Bachelor Degree Project

UNIVERSITY OF SKÖVDE

CSS-PREPROCESSORERS PÅVERKAN PÅ LADDNINGSTIDEN FÖR WEBBSIDOR

Bachelor Degree Project in 16
Datalogi: Computer Science

30 ECTS
Spring term 2016

Emil Eek

Supervisor: Jonas Gamalielsson
Examiner: Henrik Gustavsson

SAMMANFATTNING

CSS har de senaste åren fått en mer central roll i webbutveckling i och med responsiva webbplatser. För att underlätta arbetet för utvecklaren har CSS-preprocessorer skapats. Minimala variationer med ursprung från CSS med syftet att minimera arbetsbördan för utvecklaren. Påverkar valet av CSS-preprocessor laddningstiderna på webbsidor och i så fall varför? Experiment genomförs, för att utvärdera studiens hypotes. Tre olika preprocessorer jämförs med CSS för att kunna analysera mätningar och värdera skillnader. Bland preprocessorerna har LESS kortast laddningstider men resultaten visar att kod skriven i CSS blir mindre vilket innebär att svarstiderna blir kortare. Skillnaderna skildras tydligare desto större projektet är. Preprocessorer tillför funktioner som vanlig CSS saknar, men är inget revolutionerande. CSS kommer att utvecklas, i framtiden kommer antagligen en befintlig eller annan teknologi omfatta CSS och CSS-preprocessorers funktioner.

Nyckelord: CSS, Preprocessorer, Laddningstid, LESS, SASS, Stylus

Innehållsförteckning

1 INTRODUKTION 1

2 BAKGRUND 2

2.1 CSS 2
2.2 CSS-preprocessorer 2
2.3 Node.js 4
2.4 Ruby 5
2.5 Konverterare 5

3 PROBLEMFORMULERING 6

3.1 Problemet 6
3.2 Frågeställning, hypotes 7
3.3 Metodbeskrivning 7

4 GENOMFÖRANDE 10

4.1 Skriptets användning och placering 10
4.2 Tidtagning 11
4.3 Pilotstudie 11
4.4 Experiment 11
4.5 Mann-Whitney-testning 12

5 RESULTAT 14

5.1 Pilotstudie 14
5.2 Experiment 16
5.3 Mann-Whitney-testning 21

6 DISKUSSION 23

6.1 Sammanfattning 23

REFERENSER 26

1 INTRODUKTION

Idag används internet i allt större utsträckning för var dag som går. Fler människor får tillgång till internet och uppkopplingen blir bättre. För att nå ut till kunder via internet är det dock inte längre tillräckligt med endast en webbsida, användningen av smartmobiler och surfplattor kräver även att webbsidorna optimeras för alla enheter för att ge den bästa användarupplevelsen (Mohorovičić 2013). Tiden innan smartmobilerna var de olika skärmstorlekarna relativt lätträknade, men det har aldrig tidigare varit så stor varietet mellan de största och minsta skärmarna som idag används för att surfa (Kadlec 2013). Detta ökar i sin tur bland annat trycket på att skriva CSS-kod som kan hantera alla olika skärmar som användarna kommer till sidan med och detta leder till mer arbete (Mohorovičić 2013). CSS (Cascading Style Sheets) är ett språk som används för att beskriva utseendet av webbsidan (Gardner 2011).

Detta arbete fokuserar på de verktyg som skapats under de senaste åren med syftet att hjälpa utvecklare att koda mer effektivt, nämligen CSS-preprocessorer. CSS anses ibland som statisk och det har funnits en önskan om att utveckla CSS, vilket har lett till CSS-preprocessorer. CSS-preprocessorerna applicerar DRY-principen (Don't Repeat Yourself) (Prabhu 2015). Med hjälp av bland annat variabler och arv kan man med CSS-preprocessorer minska upprepningen som uppstår med vanlig CSS.

Att verktygens popularitet ökar tyder på att de hjälper utvecklaren, men mindre fokus och undersökningar finns på hur det påverkar användaren. Försäljningshemsidan Amazon har genomfört studier som visar på att varje sekund en användare behöver vänta på att sidan ska ladda klart och bli operationsduglig, har negativ effekt på användarupplevelsen (Butkiewicz, Madhyastha & Seker 2011). Målet med detta arbete är därför att undersöka hur, och till vilken grad, konverteringen från CSS-preprocessorer till vanlig CSS påverkar laddningstiden på webbsidor, samt att försöka ställa det i relation till effektiviteten för utvecklaren.

2 BAKGRUND

Detta avsnitt behandlar de viktigaste teknologierna som används för att genomföra arbetet. En djupare förklaring av teknologierna görs, hur de fungerar samt hur de används i arbetet.

2.1 CSS

CSS betyder Cascading Style Sheet och är ett kodspråk som beskriver presentationsstilen av en webbsida bland annat med hjälp av typsnitt, fonter och färger. CSS är framtaget av W3C (World Wide Web Consortium), som arbetar med att utveckla tekniska protokoll standarder och programvara för webben.

2.2 CSS-preprocessorer

Många alternativ till vanlig CSS har tillkommit de senaste åren och de kallas för CSS-preprocessor. De är olika scriptspråk som med en konverterare omvandlar den skrivna preprocessorkoden till vanlig CSS (Prabhu, 2015). Webbläsare hanterar dock inte något annat programspråk än CSS för att beskriva stylingen av sidan utöver Javascript, vilket betyder att när kod skrivs i en vald preprocessor måste koden konverteras till CSS innan den kan köras i webbläsaren. Om en utvecklare bestämmer sig för att jobba med en preprocessor finns många att välja på, nästan lika många alternativ att välja finns när det kommer till konverterare. Det finns idag väldigt många konverterare på marknaden och de som kommer användas i denna undersökning tas upp senare i detta kapitel.

CSS-preprocessorer har tillkommit av en anledning men till skillnad mot många andra språk marknadsförs inte dessa med att snabbare kompilera utan att de underlättar arbetet för programmeraren. CSS-preprocessorerna har främst tillkommit för att förenkla för programmeraren och i vissa fall blir kodmängden mindre sett till storleken, men det är inte enbart detta som ska vara vinsten med att använda preprocessorer. CSS-preprocessorerna kan bland annat hantera arv och även variabler vilket inte vanlig CSS hanterar, se figur 1 och 2. Figur 3 och 4 visar hur vanlig CSS och CSS-preprocessorer hanterar nästlingar och en jämförelse av dessa visar att CSS-preprocessorerna förenklar koden för programmeraren att lättare se vad som hör ihop och vad som påverkas av vad.

```
1   .block, p, ul, ol {
2     margin: 10px 5px;
3     padding: 2px;
4   }
5   p {
6     border: 1px solid #EEE;
7   }
8   ul, ol {
9     color: #333;
10    text-transform: uppercase;
11  }
```

Figur 1 - Hur CSS hanterar arv. (Skärmdump från Sublime Text 3, utgiven av Jon Skinner)

```
1  .block {
2    margin: 10px 5px;
3    padding: 2px;
4  }
5
6  p {
7    @extend .block; /* Ärver egenskaper från '.block' */
8    border: 1px solid #EEE;
9  }
10 ul, ol {
11   @extend .block; /* Ärver egenskaper från '.block' */
12   color: #333;
13   text-transform: uppercase;
14 }
```

Figur 2 - Hur CSS-preprocessorerna SASS och Stylus hanterar arv (Skärmdump från Sublime Text 3, utgiven av Jon Skinner)

```
1  section {
2    margin: 10px;
3  }
4  section nav {
5    height: 25px;
6  }
7  section nav a {
8    color: #0982C1;
9  }
10 section nav a:hover {
11   text-decoration: underline;
12 }
```

Figur 3 - Hur CSS hanterar nästlingar (Skärmdump från Sublime Text 3, utgiven av Jon Skinner)

```
1  section {
2    margin: 10px;
3
4    nav {
5      height: 25px;
6
7      a {
8        color: #0982C1;
9
10       &amp;:hover {
11         text-decoration: underline;
12       }
13     }
14   }
15 }
```

Figur 4 - Hur CSS-preprocessorerna SASS och Stylus hanterar nästlingar (Skärmdump från sublime Text 3, utgiven av Jon Skinner)

Utöver variabler, nästlingar och arv ska även preprocessorer kunna hantera loopar och en del matematiska formler.

2.2.1 Sass/SCSS

Första releasen av Sass (*Syntactically Awsome Stylesheets*) släpptes 2006 (GitHub 2016) och är den första CSS-preprocessorn som släpptes. Den skapades av Hampton Catlin när han insåg att mycket av hans kollegors kod var repetitiv[1]. En skillnad från vanlig CSS är att varken måsvingar eller semikolon behövs när utvecklaren kodar. Detta betyder dock att indentering är ett krav för att konverteraren ska kunna utläsa koden korrekt (Prabhu 2015).

SCSS (*Sassy CSS*) ses som en egen CSS-preprocessor, men har fått grunderna från Sass. Fördelen med SCSS är att godkänd CSS-syntax går att skiva i SCSS-filer och måsvingar och semikolon har lagts till igen. Tanken med SCSS var att kombinera det bästa från båda världarna, alltså som arv, variabler.

2.2.2 Less

Less, skapad av Alexis Sellier, släpptes 2009 och har inspirerats från bland annat Sass (lesscss.org 2016). Less var från början skriven i Ruby men konverterades sedan till JavaScript. En anledning till att Less snabbt blivit omtyckt är att den fungerar väl tillsammans med *Bootstrap,* ett front-end ramverk för att skapa webbsidor med ett färdigt CSS- och JavaScriptbibliotek som effektiviserar den responsiva webbutvecklingen (Kim 2013). Detta eftersom skaparen av Less och utvecklarna av *Bootstrap* tillsammans arbetat för att optimera kompileringstiden när de används tillsammans (getbootstrap.com 2016). Less har också blivit populärt för just enkelheten som även namnet ger en hint av. En annan fördel med Less gentemot andra preprocessorer är att den tillåter real-time kompilering i webbläsaren, alltså att i användaren i webbläsarens ”granskare” kan justera värden och se förändringar direkt utan att behöva uppdatera CSS-filen (Prabhu 2015).

2.2.3 Compass

Compass är ett CSS-ramverk som är baserat på Sass och använder sig av minimerat märkspråk, alltså användningen av taggar och element. Ramverket är fyllt av återanvändningsbara delar som skaparna menar att utvecklaren använder ofta och hjälper till att koda mer effektivt (Prabhu 2015). Compass räknas dock inte som en egen CSS-preprocessor och ses som en utbyggnad av Sass, men har utöver Sass-syntax en egen utvecklad syntax (Gringl 2013).

