Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Master's Thesis

Performance-optimized A/B-Testing for E-Commerce-Websites

Aram Yesildeniz

aram.yesildeniz@studium.uni-hamburg.de
M. Sc. Informatics
Matr.-No. 6890370

1st supervisor: Dr. Wolfram Wingerath
2nd supervisor: Benjamin Wollmer

Hamburg, 1st of September 2021

A distributed system is one where the failure of some
computer I've never heard of can keep me from getting my work done.
– *Leslie Lamport*

Contents

1 Introduction

[tbd]

- Explain structure and main goal of this thesis
- Describe shortly all sections from this chapter and what the reader can expect
- Give short outlook to following chapter

1.1 Research Question

The e-commerce industry is booming and there are no signs that this trend is reversing; on the other hand. Performance plays an important role in terms of customer satisfaction and how this directly affects business revenue. To better understand e-commerce website visitors, page tagging is widely used and implemented.

Several questions and issues can arise in this area and context, such as: Does page tagging affect the website's performance? Intuitively, it can be said that loading additional JavaScript will reduce the performance of the website, depending on parameters such as the script size and network condition. But are there more unpredictable side effects? Do the various techniques of embedding a tracking script affect the data collected and measured? Will the various tracking scripts supplied interfere with each other?

A hypothesis of this work is that tracking tools slow down the monitored websites, reduce the speed and performance of the website and thus have an unfavourable effect on the user experience.

These questions are to be investigated within the scope of this thesis.

1.2 Goal of this Thesis

This thesis has several goals:

The Internet and websites in general are complicated, complex, and tangled. Although basic HTML structures are standardized, each website follows its own form and is unique and sui generis. In order to conduct a controlled experiment and test hypotheses, one goal is to approximate real websites with an artificial, laboratory-generated website that is completely controlled and manipulated by the researcher.

The aim is to create a reliable, but also convincing test environment in order to model and reproduce real behaviour.

Once the test environment is up and running, performance measurement issues need to be addressed. The aim is to measure, collect, visualize and analyse performance data.

As we will see in chapter X, there are many metrics for measuring performance. Another goal of this work is to establish something like a taxonomy of performance metrics.

1.3 Chapter Outline

[tbd]

Chapter 1 was about... In Chapter 2 we see, Chatper 3...

2 Introduction to E-Commerce and Web Analytics

The focus of this thesis revolves around web performance of e-commerce websites. Web performance can be considered a subset and specialized form of web analytics. To provide context, this chapter provides an overview of e-commerce and web analytics.

Section 2.1 provides an overview of e-commerce and explains why performance is important for e-commerce companies.

Section 2.2 discusses how e-commerce companies can be audited using web analytics and what techniques and metrics are available.

Once context is provided, this thesis takes a step further and dives into web performance in chapter 3.

2.1 E-Commerce

This section gives an introduction to e-commerce. Before going into the details, the emergence of e-commerce is described, starting with the Internet in section 2.1.1. In the following, I will briefly discuss the history of e-commerce and its types in section 2.1.2. The relationship between user satisfaction and the website performance is covered in section 2.1.3, which then leads to the section 2.2 which is about web analytics.

2.1.1 The Internet: Overview

In the last 50 years, a new technology emerged, spread over the entire world and influenced many aspects of most peoples life. Within the turmoil of the cold war, the United State's *Advanced Research Projects Agency* (ARPA) established in 1957 a communication network to bring together universities and their researches all around the country in order to be able compete against the USSR [CA11]. What started as a tool for scientific collaboration evolved half a century later into the *Internet*, a global network and phenomenon, to which every user with a dedicated device has access and can contribute to. The internet is an integral part, if not the backbone of today's everyday life. Users of the internet use it for almost everything, from sending emails, watching television, chatting with friends, order lunch, checking the weather for the next day or renting motorized scooters.

In 2021, the internet has 4.66 billion users, which is around 60% of the world population.[1] Compared to 2020, the number of internet users increased by 7.3%. In Europe, more than 90% of the population are internet users. For a developed country like Germany, the numbers are even more impressing: 94% of the German population are using the internet with an average daily time of over five hours.

Those numbers demonstrate impressive that the internet is an integral part of our daily life. Along the rise of the internet, transactions and processes falling under the term of e-commerce are climbing as well. Before discussing the term "e-commerce" and take a grasp at its history and types, some statistics are presented to demonstrate the importance of e-commerce.

2.1.2 E-Commerce

2.1.2.1 The Importance of E-Commerce

From the global data report[2], one can read out that over 90% of the world population visited an online retail site and over 76% of the world population purchased a product online. For or a western country like Germany, the figures are higher: 92.5% of the German population visited an online retail site and over 80% purchased a product online. And the usage is expanding: the growth of the amount spent within the category food and personal care is 28.6%, and 17.6% for the category fashion and beauty.

E-commerce sales have grown steadily over the past 20 years, topping to 57.8 billion in 2019.[3]

The COVID-19 pandemic with its implications had and still has an not negligible impact on the growth of e-commerce. Several measures were taken to stop the spread of the virus and the number of deaths, one of which was to minimize physical interaction between people. This leads consequently to a shift of human interactions to the internet. Along this, e-commerce benefits. Bhatti et al. [BAB+20] conclude that "e-commerce enhanced by COVID-19".

2.1.2.2 Brief History of E-Commerce

E-Commerce, or electronic commerce, is according to the *Encyclopædia Britannica* about "maintaining relationships and conducting business transactions that include selling information, services, and goods by means of computer telecommunications networks."[4] In short, e-commerce is about buying and selling products and services via the internet.

The success of e-commerce is closely linked to the tremendous advances in Internet technology in recent years: The development of the *Electronic Data Interchange* (EDI) start-

[1] Following statistics are taken from `https://datareportal.com/reports/digital-2021-germany` [14.05.2021]
[2] `https://datareportal.com/reports/digital-2021-germany` [14.05.2021]
[3] `https://einzelhandel.de/presse/zahlenfaktengrafiken/861-online-handel/1889-e-commerce-umsaetze` [14.05.2021]
[4] `https://www.britannica.com/technology/e-commerce` [19.05.2021]

ing in the 1960s standardised the communication between two machines. Personal computers were introduced in the 1980s, and one of the first examples of an online shop is the *Electronic Mall* opened by CompuServe in 1984. Another crucial milestone is the launch of the *World Wide Web* (WWW) in 1990, which made the internet accessible to everyone. With social media visible on the horizon from the 2000s, new possibilities for businesses and consumers alike to participate in e-commerce arise, for example, by enabling new marketing strategies or providing new sales channels. New devices such as smart phones and tablets lowered the barrier to participate in e-commerce. While e-commerce was available at any time, the new devices brought flexibility and mobility, making e-commerce available everywhere [Her19].

With the continued advancement in technology, e-commerce can expect a bright future with trends such as AI recommendation systems, outstanding UX thanks to virtual reality, or even more simpler payment methods through cryptocurrencies.[5]

2.1.2.3 Types

There are several types in e-commerce and they emerge from the possible combinations between the actors *Business*, *Consumer* and *Government* [SSMG17], as shown in table 2.1. The research question and the context of this thesis revolve around e-commerce websites in the sense of online shopping (B2C). B2C will be briefly discussed below, types such as C2C or C2G are not of interest and will not be considered further.

	Business	Consumer	Government
Business	B2B	B2C	B2G
Consumer		C2C	C2G
Government			G2G

Table 2.1: Types of e-commerce.

B2C Business to Consumer in e-commerce describes basically online shopping, by means of a business offering its services and products to the consumer over the *WWW*. The consumer can browse through the products and services presented within an online shop and order them directly via the website. A variety of payment and delivery options conclude the B2C type [Hei20].

For an aspiring business, there are several ready-made software solutions for setting up an online store, as for example. *Shopify*, *ePages*, *Magento* or *WooCommerce* [SBR+19].

A famous example of a B2C company is *Amazon*. On the 16th of July in 1995, Amazon launched as a website and entered the stock market on the 15th of May 1997 [SB19]. Amazon has been successful, with the stock starting at $1.5, which is at around $3200 as

[5] `https://www.spiralytics.com/blog/past-present-future-ecommerce/` [19.05.2021]

of this writing.[6] Today, Amazon employs over 1 million people[7] and serves the desires of 200 million paying prime members.[8]

By taking a quick look at the pros and cons of an online store, it becomes clear that some of the advantages are that: there is no need of a real, physical store to showcase and sell the products; the virtual shop is available to the consumer at any time and has no closing hours; there is a high potential for the online shop as it is part of growing market; online business is scalable; due to tracking algorithms, precise targeting as well as data analysis is possible; to start an online business, there is not so much floating required and there are generally lower costs; it is possible to provide a personalized customer experience.

Some disadvantages are that the speed of market is rapid, competitors arise everyday everywhere and technology evolves quickly while consumers expectations go high [Her19], [LO20].

Another disadvantage is that there is no direct or physical connection with the consumer. As described above, online shopping takes place on the virtual WWW, i.e. personal interaction between buyer and seller is not possible and the shopping experience takes place on a website, from which it follows that the overall virtual user experience must be excellent in order to compete.

In the next section, I will describe the findings between the correlation between user satisfaction and the performance of the retailers web presence.

2.1.3 User Satisfaction and Performance in E-Commerce

The aim of this thesis is not to deep dive into terms and concepts or the non-trivial problem of defining user satisfaction, usability or the like. Therefore the term user satisfaction is in this context loosely defined as how happy the user is with the website he or she interacts with.[9]

Performance can be understood as the speed of an online shop, e.g. how long it takes the page to load, how quickly the user can interact with the page, and how the user perceives the performance of the website. In section 3.3 I will discuss that measuring performance is not so trivial and there are a lot of ideas and metrics to measure it (section 3.4).

[6] https://finance.yahoo.com/quote/AMZN?p=AMZN [19.05.2021]
[7] https://www.statista.com/statistics/234488/number-of-amazon-employees/ [19.05.2021]
[8] https://www.statista.com/statistics/829113/number-of-paying-amazon-prime-members/ [20.05.2021]
[9] For a discussion see "User satisfaction measurement" in [IKOK10]

2.1.3.1 Studies about User Satisfaction and Performance

There are numerous studies that shed light on various aspects of the relationship between user satisfaction and performance. A selection is summarized here.

Unbounce [Unb] and Awwwards, in partnership with Google [GA], offer insights into user profiles. In terms of gender, young women are the most demanding consumers and are less likely to buy from slow sites. In general, people between the ages of 18 and 24 have higher expectations of a site's speed than their older counterparts. There are also differences between nations and regions, for example people from Japan have the highest expectations, which is almost certainly related to technological advancements in that country.

DeviceAtlas [Dev] provides insights into devices in use. Their studies show that mobile users are more likely to buy products and services than their colleagues using a desktop computer, where iOS users have generally more expectations regarding site speed.

Akamai [Akab] offers insights into psychological aspects of user satisfaction and performance. Their observations show that relaxed and calm users perceive pages faster than stressed or hurried users. Also users experience websites more slowly while on the go.

There are many other real world examples and studies that prove and demonstrate the importance of website speed in terms of user satisfaction and ultimately sales: *Amazon* [Lin] found out that a decrease of 100 ms in page loading leads to -1% conversion rates. If the site loads 100 ms faster, *Walmart* [CKR] observed that the revenue increases by 1%. For *Zalando* [SKG$^+$], increasing site speed by 100 ms has led to an uplift of 0.7% revenue per session.

Search engine optimization is heavily impacted by load speed: For *Google* [May], 500 ms slower sites led to a decrease of 20% in traffic. *GQs* [Mos] traffic increased by 80% after the page load went down from 7 s to 2 s. And for *Pinterest* [MAC], 40% faster loads led to 15% more SEO traffic.

User engagement and satisfaction also depend heavily on loading times: *Forrester* [For] noted an increase of 60% for the session length while brining down the load time by 80%. *Akamai* [Akaa] monitored that the bounce rate climbed up incredible 103% when the load time increased by 2 seconds. And for the *AberdeedGroup* [Abe], the customer satisfaction dropped by 16% at one more second delay in response times.

In summary, it can be said that many studies and practical examples prove and demonstrate that faster websites and online stores lead to a better user experience and usually to happier customers. In commercial terms, one can conclude that page speed equals money.

In order to properly test the effects of performance on users, a scientific method is required. A/B testing as a controlled experiment is one of them and will be explained in the next section. After discussing A/B testing, I will move on to examining *Web Ana-*

lytics, a term that encompasses methods, tools, and instruments for companies to better understand their business and customers.

2.1.3.2 A/B Testing

Controlled experiments like A/B testing are not a new tool for scientists and researchers and were used as early as the 1920s [KL17]. With the advent of the Internet in the 1990s, the concept was adopted into the online domain and is now used by large companies such as Amazon, Facebook or Google to test ideas and hypotheses directly on a live system. Controlled experiments such as A/B testing are used to aid decision making and provide a "causal relationship with high probability" [KL17]. They enable a data-driven and quantitative validation of the hypothesis [Mor18].

Controlled experiments help to test hypothesis and questions of form: "If I change feature X, will it help to improve the key performance indicator Y?"

To answer this question, two systems are needed: *Version A*, the control variant or default version, and a slightly different *Version B*, called the treatment. If more than two versions or one treatment should be evaluated at the same time, an A/B/n split test has to be implemented. With a univariable setup, only one variable differs between the systems; with a multivariable structure, several variables are changed at the same time.

Usually, the users of the system are randomly split into two groups and testing is directly performed with real users on a production system. It is advantageous, also compared to other experimental set-ups, that the users and participants are not aware that they are part of an experiment, which leads to fewer biases and side effects. In order to measure the differences and the user behaviour, web analytics has to be integrated within the system [KL17].

A brief and general discussion of controlled experiments in computer science can be found in chapter X.

To continue with the question of performance and user satisfaction, A/B testing allows two different versions of the same site to be served to two groups at the same time, one site being slow and the other being fast, without users knowing.

An implemented web analytics system makes it possible to measure how the various systems and user groups behave. What web analytics exactly is, what tools are available and what a web analytics process looks like, is discussed in the next section.

2.2 Web Analytics

This section is about web analytics. I will first discuss definitions and give a short introduction to web analytics in section 2.2.1. Then, a brief history of web analytics will be given in section 2.2.2 to help contextualize. After describing how web analytics can

be applied in practice in section 2.2.3, section 2.2.4 discusses methods for data collection. Finally, section 2.2.5 discusses metrics used in web analytics.

2.2.1 Introduction to Web Analytics

What is *Web Analytics*? A review of the literature shows that there are several definitions:

Nakatani et al. state that "Web analytics is used to understand online customers and their behaviors, design actions influential to them, and ultimately foster behaviors beneficial to the business and achieve the organization's goal" [NC11]. According to this definition, web analytics is about gaining insight into the users of the system, not only who or what they are, but also how they interact with the system. In addition, the definition emphasizes that the underlying motivation of web analytics is to achieve business goals.

Singal et al. provide a more technical definition by pointing out that "Web Analytics is the objective tracking, collection, measurement, reporting and analysis of quantitative internet data to optimize websites and web marketing initiatives" [SKS14]. Again, the ultimate goal is to drive business, but using data science methods and tools, such as big data tracking, collection and analysis.

Bekavac et al. provide a similar definition by pointing out that web analytics is "the analysis of qualitative and quantitative data on the website in order to continuously improve the online experience of visitors, which leads to more efficient and effective realization of the company's planned goals" [BGP15].

Zheng et al. describe four main use cases for web analytics [ZP15]:

- Improving the overall design and usability, for example of the navigation or layout of the website.
- Optimize for your business goals: Whatever goals the business is trying to achieve, generating conversions is the goal.
- Monitor campaigns: Understand and measure the success of advertising campaigns.
- Improve performance by examining metrics such as page load time.

Summarizing the above definitions, it is noticeable that web analytics consists of two important elements: a data-driven, information-oriented and technical element of collecting and analysing data about users, and a commercial and business-oriented element that provides the main motivation for collecting the data primarily by setting business goals.

Web analytics can also be complex, and there are some obstacles and challenges to overcome. Kumar et al. describe some of the hurdles as follows [KO20]: The analysis and interpretation of the data and the goals and measures derived from it are reactive rather than anticipatory, since "predictive modelling applications" for web analytics are not yet on the market.

Big data, that is the volume, variety and velocity of data collected, can be challenging to manage, for example, important insights can be lost or overlooked.

Privacy and the information collected from visitors and customers can be another challenge in terms of inappropriate use of data and GDPR compliance.

2.2.2 Brief History of Web Analytics

The history of web analytics can be described as a transformation from an IT domain and a technical log file analysis tool to a sophisticated, polymorphic tool for marketers.

Each time a user requests an HTML file or other resource from a web server, the server makes an entry in a special log file [SKS14]. The first log entries followed the *Common Log Format* (CLF) which provides rudimentary information such as the date, the HTTP status code or the number of bytes transmitted. In 1996, the *Extended Log Format* (ELF) was introduced with more flexibility and information in mind. Thanks to the standardized format of the log files, it was possible to create software that evaluates the log files and presents them to users in a readable form. *GetStats* was one of the first tools which generated statistics and user friendly output for analysts [CP09]. In 1995, Dr. Stephen Turner developed *Analog*, the first free software for analysing log files [ZP15]. Log file analysis will be further discussed in section 2.2.4.1.

What was initially mainly interesting for maintenance and IT staff, who for example answered the question of how many 404s occurred on the server, developed into an interesting website information pool for marketers.

As the available information increased, it became clear that the data could be used for more than just analysing server behaviour. But log file analysis was not enough to provide details about how users interact with the site. Web analytics underwent a transformation from log analysis to user data tracking, analysis, and reporting.

Croll [CP09] describes the move of analytics from IT to marketing with three steps:

1. JavaScript eliminated the need for log files and enabled marketers to maintain and deploy their analytics solutions themselves, making them less dependent on IT.

2. The introduced advertising economy of search engines like Google led to a new focus of analysts on user attraction and conversion rates.

3. New cost models allowed marketers to pay for the analytics service based on website traffic, rather than paying upfront for hardware and software. Analytics spend was thus linked to website traffic and, ideally, revenue.

In 1990 the WWW started and in 1993 one of the first widely used browsers *Mosaic* was launched. At the same time *WebTrends* developed and released one of the first analytics software. A lot more services followed, such as *WebSideStory* in 1996 [CP09] or *Quantified* by Urchin in the same year [CP09].

Page tagging made it possible to collect not only technical data, but also business-relevant information. Visitors and their behaviour were the focus, shaping questions like: How is this user behaviour related to a purchase? If a user buys shoes, will he also buy socks? The development and implementation of cookies enabled the identification of unique users. Not only business-relevant questions were asked and answered, but also studies on performance and usability [CP09]. More details about page tagging can be found in section 2.2.4.2.

In 2003, Edwards, Eisenberg and Sterne founded the *Web Analytics Association* (WAA). The WAA brings together and supports all the players in web analytics such as users, marketers and IT specialists on an international stage. Due to digitalization and its all-encompassing effects, the WAA has renamed itself *Digital Analytics Association* (DAA) in 2012 because the web is not the only area where users leave their digital footprint [SKS14].

