Mobile-first eller Desktop-first, en studie om utvecklingslösningar för den responsiva webben

Eduardo Castaneda

2 maj 2013

Innehåll

1 Bakgrund

Mobilutvecklingen har under de senaste åren gått fort framåt, vilket har lett till att mobiler numera utvecklas till att fungera likt en handburen minidator. Huvudsyftet med mobiler är inte längre att bara kunna kommunicera med andra, utan även att bland annat ha möjligheten att få ut information genom webben. Att en websida inte längre endast ses i en skrivbordsskärm har gjort att nya webblösningar krävs. En utvecklare måste ha i åtanke att webbdesignen bör fungera för en användare som sitter på kontoret och läser sidan från en skrivbordsskärm, likaväl för en användare som sitter på tåget och läser sidan från mobilen. *Responsive Web Design* är en lösning vilken beroende på skärmstorlek renderar samma sida på olika sätt. I responsive web design har det fokuserats på två olika metoder, *Mobile-first* och *Desktop-first*. Metoderna grundar sig på att utveckling sker för en typ av skärm först för att sedan utifrån det utveckla så att den sidan även passar för andra skärmar. Det finns mycket kunskap om metoderna var för sig, men inget underlag för valet av metod i olika situationer och inom olika områden. Mobilanvändarna ökar för varje dag medan skrivbordsanvändare behåller sina användare med en viss minskning. Det största antalet är användare utav båda, vilket kräver kunskap om när metoderna appliceras bäst för att få effektivare lösningar inom webutveckling.

1.1 Mobilanvändare

Sedan 2007 då Apple visade upp den första versionen av iPhone har utvecklingen för *smartphones* eskalerat markant. Smartphones ger dig möjligheten att utföra datorliknande handlingar som att få ut information från webben och använda nätbaserade applikationer. Tidigare mobiler har delvis haft den funktionen men fokus mestadels har varit på musikspelare och kamera, då surfande gick trögt och var icke användarvänlig. Med smartphones, vilket innebar större pekskärmar, gjorde åtkomsten till internet på mobilen mer användarvänlig och har under de senaste åren utvecklats till en oundviklig funktion i dagens mobiler.

Dagens användare av mobiler är mellan 15-79 år gamla, av dessa är det cirka 40 % som använder sig av mobilt internet, i jämförelse med 2008 har det ökat med över 50 %. Telefonoperatörer har anpassat sig till den nya marknaden och erbjuder abonnemanger vilka innehåller en fast kostnad för surf på ett 3G nät. Då mobilsurf anses som en nödvändig funktion, sätts krav hos mobiloperatörer i form av att hög åtkomlighet och en hög hastighet för 3G nätet. Effekten av att mobilanvändarnas åtkomst till internet ökar, leder till mobilt internet används mer och äldre tjänster ersätts. En tidning i tunnelbanan är inte alls lika vanlig nu som för fem år sen, tidtabell vid busshållplatsen är inte enda sättet att få ut information om busstider och att checka-in inför en flygresa behöver inte nödvändigtvis ske via en check-in disk.

Konkurrensen som finns i dagens mobilmarknad har tvingat ledande företag att skapa mobiler med ny teknik till allt lägre priser. Pris, tillgänglighet och användbarhet har gjort att användning av internet via mobilen i världen närmar sig antalet användare av internet

via en dator[fotnot 1]. Analytiker som förutspådde mobilen till att slå i marknaden har i dagsläget fått det bekräftat och förutspår att användare av mobilt internet i världen kommer att passera antalet användare av Desktop internet [fotnot 2] under år 2014. Detta kräver från webbplatser att följa målgruppen användare och anpassa webbsidor utifrån de enheter som webbsidor ses ifrån, vilket till stor del är från mobilen.

1.2 Desktop användare

Att mobilanvändare ökar för varje dag har till viss del ett samband med minskningen av antalet *desktop* användare. Men datorer har i dagsläget funktioner som gör det osannolikt för mobiler att helt kunna ersätta datorer, funktioner som kräver att man sitter vid en dator och på så sätt sker informationssökandet i webben via desktop.