2.2.4 Stylus

En av de senare CSS-preprocessorerna som tillkommit är Stylus och har inspirerats från både Sass och Less. Kod skriven i Stylus behöver likt Sass inte skrivas med måsvingar och semikolon. CSS-preprocessorn ses som den tredje största (css-tricks.com 2016) och är skriven i Node.js och JADE av TJ Holowaychuck (GitHub.com 2016). För att kunna använda sig av och konvertera Stylus behöver man Ruby tillsammans med Node.js.

2.3 Node.js

NodeJs är ett program skrivet i JavaScript och används på serversidan för att kommunicera med exempelvis en webbklient. Programmet är eventbaserat och har kommit till för att lösa realtidsbehov. Det som gjort Node.js populärt är att den snabb, skalbar och minnessnål (utvbloggen.se, 2016).

[1] Hampton Catlin skapare av Sass, mailkontakt 2016-02-21

2.4 Ruby

Ruby är ett dynamiskt, objektorienterat, open-source programspråk. Till Ruby finns även ett antal ramverk för att skriva i, ett av de mer kända heter Ruby on Rails som använder sig av Ruby. Ruby är till för att skapa webbaserade programvaror (David Elbe, 2015).

2.5 Konverterare

Webbläsarna kan inte läsa koden I det formatet som CSS-preprocessorerna är skrivna i, de kan bara läsa koden som CSS. Därför behövs programvaror som kan omvandla koden till CSS innan filen laddas upp på servern. I detta arbetet används tre stycken olika konverterare.

2.5.1 Grunt.js

Grunt.js är en task-runner på engelska, en programvara som hjälper till att automatisera uppgifter för användaren och hanteras via pakethanteraren npm i Node.js. Programvaran har plugin som går att installera för att kunna hantera konvertering till bland annat det största CSS-preprocessorerna. För att hantera Grunt.js arbetar man genom Terminalen eller kommandopromten. När man skriver, till exempel Less-kod krävs inget extrajobb med att konvertera koden manuellt varje gång ändringar eller tillägg görs i koden. Varje gång man sparar .less-filen läser Grunt.js igenom filen och konverterar filen automatiskt till CSS.

2.5.2 Koala

Likt Grunt.js är även Koala ett en automatiserande programvara för att hjälpa användaren med bland annat konvertera en kod till annan. Skillnaden mellan Koala och Grunt.js är att Koala har ett grafiskt gränssnitt.

2.5.3 Beautifytools

Beautifytools är ingen programvara som användaren behöver ladda ned utan är en webbaserad tjänst som finns på beautifytools.com. Tjänsten är simpel och kräver bara kopiera och klistra in från användaren. Utöver konvertering mellan CSS och preprocessorer erbjuder de konvertering mellan nästan vad som helst, Excel till SQL, Html till XML för att nämna några.

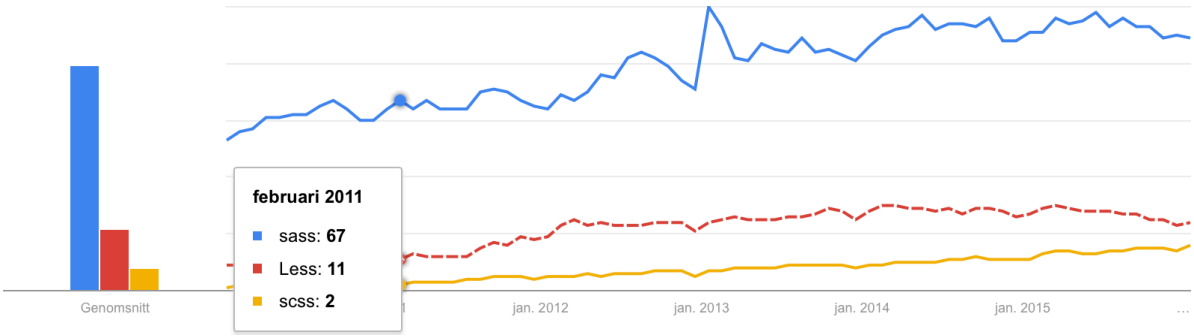
3 PROBLEMFORMULERING

Detta avsnitt behandlar utvecklingen av preprocessorer och vad de betyder för utvecklingen av webbutveckling. Avsnittet tar även upp hur arbetet utförs med val av vetenskaplig metod och tar upp hypotesen och frågeställning.

3.1 Problemet

Webbutveckling har under många år varit en bransch med stor innovationsförmåga och är under konstant utveckling. I grunden finns många språk som skapats för att genom kod kunna få en dator att uträtta uppgifter som skaparen vill. Nya språk så som PHP (som ursprungligen stod för Personel Home Page) och JavaScript, båda med sina rötter från objektorienterad programmering (developer.mozilla.org, 2016; php.net, 2016) har skapats med målet att göra webbsidor bättre. Detta genom att göra dem interaktiva, databaskompatibla och dynamiska. Dessa är bara två exempel i ett hav där det idag finns ett stort antal olika teknologier som skapats för att påstå sig vara bättre än föregångaren. Det är till fördel för programmeraren och alltså användaren att ha fler alternativ för att hitta någonting som passar hen bäst. Dock arbetar många inom webbindustrin hårt för att skapa standarder som syftar till att underlätta för programmerare att jobba korrekt (W3C 2016).

Ett exempel där alternativ under de senaste åren bubblat upp är i designgenren av webbutveckling, alltså CSS (Cascading Style Sheets) språk för att beskriva utseendet av en webbsida (Gardner, 2011) av vilken CSS-preprocessorer tillkommit. Alltfler programmerare testar och går över till de olika preprocessorerna (css-trick.com, 2016). Det finns ett stort intresse för preprocessorer vilket går att se från alla forum och bloggar som skriver om dem. Intresset för de olika CSS-preprocessorerna ökar dessutom, vilket kan ses på *Google Trends*-verktyg, se figur 5.



Figur 5 - Med verktyget Google Trend redovisas hur populära olika sökningstermer av preprocessorer är. Söktermerna i grafen är SCSS, SASS och LESS (Google.se/trends, 2016)

Prabhu (2015) menar att programmera *stylesheets* med hjälp av en preprocessor inte är det första man börjar med som nybörjare i programmeringsvärlden. Han säger även att preprocessorerna inte kan skriva bättre CSS av sig själva, utan det är först när utvecklaren känner sig trygg med vanlig CSS som hen kan förstå och dra nytta av fördelarna CSS-preprocessorerna erbjuder. Det finns idag väldigt många CSS-preprocessorer och för många för att i det här arbetet kunna testa alla. Därför har de tre mest populära valts ut. För att komma fram till vilka som är de största och mest populära CSS-preprocessorerna har GitHub varit en källa genom att titta på *commits*, *releases*, *contributors* och internetsökningar. Dessutom gjordes en omröstning 2012 på css-tricks.com där 12712

personer deltog för att svara på flera frågor. Bland annat om man testat CSS-preprocessorer och om vilken som var en favorit (sitepoint.com, 2016).

Ett problem med CSS-preprocessorer är att det finns så många alternativ och ingen självskriven vinnare (sitepoint.com, 2016). När en internetsökning görs på CSS-preprocessorer får man upp mängder av bloggpostartiklar som säljer in argument om varför CSS-preprocessorer är bra och något färre artiklar om varför artikelförfattarna inte tycker om dem. I de flesta blogginlägg där diskussion och åsikter uppmuntras, har de flesta en egen favorit. De flesta tar upp skillnader mellan funktioner men ingen diskuterar hur prestandan skiljer dem åt. Idén om att skriva examensarbete om detta ämne uppkom av att en blogg just tagit upp denna infallsvinkel och skapat egna testfall (Solitr.com 2016). Testfallen är dock endast utförda en gång per CSS-preprocessor och redovisar inte mer än så. Enligt bloggaren finns avsevärda skillnader vilket gör ämnet intressant som forskningsområde.

Ett examensarbete från 2013 (Gringl 2013) behandlar, såsom bloggaren ovan, CSS-preprocessorer och kommer användas som bakgrundsmaterial för vidareforskning. Det är därför av intresse att utföra tester på de olika CSS-preprocessorerna för att påvisa eventuella skillnader och dess omfattning. Olika webbläsare kommer även testas för att undersöka dess påverkan. Störst fokus kommer vara på utvecklandet av CSS och olika CSS-preprocessorer men även en webbsida med minsta möjliga HTML-kod för att påverka testen så lite som möjligt. Det intressanta med att testa på webbläsare är att stöd för CSS och JavaScript skiljer sig åt mellan webbläsare (Harb et al., 2011), vilket kan ge upphov till eventuella skillnader i mätningarna.

3.2 Frågeställning, hypotes

Hypotesen är att valet av CSS-preprocessor påverkar laddningstiden av webbsidor negativt till skillnad mot att skriva vanlig CSS. Hur påverkar valet av CSS-preprocessor laddningstiden av webbsidor?

Under genomförandet finns ett antal delmål som behöver uppnås för att slutföra arbetet. M. Berndtsson et al. (2008) menar att det kan behövas olika metoder för att uppnå målen. Nedan finns en lista som redovisar de olika delmålen, bland annat kommer två olika metoder behövas; implementation och experiment.

1. Val av data
2. Konstruerande av script för tidtagning (implementation)
3. Design av experimenten (experiment)
4. Genomförande av experiment
5. Utvärdering av experiment

3.3 Metodbeskrivning

I denna del noteras vilka typer av vetenskapliga metoder som kommer att användas för att kunna utföra tänka test och mätningar, så som implementation, experiment. Denna del belyser även metoddiskussion och forskningsetik.

3.3.1 Implementation

För att kunna utföra tidtagning av datan behövs script skrivna i JavaScript utvecklas. Implementationen av scripten är inte något nyskapade utan någonting som kommer behövas för att kunna klara av nästa del av arbetet. M. Berndtsson et al. (2008) åberopar vikten av att, i detta fall, tidtagningsscriptet är korrekt för att kunna försäkra giltigheten och tillförlitligheten i arbetet för att inte påverka mätningarna negativt. Dessa script är nödvändiga för att kunna utföra mätningar på de olika CSS-preprocessorerna som har egna script för att konvertera preprocessorkoden till vanlig CSS. Om det skulle uppstå problem med skapandet av tidtagningsscripten finns en alternativ plan där Rajamony & Elnozahy (2001) skapat ett program för att mäta responstider i webbläsare med JavaScript som möjligen skulle kunna implementeras för att utföra mätningarna.

