Aalto University
School of Science
Degree Programme of Information Networks

Pyry Kröger

Tools for Visualizing Geographical Data on the Web

Reducing the Work Needed by Eliminating Boilerplate

Master's Thesis
Espoo, October 14, 2014

DRAFT! — November 7, 2014 — DRAFT!

Supervisor: Professor Petri Vuorimaa
Instructor: Sami Vihavainen D.Sc. (Tech.)

Aalto University
School of Science
Degree Programme of Information Networks

ABSTRACT OF
MASTER'S THESIS

Author:	Pyry Kröger		
Title:			
Tools for Visualizing Geographical Data on the Web - Reducing the Work Needed by Eliminating Boilerplate			
Date:	October 14, 2014	**Pages:**	viii + 83
Professorship:	Media	**Code:**	T-111
Supervisor:	Professor Petri Vuorimaa		
Instructor:	Sami Vihavainen D.Sc. (Tech.)		

A dissertation or thesis is a document submitted in support of candidature for a degree or professional qualification presenting the author's research and findings. In some countries/universities, the word thesis or a cognate is used as part of a bachelor's or master's course, while dissertation is normally applied to a doctorate, whilst, in others, the reverse is true.

!FIXME **Abstract text goes here (and this is an example how to use fixme).** FIXME! Fixme is a command that helps you identify parts of your thesis that still require some work. When compiled in the custom `mydraft` mode, text parts tagged with fixmes are shown in bold and with fixme tags around them. When compiled in normal mode, the fixme-tagged text is shown normally (without special formatting). The draft mode also causes the "Draft" text to appear on the front page, alongside with the document compilation date. The custom `mydraft` mode is selected by the `mydraft` option given for the package `aalto-thesis`, near the top of the `thesis-example.tex` file.

The thesis example file (`thesis-example.tex`), all the chapter content files (`1introduction.tex` and so on), and the Aalto style file (`aalto-thesis.sty`) are commented with explanations on how the Aalto thesis works. The files also contain some examples on how to customize various details of the thesis layout, and of course the example text works as an example in itself. Please read the comments and the example text; that should get you well on your way!

Keywords:	work, in, progress
Language:	English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan tutkinto-ohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Pyry Kröger		
Työn nimi:			
Työkaluja geografisen datan web-visualisointiin - Tarvittavan boilerplate-koodin vähentäminen			
Päiväys:	14. lokakuuta 2014	**Sivumäärä:**	viii + 83
Professuuri:	Media	**Koodi:**	T-111
Valvoja:	Professori Petri Vuorimaa		
Ohjaaja:	Tekniikan tohtori Sami Vihavainen		
Tämä on diplomityö.			
Asiasanat:	vähän, vielä, kesken		
Kieli:	Englanti		

Acknowledgements

So long, and thanks for all the fish.

Espoo, October 14, 2014

Pyry Kröger

Abbreviations and Acronyms

API	Application Programming Interface
CC	Cyclomatic Complexity
CSS	Cascading Style Sheets
CSS3	Cascading Style Sheets level 3
CSV	Comma-Separated Values
ECMA	European Computer Manufacturers Association
GeoJSON	Geographic JavaScript Object Notation
GIS	Geographic Information System
HD	Halstead Difficulty
HE	Halstead Effort
HTML	HyperText Markup Language
HTML5	HyperText Markup Language version 5
JSON	JavaScript Object Notation
LOC	Lines of Code
LLOC	Logical Lines of Code
POI	Point of Interest; a piece of data with geospatial dimension
REST	Representational State Transfer
SPA	Single-Page Application
UI	User Interface

Contents

Chapter 1

Introduction

Geographical data is data with geospatial dimension, such as Point of Interest (POI) with location data as coordinates (Kraak and Ormeling, 2011, chap. 1.2). The most natural method for visualizing geographical data is usually with various maps. In the past, geographical data was predominantly visualized by cartographers, but it has been recognized (Kraak and MacEachren, 1999) that the situation has changed, with people from increasing number of fields having a need – and the possibility (Slocum and McMaster, 2014, chap. 1) – for visualizing geographical data. Moreover, the popularity of Google Maps (Google, 2005b) along with its Application Programming Interface (API) (Google, 2005a) has proved that in addition to experts of other academic fields, there is a definite demand for web map visualizations within consumers as well.