Webbdesignen på en mobil är kompakt och uppfyller den nödvändigaste användbarheten. På desktop är designen mer informationsrikt, vilket ger en större inblick till webbsidans innehåll och en mer simpel navigering på webbsidan. Det gör att en användare beroende på komplexitet och säkerhet av handlingen väljer att utföra den via en desktop, än via mobilen. En sådan handling kan t.ex. vara bankärenden eller webshopping. Om bankärendet gäller en översikt av saldo i kontot eller överföring mellan egna konton, anses mobilen vara en smidig enhet att använda sig av. Däremot om handlingen innefattar att betala räkningar eller överföra stora summor pengar finns behovet av att utföra dessa handlingar via desktop, då det ger en större säkerhet och en mer simpel navigering på websidan. Även webshop faller i samma kategori, då användare väljer att söka information om produkten via mobilen, men väljer sedan att utföra köpet via desktop[fotnot 3]. Detta tyder på att användare av mobilt- eller desktop internet inte är endera, utan är användare utav båda, beroende på situation, miljö och kontext på webbsidan.

Det finns ingen grundlig faktum som tyder på att mobilt internet kommer att ta över all desktop internet användande, endast att de kan bli fler. Därför krävs det att man förstorar vyerna kring webutveckling och lösningar som gynnar både mobil- och desktop användning av internet. För även om mobila användare blir fler, så går det inte att förbise desktop användare.

1.3 Responsive Web Design

Responsive web design är ett koncept som innebär att gränssnittet på en websida ändras beroende på skärmstorlek, vilket således ger möjligheten att ha olika layout på en och samma websida anpassat efter en enhet. Konceptet definierades av Ethan Marcotte (2010) i en artikel kallat *A List Apart*, som sedan blev en del av boken *"responsive web design"* där teorier och praktiska exempel används för att förklara begreppet.

Syftet med Responsive Web Design är att kunna rendera olika delar av sidan beroende på skärmstorlek för att ge en optimal vy för den enhet webbsidan ses ifrån. Om tre element

renderas bredvid varandra på en desktop sida, så vore det optimala för en mobilsida att kunna renderar elementen under varandra och även skala ner de till en rimlig storlek. På så sätt försvinner inte artiklarna från webläsarfönstret eller tar för stor plats på skärmen.

Då endast layouten på sidan ändras behövs inga särskilda versioner för varje enhet, vilket gör det möjligt för en webbutvecklare att på ett enkelt sätt utföra en ändring i en fil, istället för antalet versioner. Däremot krävs en flexibel grundlayout för att responsive web design skall fungera, vilket enligt Ethan görs genom tre grundtekniker, *Fluid Grid*, *Fluid Images* och *Media Queries*.

1.3.1 Fluid Grid

På en webbsida kan storleken på element definieras på olika sätt. Ett vanligt sätt att definiera element är med bredd och höjd i pixlar. När det definieras i pixlar betyder det att storleken på elementet är förutbestämd vare sig upplösning eller storlek på skärm. I tidigare skede informerade webbutvecklare användare vilken upplösning som renderade sidan på bästa sätt och sedan var upp till användaren att ändra upplösning på skärmen för att få den önskade layouten på sidan. I dagsläget finns många olika skärmar, och många olika alternativ till upplösning vilket gör en förutbestämd storlek inte lika optimalt.

Fluid Grid är en teknik vilken använder sig av procentsatser istället för pixlar vid definering av ett elements storlek. När procentsatser används förstoras eller förminskas elementets storlek relativt till webbläsarfönstrets höjd och bredd, vilket gör att sidan istället anpassas efter användaren. Om ett element skall täcka hela bredden på skärmen, sätts bredden till 100 %, en fjärde del av skärmen, 25 % osv. För att kunna bestämma elementets storlek i procentform härleder Ethan Marcotte ett sätt vilket innebär att, i pixelstorlek, dividera elementets storlek med behållarens storlek. Det ger ett resultat i procent, där elementet storlek ändras relativt till hållarens bredd och höjd.