3.3.2 Experiment

Detta arbete kommer att utföras som ett experiment då det är mätningar på kompileringstider och storlek av filer som ska utföras, då det är den bästa metoden för att få beprövade värden för att kunna ställa mot hypotesen. Wohlin et al. (2012) skriver att experiment tillåter kontroll över utförandet objekt och instrument, vilket i sin tur leder till att kunna dra fler och bättre slutsatser. De menar också att experiment är bra för att kunna utföra replikering av tidigare arbeten. Under experimentet kommer statistiska analyser göras av de olika preprocessorerna där medelvärde och standardavvikelser jämförs för att försöka särskilja dem. Även om experiment i detta fall är det bästa alternativet för att få fram konkreta värden menar M. Berndtsson et al. (2008) att man ska dra sig från att dra snabba slutsatser av positiva resultat då det inte bevisar att man har rätt då det kan finnas flera andra orsaker som man inte tagit hänsyn till eller missat som är orsaken till varför resultatet ser ut som det gör.

För experimentet kommer testfall utformas för att ställa CSS-preprocessorerna mot varandra i olika situationer som de kan användas i för att likna riktiga scenarion. M. Berndtsson et al. (2008) skriver att det är viktigt att undersöka att testfallen är relevanta och intressanta för arbetet, annars riskeras värdet av resultatet bli begränsat, i negativ mening.

3.3.3 Metoddiskussion

Syftet med detta arbete är att se om och hur mycket CSS-preprocessorer påverkar laddningstiden av en webbsida. Arbetet kommer att utvärderas genom att sammanställa kompileringstiden för de olika CSS-preprocessorerna i olika testfall och mellan olika webbläsare. Experimenten kommer att utföras på två plattformar där flera webbläsare testas. Plattformarna kommer att vara Windows 10 och Mac OS X, webbläsarna som skiljer plattformarna emellan är Edge för Windows och Safari för Mac OS X. Detta kan ses som ett problem då objektiviteten kan ifrågasättas, men då Edge inte är kompatibelt med OS X och det för Safari inte finns en stabil version till Windows men ändå är populära webbläsare för respektive operativsystem är de intressanta att testa.

För att uppnå en så hög nivå av kontroll som möjligt kommer detta arbete att utföras offline då faktorer som uppkoppling och belastning kan uteslutas när mätningarna genomförs. (Wohlin, et al. 2012). Att utföra mätningar mot internet innebär att faktorer som uppkoppling mot internet och belastningar som kan uppstå på nätverket kan spela en roll då det bara går att kontrollera till viss grad.

3.3.4 Forskningsetik

Målet med testerna är att de ska vara återupprepningsbara i framtiden, därför kommer all information som är nödvändig för att kunna återupprepa redovisas i arbetet så som hårdvaruspecifikationer, OS, webbläsare, testdata och resultat. Cai, Nerurkar & Wu (1998) skriver bland annat att ett bra test är representativt, optimalt och återupprepningsbart. Med andra ord är det viktigt att redovisa vad och hur någonting är utfört.

För att göra testet rättvist och så oberoende som möjligt kommer flera webbläsare testas där de som bedöms vara de största vara av intressanta att testa. W3Schools (2016) sammanför statistik för de mest frekvent använda webbläsarna. Statistiken från december 2015, kan ses i figur 6 nedan.

2015	Chrome	IE	Firefox	Safari	Opera
December	68.0 %	6.3 %	19.1 %	3.7 %	1.5 %

Figur 6 - Statistik över de mest använda webbläsarna december, 2015 (W3Schools.com 2016)

I experimentet kommer inga testpersoner, företag eller verksamheter involveras vilket gör att de etiska aspekterna kan uteslutas.

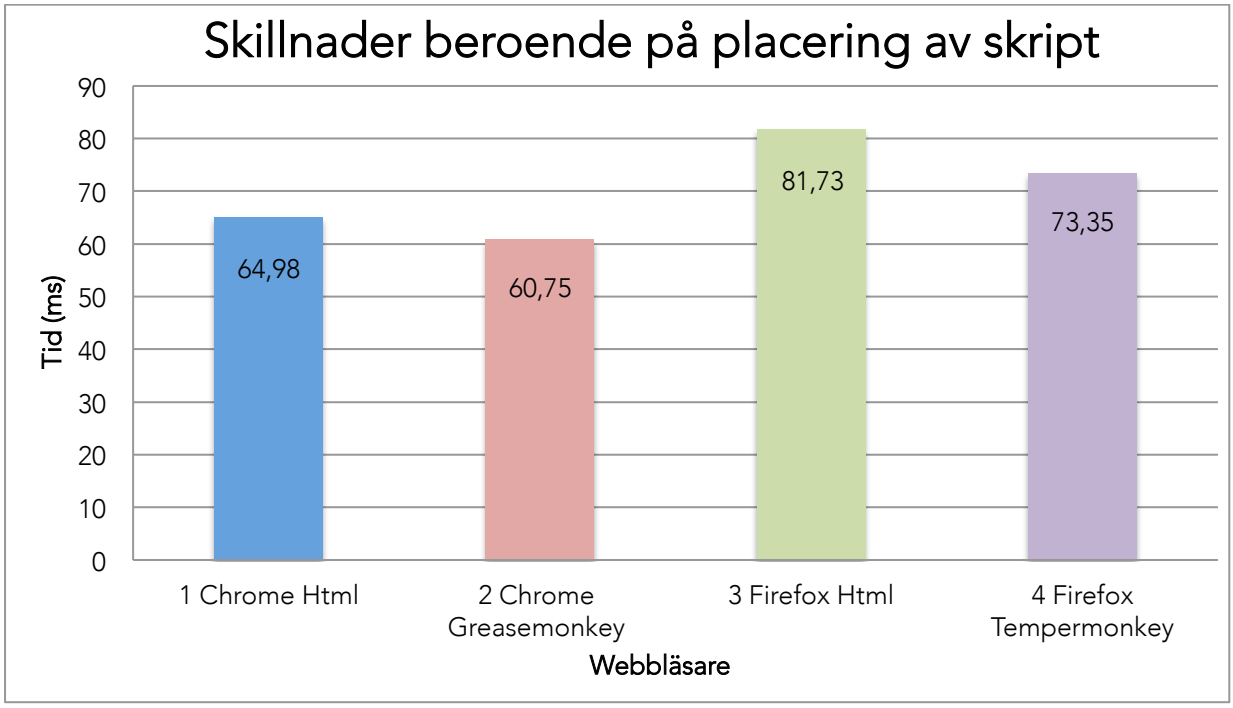
4 GENOMFÖRANDE

Detta avsnitt beskriver genomförandet av arbetet och innefattar beskrivning av tidtagning, hur pilotstudien genomfördes samt hur det större experimentet genomfördes.

4.1 Skriptets användning och placering

I starten av arbetet var tanken att placera skriptet i mjukvarorna Greasemonkey för Chrome och Edge, samt Tempermonkey för Firefox, men eftersom testplattformen ändrades från Windows till OS X krävdes vissa ändringar. Då Edge inte finns till OS X valdes istället att använda Safari. Detta gjorde att Greasemonkey uteslöts då ingen liknande programvara finns för Safari och för att erhålla jämförbara resultat uteslöts även Tempermonkey.

Mätningar genomfördes för att kontrollera om uteslutande av Greasemonkey och Tempermonkey skulle medföra någon skillnad. Resultatet visade sig att det finns skillnader i laddningstider när tidtagningsskriptet är placerat i tredjepartsporgrammen i webbläsarna jämfört med när den ligger direkt Html-filen, se figur 7.



Figur 7 - Skillnader beroende på webbläsare och var JavaScriptet för tidtagningen är placerat

Mätningarna visar att det går fortare att ladda sidan när tidtagningsskriptet ligger i tredjepartsprogrammen, men skillnaden är marginell. Tidsskillnaden mellan stapel (Chrome Html) och 2 (Chrome Greasmonkey) är 4,23 millisekunder och alltså ca 7 % snabbare när skriptet körs i tredjepartsprogrammet. Det är svårt att fastställa om det beror på brus på nätverket eller om det är på grund av att Html-filen blir mindre (i och med att skriptet flyttas från Html-filen till tredjepartsprogrammet) och då tar mindre tid att ladda. Skillnaderna mellan stapel 3 (Firefox Htlm) och 4 (Firefox Tempermonkey) är snarlika men aningen större, 8,38 millisekunder och cirka 10 % snabbare när skriptet körs i tredjepartsprogrammet. Efter dessa mätningar beslutades att lägga skriptet direkt i Html-

filen även om mätningarna påvisar en mindre tidspåläggning. Det ansågs nödvändigt med anledning av att kunna testa en till webbläsare under liknande förhållanden (Safari)

4.2 Tidtagning

Ett skript i JavaScript skapades för att mäta hur lång tid det tar att ladda sidan med CSS. Scriptet startar direkt när sidan börjar ladda och en tidsstämpel skapas. När sidan är färdigladdad skapas en ny tidsstämpel och laddningstiden är då differensen mellan den första och den sista tidsstämpeln. Skriptet uppdaterar sidan automatiskt valt antal gånger och utifrån utfallet från dessa mätningar kan en statistisk analys göras.

4.3 Pilotstudie

En mindre version av arbetet genomfördes för att kunna kontrollera att det gick att utföra mätningar av CSS-preprocessorerna relevanta för arbetet. Första steget var att skapa ett skript som fungerar likt ett tidtagarur och sedan ladda in de olika CSS-preprocessorerna. I pilotstudien utfördes mätningar på vanlig CSS och LESS. Mätningarna av CSS krävde inga tillägg, LESS behövde två extra programvaror installera för att kunna köras, Node.js och även Grunt.js. De två programvarorna hjälper till att kompilera och konvertera LESS-filerna till vanlig CSS. I Html-filen anropades en CSS-fil baserad på koden skriven i LESS. I pilotstudien genomfördes två enklare testfall av CSS och LESS för att undersöka mätningarnas genomförbarhet. Ett testfall med mindre antal rader kod och ett med fler antal rader kod. Detta för att undersöka om andra faktorer påverkar laddningstiderna. Pilotstudien genomfördes genom att ladda sidan i webbläsaren 100 iterationer per omgång och totalt 5 omgångar per webbläsare, sedan sammanställdes all testdata en graf. Testerna utfördes på en MacBook Pro 2012, 2,6Ghz i7 med 16 GB RAM-minne.