The web makes publishing and bundling map visualizations extraordinarily straightforward when compared to traditional desktop-based Geographic Information System (GIS) applications: traditional desktop-based GIS system requires an installation of the GIS application and often additional tools and accounts for publishing the visualization, while using web-based mapping software ideally requires no additional software or tools or even accounts. This is especially important when the visualizations are made by non-cartographers who only make visualizations occasionally and lack the needed resources and experience for more complex publishing process (Miller, 2006). However, as the web platform is primarily designed for static documents (Berners-Lee, 1989; Berners-Lee et al., 1992) instead of dynamic applications (Jazayeri, 2007), there are some additional concerns to address when making a complex data visualization on the web.

Data visualization enables the users to obtain insight about data quickly and efficiently (van Wijk, 2005). Visualizing geographical data helps users perceive geospatial relationships and patterns. Maps can be used for determining information on distances, directions and areas (Kraak and Ormeling, 2011, chap. 1.1). Also, using geographical visualizations, it is also possible to organize data spatially and visually, allowing more efficient memorization of data.

1.1 Problem Statement

Currently, there are several libraries available for displaying maps and simple visualizations (Google, 2005a; Agafonkin, 2011; MetaCarta, 2006). However, the problem is that none of the mainstream libraries is of sufficiently high abstraction level for building map visualizations efficiently, resulting in the need for writing *boilerplate* code that does not directly contribute to the visualization. Moreover, the libraries are not designed primarily for visualizations and therefore do not encourage or push the visualizer to create visually and cognitively effective visualizations, resulting in subpar visualizations (Slocum and McMaster, 2014, chap. 1).

In the scope of this thesis, we adopt the process definitions of van Wijk (2005) by making the difference between an efficient and an effective process. By an efficient process, we mean a process which requires as little as possible effort and other resources to complete. By an effective process, we mean a process which reaches its target results sufficiently. Therefore, building a visualization efficiently indicates that the building process is as effortless as possible, and building effective visualization indicates that the resulting visualization conveys its intended message appropriately.

1.2 Objectives and Scope

Our objective is to make creating map visualizations for the web more efficient by building a higher abstraction level reusable software system for map visualizations, and to evaluate the efficiency of the system. This system should provide the structure for creating the visualization as well as common web application features needed in modern web applications.

In order to find the solution for the problem, it is necessary to study geographical visualizations and software reuse. The process of making geographical visualizations should be studied to ensure that the system enables creating *effective* visualizations. In addition, software reuse should be studied in order to be able to create visualizations *efficiently*. Therefore, we select the following research questions for this thesis:

RQ1 Does a reusable software system enable creating geographical visualizations *efficiently*?

RQ2 Does a reusable software system enable creating *effective* geographical visualizations?

1.3 Approach

In order to build an efficient system for visualizing geographical data, it is needed to study (a) how to visualize geographical data and (b) how to build reusable software. We begin by first studying the basics of data visualization with an emphasis on geographical data, maps and the visualization process. After visualization, we study the essence of software reuse, focusing on building and evaluating reusable software. Then, we create a reusable visualization system and evaluate its effect on visualization effectiveness and efficiency of the building process. !FIXME **Remove this section?** FIXME!

1.4 Structure of the Thesis

!FIXME **This section is an early draft and will change considerably.** FIXME! Chapter 1 (this introduction) presents the motivation for this thesis as well as the problem statement. Chapter 2 describes the background of the work. In particular, the chapter describes how to visualize geographical data and the essence of web software reuse. Chapter 3 presents the web technology, standards and other needed material for building a web visualization as well as describing some existing map visualizations.

In chapter 4, we discuss the methods used to examine the problem and evaluate solutions we will propose. In chapter 5, we describe the methods used to solve the problem. In chapter 6, we evaluate the implementation and its results. !Fixme **Chapters 7 and 8 missing. Also, maybe elaborate description about chapters 4-6 a bit** Fixme!

!Fixme **Rewrite. No active passive mixing etc.** Fixme!

Chapter 2

Data Visualization

2.1 Definition

According to Kosara (2007, chap. 3), there is no universally accepted definition of visualization. He proposes the following for a "minimal set of requirements for any visualization":

- It is based on (non-visual) data
- It produces an image
- The results are readable and recognizable

According to him, while visualizations can also have other properties or qualities, such as interaction or visual efficiency, the requirements above are the ones needed for technical definition of the term. Moreover, it should be emphasized that according to this definition, visualization is the *process* itself, not the result of it.