1.3.2 Fluid Images

Fluid Images bygger på samma princip som Fluid Grid, däremot är lösningen mer komplex då det innefattar skalning av bilder. Om en bild skalas på fel sätt leder detta till att bilden ser för utsträckt eller för intryckt ut, men om skalning inte görs, kan en bild i princip ta upp all plats på webbsidan och även försvinna ut i kanterna. Inom responsive web design används Fluid Images, vilket innebär en flexible hållare med en önskad storlek, med en bild anpassad till hållaren.

Lösningen som tas upp av Ethan Marcotte är användningen av egenskapen "**max-width: 100 %**" på bilden. Med egenskapen "**max-width:100 %**" talar man om för webbläsaren att bilden skall skalas efter hållaren men endast till max av den originella storleken på bilden. Egenskapen **"width:100%"** innebär att bilden bara får anpassa sig helt efter hållaren, samtidigt som den får en normal skalning när storleken på webbläsaren minskar och hållaren

krymper. Egenskapen **Max** i **"max-width"** innebär att bilden aldrig blir större än bildens verkliga storlek. Om *"max"* inte sätts, finns risk att bilden skalas ut, vilket gör pixelkanter synliga. Däremot krävs det att man har en bild som redan från början är av önskad storlek. En för liten bild kommer den inte att förstoras då den som störst kan vara 100 % av bildens ursprungliga storlek. Detta är ett sätt att ta använda sig av Fluid Images som fungerar för responsiva webbsidor, det kan uppstå andra komplikationer i form av att bilden inte skalas enligt önskemål där andra lösningar krävs, men grundtanken för att upprätthålla fluid images är att bilder skalas efter webbläsarfönstrets storlek.

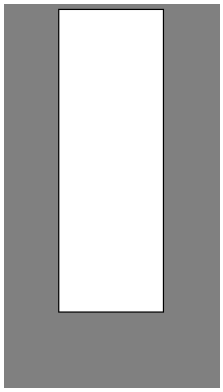
1.3.3 Media Queries

Fluid Grid och Fluid Images skapar tillsammans en del av en responsiv sida då storleken på element i sidan renderas utifrån storleken på skärmen. Minskningen eller förstoringen av elementen fungerar däremot endast till en viss gräns. Vid tillfällen då storleken på webbläsaren för liten, finns risk att element kommer för nära varandra och skapar konflikt i layouten, det leder till att element hamnar på felaktiga positioner och ger en vy som designmässigt ser förstörd ut. Blir storleken däremot för stor, finns risk att *max-width* på bilderna uppnås och stannar i storlek, medan andra element följer förstoringen hos webbläsaren, blir för stora och skapar en osymmetri i layouten. Media Queries är en lösning som tillåter element att ha olika värden beroende på skärmstorlek.

Inom responsive web design är bredden och höjden viktiga egenskaper då de avgör hur mycket av en sida skall synas på webläsarfönstret. Media queries är anrop som görs i CSS filen, där man kollar antingen höjden och eller bredden för att utföra nödvändiga ändringar i layouten för att upprätthålla en bra design. Tillsammans med Fluid Grid och Fluid Image skapar Media Queries en responsiv webbdesign.