4.3.1 Första mätningar

I pilotstudien valdes att sätta upp en enklare mätning mellan CSS och LESS. I de grundläggande mätningarna har enbart Bootstraps grundkod använts för att kontrollera att mätningarna fungerar och fångar korrekt data. Bootstrap finns för nedladdning både som CSS, LESS och SASS, detta innebär att ingen konvertering för att skapa LESS-kod har behövts för pilotstudien. Däremot har mjukvaran Grunt.js tillsammans med Node.js använts för att konvertera koden från LESS till CSS före mätningarna kunnat genomföras i de olika webbläsarna. I de första mätningarna har enbart Grunt.js använts vid konverteringen, vid huvudsakliga mätningar kommer flera konverterare testas för att säkerställa att konverteringen kommer ut likadana, alltså att ingen konverterare översätter annorlunda.

4.3.1.1 Skillnaden mellan kodmängden och tid

När mätningar genomfördes med både originalkoden och det förstorade projektet uppfattades skillnaderna mellan mätningar inte vara linjära trots att kodmängden hade ökat med 10 gånger. Därför skapades ett antal grafer från mätningarna med Chrome, Firefox och Safari för att tydligt redogöra hur stora skillnader det var mellan mätningarna för att ta reda på den konstanta kostnaden av laddningstiderna.

4.4 Experiment

Pilotstudien gav insikt i att vissa ändringar behövde göras, dessa redovisas under avsnitt 5. Då ändringarna gjorts utfördes experimentet.

4.4.1 Konverterare

De två mest populära preprocessorerna LESS och Sass tillhandahåller Bootstrap på sin hemsida, men koden har sparats och behövts konverterats till vanlig CSS med hjälp av bland annat Node.js och Grunt.js för att kunna utföra mätningarna. Grunt.js är en av många tredjepartsprogram som finns för att bland annat kunna konvertera kod till önskad form. Det är svårt att kontrollera vad som är en korrekt konvertering, det sätt som går att undersöka är att slutprodukten i CSS blir lika. Utöver att kontrollera eventuella skillnader mellan slutprodukten i CSS har det även kontrollerat att det inte uppstått några felmeddelanden. Därför var det viktigt att kontrollera flera konverterare för att säkerställa att konverteringen från en kod till en annan skedde på ett bra eller för alla tre, lika sätt. I detta arbete testades tre stycken olika konverterare. Den första är programvaran Grunt.js, den andra är programvaran Bower och den tredje är verktyget beutifytools.com.

4.4.2 Testfallsmätningar

I denna del redovisas grundmätningar som visar hur CSS står sig mot CSS-preprocessorerna. Resultaten av dessa mätningar redovisar vilken teknologi som fungerar bäst tillsammans med vilken webbläsare. Detta gör att koden, som konverterats från de olika preprocessorerna till CSS för att ställas mot originalkoden skriven i CSS, har väldigt små skillnader i slutprodukten.

4.4.2.1 Litet Projekt

I pilotstudien var det tydligt att när storleken av koden var mindre var skillnaderna mellan CSS och LESS liten. Det kan tyckas självklart för den första processorn, för den första preprocessorn men inte självklart för de två andra. Det var därför av intresse att undersöka om det mönstret stämmer överens med de andra två preprocessorerna som skulle undersökas. Koden i denna mätning innehåller Bootstraps grundkod och representerar i verkligheten kodmängden av ett mindre projekt.

4.4.2.2 Stort projekt

Då grundmätningen representerar ett mindre projekt var det också av intresse att undersöka hur de olika preprocessorerna hanterade större projekt. Grundkoden för mätningen i testfallet som representerar stora projekt har dubblerats tio gånger.
Under pilotstudien upptäcktes att kodmängden och tiden inte följde en linjär utveckling, därför genomfördes ytterligare mätningar på grundkoden för att se om det stämmer för alla preprocessorer.

4.5 Mann-Whitney-testning

För att utvärdera resultaten av mätningarnas har bland annat medelvärde och standardavvikelser undersökts och analyserats, men för ett konkret svar på om resultaten är signifikanta har Mann-Whitneys u-test använts (Mann & Whitney 1947). Testet kontrollerar om det finns en signifikant skillnad, om en mängd testpunkter är större eller mindre än den andra och resultatet är P-värdet. För att kunna bestämma att någonting har en signifikant skillnad behöver P-värdet eller svaret som u-testet ger vara högre än 0,95 eller lägre än 0,05 i en skala från noll till ett.

Undersökningen genomfördes genom att sätta in mätpunkterna från testfallet som liknande ett stort projekt. I Mann-Whitney u-test kan enbart en mätserie ställas mot en annan per test

vilket gör att en tabell skapades för att lättare visualisera resultaten från den. I resultatet presenteras tabellen där alla preprocessorer ställs mot varandra en gång.

5 RESULTAT

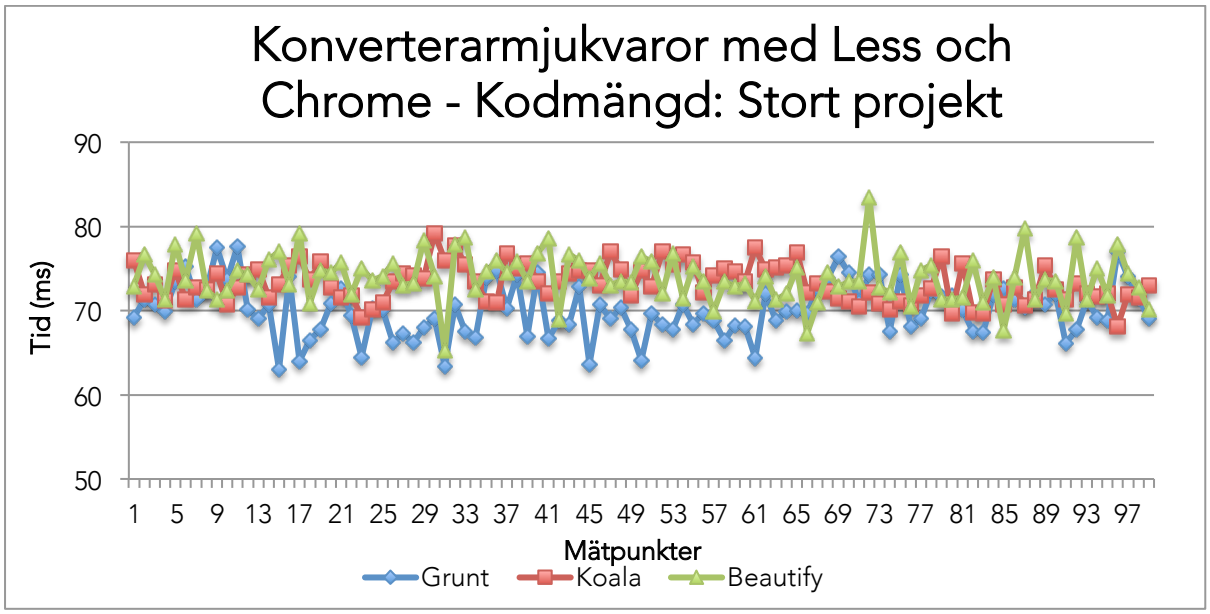
Detta avsnitt redogör resultatet av arbetet och inleds med genomgång av pilotstudien och vilka ändringar som gjordes. Vidare redovisas resultatet från det större experimentet.

5.1 Pilotstudie

Testerna förändrades då testmiljön byttes från PC till Mac. Detta medförde dock inte några problem. För mätningarna har flera olika programvaror installerats, varav alla har installerats utan problem.

5.1.1 Konverterare

Eftersom det krävs tredjepartsprogramvaror för att konvertera preprocessorkoden till CSS behöver flera konverterare mätas för att se om konverteringen mellan programvarorna leder till skillnader. I mätningarna för detta test har tre olika konverterare testats, Grunt.js, Koala och Beautifytools. Figur 8 visar resultatet av genomsnittet för de olika konverteringsprogrammen och visar på att det inte är någon märkbar skillnad dem emellan. När koden för respektive programvara granskades påträffades ingen skillnad, utöver att koden som Beautifytools konverterade var sämre indenterad samt att ny rad fanns efter varje class och id vilket ledde till att antalet rader för koden blev ca 2000 rader fler. Dock går denna skillnad ej att utläsa i figur 8. Något som inte går att förklara att Grunts mätning (blå) som verkar ha en något kortare laddningstid än Koala och Beautifytools. Notera att värdet på y-axeln i figur 8 startar på 50 (ms) för att tydligare se skillnader mellan konverterarna.

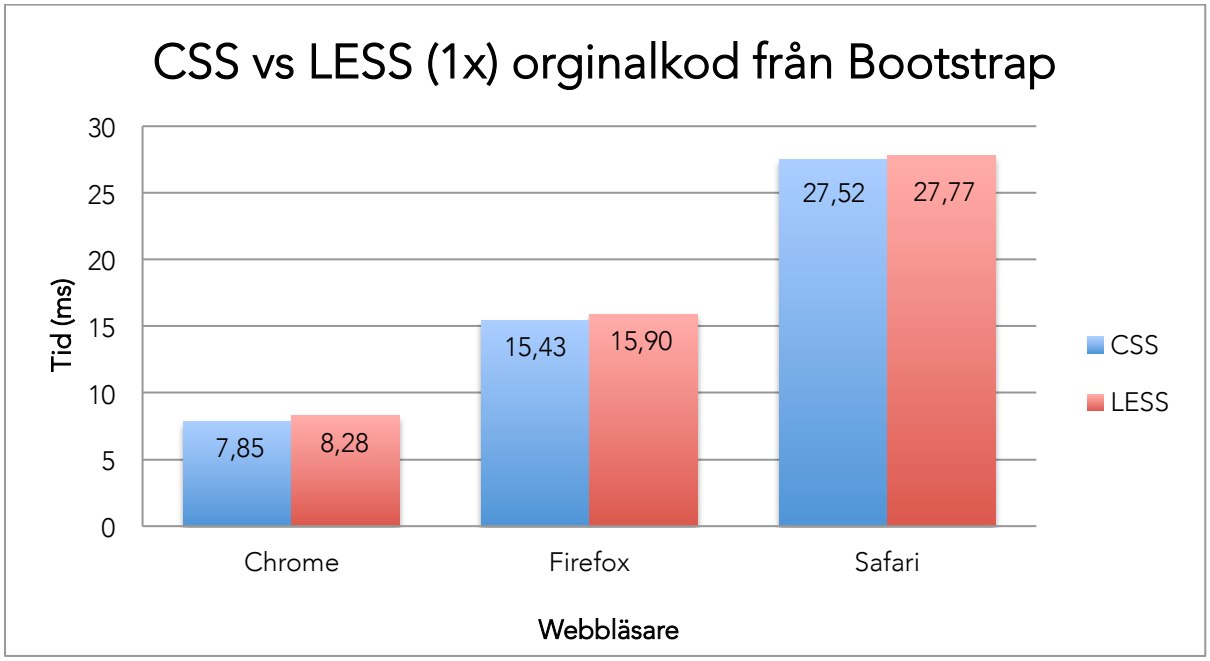


Figur 8 - Undersöker skillnaden mellan olika konverterare i Chrome

Efter undersökningen togs beslutet att det ej var nödvändigt fortsätta testa de olika konverterarna i olika testfall. Här togs beslutet att genomföra experimenten i arbetet med Grunt.js eftersom det ej finns någon märkbar skillnad dem emellan och den tycks ha bäst dokumentation och stöd för felhantering.