Kosara (2007, chap. 4) argues that visualization is separated into two types, *pragmatic* and *artistic* visualization. !FIXME **This is related to Hassenzahl's UX definition. May be nice to refer to that as well somewhere.** FIXME! Pragmatic visualization focuses on the analysis of the data in order to show its relevant characteristics as efficiently as possible. Artistic visualization on the other hand concentrates on the communication of a concern, not the display of the actual data. Therefore, artistic visualizations may emphasize or even exaggerate some of the features of the data. Kosara states that while these types focus on the opposite sides of the visualization spectrum, it may be possible to close the gap using, e.g., interaction.

The first requirement for visualizations by Kosara (2007) dictates that the visualization is based on data. This is in alignment with the principle of Tufte (1986) which states that visualizations should, above all else, show the data. This is an essential characteristic of *data* visualizations: the visualization is a function which takes data as an input and produces a visual object as an output. In less technical terms, this means that the visualization turns data into visual, effortlessly and efficiently digestible format. !FIXME **Comment by Sami: Tää on hyvä peruspointti, eli miksi visualisaatiota ylipäätään tehdään. Voisi ehkä laajentaa/tuoda enemmän esiin.** FIXME!

This leads to the fact that the data and visualization are not inherently tied to each other; the visualization "function" can be independent of the data and thus it may be possible to create a visualization framework or platform which is able to function on a potentially wide range of data.

!FIXME **The goal of visualization is (usually) better understanding of the data.** FIXME!

2.2 Principles for Successful Data Visualization

The requirements presented in the previous section are sufficient for the definition of data visualization. However, they do not convey any information about visualization quality. In order to discover the characteristics for successful data visualization, additional principles are needed. Tufte (1986, p. 13) states that excellent graphics (i.e., results of visualizations) consist of "complex ideas communicated with clarity, precision and efficiency". In practice, this means that the graphics should emphasize the actual data and its nuances above everything else, while serving a clear purpose.

In addition to graphics principles presented in the previous paragraph, Tufte (1986, p. 93) presents the concept of *data-ink*. Data-ink represents the ink used for displaying the data in a visualization. He argues that in an excellent visualization, most, if not all, ink used should contribute to display of the data. However, research by Inbar et al. (2007) suggests that maximizing the share of data-ink may not be beneficial to the user experience of the visualization: while, e.g., axis lines in a chart are not data-ink according to the definition of Tufte, they may be beneficial to the user experience of the chart by providing visual structure.

The principles presented above are essential, but too abstract in order to be used as a sole basis for defining a good visualization. However, when combined with Kosara's data visualization definition stated above, the principles become considerably more useful and concrete. Azzam and Evergreen (2013) propose an adapted version of the definition by Kosara (2007): "Data visualization is a process that (a) is based on qualitative or quantitative data and (b) results in an image that is representative of raw data, which is (c) readable by viewers and supports exploration, examination and communication of the data". The most significant differences are that the definition complements the second requirement of Kosara ("It produces an image") by requiring the produced image to represent the data truthfully, and requires the visualization to be enlightening instead of just readable. This definition effectively combines the definition by Kosara (2007) with the principle of showing data introduced by Tufte (1986). The adapted definition facilitates the process of creating a successful data visualizations by offering a more concrete version of Tufte's principles. It gives the developer of the visualization slightly more concrete checklist for representing the data: make sure the representation does not (a) omit or (b) overrepresent any information, and (c) helps the viewer gain knowledge (Azzam and Evergreen, 2013).

For the most concrete principles, Zuk et al. (2006) provide a list of heuristics for visualizations established in perceptual, cognitive and usability research. The list combines several heuristics from multiple heuristics sets by Shneiderman (1996); Zuk and Carpendale (2006); Amar and Stasko (2004). While the combination results in potentially conflicting or redundant heuristics, it nevertheless results in a concrete checklist for making successful visualizations. The heuristics, along with generated identifiers for easier referring, are presented in table 2.1.

