på klasserna inte ger tillräcklig förståelse. Forskning kring användbarhet och API:er pekar på att det är enklare att använda API:er där klasser har namn och metoder som är tydligt relaterade till problemområdet (Clarke 2005).

Utförlig och väl designad dokumentation

Erbjud användaren förståelse på låg nivå (datastrukturer) men även på högre nivå (konceptuell). Ordna dokumentationen hierarkiskt och gör den sökbar.

Motivering: Flera identifierade problem motiverar den här heuristiken: Svårt att veta var man ska börja, svårt att få dynamisk förståelse, dokumentation, bra exempel, bristfällig dokumentation, förståelse för begrepp i domänen. Enkätsvaren talade sitt tydliga språk och motiverar att dokumentation är viktigt. Ett genomgående tema i den litteraturinsamling vi gjorde var att dokumentation är väldigt viktigt.

Konsekvent struktur

Ha en konsekvent struktur för anrop, både vad gäller hur man anropar och vilken data och parametrar som krävs för liknande anrop. Om användaren har gjort anrop tidigare så ska denne genom erfarenhet förstå hur man gör liknande anrop.

Motivering: I enkäten säger alla svar att vara konsekvent med namngivning och struktur är kritiskt det vill säga viktigast. I experimentet hade vi identifierat detta som ett symptom på bra design då vi lätt kunde använda många funktioner då vi lärt oss använda en av dem. Under vår litteraturinsamling såg vi stöd för denna heuristik hos Henning (2009) där han beskriver en designriktlinje som säger att API:er ska vara ergonomiska.

Förhindra fel genom tydlighet

Generellt språkbruk exempelvis status inbjuder till tvetydighet då man inte tillhandahåller information i vilken kontext ordet används. Eftersom "dialogen" mellan användaren och webb-API:et sker genom anrop och svar måste svaren vara väldigt tydliga med vad de säger.

Motivering: Vi relaterar följande användbarhetsproblem till denna heuristik: Namngivning som är inkonsekvent och tvetydig, status för skapade objekt, användare behöver skapa klasser som motsvarar systemets klasser. Det här blir en sammanslagning av Nielsens (1995) ursprungliga heuristiker av som berör kommunikation med användaren, exempelvis *Match between system and the real world*.

Flexibilitet och effektivitet

Erbjud acceleratorer, verktyg och genvägar, som gör lärandet och användandet snabbare både för experter samt nybörjare. Använd protokoll och datastrukturer som tillåter användaren att själv välja programmeringsspråk. Tillhandahåll domänklasserna i bibliotek så användaren inte behöver gissa och skapa egna.

Motivering: Under experimentet upptäckte vi att administratörsgränssnittet fungerade som en accelerator eller ett verktyg där vi kunde använda det för att underlätta systemutvecklingen. Den här heuristiken är en anpassning av Nielsens (1995) heuristik *Flexibility and Efficiency of use*.

6.4 Utvärderingsmetod för våra heuristiker

Som tidigare nämnt har den ursprungliga heuristiska utvärdering syftet att utvärdera visuella gränssnitt. Heuristisk utvärdering beskrivs som sessioner på en till två timmar där experten går igenom ett gränssnitt flera gånger för att hitta användningsproblem. Om det är ett större gränssnitt får man öka tiden eller ha flera sessioner där man fokuserar på olika delar av gränssnittet. Under utvärderingen ska hela gränssnittet granskas åtminstonde två gånger och alla användningsproblem som identiferas ska kopplas till någon av heuristikerna. Syftet med första granskningen är att ge utvärderaren en känsla för helheten i gränssnittet. Andra granskningen fokusering på detaljerna och hur de passar in i den större bilden. Utvärderaren bestämmer själv i vilken ordning gränssnittet ska granskas (Nielsen 1995).

En av svårigheterna med att anpassa heuristisk utvärdering för webb-API:er är hur man ska genomföra utvärderingssessioner. Var börjar interaktionen och vart tar den slut? Vid utvärdering av ett grafiskt gränssnitt finns en tydlighet i vad som utvärderas eftersom det finns framför utvärderarens ögon. Men vad är motsvarigheterna till en struktur av en meny, positionering av ett par knappar eller en logik vid en sekvens av skärmbilder då man utvärderar ett webb-API?