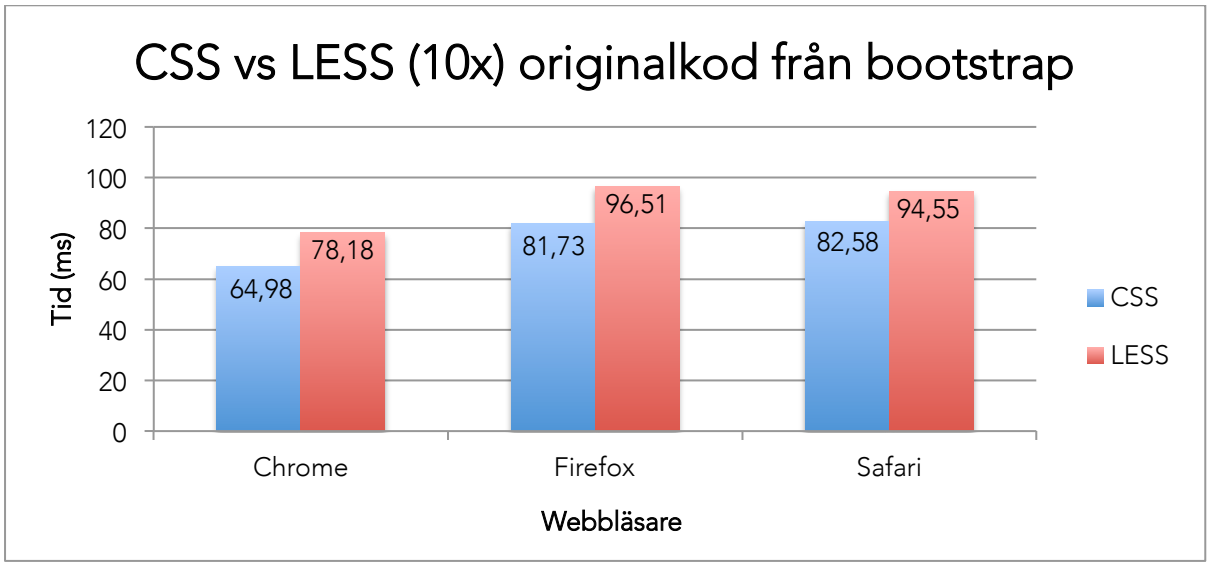
5.1.2 Mätningar

Stapeldiagrammet i figur 9 visar resultatet för tre olika webbläsare och skillnaden i laddningstid mellan CSS som originalkod och CSS konverterad från LESS. Värdena för staplarna visar medelvärdet för resultaten från 100 iterationer per omgång och 5 omgångar per webbläsare och preprocessor.



Figur 8 - CSS vs LESS – Resultaten från mätningarnas med grundkoden från Bootstrap

När mätningarna gjordes upplevdes tiden för mätningarna aningen höga. Koden som testades när CSS var originalkod var på 48 rader i Html-filen och har 6203 rader kod i CSS-filen. Därför skapades ytterligare ett test för att undersöka om tiden korrelerar med mängden kod och för att på så sätt kunna utesluta att annat påverkar. I det andra testet multiplicerades CSS-koden med 10, se figur 10.



Figur 9 - CSS vs LESS – Resultaten från mätningarna med grundkoden gångrad med 10

Av figur 10 att döma är de mätningarna oskiljaktiga. Genomsnittet för tre blåa staplarna är aningen lägre än för de tre röda, men om standardavvikelserna tas i beaktning skulle det med störst trolighet inte gå att skiljas åt.

En analys av figur 10 visar på att skillnaden är större mellan de olika mätningarna till skillnad mot den i figur 9. Utöver att resultaten som presenteras i figur 10 skiljer sig mer gentemot figur 9, speglar de även ett resultat som liknar ett större fullskaligt webbprojekt.

5.2 Experiment

I detta avsnitt redovisas mätresultaten för experimentet. Utvärderingen fokuserar på jämförelsen mellan de olika plattformarna och analysera de utförda mätningarna.

5.2.1 Plattform, hårdvaruspecifikation och parametrar

För mätningarna med olika testfall som genomförts användes en plattform tillsammans med olika webbläsare för att undersöka om hypotesen stämmer eller ej.

Plattform	MacBook Pro (Retina mitten 2012)
Operativsystem	OS X El Captain (10.11.3)
Processor	2,6 GHz Intel Core i7
Minne	16 GB 1600 MHz DDR3
Grafik	NVIDIA GeForce GT 650M 1024 MB
Lagring	Flashlagring 256 GB

Tabell 1 - Specificering av vad undersökningens mätningar är gjorda med

Hårdvara	Chrome	Firefox	Safari
MacBook Pro	Ja (V 50.0.2661.102 (64-Bit))	Ja (V 46.0.1)	Ja (V 9.0.3 (11601.4.4))

Tabell 2 - Webbläsare som testats i mätningarna

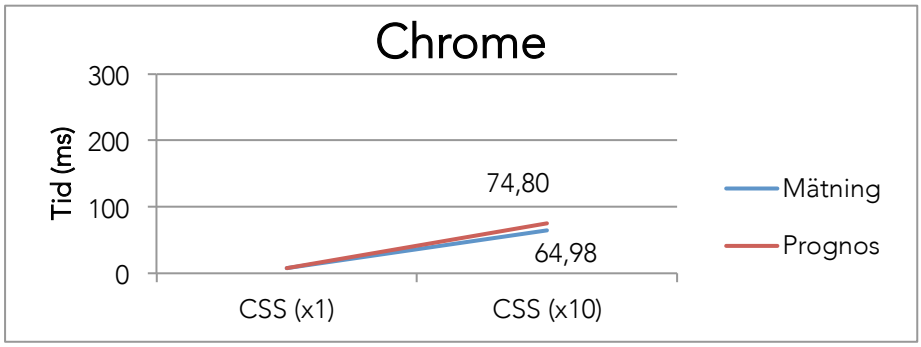
Plattform	Chrome	Firefox	Safari
CSS	Ja	Ja	Ja
LESS	Ja	Ja	Ja
Sass	Ja	Ja	Ja
Stylus	Ja	Ja	Ja

Tabell 3 - Vilka preprocessorer som testats på vilka webbläsare

5.2.2 Skillnaden mellan kodmängd och tid

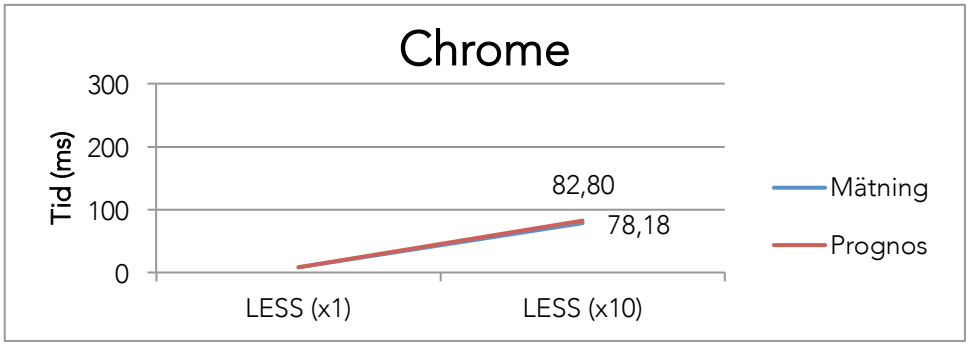
Efter att ha analyserat figur 9 och 10 från pilotstudien var det intressant att undersöka varför skillnaderna visade sig vare så stora testfallet litet projekt och stort projekt. Eftersom koden ökats gånger 10 borde även tiden öka lika många gånger. I figur 11 till 14 redovisas prognosen som en linjär ökning (röd linje) samt det faktiska resultatet (blå linje). Graferna visar att mätningarna i Chrome följer den tänka utvecklingen men Firefox inte. Mätningar har även gjort med Safari med lika utstickande utfall som Firefox.

I testet med CSS i Chrome i figur 11 är resultatet från mätningen under prognosen, men bara med 9,82 millisekunders skillnad. Chrome ligger nästan i linje med prognosen.



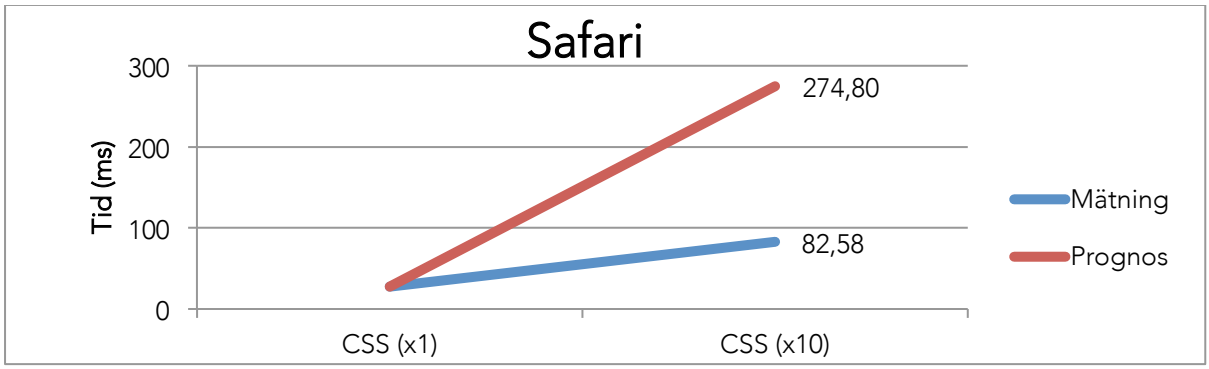
Figur 10 - Visar prognos och faktiskt resultat med CSS och Chrome vid jämförelsen av testfallet litet och stort projekt

Resultatet från mätningar av LESS i Chrome visas i figur 12. Denna visar på stora likheter med mätningar för figur 11, där *mätningen* befinner sig väldigt nära *prognosen*, men här med 4,62 millisekunders skillnad. I mätningarna med Chrome visade det sig att *mätningarna* är nästintill linjära såsom *prognosen* förutspådde.