Identifier	**Description**
Visual variable	Ensure visual variable has sufficient length
Color order	Don't expect a reading order from color
Color size	Color perception varies with size of colored item
Local contrast	Local contrast affects color
Color blindness	Consider people with color blindness
Preattentive benefits	Preattentive benefits increase with field of view
Size variation	Quantitative assessment requires position or size variation
Graphic dimensionality	Preserve data to graphic dimensionality
Most data	Put the most data in the least space
No extra ink	Remove the extraneous (ink)
Gestalt laws	Consider Gestalt Laws
Levels of detail	Provide multiple levels of detail
Integrate text	Integrate text wherever relevant
Overview first	Provide overview first
Zoom and filter	Zoom and filter out uninteresting data
Details on demand	Provide details on demand
Relate	Consider relationships among items
Extract	Allow extraction of data and its subsets
History	Keep history of actions
Uncertainty	Expose uncertainty
Relationships	Concretize relationships
Domain Parameters	Determination of domain parameters
Multivariate	Provide multivariate explanation
Cause & effect	Formulate cause & effect
Hypotheses	Confirm Hypotheses

Table 2.1: Heuristics presented by Zuk et al. (2006) with generated identifiers.

2.3 Visualizing Geographical Data

As geographic data is associated with a specific location, the most natural way of visualizing it is by using a map (Kraak, 1998; Kraak and Ormeling, 2011, chap. 1). This technique is called *thematic mapping* (Slocum and McMaster, 2014, chap. 1). Thematic mapping does not require any specific format of data, except for the geographical dimension (Kraak and Ormeling, 2011, chap. 1). However, the nature of the data has a great effect on the method, or type, of thematic mapping.

2.3.1 Methods for Thematic Mapping

As stated above, there are several types of geographical data, many of which are fundamentally different requiring different visualization methods. Therefore, several different thematic mapping methods have been developed. Slocum and McMaster (2014, chap. 14-18) list some of the most typical ones:

Choropleth Map Choropleth maps are used primarily for visualizing data which coincides with predefined enumeration units. This method is most naturally used for situations when the enumeration units are directly linked to data results, such as votes in an election for each voting area. However, choropleth maps are also used to depict "typical" values for an area even when in reality, the area is heterogeneous in relation to the measured quality. Choropleth maps are commonly visualized grouping a specified area using a constant color or a common symbol. Figure 2.1 depicts an example of a choropleth map. (Ibid.)

Isarithmic Map Isarithmic maps are map visualizations depicting continuous or smooth phenomena. Therefore, isarithmic maps excel at visualizing natural properties such as elevation. The most commonly used type of isarithmic mapping is contour map which consists of the measured property visualized as gradient colors in addition to *contour lines* used as value symbolization. Figure 2.2 contains an example of the isarithmic mapping method. (Ibid.)

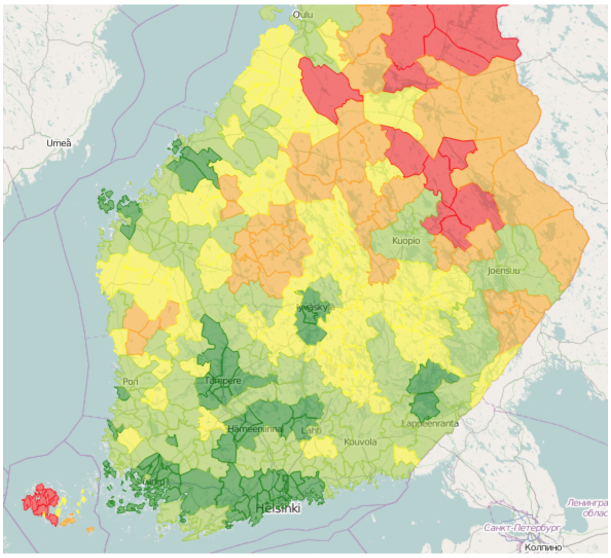

Figure 2.1: A choropleth map depicting regional voter turnout in Finnish presidential elections of 2012

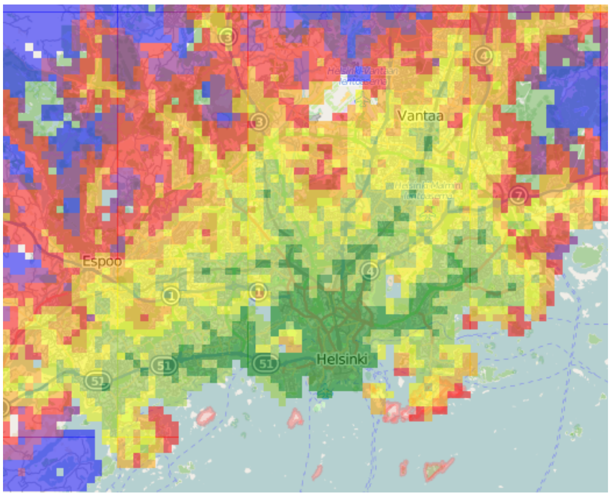

Figure 2.2: An approximated isarithmic map depicting travel times to Helsinki center.