Vid användning av fluid grid och fluid image sätts egenskaper hos element i procentform, t.ex 50 %. Med en centrering på elementet, blir resultatet en ruta i mitten av skärmen som täcker 50 % och har 25 % utrymme på var sida. När webbläsarfönstret minskas till storleken av en mobilskärm anpassas elementet relativt till förändringen, då fluid grid tekniken används för bredden. Vilket ger oss dessa två vyer, ena med webbläsarfönstret i storlek av en skrivbordsskärm och den andra i storlek av en mobilskärm:

Figur 1: Utifrån desktop



Figur 2: Utifrån mobilen

Denna design har endast ett element med egenskaperna:

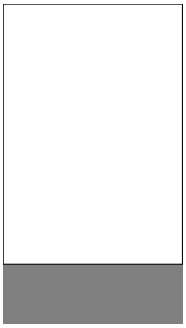
```
div {
        margin: 0 auto; //centrerad
        width:50 %; //bredd
        height:500px; //höjd
}
```

Designen fungerar bra för skrivbordsskärmar*(Figur 1)*, då rutan är centrerad och symmetrisk. 25 % av utrymmet på var sida av rutan gör designen läsvänlig och användbart då fokusering sker i mitten av skärmen där förmodlingen information kommer att samlas. Däremot är den inte lika optimal när skärmen blir mindre*(Figur 2)*. Rutan är fortfarande 50% av skärmen, vilket gör att designen blir kompakt i ett utrymme där förmedling av mycket information sker. 50% procent av skärmen går till spillo åt utrymme mellan rutan och kanter, vilket får rutan att se ihoptryckt ut. Därav en försämring på design och användbarhet när webbsidan ses ifrån en mobilskärm.

Med media anrop i CSSen, kallat för media queries, tillåter vi att värden hos ”div” elementet skrivs över när bredden på webbläsaren underskrider 380px.

```
@media all and (max-width: 380px){
       div {
                margin:0;
                width:100%;
       }
}
```

Med media anropet kommer *"width"* och *"margin"* skrivas över när skärmstorleken är 380px eller mindre, *"height"* förblir detsamma. Utrymmet mellan kanterna och rutan ändras till "0öch bredden till "100 %", vilket innebär hela skärmbredden. Detta leder till att sidan får andra designvärden när den ses från en mobil och andra enheter där skärmen har som störst 380px i bredd. Det ger en mer optimal lösning i både användbarhets och design perspektiv, då mer plats ges åt information och gör navigeringen lättare för en användare.



Figur 3: Resultatet vid användning av media queries

På så sätt kan samma CSS-fil låta element ha olika värden på egenskaper beroende på webbläsarstorlek. Responsive web design fyller sin funktion genom att tillföra en användbar design oavsett enhet som används för att se webbsidan.

1.4 Verktyg

Inom webdesignsutveckling används främst verktygen HTML, CSS och javascript, vilket är det verktyg som kommer att användas under implementeringsfasen i examensarbetet. I kommande sektion av rapporten kommer verktygen att förklaras för att få en bättre förståelse till kod och termer som används.

1.4.1 HTML

HTML står för *Hypertext Markup Language* och är ett format som definierar strukturen och logiken på en webbsida. HTML beskriver strukturen genom att märka upp olika delar av sidan med hjälp av *taggar* som beskriver typen av ett element. En webbläsare läser av HTML-koden och kan på så sätt rendera sidan med önskvärd layout.

Det finns olika sorters taggar inom HTML, stycke, rubrik, tabeller, länkar, listor, sektioner är en av dessa. Inom webbutveckling har användning utav taggen * * blivit vanlig för att strukturera upp en sida i olika sektioner, vartefter kan en sektion innehålla andra element.

```
<div>
          <h1> Mobile-first eller Desktop-first?</h1>
          <p>Frågan en webutvecklare ställer sig inför skapandet av
           en responsiv websida
          </p>
</div>
```

I ovanstående HTML-kod har vi definierat en sektion på webbsidan med * * taggen, inom den sektionen har vi en rubrik som definieras med taggen *<h1>* och ett stycke som definieras med taggen *<p>*. Taggarna i html-koden bildar tillsammans strukturen på sidan, däremot så har inte html-koden någon kontroll över utseendet för taggarna, det sköts utav CSS.