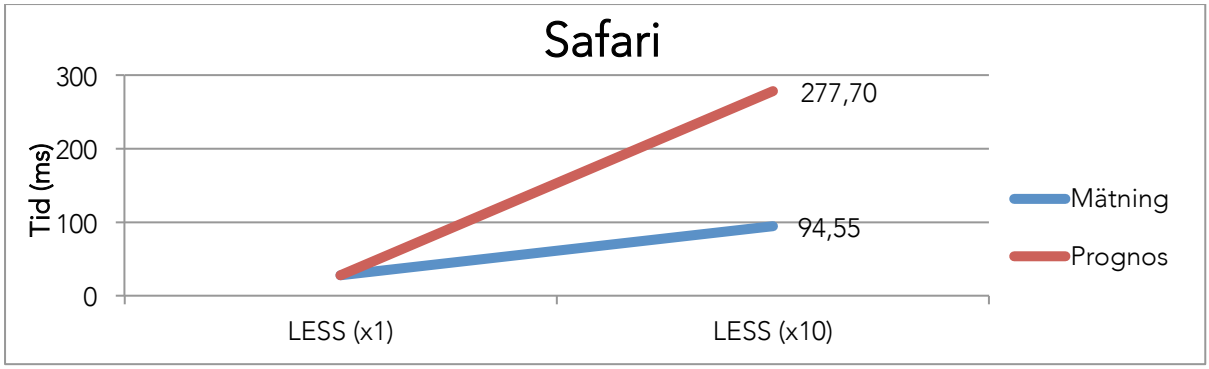


Figur 11 - Visar prognos och faktiskt resultat med LESS och Chrome vid jämförelsen av testfallet litet och stort projekt

Figur 13 visar resultatet från mätning av CSS i Safari. Till skillnad från Chrome skiljde sig *mätningarna* från *prognosen* mycket mer. Den faktiska tiden är mindre än den förutspådda *prognosen*. Det är tydligt att resultat inte följer en linjär utveckling.



Figur 12 - Visar prognos och faktiskt resultat med CSS och Safari vid jämförelsen av testfallet litet och stort projekt

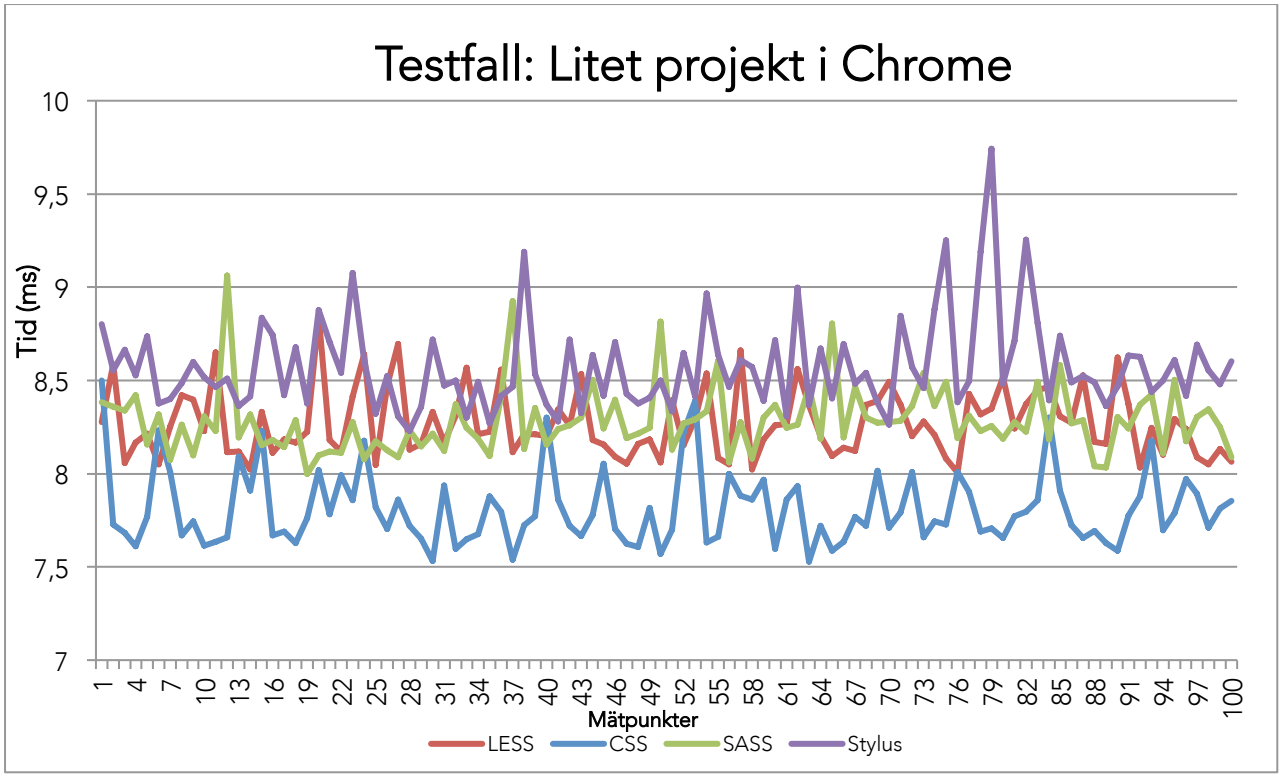


Figur 13 – Visar prognos och faktiskt resultat med LESS och Safari vid jämförelsen av testfallet litet och stort projekt

Firefox och Safari ger ett mycket högre resultat i mätningarna av grundkoden än vad Chrome gör i testfallet stort projekt. Vad detta beror på är svårt att säga, men någonting påverkar resultaten. Det är tydligt att Chrome i dessa mätningar är den webbläsare med kortast laddningstid, men det är dessutom den webbläsaren med minst skillnad mellan prognosen och den faktiska mätningen. En reflektion är att det finns tidskrävande uppgifter för webbläsarna som är konstanta. Dessa uppgifter har en procentuellt större påverkan ju mindre projektet är.

5.2.3 Testfall: Litet projekt

Figur 15 visar testfallet som representerar ett litet projekt. Resultaten från de olika preprocessorerna är väldigt klustrade, men laddningstiderna för CSS ligger sett till medelvärdet aningen under preprocessorerna. Det går dock inte utse en klar vinnare då standardavvikelserna från alla preprocessorer överlappar varandra. Överlappningarna är så stora och många att de plockats bort för att tydligare kunna visa medelvärdena. Resultaten i denna mätning är det sammanslagna av 5 omgångar med totalt 100 mätningar på omgång. Notera att värdet på grafens y-axel i figur 15 börjar på 6 (ms). Detta beslut togs då skillnaderna inte gått att urskilja om y-axeln börjat på 0.

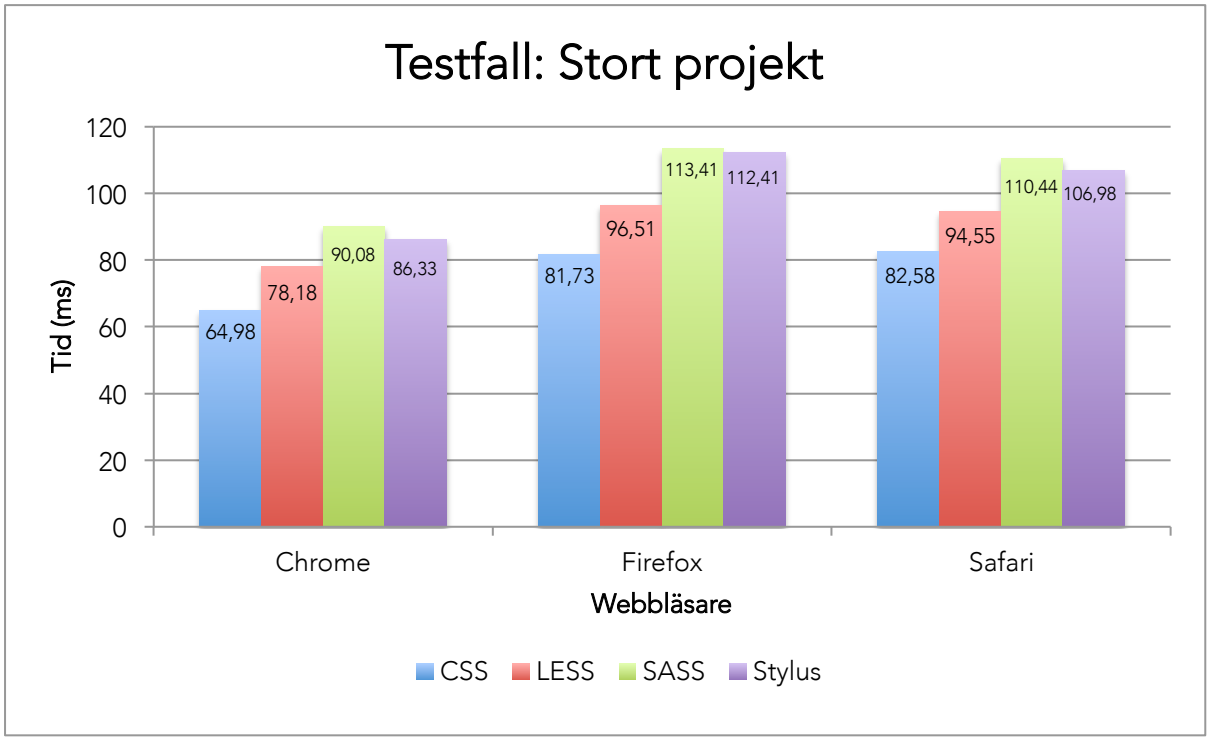


Figur 14 - Testfall: litet projekt – Visar alla mätresultat från CSS och de tre preprocessorerna