Dasymetric Map Dasymetric mapping is closely related to choropleth mapping, with the exception that in dasymetric mapping, the enumeration units are not predefined, but rather defined by the data coherency. When creating exact dasymetric maps computationally, the properties can be approximated using a number of techniques, as presented in chapter 7.2.2. Dasymetric mapping is most naturally used for data which consists of a set of internally cohesive blocks of area, such as land use (i.e. the distribution of roads, cities, forests etc.) as illustrated in figure 2.3. (Ibid.)

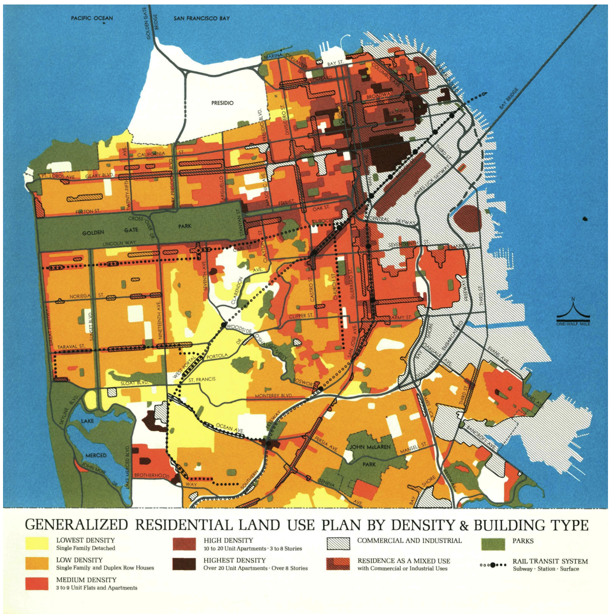

Figure 2.3: Dasymetric map depicting land use density in San Francisco (Fischer, 2012).

Dot Map Dot maps are used to represent data which is associated with locations. With dot maps, the data used can be *true* (truly associated with a single point) or *conceptual* (aggregated to a point). Moreover, dots can be clustered or combined. In practice, dot maps can be used for visualizing, e.g., store locations or the number of homicides in different cities. An example of a dot map is presented in figure 2.4. (Ibid.)

Figure 2.4: Dot map depicting cholera cases during the London epidemic of 1854 (Snow, 1854).

Proportional Symbol Map Proportional symbol maps are closely related to dot maps. They are typically used for visualizing numerical data associated with a location, but unlike with dot maps, the symbols on a map are resized proportionally to the data. Symbols can be geometric or pictorial, and the sizes can be determined using several different methods, e.g., purely mathematical scaling or perceptual scaling which takes human visual inaccuracy into account. An example of a proportional symbol map is presented in figure 2.5. (Ibid.)

Multivariate Mapping Multivariate mapping denotes displaying multiple attributes simultaneously. This can be achieved in several ways. The visualized attributes can be either visualized using a single map or a separate map for each attribute. Additionally, the attributes can be either overlaid (placed on top of each other) or combined (use a single symbol depicting all attributes). (Ibid.) !Fixme **add example image** Fixme!