Rogers et al (2011) ger ett exempel på hur utvärdering av ett nytt system för socialt nätverkande går till. Utvärderaren har som mål att lägga till en vän till sin vänskapskrets. Alla steg som måste tas för att nå målet utvärderas. För att få någon slags struktur på utvärderingsprocessen vid utvärdering av webb-API:er måste liknande mål finnas. Inför en utvärderingssession bör det skapas scenarier eller användningsfall som ger tydliga mål för vad det är som utvärderaren förväntas uppnå. Genom att skapa användningsfall eller scenarier som kräver att utvärderaren använder många klasser och funktioner i systemet kan fler användningsproblem upptäckas.

Varje systemutvecklare har till viss del en egen process för problemlösning och programmering. Faktorer som spelar in är vilken typ av utbildning man har, vilka erfarenheter man har och vilken programmeringsmetodik man följer. Liknande resonemang kan sägas vara applicerbara på utvärderare av grafiska gränssnitt då symboler och färger kan betyda

olika saker beroende på vem du frågar. Detta fenomen är relevant vid utvärderingssammanhang då du kommer att få olika användbarhetsproblem beroende på vem som utför utvärderingen. Nielsen (1995) föreslår minst tre, men helst fem utvärderare för att utvärdera ett gränssnitt. Även om det upptäcks fler användningsproblem om det är fler än fem utvärderare så avtar antalet identifierade problem exponentiellt. Det blir då en avvägningsfråga om hur mycket resurser man har till förfogande gentemot hur många problem man vill upptäcka.

I en situation där man utvecklar ett webb-API och vill utvärdera dess användbarhet bör man alltså göra följande förberedelser:

- Sätt ihop en grupp av experter med varierande bakgrund
- Skapa användningsfall/scenarier där huvudfunktionalitet kommer testas, så mycket som möjligt ska täckas av sessionerna
- Så fort experterna fått användningsfallet/scenariot ska de skriva ner de användingsproblem de upplever och knyta dem till någon heuristik. En del problem kan kanske knytas till flera heuristiker
- Det är inte fel att påpeka fel som inte faller under någon heuristik. Det kan tyda på att heuristikerna behöver modifieras eller någon heuristik behöver läggas till

Om vi hade genomfört heuristisk utvärdering av MobileResponse hade vi kunnat använda vårt användningsfall som grund för heuristisk utvärdering.

7. Diskussion/Slutsats

Under rubrikerna nedan diskuterar vi och presenterar de slutsatser vi dragit från projektet.

7.1 Diskussion

Under experimentet var vårt syfte att samla in data som pekade på problem eller lösningar som möjliggjort att många problem kunde undvikas. Det kunde ibland vara svårt att veta om problemet låg hos den som utförde experimentet eller hos själva webb-API:et. Vi anser att vi under analysen kunde resonera kring de problem vi skrivit ner och fånga upp det som var relaterat till hur webb-API:et och dess dokumentation.

En annan reflektion kring experimentet och dess data är huruvida det utvecklingstillstånd som webb-API:et var under vårt experiment. Eftersom det fortfarande var under utveckling och ej öppet för allmänheten kan det vara så att en del av de problem som vi upptäckte inte skulle

upptäcks om vi genomfört experimentet under ett senare moment i webb-API:ets utveckling. Däremot hade vi kanske upptäckt andra saker som uppstått.

Vid analysen av den insamlade datan var vår ansats att komma fram till orsaker till de problem vi identifierat. I vissa fall kunde något som först såg ut att vara ett problem egentligen vara ett symptom för ett problem. Vi har försökt att analysera och kategorisa problem så att vi beskrev dem på en lagom liknande nivå. Detta var för att våra heuristiker ska bli så användbara som möjligt med att identifiera så många problem som möjligt utan att förvandlas till en checklista.