1.4.2 CSS

CSS står för *Cascading Style Sheets* och är ett språk som beskriver utseendet på en webbsida. Med hjälp av CSS-kod kan olika element i html-koden få ett speciellt utseende i form av storlek, färg och position. Det finns olika sätt att definiera css, man kan definiera det direkt i taggen t.ex *<p style="color:blue; width:50px">Frågan en webbutvecklare ställer sig inför*

skapandet av en responsiv webbsida</*p*> vilket kallas för *inline-styling*. Man kan även i html filen, definiera utseendet i <*head*> taggen, vilket kallas för *internal styling*:

```
<head>
            <style>
            p {
            color:blue;
            width:50px;
            }
            </style>
</head>
```

På så sätt får alla <*p*> taggar i html-koden egenskaperna som sätts. Trejde sättet är att definiera det i en egen css-fil och länka till det från <*head*> taggen i html-koden, vilket är det vanligaste av de tre sätten.

```
<link rel="stylesheet" type="text/css" href="main.css">
```

I css-kod definieras utseendet av olika taggar. Det finns möjlighet att definiera samma tagg med olika utseenden, detta görs genom att sätta *Klasser* och *IDs* för varje element. Ett element kan ha olika klasser eller ids, vilket definieras med ett utseende i css-koden. Det gör att varje tagg som delar klassen, delar även utseendet:

I html filen:

```
<div id=''main-container''>
          <p class=''paragraf''>Mobile-first vs Desktop-first</p>
</div>
```

I css filen:

```
#main-container {
        width:50%;
        background-color:grey;
}

.paragraf{
        font-size: 125%;
        color:black;
}
```

I ovanstående kod kommer sektionen få utseendet som definieras som ID: *main* i css-koden och paragrafen utseendet som definieras som Klass: *paragraf*.

Dessa tre olika sätt att definiera utseendet visas enligt prioritering, inline-styling kod först, sedan internal och sist external. Anledningen till att external väljs främst är för att utseendet inte behövs definieras flera gånger, allt är samlat i en fil och det går snabbare att ladda sidans utseende. Därmed hålls det ur en webbutvecklares synvinkel, en bra struktur. Vilket tillåter webbutvecklaren att på upprätthålla teknikerna fluid grid, fluid images och på ett enkelt sätt använda sig av media queries vid implementering av en webbsida.

1.4.3 JavaScript

JavaScript är ett scriptspråk som inom webbutveckling främst används för att hantera dynamiska funktioner för beteendet hos en websida, beteenden som skapas från klientsidan. Med JavaScript kan man t.ex. läsa en användarens klick i webbsidan och utifrån det kalla på funktioner som kan ändra webbsidans innehåll eller utseende. Eftersom JavaScript är ett skriptspråk behövs ingen kompilering och koden kan skrivas direkt i html-filen.

För att JavaScript skall fungera korrekt krävs det dock att webbläsaren stödjer det. I vissa fall har webbläsaren den funktionen men har den avstängd. Tidigare ett problem varit att mobilwebbläsare inte har haft stöd för JavaScript, vilket har lett till att menyer och pop-up fönster inte har fungerat korrekt när man surfar till sidan via mobilen, men tekniken har utvecklats och numera klarar mobila webbläsare av JavaScript. Det finns fortfarande en del problem när det kommer till mer komplicerade JavaScript funktioner och bibliotek som mobilwebbläsare inte kan hantera. Vid webbsidor som har helt separata mobilsidor kan det vara en fördel då man väljer att inte läsa in all JavaScript som behövs, med responsiva webbsidor använder man samma webbsida för varje enhet, vilket leder till att en mobilenhet laddar all JavaScript vilket försämra webbsidans respons. En webbutvecklare måste ha i åtanke och redan från början utveckla en hemsida vars JavaScript inte kommer i vägen för den responsiva webbsidans funktionalitet.