As described in the section above, participant and user numbers on the Internet continue to rise and now all Fortune 500 companies operate websites, with web analytics a key marketing tool [KO20].

Some of the most established tools today are *Google Analytics*, *Adobe SiteCatalyst*, *Webtrekk* and *Piwik* [Hei20]. Google Analytics will be further discussed in section 3.3.4.

Looking ahead, Zheng et al. identify several trends for web analytics, such as mobile web and application-specific analytics like video, search, learning, or social media analytics [ZP15].

2.2.3 Web Analytics in Practice

Web analytics can be described as a process in which the main goal is usually to increase sales. The literature cites two main ideas and processes, the first from the Web Analytics Association and the second from industry best practices. They are briefly described in this section.

Both processes have in common that they are aimed at improving the website and thus increasing business revenue.

Key Performance Indicators (KPIs) are the ideal tool for the instrumentation of web analytics and help to identify areas and potential for improvement. They are an integral part and play an important role in any web analytics process as they provide a "in-depth picture of visitor behavior on a site" [Jan09].

KPIs can differ depending on the business in which they operate. For commercial domains, common KPIs are conversion rates, average order or visit value, customer loyalty, bounce rate, etc. [SKS14].

Defining the right KPIs and aligning them with business goals is a critical step in any web analytics process.

2.2.3.1 WAA Process Guide

The Web Analytics Association offers a web analytics guide that consists of nine steps. They are [Jan09]:

1. Identify key stake holders
2. Define primary goals of website and prioritize them
3. Identify most important site visitors
4. Determine key performance indicators
5. Identify and implement the right solution
6. Use multiple technologies and methods
7. Make improvements iteratively
8. Hire and empower a full-time analyst
9. Establish a process of continuous improvement

2.2.3.2 Industries Best Practices

On the contrary, Waisberg and Kaushik [WK09] derive a five-step process from industry best practices with the main goal of improving the website and increasing sales:

- Define Goals
- Build KPIs
- Collect Data
- Analyse Data
- Implement Changes
- Repeat last two steps

When comparing the two proposed processes, it becomes clear that both focus on identifying and being aware of the most important business goals. The WAA process is a finer-grained and more practical approach, with Waisberg and Kaushik abstracting the main activities.

2.2.4 Data Collection: Log File Analysis and Page Tagging

There are four main methods of collecting data for web analytics: through log file analysis, JavaScript page tagging, web beacons, and packet sniffing [WK09].

As mentioned in section 2.2.2, the two main methods of data collection are log file analysis and page tagging. In this section I will briefly describe and compare both mechanisms.

2.2.4.1 Log File Analysis

As described earlier, log file analysis is about gaining insights from the log file records of the web server. Log file analysis is considered the traditional and original approach to web analytics [Mar15], [ZP15]. Once the user types a URL into the browser and presses enter, the request is forwarded to a web server. The server then creates an entry in a log file and sends the requested page or resource back to the client as a response [WK09]. The information within the log entry can vary depending on the log format. Typically, the IP, browser, timestamp, time taken, bytes transferred, whether there is a cache hit, and the referrer are specified [WK09]. The standardized Common Log format includes host, ident, authuser, date, request, status, and bytes, as shown in listing 2.1.[10]

Listing 2.1: CLF

```
127.0.0.1 user-identifier frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326
```

Standard formats of log files and entries allow log file analysis software such as *Analog*, *Webalizer* or *AWStats* to process, analyze and report valuable statistics to users [ZP15].

Below are some points that describe the advantages and disadvantages of log file analysis. The advantages of log file analysis are ([WK09], [NC11], [SKS14], [ZP15]):

- JavaScript and cookies are not required on the client side
- Maintainer of the website and server owns the data
- Bots and web crawler requests are also logged
- History of data is available
- Log entries are reliable
- The standard format of log files enables easy switching of analysis tools
- The web server also logs failed requests
- No modifications to the website required
- Does not demand more bandwidth

Some disadvantages of log file analysis are ([Mar15], [ZP15]):

- The log entries mainly contain technical information that is not very useful for analyzing user behavior. Business-related metrics such as bounce rates are not available.

[10] https://www.w3.org/Daemon/User/Config/Logging.html#common-logfile-format [03.06.2021]

- Only direct requests from the client to the web server are logged: Any user interaction in the web browser that does not trigger a request is not logged. Responses from caches and proxies are also not visible in the web server log file. Only interactions with the web server are logged.

2.2.4.2 Page Tagging

In a nutshell, page tagging describes the analytics method where JavaScript is embedded into the website that collects data and sends it to an analytics server [Mar15]. For this purpose, JavaScript must be integrated on each page to be analysed [WK09]. Page tagging is the most important method of web analytics today [ZP15].

Croll provides a illustration (figure 2.1) of the page tagging process, which is explained below [CP09].

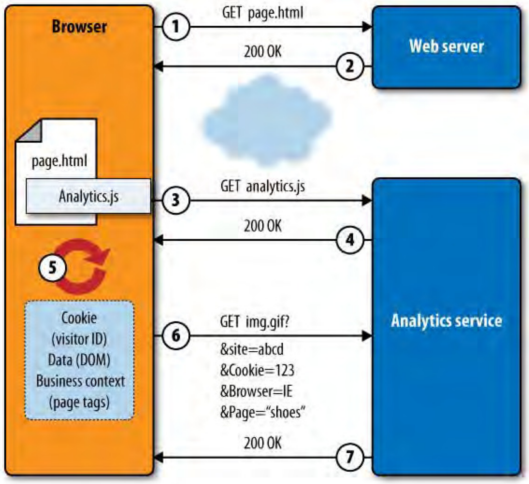


Figure 2.1: Page Tagging, taken from [CP09]

The client (browser) requests a page from the web server (1, 2). Within the HTML file, an external JavaScript resource, the analysis code, is linked and received by the analysis server (3, 4). The analysis script tracks and measures user behaviour and finally sends the data back to the analysis server (5, 6, 7).

The collected data may also be stored in cookies that contain data beyond one session and allow the identification of the user, for example, the next time the user visits the site [KO20].

The advantages and disadvantages of page tagging are as follows, starting with the advantages ([WK09], [NC11], [Mar15], [SKS14], [ZP15]):

- Every page visit is counted
- The analytics service is outsourced, which includes the storage of the data, but also the data analysis and reporting

- Page tagging is rather easy to implement and favourable when the analyst does not have access to the web server
- Highly customizable: Everything that JavaScript enables to measure, collect, and track is available. This also includes information about the client such as screen size, device used or color depth.
- Ability to track events and actions such as mouse clicks that do not send requests to the web server. This is especially important for single-page or progressive web applications that do not generate requests as often.
- Mechanics of cookies provide identification of unique and repeat visitors
- Real time reporting is possible

Some disadvantages are mainly privacy concerns and user permission to collect data, that the analytics process relies on the use of JavaScript and cookies that can be disabled by the user [Mar15], that any page that needs to collect data must include the analytics script, and that it is quite difficult to switch tools due to the use of a third-party analytics service [SKS14].

Page tagging is the underlying technique of Real-User Monitoring, which is discussed in 3.3.3.

2.2.5 Web Analytics Metrics

In this thesis, two kinds of metrics occur: Web Analyitcs metrics and Web Performance metrics. Web analytics metrics are metrics that reflect and quantify all business-related aspects of the website. I use the term "business" or "non-performance" metrics to draw a line between metrics that are primarily concerned with website performance and metrics that reflect other web analytics aspects. I will use the terms "business," "non-performance," and "web analytics" metrics interchangeably. Business metrics are not directly responsible for capturing performance data such as website load speed. Performance metrics, on the other hand, are primarily concerned with the performance of a website, such as website speed, as discussed in 3.4.

The selection of any type of metrics and especially their relevance to the business development of the website is crucial. It is important to measure what is important and relevant for the users and the achievement of the business goals, while "each business has its own definitions of success". Therefore, metrics are ideally tailored to the website and should track if "your business benefited in some way from their [the users] visits." Just as each website is unique and serves a different purpose, so should be the metrics that quantify it. Metrics can be developed for any specific business question or use case, and there are an unlimited number of metrics.

Metrics are only helpful if they are useful. This means that the collection and measurement of metrics should be consistent and their presentation should be user-friendly and

understandable. Kumar explains that ideally, good metrics are not too complex, relevant, timely, and immediately useful.

In this section, I describe metrics that correlate more with and map to business issues, such as conversion rate or bounce rate. A core set of web analytics metrics can be found in the literature, and there are several ideas for structuring and categorizing these metrics. The different ways of categorizing them are discussed below. Then, one approach to categorization is used to list a selection of common web analytics metrics.

2.2.5.1 Web Analytics Metrics Categorizations

Categories help to better grasp and understand key figures, and they can provide structure and order. Below are some examples of categorizing business metrics as found in the literature.

Peterson argues from a marketing perspective and categorizes metrics according to the customer lifecycle, which includes the reach, acquisition, conversion, and retention phases.

Metrics for measuring *reach* are, for example, the number or percentage of new visitors. The category *Acquisition* contains metrics such as average number of visits per visitor or average pages viewed per visitor. The *Conversion* category contains metrics such as conversion or abandonment rates. And the last category *Retention* includes metrics like the number of returning visitors. This proposed categorization is very customer-centric and customer-focused, from the marketer's point of view.

The Web Analytics Association defines three types and, accordingly, categories of metrics: *Counts*, *Ratios*, and *Key Performance Indicators*. Counts are metrics that represent single numbers such as the number of visits, while ratios consist of counts divided by other counts, such as page views per visit. KPIs are counts or ratios with a specific meaning and importance to the business in question.

Jansen defines four categories for web analytics metrics: *Site Usage*, which includes metrics such as demographics and system statistics, as well as visitor length and type,, *Referrers*, which are metrics that describe the referring URL, *Site Content Analysis*, which includes metrics such as top pages or visitor paths, and *Quality Assurance* which includes metrics that reflect errors or other quality aspects of the system being measured.

Croll argues that all metrics answer at least one of four user-centered questions: *What did they do?*, *How did they do it?*, *Why did they do it?*, and *Could they do it?*. Accordingly, the metrics are categorized by the question they answer.

Bekavac breaks down metrics by what they describe: *Visits*, such as entry or exit page, *Visitors*, such as metrics that capture new or returning visitors, metrics such as page exit rate or bounce rate that describe *Visitor Engagement*, and metrics that describe *Conversions*, such as conversion rate..

Hassler proposes a classification that is very similar to the categorization proposed

by WAA. The metrics are ordered according to their type: *Counts* are absolute values like visitors or total revenue, *Relations* put counts in relation, e.g. as percentages, such as page views per visitor. and *Values* include non-numeric metrics such as referrers or search terms.

Gessert et al. propose a semantic categorization of metrics. They structure metrics according to their meaning and domain membership. The categories are *performance*, which includes performance metrics (as described in section 3.4), *User Engagement* metrics such as session duration or bounce rate, *Business KPIs* such as cart size or transaction amount, as well as conversion rates and revenue, and *QA metadata* that summarize technical metrics such as JS errors or browser distribution.

Value-Driven	{Counts, Ratios, KPIs}, {Counts, Relations, Values}
Semantic-Driven	{Performance, User Engagement, Business KPIs, QA Metadata}, {Site Usage, Referres, Site Content Analysis, Quality Assurance}
Marketing-Driven	{Reach, Acquisition, Conversion, Retention}, {Visits, Visitors, Visitor Engagement, Conversions}

Table 2.2: Possible categorizations of web analytics metrics

As described above, several categorizations of metrics have been proposed in the literature. The different ways of categorizing them are summarized in table 2.2. While some authors group metrics by the nature of their value, e.g., whether they count something or represent a ratio, others use a more semantic approach and group metrics by their meaning, with some extreme examples of authors grouping only from a marketing perspective.

In the following, I will list and briefly explain some metrics, following the categorization of counts, ratios, and values. After that, the focus will be narrowed down to web performance and the corresponding metrics to capture it.

2.2.5.2 Web Analytics Metrics Examples

Table 2.3 contains web analytics metrics that are not directly related to performance. The categories used to structure the metrics are *Counts*, *Ratios* and *Non-Numeric Values* as already defined by Hassler or WAA as described in the section above.

Counts	
Hits	Represents the number of requests to the server.
Visits or Sessions	Counts how many users visit the website.
Session Length	The duration of the session.
Page Views	How many times a page has been viewed.
Single Page Visits	Sessions in which only one page was viewed.
JS Errors	How many JS errors occurred.
Cart Size	How many items are in the shopping cart.

Conversions	How many visitors perform the desired action. In e-commerce, this could be the purchase of a product, for example.
etc.	
Ratios	
Conversion Rate	Percentage of users that result in a conversion (perform a desired action).
Bounce Rate	Percentage of all sessions in which only a single page was viewed and no interaction took place.
Click Through Rate (CTR)	Percentage of visitors who have clicked on a specific link. Used in advertising campaigns, for example.
etc.	
Non-Numerical Values	
Demographics	Information about visitors to the website, such as age, gender, geographic location, language, etc.
Referrer	The website from which the user came to the current website.
etc.	

Table 2.3: Web Analytics or Business Metrics Examples, taken from

The selection of common business metrics in table 2.3 is not exhaustive and provides only a glimpse of web analytics metrics.

The main focus of this thesis is web performance. What web performance is, what factors play a role in website performance, and what measurement methods and metrics exist will be covered in the next chapter.

3 Web Performance

In the last chapter, we saw how important performance is in e-commerce and how web analytics help measure many aspects of e-commerce websites. This provides the context for the following chapter.

This chapter is about web performance. Web performance is also about analysing the website, but not about business-related issues such as the type of users, how they interact with the website, or whether they lead to conversions. Web performance is about the website itself, specifically how fast it is and how users perceive performance.

The introduction section 3.1 gives an overview of some general ideas and aspects of web performance. Some of them will be discussed in more detail. To understand web performance, one must know how websites are loaded into the browser and presented to the user. Section 3.2 explains this process, focusing on the steps within the network and the front end. What techniques exist for measuring and evaluating the performance of a website are discussed in section 3.3. Finally, section 3.4 discusses metrics that reflect and capture the performance of a website.

Before describing the approach and practical work done to address the research question in chapter 5, the next chapter 4 discusses some other research in this area.

3.1 Introduction

As described in section **??**, web performance plays a non-negligible role in user satisfaction and business success. The studies cited in the relevant section show that increasing website performance also increases sales, or as Google puts it, "Performance plays a major role in the success of any online venture."[1]

A good overview of web performance topics and goals is provided by the MDN Web Docs[2], which serve as an outline for this chapter and are briefly described below.

Reducing load time To understand how load times occur, the technical aspect of how websites are loaded and presented to the user by the browser must be understood. This subject is addressed in section 3.2.

Concrete optimization steps and techniques to reduce loading times are not the subject of this work.

[1] `https://web.dev/why-speed-matters/` [03.06.2021]

[2] `https://developer.mozilla.org/en-US/docs/Learn/Performance/What_is_web_performance` [03.06.2021]

Usability and interactivity There are several metrics that attempt to reflect areas of performance such as load time, smoothness, and interactivity, and there are specific metrics to distinguish between technical and user-perceived performance. Web performance metrics are discussed in section 3.4.

Performance perception Perceptions of performance are generally subjective. As seen earlier in section **??**, there are some quantifiable time intervals that correlate with human psychology regarding received performance. Table 3.1 provides "unofficial rules of thumb" for delay thresholds [Gri13].

Delay	User Perception
0-100 ms	Instant
100-300 ms	Small perceptible delay
300-1000 ms	Machine is working
> 1 s	Likely mental context switch
> 10 s	Task is abandoned

Table 3.1: Rule of thumbs for delay

Interpreting the numbers from the table, one can make the statement that it is desirable to keep load times below one second. Thresholds for certain performance metrics and the psychological rationale for setting them are briefly discussed in section 3.4.3.3.

Performance measurements Measuring the performance of a website is an important and non-trivial task. Two main methods for measuring the performance of a website exist: *Synthetic Monitoring* is discussed in section 3.3.1, *Real User Monitoring* (RUM) is covered in section 3.3.3.

3.2 The Website Loading Process

To understand web performance metrics and the methods used to measure them, it is critical to have a basic understanding of the technical aspect of loading a web page into the browser. In general, this process involves establishing a connection between a client and a server, and the browser's task of converting the data received from the server into a readable, ready-to-use website.

It is helpful to divide the components of the website loading process into three units: The front-end, the back-end, and the network. The front-end takes care of everything the user sees on the screen, such as rendering the UI, and consists of a client (browser) that requests data from the back-end. The back-end, usually a web server, processes the client's requests and sends the response back to the front-end. The data is transmitted over the network, which connects the front and back ends and consists of network components such as cables, routers, switches, network protocols, and so on.

Figure 3.1 shows the three entities and their tasks during the website loading process. What happens in the network phase is described in section 3.2.1. The front-end's task of receiving data and transforming it into a website is covered in section 3.2.2. How the back-end processes requests and creates responses is not part of this work.

Figure 3.1: Timing Overview

3.2.1 The Website Loading Process in the Network

The network connects the front-end to the back-end. When a client requests resources from a server, data must be transferred over the network.

Latency and bandwidth are two important factors in network performance. How they affect performance is discussed in section 3.2.1.1. The procedure and other mechanisms for establishing a reliable and secure connection between the front-end and the back-end are described in section 3.2.1.2. Web performance metrics measure, among other things, how long certain steps take during connection establishment, as described in section 3.4.2.

3.2.1.1 Latency and Bandwidth

There are two important factors when discussing network performance: latency and bandwidth. As will be examined below, latency is the bottleneck of network performance, not bandwidth.

Bandwidth is the "maximum throughput of a logical or physical communication path" [Gri13]. In other words, bandwidth describes the amount of data that can be sent in parallel from one node in a network to another. Physical communication paths are usually cables such as metal wires or fiber optic cables, with fiber optic cables having less signal loss and lower lifetime maintenance costs. Using methods such as wavelength division multiplexing (WDM), it is possible to transmit up to 70 Tbit/s over a fiber optic link. This high technology is only used in the backbone infrastructure, e.g. for the connection between Europe and America. For the end user, the bandwidth is much lower, and the average was only 5.1 Mbit/s at the end of 2015. High bandwidth is useful for bulk or large-scale data transfers like streaming video or audio. But for loading a website or any other browser activity that depends on many requests fetching data from many different locations around the globe, the performance bottleneck is latency [Gri13].

Latency is "the time from the source sending a packet to the destination receiving it" [Gri13]. Latency is measured in seconds and can be the time it takes to travel a single distance or, more commonly, how long it takes the transmitted data packet to round-trip time (RTT) from source to destination and back. In other words, latency "describes the amount of delay on a network or Internet connection" [MDNc]. For the very first request when establishing a connection, the latency is longer due to protocols such as DNS lookups, TCP and TLS handshakes [MDNc]. These are discussed in section 3.2.1.2.