5.2.4 Testfall: Stort projekt

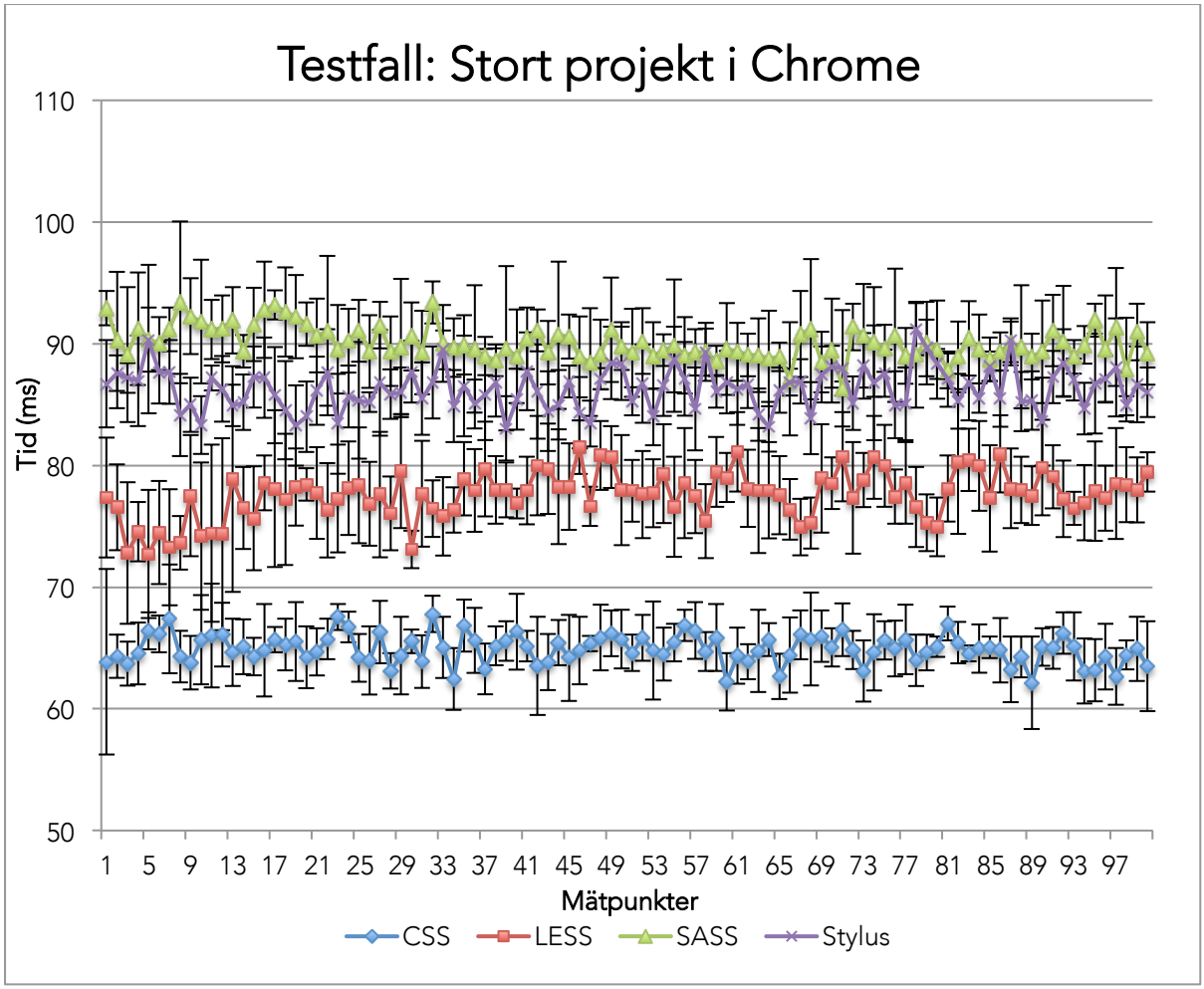
Vidare gjordes ett testfall för ett stort projekt. Figur 16 och 17 visar mätningarna baserade på Bootstraps grundkod. I figurerna presenteras både ett stapeldiagram med snittvärdet av alla mätningar för respektive preprocessor och webbläsare, men även en figur med graf för att kunna studera resultaten med standardavvikelser. Standardavvikelserna redogör för hur mycket värdena avviker från medelvärdet, det är ett sätt som ger en fingervisning om att en teknologi med ett värde är bättre än en annan. Värt att notera är att grafens y-axel i figur 16 börjar på 50 (ms) för att göra det enklare att se skillnaderna mellan CSS och preprocessorerna.

Figur 16 visar medelvärdet av totalt 500 mätvärden. Ur figuren går att utläsa att CSS har kortare kompileringstid än CSS-preprocessorerna. De olika webbläsarna skiljer sig åt i storlek men har liknande struktur. Googles webbläsare Chrome laddar sidan snabbare än både Firefox och Safari i samtliga tester. De två senaste har överlag liknande laddningstid i samtliga tester, dock har Safari något kortare laddningstid i tre av fyra fall. För mätningarna med Stylus är Safari aningen snabbare än Firefox med samma preprocessor. I detta testfall finns det första resultaten som tydligt pekar på motsatsen av Gringls (2013) resultat, men fler mätningar behöver analyserar innan någonting går att fastställa med säkerhet.



Figur 15 - Visar stapeldiagram över skillnaden mellan CSS, preprocessorer och hur de presterat i olika webbläsare

En tydligare bild av skillnaderna mellan CSS och CSS-preprocessorerna går att se i figur 17. Som figur 16 tidigare visade har CSS-mätningarna kortast laddningstider, men det blir ännu tydligare med standardavvikelserna beräknade i figur 17. Det är endast fåtal mätpunkter vars standardavvikelser korsar med den närmst liggande preprocessorn, LESS. Även LESS har värden som är relativt separerade i mitten av grafen. Däremot är det svårt att skilja på SASS och Stylus. Stylus medelvärde är aningen lägre än SASS men majoriteten av standardavvikelserna går in i varandra, vilket gör att det är inte går att konstatera med säkerhet att den enda är snabbare än den andra. Notera att värdet på grafens y-axel i figur 17 börjar på 50 (ms) för att tydligare se skillnaderna mellan CSS-resultaten och preprocessorerna.



Figur 16 - Graf med standardavvikelser för att visa på hur mycket de olika webbläsarna skiljer sig åt i Chrome

5.3 Mann-Whitney-testning

Den andra typen av testning som genomförts är Mann-Whitneys u-testning. Testet utfördes för att undersöka om något resultat från mätningarna har en signifikant skillnad. Mann-Whitney testet har genomförts på de två testfallen, litet projekt och stort projekt. Tabell 4 och 5 visar resultatet från testfallet för litet och stort projekt. För att ett resultat ska ses som signifikant, behöver P-värdet vara högre än 0,95 på skalan 0-1. Värdena i tabell 4 visar på att det inte är signifikanta skillnader mellan alla preprocessorer. Många av jämförelserna är inte nära gränsen (0,95 eller över) vilket också är förståeligt när man tittar tillbaka på figur 15 som Mann-Whitney-testerna är baserade på.

Testfall: Litet projekt i Chrome

Plattform	CSS	LESS	Sass	Stylus
CSS	X	0,85	0,95	0,95
LESS	X	X	0,0024	0,33
Sass	X	X	X	0,14
Stylus	X	X	X	X

Tabell 4 - Mann-Whitney u-test i Chrome – Testfall: litet projekt

Tabell 5 visar mätningarna från testfall av stort projekt. Här kan utläsas att alla test har ett P-värde högre än 0,95 vilket innebär att det är signifikanta skillnader dem emellan.

Testfall: Stort projekt i Chrome

Plattform	CSS	LESS	Sass	Stylus
CSS	X	0,95	0,95	0,95
LESS	X	X	0,95	0,95
Sass	X	X	X	0,95
Stylus	X	X	X	X

Tabell 5 – Mann-Whitney u-test i Chrome – Testfall: stort projekt

Mann-Whitneys u-test visar tydligt på skillnaderna mellan det lilla och stora testfallet. Ju större projektet blir desto större blir skillnaderna mellan preprocessorerna.

6 DISKUSSION

Detta avsnitt sammanfattar och för diskussion kring de utförda mätningarna och dess utfall, slutsatser dras och reflektioner kring arbetet som stort görs. En återkoppling till de frågeställningar som ställdes inför detta arbete görs även.

6.1 Sammanfattning

Frågeställningen och hypotesen som undersöktes i arbetet var om användningen av CSS-preprocessor istället för vanligt CSS på något sätt påverkade slutprojektet och om ja, på vilket sätt. Efter undersökningen går det med facit i hand att urskilja från mätningarna att alla preprocessorer som testats i undersökningen påverkat kompileringstiden av testwebbsidan negativt i olika grader. På testfallet som representerade ett stort projekt gick det redan att se på grafen (figur 17) att CSS som originalkod hade kortare laddningstid i genomsnitt. Detta innebär att resultaten i studien pekar på det motsatta jämfört med Gingles (2013) undersökningar.

Undersökningen i det stora hela har gått bra, med endast några få ändringar och problem på vägen. Bland annat missuppfattades hur arbetsprocessen med CSS-preprocessorer såg ut, vilket ledde till att strukturen av arbetet fick förändring. Det har även behövts många olika programvaror för att kontrollera att mätningarna är optimerade och rättvisa.

6.1.1 Diskussion

Resultaten i denna studie pekar på motsatsen av Gringls (2013) resultat. Även om hon i sin studie även använt SASS går det inte att dra allt för stora slutsatser, då datan som användes i hennes arbete inte är redovisat. Hennes resultat pekar dock inte på en klar skillnad mellan vanlig CSS och CSS-preprocessorn SASS, snarare en marginell, till SASS fördel. Resultaten i denna studie pekar på en större och klarare skillnad med lägre laddningstid för CSS. Den påverkande skillnaden mellan CSS och preprocessorernas laddningstid i denna undersökning är kodmängden. Kodmängden blir större och tar därför längre tid att kompilera. Skillnaden mellan preprocessorerna blir större och tydligare ju större projektet är.

Men med facit i hand har CSS lägre laddningstider i samtliga testfall som gjorts. När koden konverterats och sedan testats har det inneburit att den transformerade CSS-koden blivit större. I slutändan är detta inte bra då det betyder längre tid för att webbsidan ska ladda klart. CSS-preprocessorer har tillkommit med utvecklaren i fokus, medan användaren verkar ha glömts bort i sammanhanget. En utmaning är att väga dessa två mot varandra, utvecklarens effektivitet mot användarens laddningstid, och till vilken grad det är mest fördelaktigt med CSS-preprocessorer utan att användaren påverkas negativt.

När Mann-Whitneytestet genomfördes blev resultaten tydligare. I testfallet litet projekt är skillnaderna inte stora (tabell 4), CSS hade signifikanta skillnader endast mot SASS och Stylus, men endast ett P-värde på 0,85 när den jämfördes mot LESS. När mätningarna från testfallet stort projekt jämfördes i Mann-Whitney testen blev resultatet desto tydligare, CSS blev en tydlig vinnare men nästan alla preprocessorer hade mätvärden som hade signifikanta skillnader sinsemellan. CSS som originalkod hade ett P-värde på 0,95 eller högre mot alla de andra preprocessorerna, vilket betyder att det finns en signifikant skillnad.

Användningen av u-testet upplevdes givande, då det ibland kan upplevas tydligare med värden i en tabell än mätpunkter och värden i grafer. Med Mann-Whitneys u-testet underlättade det speciellt då det enkelt gick att tyda om ett resultat av mätningen hade ett högre eller lägre värde än 0,95.