1.5 Mobile-first och Desktop-first

Tekniken för att skapa responsiva webbsidor existerar, med hjälp av utvecklingen av ovanstående tekniker och verktyg finns i nuläget möjligheten att skapa webbsidor som beroende på skärmstorlek har olika designlayouts med en och samma kodgrund. Det finns även kunskap om hur man skall gå tillväga för att skapa en responsiv webbsida, där utvecklare med former och riktlinjer kan lära sig att använda fluid grid, fluid images och media queries på bästa sätt. Men vägen till att skapa en responsiv websida är olika. I bloggar och forum på webben diskuteras flitigt valet av utvecklingsmetod när det kommer till responsiva webplatser och innan implementeringen av en webbsida är detta en fråga som med stor sannolikhet dyker upp hos webbutvecklare. Utvecklingsmetoderna man diskuterar om är *Mobile-first* och *Desktop-first*. Mobile-first och Desktop-first är två olika metoder där det responsiva

angreppssättet används. Båda strävar efter samma mål men skiljer sig i prioriteringen av vilken enhet webbsidan skall vara anpassad för först.

1.5.1 Mobile-first

Mobile-first metoden bygger på att man skapar en webbsida anpassad för mobilskärmen först, för att sedan med hjälp av media queries och en flexibel layout renderera elementen på webbsidan desto större webbläsarfönstret blir. På så sätt är grund layouten designad för en mobil. Mobile-first anses som en rimlig utvecklingsmetod då man tar till hänsyn antalet mobilanvändare och det behov som finns vid navigering och informationsintagelse från mobilskärmar. Det betyder att innehåll prioriteras då bristen på plats är ett faktum och fokus läggs på de delar som informationsmässigt är de viktigaste. Det behöver inte nödvändigtvis betyda att hänsyn till design för mobilen inte tas i desktop-first, utan snarare att implementationen för mobilskärm sker vid ett senare skede i utvecklingen än vid användning av mobile-first.

Kodmässigt är grunden anpassad för mobilen, det vill säga att media anrop sker när skärmen blir större, där media queries ser till att skriva om värden hos valda element.

```
@media all and (min-width: 480px){
        .main {
                margin:0 auto;
                width:50%;
        }
}
```

I fallet ovan sker ett anrop när bredden på websidan är som minst 480px, som gör att elementen med klassen main får "*margin*" och "*width*" skrivs över med nya värden. I fall då mobiler inte klarar av att läsa media queries(fotnot), vilket i dagsläget är få, är detta en optimal lösning då grunden redan är skrivet för mobilen och en läsning av media queries endast kommer att krävas från en skrivbordsskärm vilket de flesta webbläsare klarar av.

1.5.2 Desktop-first

Desktop-first metoden bygger på att man utvecklar för skrivbordskärmen först och därefter rendererar sidan desto mindre skärmen blir. Det behöver inte nödvändigtvis betyda att en färdig sida för desktop i efterhand designas om till mobil, utan tanken för mobil finns redan från början men implementeringen sker i synnerhet för skrivbordsskärmen först. Webbsidor som i efterhand skapas till mobilen, brukar innebära en separat mobilsida då refaktoreringen

av kod i samband med att förvandla sidan till responsiv kan innebära en del komplikationer.

Kodmässigt är grunden anpassad för skrivbordsskärmen. Det vill säga att media anrop sker när skärmbredden når en minimum gräns, där media queries ser till att skriva om värden på valda element.

```
@media all and (min-width: 380px){
        .main {
                margin:0 ;
                width:100\%;
        }
}
```

I fallet ovan sker ett anrop när bredden är som max 380px. Anropet ger "*margin*" och "*width*" i klassen main nya värden, anpassade till mobilen. Desktop-first används oftast i samband med hemsidor där man vill skapa en upplevelse och vill att webbsidan skall nå ett design maximum när den ses utifrån en skrivbordskärm, upplevelsen utifrån en mobilskärm är inte lika högt prioriterad, men bör innebära en webbsida med den nödvändigaste funktionaliteten.