To get an idea of how the two aspects of bandwidth and latency affect web performance, Mike Belshe conducted a study [Bel]: There are two configurations. One has fixed latency and variable bandwidth, and the other is configured the other way around. He then compared the performance of the two configurations using the Page Load Time metric (figure 3.2).

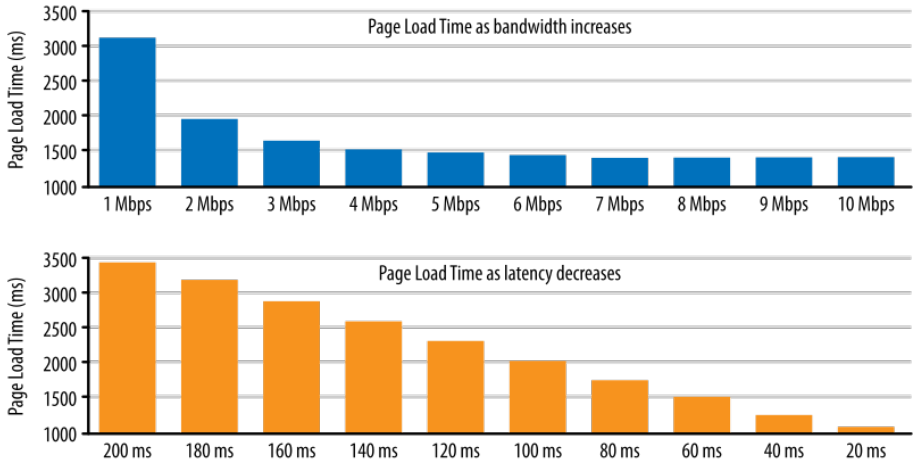


Figure 3.2: Latency vs Bandwidth, taken from [Gri13]

We see that the influence of the bandwidth is trivial: if the available bandwidth is doubled, e.g. from 5 to 10 Mbit/s, there is no change in the loading time. For latency, on the other hand, the picture is different: If the latency can be reduced by half, e.g. from 120 ms to 60 ms, the page load time also drops by half. Or as Belshe puts it, "[reducing] cross-atlantic RTTs from 150ms to 100ms [...] would have a larger effect on the speed of the internet than increasing a user's bandwidth from 3.9Mbps to 10Mbps or even 1Gbps" [Bel].

These observations can be explained by the many short, small connections and requests made while browsing websites and the contrary basic structure of the communication protocols, which are "optimized for long-lived connections and bulk data transfers" [Gri13]. But simply reducing latency is not easy: the speed of data transmission is already 2/3 the speed of light, but the physical conditions are the limiting factor, e.g. there is a minimum distance between London and New York that cannot be "optimized" further [Gri13].

Another aspect of latency is that latency is even higher for wireless connections and

thus for mobile devices, "making network optimization a critical priority for the mobile web" [Gri13]. Latency is high for mobile users due to mobile network infrastructure (see also "Why are mobile latencies so high?" in [Gri13]).

Although latency is an important factor, what happens on the front end is still important. At the same time, this thesis focuses on metrics to measure performance on the front end. Before going into what happens in the browser once the website data arrives, I will briefly describe the preceding steps of establishing a connection between the browser (client) and the server, which can still be considered part of the network.

3.2.1.2 Network Navigation Steps

When the user enters a URL into the browser, a series of steps are taken to render and present the website to the user. These steps consist of DNS lookup, TCP and TLS negotiation, and the HTTP request, among others. These steps are necessary to establish a secure and reliable connection between the browser (client) and the server, and are described below.

DNS Lookup If the requested resource cannot be loaded from the browser's cache, the first step in establishing a connection is a DNS (Domain Name System) lookup (or DNS resolution).

This step is about translating the URL into an IP address. It must be performed for each unknown URL, for example, if linked images on a website come from different servers, a DNS lookup must be performed for each individual URL. The URL to IP mapping can be cached by the browser, which makes repeated calls faster [MDNb].

DNS lookups take between 20 and 120 milliseconds on average [Key].

TCP Handshake The goal of TCP (Transmission Control Protocol) is to establish a reliable connection in an unreliable network. TCP "guarantees that all bytes sent will be identical with bytes received and that they will arrive in the same order to the client" [Gri13]. The TCP 3-way handshake is a technique for establishing a reliable connection. In terms of performance, the handshake adds two more roundtrips, which, as we have seen, is bad for performance because of latency.

There are many algorithms and techniques for optimal data transmission and avoiding congestion, e.g. Slow-Start. Slow-Start is an algorithm that determines the maximum usable bandwidth by gradually increasing the amount of data sent. Slow-Start prevents the full capacity of the network from being used from the start, which in turn leads to more round trips and latency [Gri13]. For a detailed discussion of TCP, see "Building Blocks of TCP" in [Gri13].

TLS Negotiation TLS (Transport Layer Security) is another protocol whose goal is to establish a secure connection in the sense of data encryption. Data that is transmitted

over the network must be encrypted so that outsiders cannot read or manipulate the data. For encryption, a cipher must be defined that is exchanged between client and server during TLS negotiation [MDNb].

From a performance perspective, TLS in turn means more roundtrips, which has a negative impact on performance. For a detailed discussion of TLS, see "Transport Layer Security (TLS)" in [Gri13].

HTTP Request and Response Now that a secure connection has been established, the client fetches the first resources via HTTP GET request. In most cases, the server responds by returning the index.html file, which can then be used by the browser to build the website [MDNb].

Normally, many more resources are requested from the browser to complete the construction of the website. Currently, the average is about 70 requests per website [HTT].

A request is not the same as a connection: Once the connection is established using the methods described above, such as DNS lookup, TCP, and TLS handshakes, multiple requests can be transmitted over the same connection. Typically, the number of requests is much higher than the number of connections to load a website because the browser maintains connections to keep them open for multiple requests. The average number of connections for a website today is about 13 [HTT]. Modern browsers like Chrome allow up to six connections open in parallel [Hog14].

At this point, the browser has received the first data about the website and it can start rendering the page. Exactly how this happens is explained in the next section.

3.2.2 The Website Loading Process in the Frontend: Critical Rendering Path

This section explains what happens after the first bytes of the website arrive in the browser. The following processes are usually subsumed under the term *Critical Rendering Path* (CRP). The CRP is the final part of the navigation process, as shown in figure 3.1.

The CRP is the minimum number of steps the browser must take from receiving the first HTML byte to rendering pixels on the screen for the first time. Rendering is "critical" because it is the very first rendering, the first visible content the user sees on the screen. The resources needed to render the page for the first time are considered critical. Without the critical resources, the browser cannot display the content on the screen. An example of a critical resource is the first HTML file that the browser receives, because without it, nothing can be seen on the screen. Non-critical resources, on the other hand, do not prevent the browser from displaying the first content on the screen [Joh].

There are a number of steps that the browser goes through to render the page. The basic idea is to convert HTML, CSS, and JS into actual pixels on the screen.

Figure 3.3 illustrates the flow of the CRP: Once the HTML code is received, the browser starts parsing the HTML code and translates it into the DOM. The content of the CSS files

is parsed into the CSSOM. JavaScript must be retrieved and executed. Once the DOM and CSSOM are available, the render tree is created. When the render tree is available, the layout is created. Finally, the pixels can be output to the screen [MDNa].

The individual steps are described in more detail below.

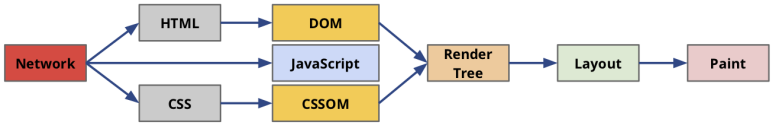


Figure 3.3: Critical Rendering Path

DOM Construction from HTML As soon as the browser receives the first bytes of the HTML file, it begins to decompose it into the *Document Object Model* (DOM). Building the DOM is the first step the browser takes when it receives data. The DOM is a tree structure and an internal representation of the HTML for the browser [MDNb].

The general parsing process consists of translating bytes into characters, into tokens, into nodes, and finally into the object model [Gria]. The specification of the DOM is maintained by the WHATWG living standard (for standards, see section 3.4.2.1).

The DOM tree contains information about the content of the document, but not about its style. The styling is defined in the CSS (see next section). Once HTML and CSS have been transferred and processed by the browser, the *Render Tree* can be created to reflect the actual information and its styling that the browser can display. In this context, it is possible to categorize resources into render-blocking and non-render-blocking: A render-blocking resource is a resource that prevents the browser from rendering the content on the screen. HTML and CSS are render-blocking resources because the parsing process of these files prevents the browser from rendering the page on the screen [Grib].

As soon as the first data packets of HTML arrive in the browser, the parsing process begins [MDNb]. The DOM is created incrementally, which means that the browser can begin processing the HTML before all of the content is transmitted over the network.

Usually, external resources are linked in the HTML code that are necessary for the completeness of the website, such as CSS or JavaScript. When parsing the HTML step by step, a reference to such an external resource will appear at some point. How the external resources CSS and JavaScript are handled by the browser is explained below.

CSSOM Construction from CSS The CSS resource contains all the information about the styling of the page. As with HTML, CSS is converted from bytes to characters, tokens, nodes, and finally to the *CSS Object Model* (CSSOM) [Gria]. CSSOM construction is usually very fast and is standardized by the W3C [MDNb].

Unlike the HTML parsing process, CSS cannot be translated into CSSOM incrementally. The reason for this is the cascading nature of stylesheets, which can cause styling

rules defined at the beginning of the file to be overwritten by rules defined at the very end of the CSS file. Partial CSSOM is therefore not possible. Thus, the browser needs the entire CSS file before it can create the CSSOM [MDNa].

As soon as the parser encounters a reference to an external stylesheet such as in listing 3.1,

Listing 3.1: Link to a CSS file from the main document

```
<link rel="stylesheet" href="styles.css">
```

it requests the resource and proceeds to parse the HTML. CSS is not a resource that blocks the parser. When the CSS arrives in the browser, CSSOM construction begins.

CSSOM creation does not block the parser, but it does block rendering. The browser blocks rendering of the page until it has received and parsed all the CSS. Rendering content to the screen is possible only if CSSOM and thus CSS are available.

Once the DOM and CSSOM are created, they can be merged to form the render tree, which then handles the layout and rendering on the screen. Before describing this process, I'll discuss how JavaScript is handled.

JavaScript in the Critical Rendering Path JavaScript (JS) resources add functionality and interactivity to a website. When the browser encounters a script tag like in listing 3.2,

Listing 3.2: Link to a JS file from the main document

```
<script src="myScript.js"></script>
```

it pauses its current task of parsing, immediately fetches the resource and executes its contents, and only then continues to create the DOM (see figure 3.4).

This means that JS blocks the parser: JS can manipulate and query the DOM tree and modify the HTML file directly. Since the HTML file is the input stream for the parser, the parser stops until the JS is downloaded and executed. Only when the execution of the script is complete, HTML parsing continues. Since JS execution blocks DOM creation and HTML processing itself is rendering blocking, JS is implicitly rendering blocking as well. The behaviour is identical for an external JS file and a script inserted directly into the HTML code.

Since JS can also manipulate the styling of the page, its execution is blocked until the CSSOM is available. This means that JS execution is on hold until the CSSOM is ready.

In summary, while JS blocks the parsing of the HTML into the DOM, JS execution itself is blocked by the creation of the CSSOM. CSSOM blocks JS, and JS blocks the building of the DOM.

Several attributes for the script tag can change the behavior of the browser. *Async* and *defer* are options that counteract the blocking nature of the script tag. They are discussed now.

With the async attribute, the browser downloads the JS in the background while it continues parsing the HTML (see figure 3.4). Parsing is not blocked and the browser can continue with its task. Once the JS is downloaded and available, the parser is blocked: The browser stops parsing and executes the JS.

The order of all asynchronous scripts within the document is no longer maintained. As soon as a script is downloaded and available, it is executed. It does not matter whether an asynchronous script is inserted at the beginning or the end of the HTML document.

As with async, scripts with the defer attribute allow the browser to download the script in parallel while it continues parsing the HTML (see figure 3.4). Unlike async, defer scripts are not executed until the page parsing is complete and the DOM tree is fully built, and the order of the scripts is maintained.

Figure 3.4: Scripts

The async and defer attributes do not apply to inline scripts. The HTML standard states that "scripts may specify defer or async, but must not specify either unless the src attribute is present." Since inline scripts do not contain a src attribute because the source of the script is inside the script tags and not external, the async and defer attributes are not applicable.

HTML, CSS and JS are now processed and the CRP proceeds to build the render tree as described below.

Building the Render Tree As described above, both HTML and CSS are rendering blockers because they prevent the page from rendering. Rendering can only occur once the render tree is available. The render tree is the combination of DOM and CSSOM and captures all visible content with their styles to be displayed on the screen. If an element has a CSS property like `display: none;`, it is not displayed in the render tree.

The calculated render tree is then used for the layout of the content on the page.

Layout In the layout process, the position and size of the nodes are calculated from the render tree. New layout calculations or re-flows are triggered as soon as the screen area changes, e.g. by rotating the device or resizing the window, or when the DOM and the render tree change.

Once the layout is resolved, the browser continues painting pixels on the screen.

Paint Finally, the browser can paint the content on the screen. When content changes, browsers are optimized to redraw only the affected areas on the screen.

3.2.3 The Website Loading Process: Conclusion

The loading process of a website is complex and multi-layered. The goal of this section was to provide a general understanding and idea of it. For this purpose, the loading process is divided into three components: The front-end, the back-end, and the network. We have seen that within the network component there are several mechanisms to ensure a reliable and secure connection between the front-end and the back-end. Within the front-end component, the Critical Rendering Path describes the steps that the browser must take to retrieve and render a website.

It is important to have a general understanding of the loading process of a website in order to understand web performance metrics. Before discussing web performance metrics in section 3.4, methods for measuring them are described in the next section.

3.3 Measurement Methods

In this section, I will discuss what methods and techniques are available to measure the performance of a website. The two most important methods are synthetic monitoring, as described in section 3.3.1, and Real-User Monitoring, as discussed in section 3.3.3.

A concrete example is given for each of the methods. WebPageTest as a synthetic monitoring solution in section 3.3.2, and Google Analytics as a Real-User Monitoring tool in section 3.3.4.

Both are used in the controlled experiment that addresses the research question, as explained in chapter 5. After this section on measurement methods, I describe which performance metrics are measurable.

3.3.1 Synthetic Monitoring

3.3.1.1 The "Synthetic" Aspect in Synthetic Monitoring

As the name implies, synthetic monitoring is a measurement method performed in an artificial, laboratory-like, synthetic environment. Test agents simulate real users and are configured to run a browser, load the observed website, and collect performance data in the process. Synthetic monitoring does not take into account real user traffic.

Performance data can be collected via common performance APIs as described in section X. In addition, user-centric metrics such as Speed Index can be calculated through video recording and analysis (see Section X).

Synthetic monitoring controls many possible configurations and variables of the test agent (client), such as location (geography), network conditions, device type, browser version, and so on. So the tester has control over many variables that affect performance, and can also change these variables to test how they affect performance.

The controlled environment enables the collection of performance data for a specific configuration, such as the location of the test agent or browser version, which can help identify issues with specific user segments. For example, a test could check the performance of all users using Firefox on macOS in Germany.

In addition to defining the technical configuration, such as browser version or network conditions, the tester can also define artificial user journeys to simulate real user behaviour.

A feature of the controlled environment provided by synthetic monitoring is that the measured data and test results are relatively consistent with little variability, providing a performance baseline for the site being monitored and facilitating performance optimization.

Synthetic Monitoring is not about Real Users Synthetic monitoring does not collect data from real users, as traffic on the website is generated artificially. Real user behaviour is approximated by simulating users, e.g., by prescribing user paths. Measured performance data does not necessarily reflect the actual user experience, and the tester should not assume that "synthetic results are like real-user metrics".

Capturing the wide variety and diversity of real users, such as the pages visited, the user's general configuration such as CPU, GPU, and memory performance, what data is stored in the browser cache and what browser version is used, screen size, operating system, and network connection, to name a few, is difficult to represent in synthetic monitoring. The selected test configuration in synthetic monitoring only reflects a specific use case and can only approximate what a user with a similar configuration might experience. In short, synthetic monitoring test results "are synthetic and therefore not representative for actual user data".

3.3.1.2 The "Monitoring" Aspect in Synthetic Monitoring

Synthetic monitoring can be automated and used to monitor a system's performance in real time and provide up-to-date reports for the system administrator. Monitoring makes it possible to check site availability around the globe, detect performance problems before users notice them, and is generally useful for continuous "health checks" of the running system.

Since each website can be tested synthetically, it is possible to compare the performance data of several competitors.

There are many synthetic monitoring tools (see, for example, Kaur 2016 for a list of tools). *WebPageTest* is one of them and is discussed in section 3.3.2.

As described in Section X, the performance of an online store correlates with sales. Synthetic monitoring enables the collection of performance metrics independent of real user behaviour. Real user behaviour is not measured in synthetic monitoring. Since only real users are able to generate revenue, synthetic monitoring cannot detect correlations between user satisfaction (and revenue) and performance (as described in Section X).

Understanding the correlation between user satisfaction and website performance requires Real-User Monitoring (RUM). RUM enables the collection of data from each real user. RUM will be covered in section 3.3.3.

Before that, a common synthetic monitoring tool, *WebPageTest*, is briefly discussed.

3.3.2 WebPageTest

As will be discussed in Section X, I will use a synthetic monitoring tool to measure and compare performance metrics in the context of a controlled experiment, which will help answer the research question. There are many synthetic monitoring tools available. The tool used is WebPageTest (WPT) and will therefore be briefly discussed here.

"WebPageTest is a web performance tool providing deep diagnostic information about how a page performs under a variety of conditions." WPT was made available to the public in 2008 and was acquired by Catchpoint in September 2020[3]. WPT is open source[4] and offered as a free web service[5] and can be used by anyone.

Some of the key features of WPT are that it measures and provides a variety of metrics (see Section X), it facilitates saving and sharing performance test results, it provides waterfall graphs, a breakdown of the website's content, or even film strips and video recordings of the loading process.

The main goal of WPT is to "provide detailed information to developers about the loading performance of their pages in a realistic end user environment."

WPT is widely used, "the leading web performance tool in the world today", and an "indispensable power tool in your web performance toolkit".

3.3.2.1 Configuration

WPT is highly configurable and allows performance testing for almost any desired configuration. One of the key configuration options is that testing can be performed from multiple geographic locations, with test sites provided by over fifty companies and individuals.

Other configurations include, for example, the possible selection of different devices and browsers, e.g., Chrome emulation from an iPad in Taiwan; the setting of Repeat View, which instructs the browser to load the website not only the first time but also

[3] `https://www.catchpoint.com/webpagetest-joins-catchpoint` [04.08.2021]
[4] `https://github.com/WPO-Foundation/webpagetest` [04.08.2021]
[5] `https://www.webpagetest.org/` [04.08.2021]

the second time, which provides insight into the caching setup of the website; and the throttling of the network, e.g., whether to use DSL or 5G.