Grill et al. (2012) presenterade en lista med heuristiker för utvärdering av API:er. Även om deras heuristiker borde vara liknande vår lista så finns det två faktorer som gör att våra heuristiker ser annorlunda ut. Den första faktorn är att deras heuristiker handlar om API:er medan våra handlar om webb-API:er. Ett exempel på hur det skiljer sig här är att deras heuristik *Factory Pattern* som handlar om att man inte bör använda det designmönstret. Denna heuristik går inte att applicera i en kontext av webb-API:er. Den andra faktorn är att vi inte ville ha heuristiker av denna typ då vi anser att den handlar endast om en enda typ av problem, inte en arketyp av problem. Om en heuristik endast används för att checka av ett specifikt problem blir den mindre användbar och passar bättre som ett kriterie i en större checklista.

Om vi hade haft mer tid och resurser hade vi kunnat välja experimentdeltagare som haft mer varierande bakgrunder och angreppsmetoder för systemutveckling. Detta hade kunnat resultera i en ännu mer komplett lista av användningsproblem och gjort resultatet mer generaliserbart. Vi hade även kunnat genomföra experiment på olika webb-API:er för att se om det finns olika typer av användningsproblem som inte är generella för denna typen av webb-API:er. Detta hade kunnat leda till mer precisa formuleringar för våra heuristiker. Vi anser dock fortfarande att vi fångade upp användningsproblem på olika nivåer då erfarenheten av programmering och MobileResponse i sig självt varierade hos deltagarna.

7.2 Slutsats

Vårt syfte med denna uppsats var att presentera en lista med användbara heuristiker för utvärdering av webb-API:er. Genom att kombinera teori kring användbarhet och utvärdering av webb-API:er med data från vårt egna experiment och enkätstudie anser vi att de heuristiker vi presenterar har en teoretisk och praktisk grund.

Ytterligare forskning skulle kunna bekräfta hur användbara de är och eventuellt modifiera listan lite då vår fallstudie inte kan garantera ett resultat som är helt generaliserbart.

7.3 Förslag för framtida forskning

För bekräfta och se hur pass användbara våra heuristiker är skulle man kunna genomföra en undersökning. Det skulle krävas en situation där det finns ett webb-API och man redan har identifierat ett antal användbarhetssproblem, exempelvis alla problem som kommit in till support. Undersökningen skulle då ske genom att försökspersoner använder våra heuristiker för att se hur många problem de lyckas identifiera. Resultatet skulle visa på hur stor del av problemen som våra heuristiker hjälper till att identifiera.

Litteraturförteckning

Beaton, J., Myers, B. A., Stylos, J., Jeong, S. Y., & Xie, Y. C. (2008). Usability Evaluation for Enterprise SOA APIs. Leipzig: ACM.

Beck, K. (2002). *Test Driven Development: By Example* (1 uppl.). Addison-Wesley.

Bloch, J. (2008). *Joshua Bloch: Bumper-Sticker API Design*. Hämtat från http://www.infoq.com/articles/API-Design-Joshua-Bloch den 14 05 2013

Bodle, R. (2011). Regimes of sharing. *Information, Communication & Society, 14*(3), 320-337.

Clarke, S. (2004). Measuring API Usability. *Dr. Dobb's Journal Special Windows/.NET Supplement*, 6-9.

Clarke, S. (Maj 2005). Describing and Measuring API Usability with the Cognitive Dimensions. Dallas: IEEE Computer Society.

DeLine, R., & Robillard, M. P. (2011). A field study of API learning obstacles. *Empirical Software Engineering , 16*(6), 703-732.

Ellis, B., Stylos, J., & Myers, B. (2007). ICSE'07. IEEE Computer Society.

Grill, T., Polacek, O., & Tscheligi, M. (2012). Methods towards API Usability: A Structural Analysis of Usability Problem Categories. *HCSE'12 Proceedings of the 4th international conference on Human-Centered Software Engineering*, 164-180.

Henning, M. (2009). API Design Matters. *Communications of the ACM, 52*(5), 46-56.

Larman, C. (2004). *Applying UML And Patterns* (3 uppl.). Prentice Hall.

Letia, I. A., & Chifu, E. S. (2011). A Neural Model for Semantically Enhancing Web APIs. Cluj-Napoca: Intelligent Computer Communication and Processing.

Löwgren, J., & Stolterman, E. (2004). *Design av Informationsteknik* (2 uppl.). Lund: Studentlitteratur AB.