En intressant reflekion är att preprocessorerna marknadsförs som minimala för utvecklaren, men när koden konverteras till den slutliga CSS-koden för att kunna köras i webbläsaren blir den större än om den skrivits i CSS direkt. Problemet tros ligga i konverteringen, kod skriven med någon av preprocessorerna blir inte likt CSS-koden, vilket i slutändan leder till längre laddningstid. Frågan är om utvecklare ska lägga fokus på att lära sig att arbeta med någon av preprocessorerna när de i slutändan verkar förlänga laddningstiden för användaren? Amazon och Google har som tidigare nämnts, undersökningar som pekar på att de förlorar intäkter till ett värde av 1 % av deras omsättning för varje sekund deras hemsida laddar. Google har sett att när en sökning tar 5 sekunder eller längre minskar trafiken med 20 %. Det finns även undersökningar som menar att börsmäklare kan för förlora upp emot 30 miljoner kronor när deras system är 5 millisekunder efter på plattformen (Liddle 2008). För företag handlar det om förlorade intäkter, kunder vill inte förloras på grund av en långsam laddning på sidan.

Ett problem av etisk karaktär som uppstod är var vid valet av var scriptet som utförde tidtagningen skulle placeras. I figur 8 redovisades skillnaderna mellan de två olika alternativen, när scriptet placerades direkt i Html-filen jämfört med när det placerades i tredjepartsprogramvarorna Greasemonkey och Tempermonkey för Chrome och Firefox respeltive. Testet för att undersöka skillnaderna innehöll samma kodmängd som testfallet för testfallet stort projekt. Resultatet visade på några millisekunders skillnad för Chrome och mellan de två alternativen, samma resultat kunde även konstateras för Firefox. När testet genomfördes antogs att denna skillnad inte var en konstant kostnad av tid utan en snarare en linjär ökning. Vid eftertanke borde ett liknande test utförts med testfallet litet projekt för att undersöka eventuella skillnader. Detta har dock inte genomförts utan enbart ett test med ett testfall har undersökts. Efter att mätningen hade genomförts togs beslutet att placera skriptet i Html-filen då detta innebar att undersökningen även kunde innefatta Safari som inte har en tredjepartprogramvara liknande Greasemonkey eller Tempermonkey. Om placeringen av skriptet medför en annan skillnad än den som yttrar sig i den mätningen som genomfördes är oklart, men det är någonting som bör tas i beaktning vid utveckling eller fortsatt arbete.

Någonting som skulle kunna förändras är att lägga till en till plattform att köra på, alltså t.ex. PC, då antalet användare av PC är mycket fler än till Mac. Några större skillnader i mätningarna är svårt att säga om det skulle bli, men den främsta anledningen till att köra PC som plattform är att kunna utföra mätningar i Microsofts webbläsare Explorer/Edge som är den tredje största webbläsaren enligt W3Schools (2016).

I senare ett skede av arbetet krävdes en förändring och det var vilka testfall som skulle undersökas och genomföras. Någonting som nämns som en av fördelarna med preprocessorer är matematiska beräkningar. Det är alltså möjligt att göra beräkningar direkt i preprocessorn som i annat fall skulle behöva göras i JavaScript. Det var därför länge intressant att utföra mätningar där det ingick, men efter research upptäcktes ingen bra

infallsvinkel för att kunna erhålla jämförbara mätningar. Dock är det möjligt att det skulle kunnat gå att jämföra bara mellan preprocessorerna.

I denna undersökning användes Bootstraps egna open-scource-kod för att utföra mätningar på. För att säkerställa att resultaten är korrekta skulle det vara bra om även annan kod gjorts mätningar på. Det skulle lett till att resultaten i denna studie kunnat vetenskapligt säkerställas. Optimalt skulle vara att få tag i en ”riktig” webbsidas kod, till exempel Amazon, eBay eller SJ för att på så sätt kunna verifiera resultaten.

En intressant notis är att av alla mätningar som genomförts under undersökningen har fläkten gått igång under mätningssessionerna men endast när mätningarna genomförts med Safari som webbläsare. En tanke före mätningar påbörjades var att Safari skulle vara den webbläsare som skulle prestera bäst när det blev klart att mätningarna skulle genomföras på Mac i stället för PC. Vad det beror på är svårt att uttala sig om, för resultaten har ju inte varit de sämsta av de tre webbläsarna som

6.1.2 Framtida arbete

I webbutveckling är CSS och CSS-preprocessorerna idag väldigt centralt, så lär det fortsätta några år till. I framtiden är det svårt att veta hur det kommer se ut, men att det kommer vara annorlunda är självklart, IT-branschen är i ständig utveckling. Preprocessorerna ger några fler funktioner som vanlig CSS saknar men inget revolutionerande och kanske är det just något revolutionerande som CSS behöver för att kunna överleva som en egen identitet. För det första måste konverteringen optimeras, men nummer två är flera nya funktioner. Annars är risken att det kan komma en annan lösning som kan kapsla in CSS. Redan idag kan JavaScript jobba med CSS implementerat i koden. Dock finns ingen lösning för att skapa hela projektets CSS i JavaScript, men eftersom det till viss del redan går idag, är en full övergång ett möjligt scenario att föreställa sig. Dock är motsatsen, att CSS får fler funktioner och större ansvarsområde desto svårare att tänka.

Någonting som skulle vara intressant är att skapa en preprocessor med större fokus på konverteringen, för att kunna optimera och få ett resultat likt koden skrivet i CSS från början. Om man skapade en egen preprocessor skulle det även vara intressant undersöka om att eventuellt att lägga till funktion eller förändra kodstrukturen.

REFERENSER

Beautifytools.com (2016) – Tillgänglig: http://beautifytools.com/css-to-stylus-converter.php [2016-03-29]

Blog.millermedeiros.com (2016) – Tillgänglig: http://blog.millermedeiros.com/the-problem-with-css-pre-processors/[2016-02-23]

Butkiewicz, M., Madhyastha, H. V., & Sekar, V. (2011). Understanding website complexity: Measurements, metrics, and implications. In proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference (pp. 313-328). ACM.

Cai, J.-Y., Nerurkar, A., Wu, M.-Y., (1998). Making benchmarks uncheatable, Presented at the Computer Performance and Dependability Symposium, 1998. IPDS '98. Proceedings. IEEE International, pp. 216–226. doi:10.1109/IPDS.1998.707724

css-tricks.com (2016) - Tillgänglig: https://css-tricks.com/poll-results-popularity-of-css-preprocessors/ [2016-02-20]

developer.mozilla.org (2016) - Tillgänglig: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript [2016-02-23]

Gardner, B. S. (2001). Repsonive Web Design: Enriching the User Experience. Sigma Inside the digital ecosystem, 11(1), pp. 13-19.

Getbootstrap.com (2016) – Tillgänglig: http://getbootstrap.com/2.3.2/extend.html [2016-02-24]

GitHub.com (2016) – Tillgänglig: https://github.com/sass/sass/releases?after=1.0.4 [2016-02-22]

Google.se (2016) – Google Trends – Tillgänglig: https://www.google.se/trends/explore#q=Sass%2C%20%2Fm%2Fogjd0jv%2C%20SCSS&date=1%2F2010%2073m&cmpt=q&tz=Etc%2FGMT-1[2016-02-22]

Harb, E., Kapellari, P., Luong, S. & Spot, N. (2011). Responsive Web Design as a Wicked Problem. In proceedings of the 9th Student Interaction Design Research Conference. 94-97. SIDeR.

Jeong, W., & Han, H. J. (2012). Usability study on newspaper mobile websites. OCLC Systems & Services, 28(4), 180-198.

Jim Liddle (2008) – Tillgänglig: http://blog.gigaspaces.com/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/ [2016-05-24]

Kadlec, T. (2013). Implementing responsive design. Berkeley, CA: New Riders.

Khan, F., Foley-Bourgon, V., Kathrotia, S., Lavoie, E. (2014). Using JavaScript and WebCL for Numerical Computations: A Comparative Study of Native and Web Technologies. In proceedings of the 10th ACM Symposium on Dynamic Languages. DLS '14. New York, NY, USA, ACM. s. 91–102.

Kim, B. (2013). Chapter 4: Responsive Web Design, Discoverability, and Mobile Challenge. Library Technology Reports, 49(6), 29-39.

Lesscss.org (2016) – Tillgänglig: http://lesscss.org/about/[2016-02-25]

L. Gringl (2013). Modular CSS in practice, Bachelor Thesis, St. Pölten University of Applied Sciences

Mann, H.B. & Whitney, D.R. (1947) On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. The Annals of Mathematical Statistics. 18 (1), s. 50– 60.

Matsudaira, K. (2013). Making the mobile web faster. Communications of the ACM, 56(3), 56-61.

Mohorovičić, S. (2013). Implementing responsive web design for enhanced web presence. In Information & Communication Technology Electronics & Microelectronics (MIPRO), 2013 36th International Convention on (pp. 1206-1210). IEEE.

php.net (2016) - Tillgänglig: http://php.net/manual/en/intro-whatcando.php [2016-02-23]

Prabhu, A. (2015). Introduction to Preprocessors, in: Beginning CSS Preprocessors. Apress, pp. 1–12.

Rajamony, R. & Elnozahy, M. (2001). Measuring Client-perceived Response Times on the WWW. In proceedings of the 3rd Conference on USENIX Symposium on Internet Technologies and Systems - Volume 3. USITS'01. Berkeley, CA, USA, USENIX Association. s. 16–16.

Sitepoint.com (2016) – Jo Liss's musings on enlightened software development. Tillgänglig: http://www.sitepoint.com/6-current-options-css-preprocessors/ [2016-02-20]

Solitr.com (2016) – Tillgänglig: https://www.solitr.com/blog/2014/01/css-preprocessor-benchmark/[2016-02-20]

Standout.se (2015) - http://standout.se/2015/08/20/skillnad-mellan-ruby-och-ruby-on-rails/ [2016-03-26]

Wei Jiang, Meng Zhang, Bin Zhou, Yujian Jiang, Yingwei Zhang (2014). Responsive Web Design Mode and Application, Workshop on Advanced Research and Technology in Industry Applications (WARTIA). IEEE, Ottawa.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B. & Wesslén, A. (2012) Experimentation in Software Engineering. Berlin Heidelberg: Springer-Verlag. ISBN 978-3642290435.

W3Schools.com (2016) – Tillgänglig: http://www.w3schools.com/browsers/browsers_stats.asp [2016-02-23]

W3.org (2016) – Tillgänglig: https://www.w3.org[2016-02-23]