The configuration options give the investigator control over many variables that can affect performance. The settings used for the experiment to answer the research question are explained in Section X.

3.3.2.2 Private Instances

The public WPT instance is reliable and configurable, but depending on the test location, it can also lead to waiting times because other people are also using it. Also, the public WPT instance can only access and thus test and evaluate publicly accessible websites. There are also daily limits on how much testing can be done.

Private WPT instances help to counteract the problems described. Since WPT is open source, it is possible to create and use your own version of WPT. Private instances allow unlimited testing and give the tester full control over the infrastructure. An additional and only available feature for private instances are bulk tests. Bulk tests allow a list of URLs to be passed to WPT for processing. In Section X, I describe how this feature can be used to test the same URL multiple times and how private instances are set up and used.

WPT captures a variety of performance metrics. What performance metrics are is described in Section X, and what metrics are available through WPT in Section X. The next section discusses another important method for measuring web performance, Real-User monitoring.

3.3.3 Real-User Monitoring

As the name suggests, Real-User Monitoring (RUM) is about collecting and measuring data from real users who visit the website. Unlike synthetic monitoring, which artificially generates website traffic and only approximates the performance experience of real users, RUM data relies on real user traffic and captures data directly from each user's browser. RUM measures performance as experienced by users.

3.3.3.1 The Page Tagging Technique in RUM

As described in Section X., page tagging is a technique that instruments the user's browser to collect data and report it to an analytics server. RUM is based on page tagging, i.e., a JS code snippet ("tracking code") that is loaded into the user's browser. Once this tracking code is loaded and executed in the browser, it collects data and sends it back to an analytics service. If the user blocks JS or the script cannot be downloaded for other reasons, RUM will not work. Once the data arrives at an analytics service, it must be stored and an interface must be provided for the analyst to query the data and gain insights, such as by providing a dashboard.

Since RUM relies on the JS code, the very first opportunity to measure data is when this JS code has been downloaded and executed. Anything that happens before that step is invisible to the tracking script, and therefore invisible to the analyst. Meenan notes that about 20% of the loading process is outside the RUM measurement range and that "getting a reliable start time for measurements from the real world has been the biggest barrier to using RUM".

Another facet of RUM is that the measurement of data should ideally have as little impact as possible on the site rendering process, and network capacity should not be taken up by RUM scripts and block resources of the CRP. One of the research questions of this paper is whether RUM, as a page marking technique, slows down the observed website. The evaluation of the controlled experiment to answer this question shows that RUM...., as described in section X.

RUM is independent of the user's setup or environment and collects data for all active users: Regardless of the user's device, browser, network conditions, or geographic location, RUM collects data as long as the measurement script is loaded into the user's browser. So the RUM data represents each individual user experience.

Due to the diversity of users and the unique environment of each user, RUM data tends to be more diverse and heterogeneous than data collected through synthetic monitoring.

3.3.3.2 RUM Measures Behaviour of Real Users

As discussed in section X, website performance and user satisfaction are directly related. An important component of RUM is not only the collection of performance metrics, but also the measurement of user behaviour, e.g., how the user interacts with the website and where he or she clicks. In an e-commerce context, user behaviour questions are of interest, such as whether a new campaign changes user behaviour as expected or where users leave the check-out process.

RUM enables the combination of collected metrics with user behaviour and business KPIs and can answer questions such as whether and how website performance affects user behaviour, e.g. whether users buy more or less depending on the speed of the website. Thus RUM is not only important to understand user behaviour, but also to optimize the website and increase sales. With techniques such as cookies (see section X.), it is also possible to track user behaviour not only for one page visit, but over a series of website visits, resulting in even more detailed insights about the visitor.

There are several RUM tools and JS libraries, such as Boomerang from Akamai.[6] The main player, Google Analytics, is discussed in more detail in section 3.3.4.

Google collects RUM and browser data from Chrome users who have consented to it. The data is accessible via the Chrome User Experience Report (CrUX) and is discussed below.

[6] `https://github.com/akamai/boomerang` [23.06.2021]

3.3.3.3 Chrome User Experience Report (CrUX)

The Chrome User Experience Report (CrUX) is a RUM method implemented by Google that collects real user data from Chrome users. Once the Chrome user gives consent, data collection can begin and does not need to be set up. The CrUX only collects data from Chrome users and is therefore not a complete sample of the Internet user population, as it does not collect data from users who browse the Internet using Firefox or Safari, for example. The collected data is available via Google's PageSpeed Insights, the CrUX Dashboard or the BigQuery and CrUX API.

3.3.4 Google Analytics

As will be explained in Section X, I will use a RUM tool to measure and compare performance metrics in the context of a controlled experiment, which will help answer the research question. There are many RUM tools available. The tool used is Google Analytics (GA) and will therefore be briefly discussed here.

GA is a RUM solution that helps to "determine how successful your site is at turning users into customers".[7]

GA is by far the most widely used analysis tool today.[8] The first version of GA was initially developed by Urchin Software Corporation and acquired by Google in 2005. The second version, Classic GA, was released in 2007. The third version, Universal Analytics, was released in 2013 and the latest version, Google Analytics 4, was released in 2020. Each version improved the quality of measured data and integration with other Google products [FV].

GA offers a wide range of functions.[9] Worth mentioning, for example, is analytical intelligence, a system that answers specific questions asked by the tool's users by extracting answers from the data; a variety of reports, such as real-time reports or conversion reports; or general data analysis, visualization and monitoring, such as with custom segmentation.

GA also tracks a number of web performance metrics. What these are and how they are calculated is described in Section X.

GA is a RUM method and therefore relies on the page tagging technique, as explained in Section X. Some examples and aspects of how to integrate the GA tracking script into the monitored website are explained in Section X.

3.3.5 Measurement Methods Conclusion

There are two main complementary techniques to measure the performance of a website: synthetic monitoring and Real-User Monitoring (RUM) (table 3.2).

[7] `https://marketingplatform.google.com/about/resources/analytics-product-overview/` [05.08.2021]
[8] `https://w3techs.com/technologies/overview/traffic_analysis` [05.08.2021]
[9] `https://marketingplatform.google.com/about/analytics/features/` [05.08.2021]

Synthetic monitoring measures the performance of a website in a controlled environment using test agents and is particularly useful for finding a baseline for website performance and for continuous monitoring and health checks. Performance as perceived by end users can only be approximated, as it is not captured by synthetic monitoring.

RUM collects data from each user who visits the website and reports it back to an analytics server. RUM is particularly useful when multiple metrics are combined, such as performance and business metrics, to analyse user behaviour. On the other hand, if no user visits the website, RUM does not collect any data.

Other performance measurement methods such as CrUX and surveys provide browser-specific or quality data and complement the analysts' toolkit.

Synthetic Monitoring	RUM
Artificial test agents	Real users
Controlled configuration	No control over the configuration
Approximates real user data	Collects real user data
For monitoring	Links user behaviour to business KPIs

Table 3.2: Comparison of Synthetic Monitoring and RUM

Measurement methods such as synthetic monitoring or RUM can measure metrics to map website performance or business success to some kind of value or number. As discussed in section X, there are a variety of web analytics metrics to measure any business-related aspect of the website, such as conversion or bounce rates.

Website performance can also be measured using the methods described above. There are a variety of web performance metrics. What they are, how they have evolved, and what they monitor will be covered in the next section.

3.4 Web Performance Metrics

[tbd]

[Link to Research Question]

As discussed in Section X., web analytics (or business or non-performance) metrics collect data on many different aspects of an online business, such as how many visitors the website has or what the conversion rate is. A number of specific metrics are required to measure *performance*, such as the speed of the website and how users perceive the website's performance. Performance metrics are created to measure various aspects of a website's performance and are discussed in this section.

Performance metrics make performance quantifiable because the metrics reflect performance in numbers. Once performance is mapped into numbers, they can be compared with each other, e.g., over time, after a change in the measured application, or with a competitor's numbers. By quantifying performance, metrics facilitate the analysis of website

traffic. Specific metric values can be used as targets for goal achievement and are therefore an important tool for improving websites. Ideally, metrics are precise and serve as objective criteria for evaluating the measured website.

As with web analytics metrics, there are several ways to structure and categorize performance metrics. One way to bring order to performance metrics is to distinguish between *Navigation Timing Metrics* (section 3.4.2), which measure elapsed time for specific processes and navigation steps from the CRP, and *User-Centric Performance Metrics* (section 3.4.3), which approximate user-perceived performance through visual analysis. Both classes of metrics are discussed below.

Before that, I'll discuss *Page Weight* (section 3.4.1), a metric that characterizes the size of a website. The benefits of *Custom Metrics* (section 3.4.4) and why they are important will conclude the section on performance metrics.

3.4.1 Page Weight

A naive approach to quantifying a website is to measure its size and the resources required to build it. Page weight can be used as a performance measure because the performance of a website depends on its size, especially when combined with network conditions such as download speed and latency. The more bytes that need to be transferred over the network, the longer it takes for the website to build, and the longer the wait for the user. The situation is similar for requests: the more requests are required, the more connections have to be established, which in turn takes more time.

A website consists of several resources, such as HTML files, fonts, script files or images (see table 3.3). The page weight describes the number of bytes and requests retrieved for each resource type.

HTML	CSS	JS	Fonts
Images	Videos	Others	(Total)

Table 3.3: Types of resources of a website

Page weight is an initial assessment of a website's performance. It is mainly useful for comparison with median values of other websites, but has little to no significance and meaning for evaluating the performance of a website. To gain more meaningful insights into a website's performance, other metrics are needed, such as navigation timing metrics, which are discussed in the next section. As will be described in section X, the idea of page weight is used to create a test page with an average page weight value.

3.4.2 Navigation Timing Metrics

Navigation timing metrics show the elapsed time within the website loading process, such as the network navigation process and CRP (see section 3.2). They are computed and derived from a set of standardized Web APIs.

In the following, I will briefly explain the nature of Web standards and recommendations (section 3.4.2.1). I will then discuss performance-related specifications and the metrics they provide (section 3.4.2.2).

3.4.2.1 Web Standards

Web standards are documents that describe the technology used to build the WWW. These documents, also called specifications, are maintained by various groups and organizations. There are several organizations that create and maintain standards for different areas and technologies of the Web. Such organizations include the WHATWG, which maintains the living HTML standard[10], or Ecma International, which maintains the specification for JS.[11].

The World Wide Web Consortium (W3C) is an organization that maintains various standards for the WWW, including specifications for performance measurement[12], which I will discuss below.

W3C The W3C was founded in 1994 by Tim Berners-Lee with the goal of bringing together "representatives from many different technology companies to work together on the creation of web technology specifications". The W3C manages and publishes documents that describe and define web technologies such as DOM, SVG or CSS.

The creation of a document or specification follows a "recommendation track" that describes the hurdles a document must overcome to move from an idea to a final recommendation and web standard that browser vendors implement as concrete APIs. The stages of the recommendation process are described by the "maturity levels" of the document and are as follows: First Public Working Draft (FPWD) $\rightarrow$ Working Draft (WD) $\rightarrow$ Candidate Recommendations (CR) $\rightarrow$ Proposed Recommendation (PR) $\rightarrow$ W3C Recommendation (REC). W3C recommendations are considered web standards.

Web Performance Working Group Within the W3C there are several groups, each dealing with a specific topic in the WWW universe. Founded in 2010, the Web Performance Working Group is a group whose mission is to "provide methods to measure aspects of application performance of user agent features and APIs."[13].

The Web Performance Working Group publishes and maintains a number of documents that address performance measurement, such as High Resolution Time, Navigation Timing, and Page Visibility. The published papers that are particularly relevant to performance measurement are discussed below.

[10] `https://whatwg.org/` [07/11/2021]
[11] `https://www.ecma-international.org/` [11.07.2021]
[12] `https://www.w3.org/` [11.07.2021]
[13] `https://www.w3.org/webperf/` [11.07.2021]

3.4.2.2 W3C Performance Specifications

High Resolution Time There are two versions of High Resolution Time. High Resolution Time Level 2 has the maturity level of a W3C Recommendation and was released in November 2019. The newer specification, simply High Resolution Time, is a current working draft. The High Resolution Time specification does not directly define performance metrics, but serves as the basis for other specifications for measuring and recording time values.

Prior to High Resolution Time, time values were calculated using the ECMAScripts Date object, which represents time in milliseconds since January 1, 1970 UTC. This time definition is inaccurate with respect to time shifts and system clock adjustments. It is possible that the time values derived from the Date object do not increase monotonically or sometimes even decrease.

High Resolution Time solves these problems and provides monotonically increasing time values, sub-millisecond resolution, and a starting value that refers to the beginning of the navigation process on the website instead of the year 1970, making it easier to understand and calculate timestamps and intervals. The specifications described below use the precise time information provided by High Resolution Time.

Navigation Timing There are two versions of Navigation Timing. A W3C recommendation "Navigation Timing" (Level 1) from December 2012 and "Navigation Timing Level 2", a current Working Draft.

Level 1, which is still used for backward compatibility, uses the less accurate time measurement from January 1, 1970. Level 2 replaces Level 1 and is based on the accurate high-resolution time measurement as described above. A summary of the improvements from Level 1 to Level 2 is given in X.

Navigation Timing displays timestamps of events that occur during the navigation and loading process of a website. All relevant timing information from the document loading process is accessible through Navigation Timing. The navigation or loading process represents the conversion of the received HTML document and other resources into a functioning website and takes place in the browser, as described in section 3.2.

Navigation Timing captures only time values from the main HTML document. Within the main document, requests are made to other linked resources. Timing information for these linked resources is captured and presented via Resource Timing, as described in the next paragraph.

The exposed timing events are shown in figure 3.6 and described in table 3.4. The metrics calculated directly with Navigation Timing (and Resource Timing) are listed in table 3.5.

Navigation Timing timestamps are exposed via the PerformanceNavigationTiming interface and can be retrieved in the browser via $window.performance.getEntriesByType(\text{"}navigation\text{"})$, as seen in figure 3.5.

Figure 3.5: Navigation Timings, taken from Chrome's console log

Resource Timing There are two versions of Resource Timing. A Candidate Recommendation, Resource Timing Level 1, from March 2017, and Resource Timing Level 2, a current Working Draft.

Navigation Timing outputs timing information for the main document, while Resource Timing outputs timing information for all other resources that the main document requests, and also for resources that request other resources. Other resources can be CSS, JS, other HTML documents, images and so on.

The timestamps displayed by Resource Timing are mainly network-related time values (see table 3.4). For example, the time it takes to download a particular resource.

The Resource Timing values are provided by the PerformanceResourceTiming interface and can be retrieved in the browser via $window.performance.getEntriesByType("resource")$.

Navigation and Resource Timing go hand in hand. With Navigation and Resource Timing, all relevant timing information is available for all resources on the site. The waterfall diagram (see Section X.) is an illustrative example of a visualization of the attributes provided by both specifications. The attributes provided by Navigation and Resource Timing are discussed in more detail below.

Navigation and Resource Timing Attributes All time values or attributes relevant to the navigation process and captured by Navigation and Resource Timing Level 2 are

listed in Table 3.4 and Figure 3.6. The black coloured timestamps associated with the blue boxes in Figure 3.6 are captured by Navigation Timing. The yellow coloured timestamps are captured by Resource Timing. All attributes can be accessed in the browser via *performance.getEntriesByType("navigation")*. Figure 3.6 does not contain all defined and available values from the specification definitions, but only those visible within the navigation process. For documents with different origins, the attributes in parentheses may not be available.

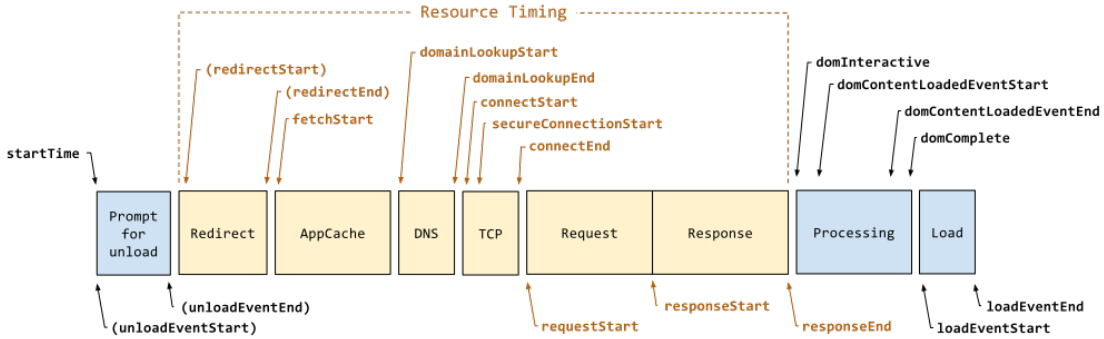


Figure 3.6: Timestamps from Navigation and Resource Timing

Table 3.4 provides a brief explanation for each timestamp. The definitions are taken directly from the W3C specifications or MDN Web Docs, unless otherwise noted. These timestamps are used to calculate performance metrics, as described in the next section.

Navigation Timings Level 2	
startTime (= navigationStart)	Start time of the navigation process. Is set to 0. Navigation Timing Level 1 exposes "navigationStart" which is set to the "time immediately after the user agent finishes prompting to unload the previous document."[14] Since some metric calculations are still based on navigationStart (see below), it is mentioned here.
unloadEventStart	Set to 0 if there is no previous document. Otherwise, the value is set to the time when the event to unload the previous document is triggered by the user agent.
unloadEventEnd	Set to 0 if there is no previous document. Otherwise, the value is set to the time when the event to unload the previous documents ends.
domInteractive	Time value equal to the time immediately before the user agent sets the current document readiness of the current document to "interactive".
domContentLoadedEventStart	Time value equal to the time immediately before the user agent fires the DOMContentLoaded event at the current document.

[14] `https://www.w3.org/TR/navigation-timing/)` [15.07.2021]

domContentLoadedEventEnd	Time value equal to the time immediately after the current document's DOMContentLoaded event completes.
domComplete	Time value equal to the time immediately before the user agent sets the current document readiness of the current document to "complete".
loadEventStart	Time value equal to the time immediately before the load event of the current document is fired. Returns 0 if the event was not triggered.
loadEventEnd	Time value equal to the time when the load event of the current document is completed. Returns 0 if event has not been fired or is not completed.
Resource Timings Level 2	
redirectStart	Start time of the fetch which initiates the redirect.
redirectEnd	Marks timestamp which occurs immediately after receiving the last byte of the response of the last redirect.
fetchStart	Marks when the browser starts to fetch a resource. Does not mark when the browser makes a network request for a resource, but rather when it begins checking caches to see if a network request is even necessary.
domainLookupStart	Returns the timestamp immediately before the browser starts the domain name lookup for the resource.
domainLookupEnd	Returns the timestamp immediately after the browser finishes the domain name lookup for the resource.
connectStart	Returns the timestamp immediately before the user agent starts establishing the connection to the server to retrieve the resource.
secureConnectionStart	Returns a timestamp immediately before the browser starts the handshake process to secure the current connection.
connectEnd	Returns the timestamp immediately after the browser finishes establishing the connection to the server to retrieve the resource.
requestStart	When the browser issues the network request.
responseStart	Returns a timestamp immediately after the browser receives the first byte of the response from the server, cache, or local resource.
responseEnd	Returns a timestamp immediately after the browser receives the last byte of the resource or immediately before the transport connection is closed, whichever comes first.