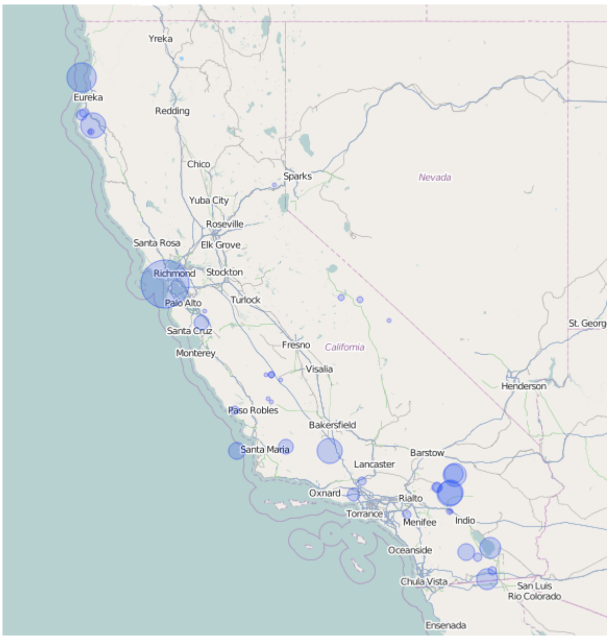

Figure 2.5: Proportional symbol map depicting the magnitude of earthquakes in California.

Cartogram Cartograms are used to distort the map based on the data. Thus, cartograms may be used to communicate relative sizes of an attribute in several areas, such as population in each country. This is advantageous when the geographical sizes and attribute values do not correlate, e.g., when some areas with high attribute value are extremely small in size. An example of a cartogram is depicted in figure 2.6. (Ibid.)

Flow Map Flow maps are maps with lines or arrows of varying width from one location to another. Therefore, flow maps excel at displaying movement-related attributes such as immigration from one country to another or wind speed and direction. An example of the flow map is depicted in figure 2.7. (Ibid.)

Even a single thematic map is often used for multiple different purposes (Schlichtmann, 2002, chap. 2). For instance, a single map can be read on the *overall level* ("where are the primary schools located in Helsinki metropolitan

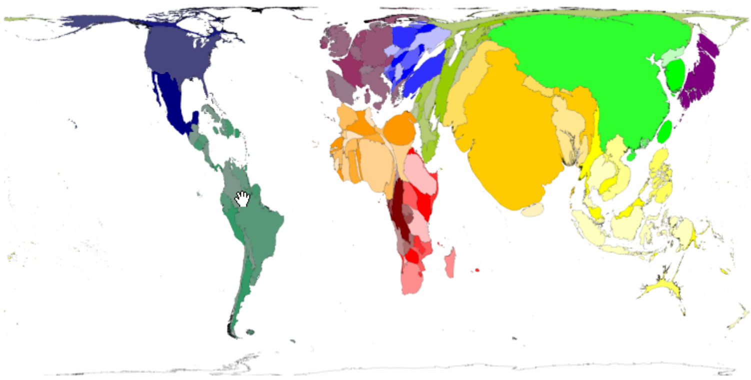

Figure 2.6: Cartogram depicting the population of the world (Hennig, 2014).

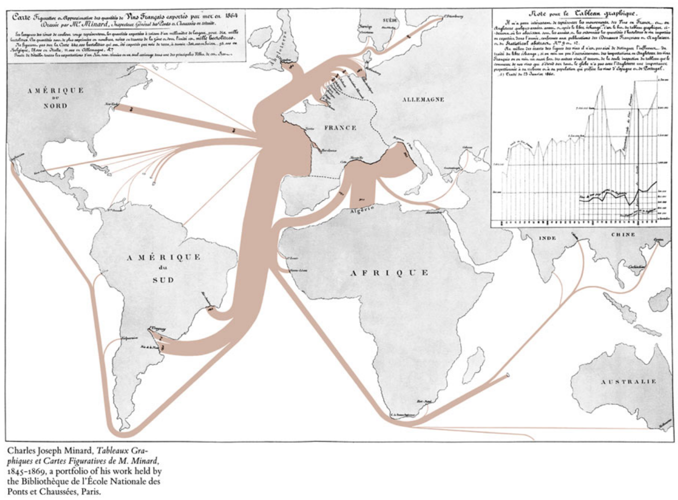

Figure 2.7: Flow map displaying the French wine exports in 1864 (Minard, 1865).

area?") and *elementary level* ("is there a primary school in Punavuori?"). Furthermore, some possible uses for a thematic map are "what is the ratio and distribution of Finnish schools compared to Swedish schools in Helsinki" or "what is the spatial distribution of sizes of schools in Helsinki". Therefore, an efficient map visualization should not lock the user to any single perspective. !Fixme **maybe move somewhere? Create a separate (sub)section for interactivity?** Fixme! !Fixme **Interactivity could help here? See (Andrienko and Andrienko, 1999)** Fixme!