1.5.3 Mobile-first eller Desktop-first, vilken skall man välja?

Utvecklingen av tekniker och verktyg har gjort det möjligt att kunna applicera metoderna vid skapandet av en webbsida. Kunskapen om metoderna var för sig har med tiden blivit större i samband med utvecklingen av responsive web design. Däremot ställs webbutvecklare kring frågan om vilken metod som appliceras bäst till den typen av webbsida som skall skapas. I dagsläget finns ingen kunskap om hur bra metoderna appliceras i jämförelse med varandra. Det finns tankar och spekulationer, därefter väljs metod utifrån dessa, komplikationer hanteras men dokumenteras inte och till slut har man en fungerande responsiv websida. Eftersom websidan är färdig implementerad finns ingen anledning till att implementera om en fungerande lösning med en annan metod, därav finns ingen större kunskap om jämförelse av implementationen med respektive metod.

Användning av mobilt internet ökar och användningen av internet via en dator kommer med stor sannolikhet bestå. Vilket gör att en grund för val av metod är högst passande, då det redan nu och i framtiden kommer att kräva mer kunskap än vad som idag kan erbjudas från bloggar. Båda metoder har sina fördelar samt nackdelar, men frågan som bör ställas är när dessa går att utnyttjas på bästa sätt, och kan beslutet av val leda till positiva faktorer vilka motsatta metod inte hade kunnat uppnå inom en specifik situation.

2 Syfte

Syftet med arbetet är att kunna hitta riktlinjer till när en specifik metod appliceras bäst. Det vill säga beroende på faktorer kring webbsidan hitta den metod som tillför det mest optimala lösningen för implementationen av webbsidan. Att hitta en metod som fungerar bäst i alla lägen är inte nödvändigtvis målet med arbetet då det existera många olika vinklar, vilka gör det svårt att hitta en specifik metod som ger det optimala resultatet oberoende på situation.

Frågeställningar som besvaras i rapporten är:

1. I vilka lägen appliceras metoderna bäst?
2. Vad är för- och nackdelarna med Mobile-first?
3. Vad är för- och nackdelarna med Desktop-first?

Frågeställning 1 baseras på miljö, målgrupp och kontext. Syftet med frågeställningen är att kunna utifrån analys hitta situationer där metoderna visar sig vara fördelaktiga under utvecklingsfasen, för att hitta faktorer hos metoden vilka motsatta metod inte har och därmed inte kan tillföra samma lyckade resultat. Syftet med *Frågeställning 2 och 3* är att kunna lyfta fram de fördelar och nackdelar som finns vid implementation med de två metoderna. För att på så sätt hitta ramar för varje metod vilka en läsare kan relatera till med en egen situation och utifrån det använda det som grund vid val av metod.

2.1 Avgränsningar

Arbetet kommer att enbart fokusera på mobile-first och desktop-first, ett mellanläge som existerar för Ipads, tablets, osv. kommer inte att tas med i arbetet. I dagsläget finns även andra lösningar för mobilt webb t.ex Appar och Hybrida Appar, jämförelse mellan dessa och responsive web design kommer inte att göras, utan jämförelsen som görs är mellan mobilalösningar inom responsive web design.

Termen *Mobile-first* kan tolkas på olika sätt. Det har förekommit tillfällen för webbutvecklare där mobile-first är prioriterad designmässigt men där implementeringen ändå har skett för desktop innan mobilen, vilket en produktägare har tolkat som mobile-first. Det är inte fallet i arbetet, utan mobile-first beskrivs i arbetet som en tanke och en implementering avsedd för mobilen i första hand, och tvärtom vid desktop-first. Testfallen under implementeringsfasen i arbetet kommer att bygga på den teori för ovanstående metoder.