Table 3.4: Navigation and Resource Timing Level 2 Attributes

The document states and events mentioned above, such as DOMContentLoaded, and their triggering by the user agent are defined in the HTML standard.

Metrics calculated with Navigation and Resource Timing The attributes provided by Navigation and Resource Timing are used to compute common metrics for navigation timing. Table 3.5 lists a selection of metrics for navigation timing and shows how they are derived from the attributes provided by Navigation and Resource Timing. Figure 3.7 provides a graphical overview of the metrics and the intervals they reflect.

As with web analytics metrics, there is no single source of truth or set of definitions for navigation timing metrics. However, a review of the literature reveals a common set of navigation timing metrics. Literature sources include the MDN Glossary, Google developer guides, blog posts by other performance specialists, research papers, or the documentation of common RUM tools.

The sources mentioned above sometimes explain the idea of metrics in a rather informal way. How exactly the metrics are implemented and calculated, e.g. which navigation timing attributes are used, is not always clear and is the responsibility of the RUM tool providers and can only be checked directly in the source code. Due to the lack of standardization and definition of the metrics, it is therefore possible that different calculations are used for the same metric.

Time to First Byte (TTFB)	$[responseStart - navigationStart]$ Indicates when the user agent received the first byte of the websites data. "Time it takes between the start of the request and the start of the response".
Page Load Time (PLT)	$[loadEventStart - navigationStart]$ The MDN Docs define PLT as the "time it takes for a page to load" and it is calculated as stated above. Across multiple authors, PLT is defined as the time from navigation start until the load event[15] has been fired. The load event is fired when the main document and its associated resources have been loaded. Grigorik notes that PLT "has been the de facto metric of the web performance world", but is inaccurate today due to the shift from static websites to dynamic web applications that load resources differently (see also 2018 Netravali) and
DNS Lookup Time	$[domainLookupEnd - domainLookupStart]$ Time it takes for a DNS lookup.
TCP Handshake	$[connectEnd - connectStart]$ "The time it takes for the TCP handshake."
TLS Handshake Time	Time it takes to secure connection with TLS. MDN Docs defines TLS time with $[requestStart - secureConnectionStart]$

[15] `https://developer.mozilla.org/en-US/docs/Web/API/Window/load_event` [16.07.2021]

	Google defines TLS Time as $[connectEnd - secureConnectionStart]$
Request Time, Server Response Time	$[responseStart - requestStart]$ How long it takes for the request. Also called Server Response Time by Akamai
DOM Content Loaded,DOM Content Load	MDN defines DOM Content Loaded as $domContentLoadedEventEnd - domContentLoadedEventStart$, that is the DOMContentLoaded event duration. Akamai defines a "DOM Content Load" time which represents the time from navigation start until the DOMContentLoaded event has been fired, captured by domContentLoadedEventEnd.
Page Download Time, Receiving Time, Transfer Time	$[responseEnd - responseStart]$ How long it takes to download the page. Also called Transfer time by Akamai or Receiving Time by MDN
Latency	$[responseStart - fetchStart]$ Latency is defined as the elapsed time from when the browser requests the resource until the first byte from it is received.
DOM Interactive	$[domInteractive - navigationStart]$ When the "browser has completed parsing the entire HTML and constructed the DOM"

Table 3.5: Metrics

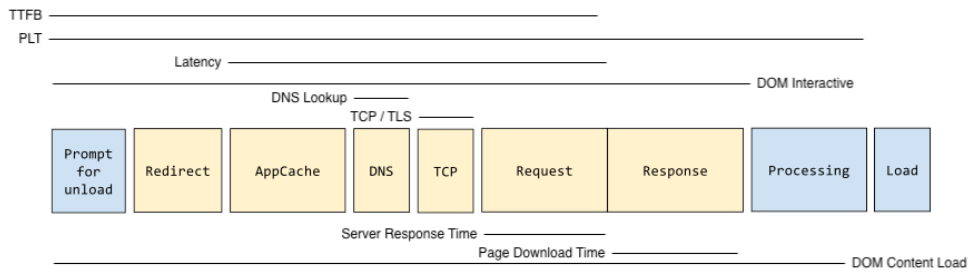


Figure 3.7: Navigation Timing Metrics

Apart from Navigation and Resource Timing, the Web Performance Working Group also maintains other performance-related specifications. Some of these are briefly discussed below.

User Timing User Timing Level 2 is a February 2019 Recommendation and User Timing Level 3 is the current Working Draft. User Timing provides methods for creating custom, specific, and unique high-resolution timestamps, querying them, and calculating intervals and elapsed time between these created timestamps. User Timing simplifies and facilitates the use of Custom Metrics (see section 3.4.4).

Performance Timeline Performance Timeline is the December 2013 Recommendation and Performance Timeline Level 2 is the current working draft.

In short, the Performance Timeline defines an API and methods that make accessible all timing values captured by other specifications such as Navigation or Resource Timing. The Performance Timeline defines an interface for capturing a variety of performance measurements. It can be used to retrieve Navigation Timings using *performance.getEntriesByType("navig*

In addition, a PerformanceObserver interface is defined that can be used to monitor the Performance Timeline and notify when new measurements and recordings are available.

Long Tasks The Long Tasks API 1 was released in September 2017 as a First Public Working Draft. The goal of the API is to detect "long tasks", i.e. tasks that take the main thread longer than 50 ms. Long tasks are critical to performance because they block other critical tasks such as user input.

The Long Tasks API is used to calculate user-centric metrics (see section 3.4.3.

Server Timing Server Timing is a Working Draft. With specifications such as Navigation and Resource Timing, the user agent has access to a variety of different timings related to the navigation process. Not visible or available are measurements and timings within server processing, such as database queries. Server Timing solves this problem by allowing a server to transmit and communicate performance-related information to the user agent.

Paint Timing Paint Timing 1 was published as a First Public Working Draft in September 2017. It specifies two "key moments" during the loading process: First Paint (FP) and First Contentful Paint (FCP). Table 3.6 provides explanations of the key moments as defined in the specification.

The metrics can be accessed directly in the browser via *performance.getEntriesByType("paint")*. Therefore, FP and FCP can be captured in RUM. Other visual metrics are mainly based on video analysis and require a synthetic monitoring facility as described in the next section.

FP and FCP indicate when the user sees something other than the default white screen for the first time. Whether the visual change is important or useful to the user is not considered by FP or FCP. Section 3.4.3 discusses visual metrics that may be of importance to the user.

First Paint (FP)	Marks the point when the browser renders anything that is visually different from what was on the screen prior to navigation
First Contentful Paint (FCP)	The point when the browser renders the first bit of content from the DOM, which may be text, an image, SVG, or even a canvas element

Table 3.6: Key Moments specified by Paint Timing

Web Incubator Community Group The Web Incubator Community Group (WICG) is a W3C Community Group and "provides a lightweight venue for proposing and discussing new web platform features." The WICG is working on and proposing a number of possible web technologies and standards.[16] These specifications are unofficial drafts that have not been reviewed by the W3C, but may be included in the W3C Recommendations track in the future. Four of the proposed "incubations" are mentioned here because they serve as the basis for some user-centric metrics, as described in Section 3.4.3.3. Details on the specifications can be found on the corresponding pages: *Element Timing API*[17], *Event Timing API*[18], *Layout Instability API*[19], and *Largest Contentful Paint*.[20].

3.4.2.3 Navigation Timing Metrics Conclusion

A variety of measurements and time intervals can be taken during the loading process of a website. The W3C and the Performance Working Group maintain and define a set of specifications with the task of measuring performance and navigation timing data. The attributes exposed by these specifications are used to calculate performance and navigation timing metrics.

Since the performance metrics are not officially defined by any authorized party or institution, their meaning may vary depending on the implementation.

Table 3.7 provides an overview of all specifications discussed, at what maturity level they are available, and why they are important.

Paint Timing is already moving in a direction that seeks to measure user-perceived performance not just by measuring elapsed time between navigation events, but rather by observing visual changes as the website loads. Although navigation times can help identify any bottlenecks or other technical problems during the loading process, they do not reflect how the user actually perceives the performance of the website loading process. Since users primarily consume websites with their eyes, visual indicators are relevant and critical.

In recent years, a variety of visual and user-centric performance metrics have emerged, some of which Google is promoting. They are discussed in the next section.

[16] `https://wicg.io/` [24.07.2021]
[17] `https://wicg.github.io/element-timing/` [24.07.2021]
[18] `https://wicg.github.io/event-timing/` [24.07.2021]
[19] `https://wicg.github.io/layout-instability/` [24.07.2021]
[20] `https://wicg.github.io/largest-contentful-paint/` [24.07.2021]

High Resolution Time	Level 2: REC November 2019 Current WD Provides accurate, stable and reliable time measurements.
Navigation Timing	Level 1: REC December 2021 Level 2: Current WD Exposes information about the navigation timing of the main document.
Resource Timing	Level 1: CR March 2017 Level 2: Current WD Exposes timing information of requested resources and the resources they requested.
Navigation and Resource Timing	Used to calculated a variety of metrics.
User Timing	Level 2: REC February 2019 Level 3: Current WD Mark and measure custom timestamps.
Performance Timeline	REC December 2013 Level 2: Current WD Subsumes collected performance data from other specifications.
Long Tasks	API 1: FPWD September 2017 Identifies tasks that occupy the main thread for more than 50ms. Basis for TTI and TBT.
Server Timing	Current WD Standard for transmitting performance-related data from the server to the client.
Paint Timing	FPWD September 2017 Exposes the "key moments" First Paint and First Contentful Paint.
Element- and Event Timing API Layout Instability API Largest Contentful Paint	WICG proposals (unoffical drafts), used for LCP, FID, CLS

Table 3.7: Summary of W3C and WICG Specifications

3.4.3 User-Centric Performance Metrics

The Navigation Timings described in Section 3.4.2 are measured directly via a set of Web APIs and are accessible via the browser. Metrics that deal with user-perceived performance focus mainly on visual changes to the website and therefore cannot be captured directly via Web APIs.

Perceived performance deals with how fast a website "feels" to a user. As described earlier in Section X., different users perceive performance differently depending on factors such as gender, region, or stress level. Perceived performance can differ from actual load times, and user-centric performance metrics are typically more difficult to measure than navigation times because visual analysis or video recordings are required to calculate the metrics.

In Section 3.4.3.1, I describe user-centric performance metrics that have become main-

stream among performance monitoring vendors. With Web Vitals, Google defines its own set of user-centric metrics and focuses on website performance. They are discussed in Section 3.4.3.2.

3.4.3.1 Visual Metrics

The term "Visual Metrics" is not defined by any specification or web technology association and is used here to subsume metrics that rely on the visual part of the website. Visual metrics are based on the video recording of the website loading process and use this recording for further analysis. Therefore, they are best suited for synthetic monitoring. Visual metrics focus mainly on how fast pixels are rendered into the viewport, the part of the screen visible to the user.

The three metrics described below appear frequently in synthetic monitoring tools and other web performance literature referenced in their respective sections.

Start Render, Time to First Paint (TTFP) The Start Render time is the time when the browser made the first pixel on the screen visible to the user. The Start Render time is calculated using video analytics instead of using browser APIs such as Paint Timing.

Start Render is defined and exposed by a number of synthetic monitoring solutions such as speedcurve or WebPageTest.

Netravali et al. use the term Time to First Paint (TTFP) for the same metric, while for Enghardt et al. TTFP is the same as FP measured within the browser and not by video analysis.

This example shows that the definitions of metrics are not clear and different authors use different definitions.

Above the Fold Time (AFT) The above-the-fold area is the area of the website that is visible to the user without scrolling. Above the Fold Time (AFT) is a non-standard metric that measures how long it takes the browser to render all the pixels in this above-the-fold area. AFT measurement must take into account both visually dynamic areas of the website, where frequent pixel changes are expected, and static areas, where pixel and color changes are not expected.

AFT is measurable by synthetic monitoring and requires video recording and analysis.

Speed Index (SI) The Speed Index (SI) is a metric developed by the founders of WebPageTest that aims to express user experience in a single number.

Like the AFT, the SI reflects the visual completeness of the website. While the AFT only measures the time at which the viewport is fully rendered, the SI also takes into account the visual loading progress of the website and reflects "the average time at which visible parts of the page are displayed". SI is not measured directly, but calculated and expressed in milliseconds.

Like AFT, SI is reliable for static websites and unreliable for websites with dynamic and changing visual content. The details of the SI calculation are covered in

3.4.3.2 Web Vitals

With the Web Vitals initiative, Google is participating in the discussion about web performance metrics. Their goal is to "provide unified guidance for quality signals that are essential to delivering a great user experience on the web." Unified guidance consists of Web Vitals, a set of performance metrics that focus on how users experience and perceive performance, as opposed to just navigation timings, as discussed in Section 3.4.2. Since performance is a factor in Google search rankings, Web Vitals automatically become important to website owners.

The set and definitions of Web Vitals metrics are not set in stone, but are constantly evolving. The definitions, calculations, recommended thresholds, or even the metrics themselves may change over time, always adapted to the rapidly evolving technologies of the WWW and advances in web performance. While Web Vitals is currently primarily about web performance, other aspects of usability, such as security, privacy, or accessibility, will be included in the future.

Web Vitals and their definitions are derived from a set of key questions (table 3.8). The key questions reflect what users ask (possibly unconsciously) while using a website. This user-centered framework serves as a guide for creating and defining Web Vitals metrics. Unlike navigation timing metrics, Web Vitals focus on users and their perceptions, and each question is a perspective or facet of the user experience. Web Vitals are "metrics relevant to users".

Is it happening?	Did the navigation start successfully? Has the server responded?
Is it useful?	Has enough content rendered that users can engage with it?
Is it usable?	Can users interact with the page, or is it busy?
Is it delightful?	Are the interactions smooth and natural, free of lag and jank?

Table 3.8: Key Questions for defining Web Vitals

In addition to the key questions, Google identifies different aspects or types of website performance (table 3.9). Like the key questions, these types provide a guide and framework for organizing, classifying, and integrating Web Vitals.

Web Vitals can be measured by synthetic monitoring or RUM. For example, RUM data is available through CrUX[21], synthetic data is available through Lighthouse[22]. A custom JS library is available and can be integrated into any website to capture Web Vitals.[23]

[21] `https://developers.google.com/web/tools/chrome-user-experience-report` [22.07.2021]
[22] `https://developers.google.com/web/tools/lighthouse` [22.07.2021]
[23] `https://github.com/GoogleChrome/web-vitals` [22.07.2021]

Perceived load speed	How quickly a page can load and render all of its visual elements to the screen.
Load responsiveness	How quickly a page can load and execute any required JavaScript code in order for components to respond quickly to user interaction.
Runtime responsiveness	After page load, how quickly can the page respond to user interaction.
Visual stability	Do elements on the page shift in ways that users don't expect and potentially interfere with their interactions?
Smoothness	Do transitions and animations render at a consistent frame rate and flow fluidly from one state to the next?

Table 3.9: Types of Web Vitals Metrics

In the following, I describe common Web Vitals metrics. Section 3.4.3.3 discusses the Core Web Vitals, the most important Web Vitals.

First Contentful Paint First Contentful Paint (FCP) deals with question *Is it happening?* and is classified as type *Perceived load speed*. This is the same FCP as used by Paint Timing (see Section 3.4.2.2), and "marks the first point in the page load timeline where the user can see anything on the screen".

Google extends the Paint Timing specification with respect to FCP by only considering pages that are in the foreground all the time, by measuring FCP even when the page has been loaded from cache, and by exposing FCP for all frames, including cross-origin iframes. Implementation details can be found in the web-vitals library.[24]

FCP is measured in seconds and can be retrieved in synthetic and real-user monitoring.

First Meaningful Paint First Meaningful Paint (FMP) is "the paint after which the biggest above-the-fold layout change has happened and web fonts have loaded".

Google states that FMP is no longer used due to "inconsistent results" and the inability to measure it in all web browsers. FMP's successor, Largest Contentful Paint, is discussed below. The replacement of FMP nicely illustrates the evolution of Web Vitals.

Time to Interactive and First CPU Idle Time to Interactive (TTI) is an indicator for *Load responsiveness* and gives insights intto the question *Is it usable?* TTI measures the "time from when the page starts loading to when its main sub-resources have loaded and it is capable of reliably responding to user input quickly." Previously, TTI was referred to as *Time to Consistently Interactive*.

Other metrics and definitions are needed to calculate TTI: FCP as a starting point and a "quiet window", i.e. a time interval in which no long tasks (as defined by the API for long tasks, see section 3.4.2.2) and no more than two in-flight GET requests occur. The TTI is then the "end time of the last long task before the quiet window". The time interval

[24] `https://github.com/GoogleChrome/web-vitals/blob/main/src/getFCP.ts` [22.07.2021]

for the quiet window is set to 5 seconds, and the starting point for the search for TTI is FCP. To summarize, "TTI measures how long it takes a page to become fully interactive."

Ideally, the time between FCP and TTI, i.e. when the content is visible but the website is not interactive, is short. The TTI can be measured by synthetic monitoring and also by RUM. The latter is not recommended because user interaction can influence the TTI, leading to unreliable results.

First CPU Idle is a metric closely related to TTI. Previously, First CPU Idle was called *First Interactive*. First CPU Idle indicates when the website is minimally interactive, as opposed to the TTI, which indicates when the website is fully interactive. Google deprecated First CPU Idle, as "the difference [to TTI] isn't significant enough to justify maintaining two similar metrics".

Total Blocking Time Total Blocking Time (TBT) captures the aspect of *Load responsiveness* and provides clues to the question *Is it happening?*. By definition, TBT measures "the total amount of time between First Contentful Paint (FCP) and Time to Interactive (TTI) where the main thread was blocked for long enough to prevent input responsiveness." The starting point of the measurement is the point of the FCP, since user input is unlikely to occur before the user interface is rendered. The end point of the measurement is the point of the TTI.

Blocking the main thread is characterized by long tasks as defined in the Long Task API (see Section 3.4.2.2). If the user interacts with the website while a long task is running, the browser cannot respond to the user until it finishes processing the long task. Therefore, the user must wait. The blocking time of a long task is the total time of the task minus the defined long task time of 50 ms. For example, a long task of 180ms has a blocking time of $180ms - 50ms = 130ms$. The sum of all these times, between FCP and TTI, is the TBT.