Maleshkova, M., Pedrinaci, C., & Domingue, J. (2010). InvestigatingWeb APIs on the WorldWideWeb. Milton Keynes: IEEE Computer Society.

Martin, R. C. (2008). *Clean Code.* Prentice Hall.

Nielsen, J. (den 1 Januari 1995). *Nielsen Norman Group.* Hämtat från http://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/ den 16 Maj 2013

O'Callaghan, P. (2010). The API walkthrough method: a lightweight method for getting early feedback about an API. New York: ACM.

Parnin, C., & Treude, C. (2011). Measuring API Documentation on the Web. Waikiki: ACM.

Patel, R., & Davidson, B. (2011). *Forskningsmetodikens grunder* (4:e uppl.). Lund: Studentlitteratur AB.

Robillard, M. P. (November/December 2009). What Makes APIs Hard to Learn? Answers from Developers. *IEEE Software, 26*(6), 27-34.

Rogers, Y., Sharp, H., & Preece, J. (2011). *Interaction Design - Beyond Human Computer Interaction* (3:e uppl.). Chichester: John Wiley & Sons .

Stylos, J., & Clarke, S. (2007). Usability Implications of Requiring Parameters in Objects' Constructors. Minneapolis: IEEE.

Tidwell, J. (2011). *Designing Interfaces* (2:a uppl.). Sebastopol: O'Reilly Media.

Turban, E., Sharda, R., Delen, D., & King, D. (2011). *Business Intelligence: A Managerial Approach* (2:a uppl.). Upper Saddle River: Pearson Education.

Zang, N., Rosson, M. B., & Nasser, V. (2008). Mashups: Who? What? Why? Florence: ACM.

Zibran, M. F. (2008). What Makes APIs Difficult to Use? *International Journal of Computer Science and Network Security, 8*(4), 255-261.

Bilaga 1 - Experiment dagboksmall

1. Vid första mötet på första dagen planerar vi mål och delmål utifrån användningsfall
2. Påbörja första användningsfallet och modellerar på en tänkbar lösning med hjälp av UML (Unified modeling language)
3. Vi delar upp lösningen i olika mindre delmål
4. Under modellering och programmering av varje delmål antecknar vi de saker som fungerade bra och de saker som fungerade mindre bra. Vi använder oss här av "varför-varför-varför?"-metoden för att komma till botten av vad problemet bestod av.

Under det fjärde steget besvarade vi frågorna nedan för att dokumentera arbetet.

Modellering

- Vad behöver jag göra mot webb-API:et för att lösa min uppgift/mitt case?
- Vilka klasser behöver jag skapa?
- Vilka anrop behöver jag göra mot webb-API:et? (hur ska jag prata med webb-API:et, skall det anropas på samma sätt som tidigare, vilket är sökvägen/anropet)
- Vad gör webb-API:et och vad gör jag? (input (vad skickar jag in) och output (vad får jag tillbaka och vad för annan output kommer från systemet)).
- Förklara stegvis hur du ska lösa uppgiften.

Programmering

Skriva test som täcker allt i den stegvisa förklaringen.

Eftertanke

Efter varje test beskriva: Vilket var testet? Vilka inblandade klasser? Var det lätt/svårt? Varför var det lätt/svårt?

Bilaga 2 - Enkät

Bakgrund

- Hur länge har du programmerat professionellt?
- Hur stor förkunskap hade du gällande webb-API:et, det programmeringsspråk som du använde?
- Vilka programmeringsspråk använder du helst?
- Brukar du modellera innan du kodar?

Programmering med webb-API:et

- Hur bra möter webb-API:et dina förväntningar på det?
- Vad upplevde du var de mesta positiva aspekterna av webb-API:et?
- Fanns det något med webb-API:et som gjorde att du kunde slutföra uppgifter snabbare än med andra webb-API:er? (tillgång till webbgränssnitt, bra dokumentation)
- Upplevde du saker med webb-API:et (dokumentation, struktur, funktionellt) som gav upphov till problem?
- Uppstod det några problem vid användandet av webb-API:et?

Deltagarna fick även gradera de problem som vi hittat under vårt experiment på en skala 1-5 där ett var ”Ej viktigt” och fem ”Kritiskt”.