2.2 Valtech AB

Examensarbetet kommer att utföras på Valtech AB i Stockholm. Valtech Sverige fokuserar främst på utveckling av användarvänliga hemsidor, webbapplikationer och intranät. Hos Valtech finns erfarna gränssnittsutvecklare som har stött på problem under utvecklingsprocessen i form av ett beslutstagande av tillvägagångssätt för en responsiv webbsida. Det finns ett behov hos utvecklare på Valtech att ha en grund för vilka faktorer som är viktigast vid beslutstagande för metod, en metod som ger den bästa lösningen för en effektiv utveckling av en responsiv webbplats. Gränsnittssutvecklarna besitter på mycket erfarenhet och kunskap, vilket kan samlas i form av intervjuer för att analyseras och sammanfattas utifrån verkliga situationer som förekommer i företag, för framtida syfte.

Referenser

[1] W3Techs - World Wide Web Technology Surveys, *Usage of JavaScript for websites.* `http://w3techs.com/technologies/details/cp-javascript/all/all` Read on April 9, 2013.

[2] Mark Bates, *Testing Your JavaScript/CoffeeScript.* `http://www.informit.com/articles/article.aspx?p=1925618` Read on April 9, 2013.

[3] Edward Hieatt and Robert Mee, *Going Faster: Testing The Web Application.* IEEE Software, p. 63 March/April 2002.

[4] Github, *pivotal/jsunit.* `https://github.com/pivotal/jsunit` Read on April 3, 2013.

[5] Running documentation, *Jasmine is a behavior-driven development framework for testing JavaScript code.* `http://pivotal.github.com/jasmine/` Read on April 3, 2013.

[6] Rudy Lattae, *Jasmine-species: Extended BDD grammar and reporting for Jasmine.* `http://rudylattae.github.com/jasmine-species/` Read on April 5, 2013.

[7] The jQuery Foundation, *QUnit: A JavaScript Unit Testing framework.* `http://qunitjs.com/` Read on April 3, 2013.

[8] TJ Holowaychuk, *mocha - simple, flexible, fun javascript test framework for node.js & the browser.* `http://visionmedia.github.com/mocha/` Read on April 3, 2013.

[9] Cory Smith, Robert Dionne, Vojta Jína, Sergey Simonchik, *JsTestDriver.* `https://code.google.com/p/js-test-driver/` Read on April 5, 2013.

[10] August Lilleaas, Christian Johansen, et al., *BusterJS: A powerful suite of automated test tools for JavaScript* `http://docs.busterjs.org/` Read on April 5, 2013.

[11] Christian Johansen, *Sinon.JS: Standalone test spies, stubs and mocks for JavaScript.* `http://sinonjs.org/` Read on April 5, 2013.

[12] Vows: Asynchronous behaviour driven development for Node. `http://vowsjs.org/` Read on April 5, 2013.

[13] Cucumis: BDD Cucumber Style Asynchronous Testing Framework for node.js `https://github.com/noblesamurai/cucumis` Read on April 5, 2013.

[14] JSpec on Github no longer supported `https://github.com/liblime/jspec` Read on April 5, 2013.

[15] Shay Artzi, Julian Dolby, Simon Holm Jensen, Anders Møller, Frank Tip, *A Framework for Automated Testing of Javascript Web Applications.* ICSE '11, Honolulu, Hawaii, USA. May 21–28, 2011.

[16] Phillip Heidegger, Annette Bieniusa, and Peter Thiemann, *DOM Transactions for Testing JavaScript.* Albert-Ludwigs-Universität Freiburg, Germany. Proceeding TAIC PART'10 Proceedings of the 5th international academic and industrial conference on Testing - practice and research techniques. Pages 211-214. 2010.

[17] Frolin S. Ocariza, Jr., Karthik Pattabiraman, Benjamin Zorn *JavaScript Errors in the Wild: An Empirical Study.* 2011 IEEE 22nd International Symposium on Software Reliability Engineering (ISSRE). Pages 100-109. Nov. 29 2011-Dec. 2 2011.

[18] Christian Johansen, *Test-Driven JavaScript Development* ADDISON-WESLEY, ISBN 9780321683915, 2010.