For some use cases, TBT can adjust and correct the TTI if the TTI does not realistically reflect the available interactivity: Since the TTI is fixed around a 5-second time window, the TTI is likely to be the same for (A) three long tasks of 51 ms spread over 10 seconds, and (B) one 10-second task. (A) is likely to be more interactive because the main thread can respond to user input between the three long tasks. In (B), on the other hand, the main thread is blocked and user interactivity is prohibited. For such use cases, TBT is a good indicator: For (A), TBT is 3ms and for (B) it is 9950ms. The lower the TBT value, the better the interactivity and user experience.

As with TTI, Google recommends measuring TBT by synthetic monitoring and not by RUM, since user input can influence and interfere with the measurement results.

Among the Web Vitals, there are three metrics that are most important. The *Core Web Vitals* will be discussed next.

3.4.3.3 Core Web Vitals

Within Web Vitals, there is a group of metrics that are most important: "Core Web Vitals are the subset of Web Vitals that apply to all web pages, should be measured by all site owners, and will be surfaced across all Google tools." The Core Web Vitals are structured around the performance aspects *Loading*, *Interactivity*, and *Visual Stability*. Each of three Core Web Vitals reflects one of those aspects (table 3.10).

The Core Web Vitals are relevant for all types of websites and can be measured with RUM. In contrast to the general development of the Web Vitals, the Core Web Vitals are relatively stable in terms of definition and recommended benchmarks.

Core Web Vitals are captured by common Google performance evaluation tools such as CrUX or PageSpeed Insights,[25] and are also made available via the web-vitals library.[26]

Largest Contentful Paint (LCP)	Measures loading performance.
First Input Delay (FID)	Measures interactivity.
Cumulative Layout Shift (CLS)	Measures visual stability.

Table 3.10: Core Web Vitals

The three Core Web Vitals LCP, FID and CLS are discussed in more detail below.

Largest Contentful Paint (LCP) Largest Contentful Paint (LCP) looks at *Perceived load speed* and answers the question *Is it useful?*. While other metrics such as FCP or SI indicate when the first color is rendered on the screen or how fast the rendering process is, LCP is a metric used to report when the main content is visible to the user, where the main content is defined as the largest element. In short, LCP is "the render time of the largest image or text block visible within the viewport, relative to when the page first started loading."

To find the largest element, for example, images, videos or block elements with text are considered. The size of an element is defined as only the area that is visible to the user. LCP candidates can change during the loading process and are reported until the user interacts with the page.

LCP is defined and disclosed by the WICG specification *Largest Contentful Paint*. As with other metrics that rely on W3C specifications and APIs such as FCP, Google makes minor adjustments and, for example, only considers pages that were loaded in the foreground. Implementation details are available in the web-vitals library.[27] The advantage over other visual metrics such as SI is that LCP is retrievable not only in synthetic monitoring but also in RUM.

Although the Core Web Vitals are intentionally kept fairly stable in their definitions, Google's performance measurement initiative is still an evolutionary process that is not

[25] `https://developers.google.com/speed/pagespeed/insights/` [23.07.2021]
[26] `https://github.com/GoogleChrome/web-vitals` [23.07.2021]
[27] `https://github.com/GoogleChrome/web-vitals/blob/main/src/getLCP.ts` [22.07.2021]

yet complete: For example, a change in Google Chrome's LCP measurement changed LCP scores by 20%. Changes and improvements to the definition and implementation of Web Vitals are not uncommon, as can be seen from the Chromium Web Vitals changelog.[28]

First Input Delay (FID) First Input Delay (FID) has to do with *Load responsiveness* and answers the question *Is it usable?*. FID provides information about website responsiveness and interactivity and measures "the time from when a user first interacts with a page (i.e. when they click a link, tap on a button, or use a custom, JavaScript-powered control) to the time when the browser is actually able to begin processing event handlers in response to that interaction." Continuous events such as scrolling or zooming are not counted as initial inputs.

FID is about the time it takes for the browser to process user input. This time depends on the tasks of the main thread: If the main thread is busy while the user is interacting with the page, the browser must wait until the main thread is idle and can respond. In technical terms, "FID measures the delta between when an input event is received and when the main thread is next idle". By this definition, FID is measured even if no event listener is registered to process the user input, since only the main thread and its idle state are included in the equation.

FID only considers *first input* because first input, and thus first impression, is critical and likely to be triggered during website loading, where interactivity and performance issues are more common than when the website is already loaded.

FID can only be measured in the field because it requires real user interaction. Since FID depends on when a user interacts with the page, i.e., some users do not interact with the page at all, some interact when the main thread is busy, and others when the main thread is idle, it is important to analyse the distribution of reported FID values. In synthetic monitoring, an alternative and proxy metric of FID is TBT, as it "correlates well with FID".

FID is measurable with the WICG specification *Event Timing API*. As with other metrics that rely on W3C specifications and APIs such as FCP, Google makes minor adjustments and, for example, only considers pages that were loaded in the foreground. Implementation details are available in the web-vitals library.[29]

Cumulative Layout Shift (CLS) Cumulative Layout Shift (CLS) is about *Visual stability* and answers the question *Is it delightful?*. CLS measures "unexpected movement of page content". *Layout shift scores* are calculated for this purpose. Layout shift scores reflect and evaluate the unexpected movement of visible elements between frames (*layout shifts*). Layout shift scores are calculated by other measures such as *impact fraction* and *distance fraction*, which evaluate the movement of visible elements. The rather complex

[28] `https://chromium.googlesource.com/chromium/src/+/refs/heads/main/docs/speed/metrics_changelog/README.md` [24.07.2021]

[29] `https://github.com/GoogleChrome/web-vitals/blob/main/src/getFID.ts` [22.07.2021]

calculations are described in Finally, layout shifts over the lifetime of a page are summed (accumulated) and reported as CLS. CLS is a score and not a value representing milliseconds.

CLS is based on the WICG specification *Layout Instability API*. As with other metrics that rely on W3C specifications and APIs such as FCP, Google makes minor adjustments to the specification, such as only considering pages that were loaded in the foreground. Details about the CLS implementation are available in the web-vitals library.[30] CLS is retrievable by synthetic or real-user monitoring.

As observed with other metrics and Web Vitals, changes to the definitions of metrics or their implementation are not uncommon. For CLS, the implementation changed in June 2021. This change shows once again that web performance metrics are rather fragile and still under development.

Core Web Vitals Thresholds For each of the Core Web Vitals, Google provides thresholds and recommended goals described as *Good*, *Needs Improvement*, and *Poor*. The rationale for these thresholds is briefly explained in this section.

Google considers values determined by researchers in the fields of Human-Computer Interaction and Human Perception for its threshold definitions. Referenced studies are for example:

When research is not available, such as with CLS, a relatively new metric, Google relies on its own studies and assessments from the field. The proposed thresholds are realistic enough to be met and achieved by normal websites. The thresholds are therefore based on available data (e.g., from CrUX). The best possible rating of "good," defined as "high-quality user experience", is defined as being exceeded by 10% of all websites. With new research, technology, and other improvements, the thresholds may be adjusted in the future.

Figure X shows the threshold values for the three Core Web Vitals LCP, FID and CLS. While LCP and FID are specified in milliseconds, CLS is a calculated value. The exact rationale for the thresholds set for the Core Web Vitals can be found in

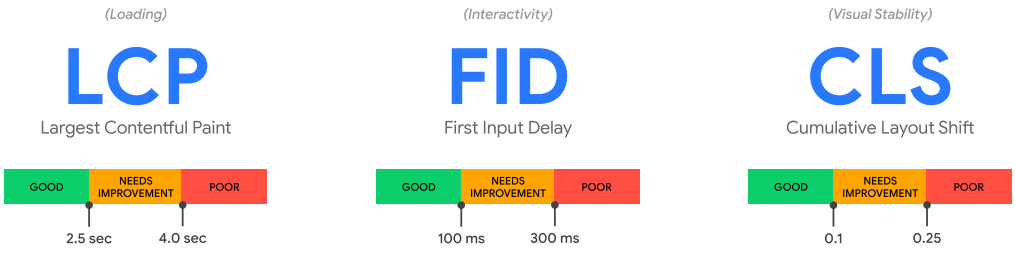


Table 3.11: Core Web Vitals Thresholds

[30] https://github.com/GoogleChrome/web-vitals/blob/main/src/getCLS.ts [22.07.2021]

The classification of a website into one of the three classes is based on the 75th percentile value of all page views: For a site to be classified as "good," at least 75% of page views must exceed this threshold. For example, if 25% of the measured page views fall into the "poor" segment, the other two segments cannot reach the 75%, so the page would be classified as "poor". The reason for the 75th percentile is that it guarantees that a large portion of users are taken into account and that the value at the selected percentile is not too influenced by outliers. The 75th percentile is a compromise between these two arguments: 3 out of 4 users experience the measured values, and 25 out of 100 samples can be outliers without affecting the result.

User-Centric Performance Metrics Conclusion User-centric performance metrics aim to capture how users perceive, experience, and feel about performance. This is in contrast to navigation timing metrics that rely only on in-browser timings that may have no meaning to the user.

Visual metrics such as Start Render or Speed Index are one approach to approximate and calculate how users perceive website performance by examining the pixels the browser renders into the viewport. They rely on video recording and analysis and work best for synthetic monitoring.

With Web Vitals, Google has introduced a set of user-centric performance metrics. They are built and modelled around a set of guiding questions and performance types, as seen in table 3.12.

Core Web Vitals are the three most important Web Vitals. They reflect the loading performance (LCP), interactivity (FID), and visual stability (CLS) of the website. Core Web Vitals can be measured using RUM. Google provides a rating system for Core Web Vitals. The thresholds for the ratings are derived from HIC and Human Perception research or derived directly from field data.

	Is it happening?	Is it useful?	Is it usable?	Is it delightful?
Perceived load speed	FCP	LCP		
Load responsiveness			TTI, TBT, FID	
Runtime responsiveness				
Visual stability				CLS
Smoothness				

Table 3.12: Web Vitals and their position in the mental model. Core Web Vitals are underlined.

As can be seen in table 3.12, user-centric performance metrics do not yet exist for all questions and types, such as runtime responsiveness or smoothness. Google indicates that new metrics are being introduced for some missing areas, but ideally custom metrics should be used in practice that capture the most important data for the business. Custom metrics are discussed in the next section.

3.4.4 Custom Metrics

The performance metrics described so far are relevant to gaining insight into page weight, such as how many requests it takes to build the website, to gaining an understanding of navigation timings, such as how long certain processes take in the CRP, and to gaining an understanding of user-perceived performance by calculating the Speed Index or measuring the time it takes to render the largest element. These metrics are good for getting a general overview of the website's performance or comparing it to the competition, but they are not able to reflect the very unique and individual performance metrics tailored to the business. Custom metrics are needed for this.

Almost all authors encourage and recommend the use of custom metrics: Meenan notes that "nothing beats application-specific knowledge and measurements", and suggest, for example, a performance metric such as "Time to First Tweet" for Twitter. And Grigorik notes that "custom and application-specific metrics are the key to establishing a sound performance strategy".

Custom JS code is required to measure custom metrics. As described in Section 3.4.2.2, several Web APIs are available to retrieve useful browser timings, i.e. paint events. User Timing, in particular, is of particular interest for custom metrics, as it allows for the creation and retrieval of custom timestamps. One disadvantage of custom metrics is that they are not comparable to any kind of benchmarks or competitors, since most likely no one else is measuring the exact same metric.

3.4.5 WebPageTest and Google Analytics Metrics

[tbd]

WebPageTest and GA are very common tools to measure web performance. One is synthetic monitoring the other one is RUM. They are already described in section X and X.

As they are both used in the experiment, the performance metrics they measure and expose are discussed here.

3.4.5.1 WebPageTest Metrics

[tbd]

covers a wide range of performance metrics: - It has waterfall charts with the associated request and response headers - It has timing metrics including time to first byte, document complete, and fully loaded. - breaks down the number of requests and bytes by content type - CPU utilization, bandwidth, and main thread timelines - focus on filmstrip views and side-by-side videos - development of the Speed Index metric

Name	Description

Successful Tests	Amount of tests who completed successfully
Document Complete	...
Fully Loaded	...
First Byte	...
Start Render	...
Bytes In (Doc)	...
Requests (Doc)	...
Load Event Start	...
Speed Index	...
Last Visual Change	...
Visually Complete	...

Table 3.13: Your caption here

3.4.5.2 GA Site Speed Metrics

[tbd]

Name	Description
Page Load Sample	The number of pageviews that were sampled to calculate the average page-load time.
Speed Metrics Sample	The sample set (or count) of pageviews used to calculate the averages of site speed metrics. This metric is used in all site speed average calculations, including avgDomainLookupTime, avgPageDownloadTime, avgRedirectionTime, avgServerConnectionTime, and avgServerResponseTime.
DOM Latency Metrics Sample	Sample set (or count) of pageviews used to calculate the averages for site speed DOM metrics. This metric is used to calculate ga:avgDomContentLoadedTime and ga:avgDomInteractiveTime.
Page Load Time (sec)	The average amount of time (in seconds) it takes that page to load, from initiation of the pageview (e.g., click on a page link) to load completion in the browser.
Domain Lookup Time (sec)	The average amount of time spent in DNS lookup for the page.
Page Download Time (sec)	The time to download your page.
Redirection Time (sec)	The time spent in redirection before fetching the page. If there are no redirects, the value for this metric is expected to be 0.
Server Connection Time (sec)	The time needed for the user to connect to your server.
Server Response Time (sec)	The time for your server to respond to a user request, including the network time from the user's location to your server.
Document Interactive Time (sec)	The average time (in seconds) that the browser takes to parse the document (DOMInteractive), including the network time from the user's location to your server. At this time, the user can interact with the Document Object Model even though it is not fully loaded.
Document Content Loaded Time (sec)	The average time (in seconds) that the browser takes to parse the document and execute deferred and parser-inserted scripts (DOMContentLoaded), including the network time from the user's location to your server. Parsing of the document is finished, the Document Object Model is ready, but referenced style sheets, images, and subframes may not be finished loading. This event is often the starting point for javascript framework execution, e.g., JQuery's onready() callback, etc.

3.4.5.3 Comparison GA and WPT Metrics

[tbd]

Navigation Timing API	WPT	GA
loadEventStart - navigationStart	Document Complete, Load Event Start	pageLoadTime
domainLookupEnd - domainLookupStart	DNS lookup, dns_ms	domainLookupTime
connectEnd - connectStart	connect_ms	serverConnectionTime
responseStart - requestStart	..	serverResponseTime
responseEnd - responseStart	..	pageDownloadTime
fetchStart - navigationStart	..	redirectionTime
domInteractive - navigationStart	..	domInteractiveTime
domContentLoadedEventStart - navigationStart	domContentLoadedEventStart	domContentLoadedTime

3.4.6 Web Performance Metrics Conclusion

[tbd]
[summary table]
[Link to Research Question]
[tbd]

4 Related Work

- Last chapter...
- This chapter: Describe shortly all sections from this chapter
- In the next chapter...
- This chapter should list research which covers and explores questions relevant for this thesis, such as:
  - Metrics: New metrics, meaning of metrics, difficulties of defining metrics, etc.
  - Overview, evaluation and comparison of measurement tools and methods
  - If available: Impact of RUM on performance

4.1 Research

- Research exists about topics like: ....
- Here i will provide a list of in my eyes relevant papers, summaries them and discuss why this is important for my research

4.1.1 some title for first category

2014 Singal I. - Describes history of web analytics and tools - Provides definitions and taxonomy for metrics - Describes log file vs page tagging - Describes KPIs

II. - Lit. overview for KPIs and Web Metrics - Lit. overview for "Trust" - Lit. overview for "Fuzzy" -> What are does categories?

III. - Some other literature worth mentioning

IV. - Describes 8 open challenges for researchers

2015 Bekavac - Two parts: - 1: Some general overview of web analytics, tools and metrics, KPIs etc - 2: Empirical study about employees satisfaction of used web analytics tools

1: - 9 web business models and 5 common goals - Hypothesis: Web analytics tools track and improve a user's satisfaction with web-based business models. - Web anayltics defintion. Log files vs Site Tagging - Web Analytics process - Tools: 5 categories, Process

of selecting tool, Table with features of different tools - Web metrics categories, Table with business models and their KPIs

2: - Which tools are used for which purpose / Activity - Users satisfaction

4.1.2 Research about Tools

Kaushik 2007 - Provides 3 questions which help to choose web analytics tools

2011 Nakatani - Gives some arguments why web analytics is important for business - Provides different categorizations for web analytics tools - Gives pros and cons of log file analysis and page tagging - Provides tool selection method based on AHP (Analytic Hierarchy Process)

"Web analytics tools collect click-stream data, track users navigation paths, process and present the data as meaningful information. - Categorizations: 1: By 4 different data collection methods 2: SaaS vs in-house 3: mobile vs non-mobile 4: Time lag

2016 Kaur - free vs paid - real time vs long term - hosted vs in-house - data portability

- free / open source tools - proprietary tools - Service Hosted Software - GA most popular one

4.1.3 Research about Metrics

Categorizations as seen in section X.

- Dont know:

5 Approach

- Last chapter...
- This chapter: Describe shortly all sections from this chapter
- In the next chapter...
- In this chapter the practical work should be documented and explained
- Elaboration of how the practical work could help answer the research question
- Discussion of real-life setup and how experiments approach it

5.1 Empirical Research Methods

- Overview of methods
- reproduceability etc.
- validity
- Justification why following approaches are conducted as controlled experiments
- Change something: Delete this item again

5.1.1 Controlled Experiment

- Short overview about controlled experiments in computer science
- Design: Show test setup image: Independent and dependent variables
- Hypothesis testing

5.1.2 Test Setup

- What is test object (website)
- What are dependent variables: Performance metrics
- What are independent variables: Specific changes in test object (see next chapter)
- Kohavi 2016: Sample size, collect right metrics, track right users, randomization unit

Variable	Values
Position	top-head, bottom-head, bottom-body
Attribute	no attribute, async, defer
Other Script	false, true

Measure effects: Dependent Variables

- Performance metrics from Lab and Field, see terms and definitions
- But also quality of RUM data. Because we could have a nice performance but RUM will be of bad quality.

Test object / HTML Template

- Depending on different approaches / Ideas (see next chapter), template looks different
- But general structure stays the same and independent variables can be defined
- Here we show different independent variables and variants

5.1.2.1 Lab and Field

- I want to collect Lab and field data for dependent variables for comparison
- This setup is a special case because lab bots (e.g. WPT) simulate at the same time real users for RUM data

Possibilites: - preconfigured Amazon Web Services (AWS) AMIs - localhost Docker

5.1.3 Independent Variables within template

- IV 1 POSITION: Position of included analyitcs script. Values: top-head, bottom-head, bottom-body
- IV 2 ATTRIBUTE: Attribute of included analyitcs script: no-attribute, async, defer
- IV 3 OTHER SCRIPT: Other tracking script included
- Other IVs not included but worth mentioning

I will compare the values from one independent variable only. Therefore, when comparing the values of one independent variable, i need to set a default value for the other independent variables. The default values are:

Position: top-head Attribute: no attribute Other Script: false

Other IVs not included but worth mentioning

- More or less infinite number of independent variables
- Again the big and important fact that each website is different

as described in section X. async and defer attributes on inline scripts are ignored. Still, as many suggest to add them, i will test this as well. it looks like attribute on inline script is ignored... "Inline JavaScript script tags ignore the defer or async attribute" so those variables will not make any sense? Do i see this in the data? As i can see how the GA is included, it will create a async attribute to fetch the analytics.js

What does the code snippet do? Basically the sync part is only.. It creates another script tag which is async and which loads the tracking code.