2.3.2 Effective Thematic Maps

While the map visualizations adhere to general visualization principles, the principles can be refined by specifying a set of guidelines for maps specifically. Koeman (1969 quoted by Kraak 1998, p. 12) defines the guidelines for map visualization process as *"How do I say what to whom"*. *How* refers to the mapping methods and techniques used. *What* refers to the data used and its characteristics. *Whom* refers to the target audience of the visualization. Kraak (1998) complements the guidelines with *"and is it effective"*, referring to the self-reflective and iterative nature of visualization.

All the elements above are important to acknowledge when creating a thematic map. While they do not provide an exact formula for determining the effectiveness of a visualization, the elements are incredibly beneficial for creating an effective visualization. Therefore, created visualizations can also be examined with the help of the elements.

!Fixme **Something about thematic map interactivity (Andrienko and Andrienko, 1999; Kraak, 1998, p. 12) could be relevant in this chapter** Fixme!

2.4 How Thematic Maps Are Made

Schlichtmann (2002) describes making thematic maps as a six-step process. The steps are presented below:

1. Decide what is the knowledge the viewer should gain from viewing the visualization
2. Decide on the information to be entered to the visualization
3. Procure the data needed
4. Procure a base with the required geometrical characteristics
5. Select the appropriate graphic means and transcribe the information as necessary
6. Explain the transcription in a legend

Slocum and McMaster (2014, chap. 1) present an alternative process for thematic visualization. The process consists of five steps and is presented below:

1. Consider what the real-world distribution of the phenomenon might look like
2. Determine the purpose of the map and its intended audience
3. Collect data appropriate for the map's purpose
4. Design and construct the map
5. Determine whether users find the map useful and informative

In practice, the processes described by Schlichtmann and Slocum and McMaster concentrate on different perspectives of thematic mapping. Schlichtmann begins the process by defining the goal of the visualization while Slocum and McMaster provides a more data-centric approach starting with phenomenon definition. Unlike Schlichtmann, Slocum and McMaster emphasizes an iterative approach of the visualization. In both processes, defining and designing the visualization is emphasized, the actual visualization (turning the data into visual representation) being addressed in only one of the steps.

Slocum and McMaster (2014, chap. 1) express their concern on the utilization of the processes defined above. According to them, it is likely that naive visualizers do not follow the steps, but take shortcuts when designing the visualizations, resulting in subpar visualizations. Therefore, it is often needed to nudge the visualizers towards using (one of) the processes when building a visualization.

The steps are used to produce a visual representation (graphic) of the data. Additionally, Schlichtmann (2002) identifies several objectives for the resulting graphic, presented in table 2.2.

Name	**Description**
Clarification	Making the map clear and readable. In practice, this means that the topemes (symbols) in a map should be easily detectable and distinguishable from each other
Emphasis	Making topemes and other important characteristics of the visualization to stand out visually
Types of Entries	Having a clearly distinguishable type for each topeme.
Sets of Types	Grouping data points and symbols with similar traits in order to make them belong together visually. Ideally, the visual similarity should be related to the conceptual similarity.
Cross-Relations	Visually indicating the potential relations and similarities between different types or between entries of different types.
Local Syntax	Aligning visual properties of the topemes to prevent unintentional emphasis of single topemes.
Local Ensembles	Supporting topemes with multiple properties (such as the numbers of children and adults in an area) so that the topeme visually reflect both the individual properties and the combination of all properties.
Multilocal Ensembles	Supporting topemes with multiple geographical properties (such as spatial distribution of people)

Name	Description
Addable and Non-Addable Quantities	Differentiating addable and non-addable properties. Typically absolute quantitative properties are addable while relative and qualitative properties are non-addable. Addable properties should be visualized in a way that cognitively supports addition (e.g., with sizes of elements) while non-addable quantities should be visualized without said feature (e.g., with colors.)
The Surface Illusion	Creating an illusion of surface on the map. This can be achieved for example by using illumination and shadowing. These visual traits can convey a meaning themselves and often naturally do so.

Table 2.2: Map visualization objectives as per Schlichtmann (2002)

The objectives above are important when visualizing geographical data on a map. Therefore, it is needed to take those into account when creating a visualization tool in order to enable or even encourage the visualizers to reach as many of the objectives as possible.