5.2 Test Object: HTML Template / Test website ideas

- Several ideas are proposed
- Each idea has pro and contra: each idea should be discussed of its usefulness, advantages and disadvantages

5.2.1 WordPress

- Show usage of WordPress with some statistics: Why is it so verbreitet
- Explain Plugin system
- Explain Setup on localhost with wocommerce and GA plugin
- Elaborate why this idea was not used

5.2.2 Plain / Skeletal Website

- Idea: Lab environment to have control over all and see effects of changing independent variables
- Problem: Too far away from reality
- Use this as the simplest test possible, not even POC (POC is http archive site)

5.2.3 HTTP Archive inspired website

- Idea: Get correct page weight
- POC: Show that changing independent variables X affect result

- Chart with changes on time: apps are growing

Listing 5.1: Position 1

```
<!DOCTYPE html>
<html>
    <head>
        <!-- Google Analytics -->
        <script></script>
        <!-- End Google Analytics -->

        <title>
        <meta>
        <link>
        <script>
    </head>

    <body>
        ...
    </body>
</html>
```

Listing 5.2: Position 2

```
<!DOCTYPE html>
<html>
    <head>
        <title>
        <meta>
        <link>
        <script>

        <!-- Google Analytics -->
        <script></script>
        <!-- End Google Analytics -->
    </head>

    <body>
        ...
    </body>
</html>
```

Listing 5.3: Position 3

```
<!DOCTYPE html>
<html>
    <head>
        <title>
        <meta>
        <link>
        <script>

    </head>

    <body>
        ...
        <!-- Google Analytics -->
        <script></script>
        <!-- End Google Analytics -->
    </body>
</html>
```

Listing 5.4: Attribute 1

```
<!DOCTYPE html>
<html>
    <head>
        <!-- Google Analytics -->
        <script></script>
        <!-- End Google Analytics -->

        <title>
        <meta>
        <link>
        <script>
    </head>

    <body>
        ...
    </body>
</html>
```

Listing 5.5: Attribute 2

```
<!DOCTYPE html>
<html>
    <head>
        <!-- Google Analytics -->
        <script async></script>
        <!-- End Google Analytics -->

        <title>
        <meta>
        <link>
        <script>
    </head>

    <body>
        ...
    </body>
</html>
```

Listing 5.6: Attribute 3

```
<!DOCTYPE html>
<html>
    <head>
        <!-- Google Analytics -->
        <script defer></script>
        <!-- End Google Analytics -->

        <title>
        <meta>
        <link>
        <script>
    </head>

    <body>
        ...
    </body>
</html>
```

Listing 5.7: Other Script 1

```
<!DOCTYPE html>
<html>
    <head>
        <!-- Google Analytics -->
        <script></script>
        <!-- End Google Analytics -->









        <title>
        <meta>
        <link>
        <script>
    </head>

    <body>
        ...
    </body>
</html>
```

Listing 5.8: Other Script 2

```
<!DOCTYPE html>
<html>
    <head>
        <!-- Google Analytics -->
        <script></script>
        <!-- End Google Analytics -->

        <!-- Other Script-->
        <script></script>
        <!-- End Other Script -->

        <title>
        <meta>
        <link>
        <script>
    </head>

    <body>
        ...
    </body>
</html>
```

5.2.4 Mirroring a complete e-commerce website

- Which website / shop to clone? Show some statistics about biggest e-commerce websites in germany
- something (test, remove this line again)

Otto Re-write this to otto start page clone chapter

Manual adjustments: - Move everything to test folder because top domain is /otto

What did not work (mostly 404s): - user-set-consent-id-cookie: Cookie with name consentId is not set, user-set-consent-id-cookie returns therefore 404 - subscribeToNewsletterSnippetContent: Change path did not work... - amount.json: Not found, also wl_miniWishlistAmount in local storage does not created - a_info: Mock a_info response json does not work...

- footer - userTiming

WPT RV is returning empty csv when 404s are encountered. Therefore i mock the missing ressources so that WPT can run bulk tests successfully.

- mock image sprite_all_1ba408b2.png

- create empty file called user-set-consent-id-cookie

- change path for subscribeToNewsletterSnippetContent: This will remove the cookie banner... but then WPT works

- Idea: Close to reality as possible
- Problems when mirroring a website
- Elaborate why this idea of mirroring complete website was not used
- I used mock of start page of otto, which works fine
- Compare original otto website with mock

Comparison to original webpage

- Remove GA again from mock, so that mock and original are as similar as possible
- Run the same lab test on both pages: WPT and mabye lighthouse
- Compare both results and explain differences

- Setup: Run WPT on mock and on original website - WPT config: - Browser: Chrome - Number of test runs: 3 - FV and RV - Capture Video - Capture DevTools Timeline - Bulk testing: 100x

Diagrams with FV and RV for both cases:

Technical: - First Byte - Bytes In (Doc) - Requests (Doc)

VIsual: - Document Complete - Speed Index

Problem with Repeat View - Problem with RV, Caching: Otto sets request headers to cache-control: no-cache which means that RV basically downloads all resources again. The mock is hosted on Github, where the cache-control header is set to ... It is not possible to change the github request headers. We can modify the http request headers via html, but this is not a clean solution. Therefore I use a different e-commerce website which does not shut down caching so that the RV results are more similar.

Ideally I would host the mock website on a similar infrastructure as the original site with the same webserver configuration. This is for a masters thesis not feasible.

Zalando Idea: It looks like zalando page does not has that many cache-control headers, therefore it may be easier to clone so that RVs are more similar.

Comparison Diagrams with fixed traffic shaping:

5.3 Test Runs

- This section covers all conducted test runs
- Explain test configuration: how many runs, dependent and independent variables, etc.

5.3.1 WPT Configurations

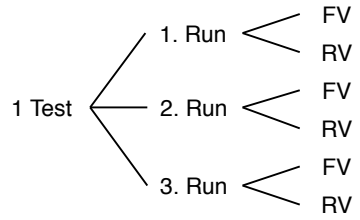
5.3.1.1 General Settings

Table 5.1: Test Runs [Sch99]

Configuration Setting	Options	GA
Test Location	Test Location	.
Browser	Firefox, Chrome	.
Connection	LAN	.
Number of Tests to Run	1 to 9	..
Repeat View	First View and Repeat View, First View Only	.
Capture Video	True or False	..
Keep Test Private	True or False	...
Label	Any String	...
Advanced Tab	...	...
Chromium Tab	...	...
Auth, Script, Block, SPOF, Custom Tabs	...	...
Bulk Testing Tab	List of URLs	...

Explanations First View: "First View refers to the cold cache setup in which nothing is served locally" Repeat View: "Repeat View refers to the warm cache containing everything instantiated by the first view" (2016 Using WPT p. 62)

Figure 5.1: Number of tests to run: 3, First View and Repeat View



Capture Video: ...

For one test, we have actually six times that the website gets loaded and tested. For e.g. 500 URLs in the bulk test list, we have a total of $500 \times 6 = 3000$ page hits.

Table 5.2: Configuration 1

Configuration Setting	Option
Test Location	Test Location
Browser	Chrome
Connection	LAN
Desktop Browser Dimensions	default (1366x768)
Number of Tests to Run	1
Repeat View	First View and Repeat View
Capture Video	True
Keep Test Private	False
Label	none
Advanced Tab	Nothing selected
Chromium Tab	Capture Dev Tools Timeline selected
Auth, Script, Block, SPOF, Custom Tabs	Nothing
Bulk Testing Tab	Test URL x times according to test plan

Configuration 1

Configuration 2 Emulate Mobile Browser

5.3.1.2 Traffic Shaping

- Important to have stable and realistic network condition
- Chromes tool is not the best for this
- Private WPT Instance docker on mac does not allow traffic shaping functionality from WPT
- I use Network Link Conditioner from Apple to slow down the whole machine. See in same blogpost that Patrick highly recommends this
- WPT also slows down their whole machines

- IN general internet connection is very unstable. If i run network link conditionier with e.g. DSL each speedtest gives different results. And other test platforms such as fast.com gives also different result.
- as long as internet connection is stable along all tests, it should not make a big difference because i compare the different variants. Therefore internet connection will fall out of the equation
- i will use the durchschnitt in germany which seems to be 40 mbit per second. or actually i use LTE profile from network conditioner which is 50 mbit per second

5.3.2 Test Object (Website) Variations

as described before, i will compare the values within one independent variable. This is needed in order to compare the impact of the different values within one IV. For example, i want to measure if there is a difference in performance between the different script attributes. To measure this, i set the default values for the other IVs and vary the values for the IV attribute

Positions: 1: Top of head element 2: just before closing head element 3: just before closing body element

Variants

Variant	Position	Attribute	Other Scripts
Variant P1	top-head	none	no
Variant P2	bottom-head	none	no
Variant P3	bottom-body	none	no
Variant A1	top-head	none	no
Variant A2	top-head	async	no
Variant A3	top-head	defer	no
Variant OS1	top-head	none	no
Variant OS2	top-head	none	yes

Table 5.3: Your caption here

I will not compare variants which are not from the same subgroup, e.g. Variant A2 will not be compared to Variant OS2. Because the first row of the variants table also includes the default values for Attribute and Other Scripts, VP1 is equal to VA1 and VO1.

With the defined IVs and variants, I can create the test objects, that is the index.html files with the corresponding setup. Because its easier to differentiate i will create for the three equal variants nevertheless own index files.

For each test variant, I will create a concrete test artefact, which is a modified index.html. This index.html needs to be uploaded to the webserver before starting with the tests.

All variants will have the same name which is index.html. This is the default file which will be delivered by the webserver when accessing root path of webpage.

Variants to measure: ——————

- Original website - Mock without GA - Position 1 - Position 2 - Position 3 - Attribute 1 - Attribute 2 - Attribute 3 - Other Script True - Other Script False

5.3.3 Test Plan. Generate the data

The Google Analytics code is more or less fixed and there are no configurations. It would be possible to change config of script, e.g. change sample rate, track other metrics etc. But it is not possible to change default tracking behaviour (?)

How the script is included into the file should reflected withing Website Variations

I will use only one WPT Configuration for all tests. Other WPT config can be used in future work, e.g. emulate mobile device.

Table 5.4: Test Runs [Sch99]

Variant	Traffic Shaping	Runs	Date
V-P1	DSL	500	2021-05-07
V-P2	DSL	500	2021-05-07
V-P3	DSL	500	2021-05-07
V-A1	DSL	500	2021-05-07
V-A2	DSL	500	2021-05-07
V-A3	DSL	500	2021-05-07
V-OS1	DSL	500	2021-05-07
V-OS1	DSL	500	2021-05-07

Pre-step: Compare Mock website with and without GA included The comparison between mock and original is part of chapter Test Object

5.3.4 Test Protocol

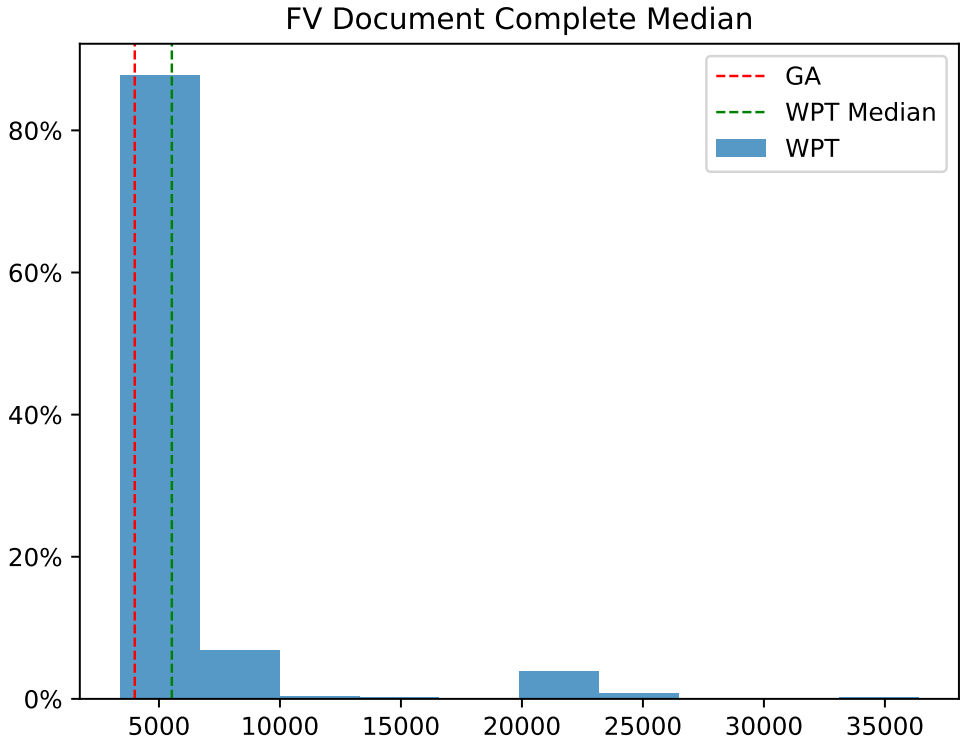
- Deploy variant of index.html by pushing to GitHub
- Start Network Link Conditioner with specified config on local machine
- Test internet speed with speedtest-cli
- Start local WPT server and agent
- Configure WPT according to specified setup and add list of urls to bulk test interface
- Run test

- When finished, download summary csv file
- On GA helper site, fetch and download data for the current day

5.3.5 Tool support for diagrams and data analysis

- python
- Matplotlib
- seaborn library

6 Evaluation



- Last chapter...
- This chapter: Describe shortly all sections from this chapter
- In the next chapter...

6.1 Test Results

6.1.1 Metrics for Evaluation

Page Weight: Measured by WPT: - bytes - bytes uncompressed - Requests

Technical Timings / API: Measured by WPT and GA: - page load time - domain lookup time - page download time - redirection time - server connection time - server response time - Dom interactive time - Dom content loaded time

Visual Metrics / Web Vitals: Measured by WPT TODO Measure also with GA / own script ??: - CLS - FCP - FMP - LCP - SI

- Visually complete ? - Time to Interactive ? Is this the same as DOM interactive time ? Core Web Vital FID ?? -> Can not be measured without real users... From WPT bulk section. Also include this somewhere for comparison ?: - Filmstrips ? - Waterfall ? - Visual Progress ? - Layout Shifts ?

6.1.2 Original vs Mock Plain

6.1.3 Mock Plain vs Position 1 (which is default position of GA: Check this again!)

TODO rename this like with GA true false ?

6.1.4 Position 1 vs 2 vs 3

6.1.5 Attribute 1 vs 2 vs 3

6.1.6 Other script True vs False

6.2 General

- For each attempt, describe: Threats to validity, generalizability

generalizability: meine Daten zeige nur für Chrome, MacBook, diese Geschwindigkeit etc. Und auch nur für diese Test-Website Die Schwierigkeit der Generalisierbarkeit ist eines der grössten Probleme bei dieser Fragestellung

6.3 Plain / Skeletal Website

- Information gained from this experiment
- Limitations and questions which can not be answered with this approach

6.4 Mirroring

6.5 HTTP Archive inspired website

- Information gained from this experiment
- Meaning and interpretation of the collected data
- Limitations and questions which can not be answered with this approach

6.6 WebPageTest Bulk Tests

- Bulk testing is a feature for private instances only
- Misuse this feature to test the same website X times

6.6.1 Bulk Test Overview: Description of test result page

- Each test has Test ID: YYMMDD_random_random
- Test results after bulk test available under `http://localhost:4000/result/{testID}/`
- For each test run, following data is available:
  - Link to test results: Test result page as same as for single test run
  - Median load time (First view)
  - Median load time (Repeat view)
  - Median Speed Index (First View)
  - Raw page data (file: [TestID_summary.csv]
  - Raw object data (file: [TestID_details.csv])
  - Http archive (.har) (file: json)
- Average First View Load Time
- Average Repeat View Load Time
- Combined Raw: Page Data (file: [TestID_summary.csv])
- Combined Raw: Object Data (file: [TestID_details.csv]). For 100 test runs, this file is appr. 20 MB, 24432 rows, 76 columns.
- Aggregate Statistics (file: [TestID_aggregate.csv])

6.6.2 Summary File for one Test

- Contains 6 rows: 3 test runs: for each test runs 1x first view and 1x repeat view
- Rows 1, 3, 5 contain FV, rows 2, 4, 6 contain data for RV

6.6.3 Aggregate Statistics File

- Contains aggregated data from bulk test
- One row for each test run: For 100 URLs in bulk test will be 100 rows in csv
- Each metric is available with Median, Average, Standard Deviation, Min, Max

- Metrics are available once from FV and once for Repeat View
- Metrics:
  - Successful Tests
  - Document Complete
  - Fully Loaded
  - First Byte
  - Start Render
  - Bytes In (Doc)
  - Requests (Doc)
  - Load Event Start
  - Speed Index
  - Last Visual Change
  - Visually Complete
- => For metric details, see Terms and Definitions

6.6.4 Compare Section

WPT has a feature to compare multiple tests. Accessible under compare URL: `http://localhost:4000/video/compare.php?tests={TestID},{TestID},...`

The compare page contains:

- Film strip
- Waterfall diagram
- Visual Progress diagram
- Timings diagram:
  - Visually Complete (First View Visually Complete Median)
  - Last Visual Change
  - Load Time (onload)
  - ...
- Cumulative Layout Shift diagram
- Requests diagram
- Bytes diagram
- Visually complete

- Last Visual Change
- Load Time (onload)
- Load Time (Fully Loaded)
- DOM Content Loaded
- Speed Index
- Time to First Byte
- Time to Title
- Time to Start Render
- CPU Busy Time
- 85% Visually Complete
- 90% Visually Complete
- 95% Visually Complete
- 99% Visually Complete
- First Contentful Paint
- First Meaningful Paint
- Largest Contenful Paint
- Cumulative Layout Shift
- html Requests
- html Bytes
- js Requests
- js Bytes
- css Requests
- css Bytes
- image Requests
- image Bytes
- flash Requests
- flash Bytes

- font Requests
- font Bytes
- video Requests
- video Bytes
- other Requests
- other Bytes

6.7 Internal, external validity

- At this point, i have the data collected and can analyse it
- The quality and quantity of the data needs to be discussed
- Quality: There are chances that some data are malformed, e.g. because internet connection was bad, etc.
- Quantity: Is the amount of data sufficient to make the evaluation generalisable

7 Future Work

- Last chapter...
- This chapter: Describe shortly all sections from this chapter
- In the next chapter...

7.1 Limitations of this thesis

- Discussion of unobserved topics
- Discussion of possible next steps

7.2 Other measurement tools and metrics

List of tools and metrics worth investigating Custom Metrics

7.2.1 Google Analytics 4

7.3 Speed Kit

7.4 PWAs, AMPs, Service Workers, Caching, HTTP2 etc.

- Overview of other web technologies and how they could be relevant for further research

8 Conclusion

- Last chapter...
- This chapter: Describe shortly all sections from this chapter
- Scope and contribution of this thesis
- Short summary of each chapter:
  - Problem statement and why it is worth to examine research question
  - Terms and definitions
  - (Related work)
  - Approach and evaluation of practical work
  - Future work

- Several topics wurden bearbeitet in this thesis, such like mocking a website for testing purposes, literature review, metrics taxonomy, and the main part which is an experiment

9 Appendix

9.1 WebPageTest Bulk Tests

9.1.1 Single Test Raw page data

WPT Metrics from summary file

Name	Description
minify_total	Total bytes of minifiable text static assets.
responses_200	The number of responses with HTTP status code of 200, OK.
testStartOffset	...
bytesOut	The total bytes sent from the browser to other servers.
gzip_savings	Total bytes of compressed responses.
requestsFull	...
start_epoch	...
connections	The number of connections used.
base_page_cdn	The CDN provider for the base page.
bytesOutDoc	Same as bytesOut but only includes bytes until the Document Complete event. Usually when all the page content has loaded (window.onload).
result	Test result code.
final_base_page_request_id	...
basePageSSLTime	...
docTime	Same as loadTime.
domContentLoadedEventEnd	Time in ms since navigation started until document DOMContentLoaded event finished.
image_savings	Total bytes of compressed images.
requestsDoc	The number of requests until Document Complete event.
firstMeaningfulPaint	...
score_cookies	WebPageTest performance review score for not using cookies on static assets.
firstPaint	RUM First Paint Time, the time in ms when browser first painted something on screen. It's calculated on the client for browsers that implement this method.
score_cdn	WebPageTest performance review score for using CDN for all static assets.
optimization_checked	Whether or not optmizations were checked.

score_minify	WebPageTest performance review score for minifying text static assets.
gzip_total	Total bytes of compressible responses.
responses_404	The number of responses with HTTP status code of 404, not found.
loadTime	The total time taken to load the page (window.onload) in ms.
URL	The tested page URL.
score_combine	WebPageTest performance review score for bundling JavaScript and/or CSS assets.
firstContentfulPaint	...
image_total	Total bytes of images.
score_etags	WebPageTest performance review score for disabling *ETag*s.
loadEventStart	Time in ms since navigation started until window.onload event was triggered (from W3C Navigation Timing).
minify_savings	Total bytes of minified text static assets.
score_progressive_jpeg	WebPageTest performance review score for using progressive JPEG.
domInteractive	...
score_gzip	WebPageTest performance review score for using gzip compression for transferring compressable responses.
score_compress	WebPageTest performance review score for compressing images.
domContentLoadedEventStart	Time in ms since navigation started until document DOMContentLoaded event was triggered (from W3C Navigation Timing).
final_url	...
bytesInDoc	Same as bytestIn but only includes bytes until Document Complete event.
firstImagePaint	...
score_keep-alive	WebPageTest performance review score for using persistent connections.
loadEventEnd	Time in ms since navigation started until window.onload event finished.
cached	0 for first view or 1 for repeat view.
score_cache	WebPageTest performance review score for leveraging browser caching of static assets.
responses_other	The number of responses with HTTPS status code different from 200 or 404.
main_frame	...
fullyLoaded	The time (in ms) the page took to be fully loaded — e.g., 2 seconds of no network activity after Document Complete. This will usually include any activity that is triggered by javascript after the main page loads.
requests	List of details of all requests on tested page.

final_base_page_request	...
TTFB	Time to first byte, which is the duration in ms from when the user first made the HTTP request to the very first byte of the page being received by the browser.
bytesIn	The amount of data that browser had to download in order to load the page. It is also commonly referred to as the page size.
osPlatform	...
test_run_time_ms	...
tester	The ID of tester that performed the page test.
browser_version	The browser version.
document_origin	...
document_URL	...
date	Time and date (number of seconds since Epoch) when test was complete.
PerformancePaintTiming.first-paint	...
osVersion	...
domElements	The total number of DOM elements.
browserVersion	The browser version.
fullyLoadedCPUms	CPU busy time in ms until page was fully loaded.
browser_name	The browser name.
PerformancePaintTiming.first-contentful-paint	...
base_page_cname	...
eventName	...
os_version	...
base_page_dns_server	...
fullyLoadedCPUpct	Average CPU utilization up until page is fully loaded.
domComplete	...
base_page_ip_ptr	...
document_hostname	...
lastVisualChange	Time in ms until the last visual changed occured.
visualComplete	Time in ms when page was visually completed.
render	The first point in time (in ms) that something was displayed to the screen. Before that user was staring at a blank page. This does not necessarily mean the user saw the page content — it could just be something as simple as a background color — but it is the first indication of something happening for the user.
SpeedIndex	The SpeedIndex score.
visualComplete85	Time in ms when page was visually completed 85%.
visualComplete90	Time in ms when page was visually completed 90%.
visualComplete95	Time in ms when page was visually completed 95%.
visualComplete99	Time in ms when page was visually completed 99%.
LargestContentfulPaintType	...
LargestContentfulPaintNodeType	...

chromeUserTiming.navigationStart	...
chromeUserTiming.fetchStart	...
chromeUserTiming.responseEnd	...
chromeUserTiming.domLoading	...
chromeUserTiming.markAsMainFrame	...
chromeUserTiming.domInteractive	...
chromeUserTiming.domContentLoadedEventStart	...
chromeUserTiming.domContentLoadedEventEnd	...
chromeUserTiming.firstPaint	...
chromeUserTiming.firstContentfulPaint	...
chromeUserTiming.firstImagePaint	...
chromeUserTiming.firstMeaningfulPaint	...
chromeUserTiming.firstMeaningfulPaintCandidate	...
chromeUserTiming.domComplete	...
chromeUserTiming.loadEventStart	...
chromeUserTiming.loadEventEnd	...
chromeUserTiming.LargestContentfulPaint	...
chromeUserTiming.LargestTextPaint	...
chromeUserTiming.CumulativeLayoutShift	...
run	The run number.
step	...
effectiveBps	Bytes per seconds, i.e.: total of bytes in / total time to load the page.
effectiveBpsDoc	Same as effectiveBps but until Document Complete event.
domTime	The total time in ms until a given DOM element (specified via domelement parameter when running a test) was found on the page.
aft	Above the Fold Time (no longer supported). The time taken to load everything in the viewport above the fold.
titleTime	Total time in ms until page title was set on browser.
domLoading	...
server_rtt	...
smallImageCount	...
bigImageCount	...
maybeCaptcha	...
bytes.html	...
requests.html	...
bytesUncompressed.html	...
bytes.js	...
requests.js	...
bytesUncompressed.js	...
bytes.css	...
requests.css	...
bytesUncompressed.css	...
bytes.image	...
requests.image	...

bytesUncompressed.image	...
bytes.flash	...
requests.flash	...
bytesUncompressed.flash	...
bytes.font	...
requests.font	...
bytesUncompressed.font	...
bytes.video	...
requests.video	...
bytesUncompressed.video	...
bytes.other	...
requests.other	...
bytesUncompressed.other	...
id	...
chromeUserTiming.InteractiveTime	...

Table 9.1: Your caption here

9.1.2 Single Test Raw object data

9.1.3 Single Test Http archive (.har)

9.1.4 Combined Test Raw page data

9.1.5 Combined Test Raw object data

9.1.6 Combined Test Aggregate data

Bibliography

[Abe] ABERDEENGROUP: *The Performance of Web Applications. Customers Are Won or Lost in One Second.* https://info.headspin.io/hubfs/Analyst%20Reports/5136-RR-performance-web-application.pdf, Abruf: 2021-06-06

[Akaa] AKAMAI: *Akamai Online Retail Performance Report: Milliseconds Are Critical.* https://www.akamai.com/de/de/about/news/press/2017-press/akamai-releases-spring-2017-state-of-online-retail-performance-repor jsp, Abruf: 2021-06-06

[Akab] AKAMAI: *Performance Matters. 9 Key Consumer Insights.* https://www.akamai.com/us/en/multimedia/documents/content/akamai-performance-matters-key-consumer-insights-ebook.pdf, Abruf: 2021-06-06

[BAB+20] BHATTI, Anam ; AKRAM, Hamza ; BASIT, Hafiz M. ; KHAN, Ahmed U. ; NAQVI, Syeda Mahwish R. ; BILAL, Muhammad: E-commerce trends during COVID-19 Pandemic. In: *International Journal of Future Generation Communication and Networking* 13 (2020), Nr. 2, S. 1449–1452. – ISSN 2233–7857 IJFGCN

[Bel] BELSHE, Mike: *More Bandwidth Doesn't Matter (much).* https://docs.google.com/a/chromium.org/viewer?a=v&pid=sites&srcid=Y2hyb21pdW0ub3JnfGRldnxneDoxMzcyOWI1N2I4YzI3NzE2, Abruf: 2021-08-03

[BGP15] BEKAVAC, Ivan ; GARBIN PRANIČEVIĆ, Daniela: Web analytics tools and web metrics tools: An overview and comparative analysis. In: *Croatian Operational Research Review* 6 (2015), Oktober, Nr. 2, S. 373–386

[CA11] COHEN-ALMAGOR, Raphael: Internet History. In: *International Journal of Technoethics* 2 (2011), April, Nr. 2, 45–64. http://dx.doi.org/10.4018/jte.2011040104. – DOI 10.4018/jte.2011040104. – ISSN 1947–3451, 1947–346X

[CKR] CROCKER, Cliff ; KULICK, Aaron ; RAM, Balaji: *Real-User Monitoring at Walmart. SF & SV Web Performance Group.* https://www.slideshare.net/devonauerswald/walmart-pagespeedslide, Abruf: 2021-06-06

[CP09] CROLL, Alistair ; POWER, Sean: *Complete Web Monitoring*. 1st ed. Beijing; Cambridge [Mass.] : O'Reilly, 2009

[Dev] DEVICEATLAS: *Android v iOS market share 2019*. https://deviceatlas.com/blog/android-v-ios-market-share, Abruf: 2021-06-06

[For] FORRESTER: *The Total Economic Impact Of Accelerated Mobile Pages*. https://amp.dev/static/files/The_Total_Economic_Impact_Of_AMP.pdf, Abruf: 2021-06-06

[FV] FRANCO, Luis ; VALDÉS, Mayra: *History of Google Analytics*. https://onward.justia.com/history-of-google-analytics/, Abruf: 2021-08-05

[GA] GOOGLE ; AWWWARDS: *Speed Matters. Designing for Mobile Performance*. https://www.awwwards.com/brain-food-perceived-performance/, Abruf: 2021-05-06

[Ges21] GESSERT, Felix: *Mobile Site Speed and the Impact on E-Commerce*. https://youtu.be/RTt1RfMUvOQ. Version: 05.06.2021. – code.talks 2019

[Gria] GRIGORIK, Ilya: *Constructing the Object Model*. https://developers.google.com/web/fundamentals/performance/critical-rendering-path/constructing-the-object-model, Abruf: 2021-08-03

[Grib] GRIGORIK, Ilya: *Render Blocking CSS*. https://developers.google.com/web/fundamentals/performance/critical-rendering-path/render-blocking-css, Abruf: 2021-08-03

[Gri13] GRIGORIK, Ilya: *High Performance Browser Networking*. Beijing; Sebastopol, CA : O'Reilly, 2013

[Hei20] HEINEMANN, Gerrit: *Der neue Online-Handel: Geschäftsmodelle, Geschäftssysteme und Benchmarks im E-Commerce*. Wiesbaden : Springer Fachmedien Wiesbaden, 2020

[Her19] HERMOGENO, Darwin L.: E-Commerce: History and Impact on the Business and Consumers. In: *International Journal of Engineering Science* 9 (2019), Nr. 3

[Hog14] HOGAN, Lara C.: *Designing for performance: Weighing Aesthetics and Speed*. First edition. Sebastapol CA : O'Reilly Media, 2014

[HTT] HTTPARCHIVE: *Report: State of the Web*. https://httparchive.org/reports/state-of-the-web, Abruf: 2021-08-04

[IKOK10] ISLAM, A.K.M. N. ; KOIVULAHTI-OJALA, Mervi ; KÄKÖLÄ, Timo: A lightweight, industrially-validated instrument to measure user satisfaction and service quality experienced by the users of a UML modeling tool. In: *AMCIS 2010 Proceedings* 8 (2010)

[Jan09] JANSEN, Bernard J.: Understanding User-Web Interactions via Web Analytics. In: *Synthesis Lectures on Information Concepts, Retrieval, and Services* 1 (2009), Januar, Nr. 1, S. 1–102

[Joh] JOHNSON, Benjamin: *How CSS works: Parsing and painting CSS in the critical rendering path*. `https://blog.logrocket.com/how-css-works-parsing-painting-css-in-the-critical-rendering-path-b3` Abruf: 2021-08-04

[Key] KEYCDN: *How to Reduce DNS Lookups*. `https://www.keycdn.com/support/reduce-dns-lookups`, Abruf: 2021-08-03

[KL17] KOHAVI, Ron ; LONGBOTHAM, Roger: Online Controlled Experiments and A/B Testing. In: *Encyclopedia of Machine Learning and Data Mining*. Boston, MA : Springer US, 2017, S. 922–929

[KO20] KUMAR, Vikas ; OGUNMOLA, Gabriel A.: Web Analytics for Knowledge Creation: A Systematic Review of Tools, Techniques, and Practices. In: *International Journal of Cyber Behavior, Psychology and Learning* 10 (2020), Januar, Nr. 1, S. 1–14

[Lin] LINDEN, Greg: *Make Data Useful. Standford Data Mining Class CS345A, 2006.* `http://glinden.blogspot.com/2006/12/slides-from-my-talk-at-stanford.html`, Abruf: 2021-06-06

[LO20] LANG, Gil ; OTTEN, Steffen: *Erfolgreicher Online-Handel für Dummies*. Weinheim : Wiley-VCH, 2020

[MAC] MEDER, Sam ; ANTONOV, Vadim ; CHANG, Jeff: *Driving user growth with performance improvements*. `https://medium.com/pinterest-engineering/driving-user-growth-with-performance-improvements-cfc50dafadd7`, Abruf: 2021-06-06

[Mar15] MAREK, Kate: *Library technology reports: expert guides to library systems and services. Using Web Analytics in the Library. Volume 47, Number 5*. 2015

[May] MAYER, Marissa: *Conference Keynote, Web 2.0, 2006*

[MDNa] MDN, Contributors: *Critical rendering path*. `https://developer.mozilla.org/en-US/docs/Web/Performance/Critical_rendering_path`, Abruf: 2021-08-03

[MDNb] MDN, Contributors: *Populating the page: how browsers work*. `https://developer.mozilla.org/en-US/docs/Web/Performance/How_browsers_work`, Abruf: 2021-08-03

[MDNc] MDN, Contributors: *Understanding latency*. `https://developer.mozilla.org/en-US/docs/Web/Performance/Understanding_latency`, Abruf: 2021-08-03

[Mor18] MORYS, André: Mit A/B-Tests die Optimierungsideen validieren. In: *Die digitale Wachstumsstrategie*. Wiesbaden : Springer Fachmedien Wiesbaden, 2018, S. 97–119

[Mos] MOSES, Lucia: *How GQ cut its webpage load time by 80 percent*. `https://digiday.com/media/gq-com-cut-page-load-time-80-percent/`, Abruf: 2021-06-06

[NC11] NAKATANI, Kazuo ; CHUANG, Ta-Tao: A web analytics tool selection method: an analytical hierarchy process approach. In: *Internet Research* 21 (2011), Januar, Nr. 2, S. 171–186

[SB19] STONE, Brad ; BEZOS, Jeffrey: *Der Allesverkäufer: Jeff Bezos und das Imperium von Amazon*. 2. erweiterte Neuausgabe. Frankfurt : Campus Verlag, 2019

[SBR+19] STEIREIF, Alexander ; BÜCKLE, Markus ; RIEKER, Rouven ; ERTLER, Bernhard ; SKIBBA, Janina: *Handbuch Online-Shop: Strategien, Erfolgsrezepte, Lösungen*. 2., aktualisierte und erweiterte Auflage. Bonn : Rheinwerk Verlag, 2019 (Rheinwerk Computing)

[Sch99] SCHWARZ, Kerstin: *DISDBIS*. Bd. 64: *Das Konzept der Transaktionshülle zur konsistenten Spezifikation von Abhänigigkeiten in komplexen Anwendungen.* Infix Verlag, St. Augustin, Germany, 1999

[SKG+] SCHELHOWE, Luetke ; KAGAWA, Shuhei ; GRUDA, Thorbjoern ; CYBULSKI, Jeff ; MARTIN, David: *Loading Time Matters*. `https://engineering.zalando.com/posts/2018/06/loading-time-matters.html`, Abruf: 2021-06-06

[SKS14] SINGAL, Himani ; KOHLI, Shruti ; SHARMA, Amit K.: Web analytics: State-of-art & literature assessment. In: *2014 5th International Conference - Confluence The Next Generation Information Technology Summit (Confluence)*. Noida : IEEE, September 2014, S. 24–29

[SSMG17] SANTOS, Valdeci Ferreira d. ; SABINO, Leandro R. ; MORAIS, Greiciele M. ; GONCALVES, Carlos A.: E-Commerce: A Short History Follow-up on Possible Trends. In: *International Journal of Business Administration* 8 (2017), Oktober, Nr.

7, 130. http://dx.doi.org/10.5430/ijba.v8n7p130. – DOI 10.5430/ijba.v8n7p130. – ISSN 1923–4015, 1923–4007

[Unb] UNBOUNCE: *Think Fast: The 2019 Page Speed Report Stats & Trends for Marketers*. https://unbounce.com/page-speed-report/, Abruf: 2021-05-06

[Wik16] WIKIPEDIA: *Wissenschaftliche Arbeit*. https://de.wikipedia.org/w/index.php?title=Wissenschaftliche_Arbeit&oldid=156007167. Version: 2016, Abruf: 2016-07-21

[WK09] WAISBERG, Daniel ; KAUSHIK, Avinash: Web Analytics 2.0: Empowering Customer Centricity. (2009)

[ZP15] ZHENG, Guangzhi ; PELTSVERGER, Svetlana: Web Analytics Overview. In: KHOSROW-POUR, Mehdi (Hrsg.): *Encyclopedia of information science and technology*. Information Science Reference, 2015, Kapitel 756, S. 7674–7683

List of Figures

List of Tables

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.
Ich bin mit einer Einstellung in den Bestand der Bibliothek des Fachbereiches einverstanden.

Hamburg, den _____ Unterschrift: _____