Chapter 3

Software Reuse

In order to create a reusable software framework for visualization, it is necessary to study software reuse along with different reuse techniques and their characteristics, advantages and disadvantages.

Krueger (1992) presents software reuse as a process of reusing existing software code (applications, libraries, functions or single lines) when building new software, while according to Mohagheghi and Conradi (2008), reuse is not restricted to code, but can also refer to other software assets such as design. However, both agree that software reuse combines several different existing pieces of code (and possibly other assets) along with new assets which are specific for the application in question. According to Mcilroy (1969) and Boehm (1999), it is one of the most effective techniques of reducing the development time and cost of complex software products.

3.1 Software Reuse Advantages & Disadvantages

When used appropriately, software reuse has several benefits. In their overview of multiple case studies, Mohagheghi and Conradi (2008) discovered that in most cases, using reused software components resulted in a considerably lower number of software defects and better productivity. Several of the studies implied that reusing software is also beneficial for software complexity and product time-to-market. However, it should be noted that since the overview only addresses case studies, its results should not be considered universally applicable.

Although reusing software is often said to decrease the effort needed (Mcilroy, 1969; Boehm, 1999; Mohagheghi and Conradi, 2008), concrete evidence for this is difficult to find (Mohagheghi and Conradi, 2008).

Given its lucrative advantages, software reuse is definitely beneficial for many software systems. However, according to Krueger (1992), software reuse can be problematic and even disadvantageous. Learning to use a specific piece of reusable software often takes considerable effort. Moreover, finding suitable code fragments may also prove to be a challenge. For uncomplicated software systems and especially reusable components, it may not be worth the effort. Therefore, developer needs to carefully consider all sides of reusing when building a software system; according to Krueger (1992, chap. 1.3), for successful software reuse scenario, the amount of intellectual effort between the concept and implementation of the system must be as low as possible. In practice, this means that the value of the reused component must be as high as possible for the developed system, while the implementation cost (resources needed to take the reusable component into use) should be relatively low.

3.2 Factors for Successful Software Reuse

Frakes and Isoda (1994) present six critical factors for successful software reuse: management, measurement, legal, economics, design for reuse, and libraries. Some of the factors are relevant only for a corporate-level reuse program, but many are critical for smaller scale reuse as well.

Successful reuse requires the **management** to commit to a long-term, top-down support, because reusing software may require years to pay off the costs. Also, **measurement** of reuse is vital to reuse software successfully. Both *reuse level* (the ratio of reused software to total software) and *reuse factors* (things affecting the increase of reuse) should be measured.

Legal issues are also important to consider when reusing software. Specifically, the rights and responsibilities of providers and consumers of reusable software should be agreed on. Moreover, using software with conflicting licenses may cause problems.

Economics present a challenge in systematic reuse scenarios. Measuring reuse costs is not straightforward as often costs of creating a reusable component are compensated by benefits in some other project using the component.

In order to **design for reuse**, a degree of domain knowledge is required. This necessitates the study of the domain when creating a *domain-specific* reusable software. In addition to that, reusable software design requires effort on encapsulation, abstraction and interfaces.

Lastly, reusable software **libraries** are required to fully benefit from the reusability effort. Libraries enable storing, retrieving and finding the reusable software.

3.3 Analyzing Software Reuse

According to Krueger (1992), in order to analyze reusing software, the reuse process to be studied should be separated into four *dimensions*. The dimensions are presented below.

Abstraction is the process of making a piece of software more generic, thus making it applicable to a wider range of software projects. Software reuse is almost always based on abstraction, but according to Krueger (1992), raising the abstraction level has proven to be difficult, thus making building reusable software a nontrivial process.

Selection facilitates finding, comparing and choosing suitable pieces of software. For example, libraries or frameworks aid selection by bundling and structuring the software components.

Specialization is the process of making the abstracted component more specific, usually by parameterizing the software or making it transformable.

Integration facilitates providing the software with reusable components, for example with a mechanism to import relevant modules or functions to the software.

3.4 Software Reuse Methods

Software reuse is not a single, uniform procedure or technique. Several different reuse techniques exist to cater different needs. Consequently, different reuse methods excel at different areas. In order to describe the advantages and disadvantages of the methods, we describe the using the reuse dimensions presented in the previous section.

Krueger (1992) and Johnson (1997), among others, present and analyze software reuse methods. From these, we have selected the most relevant for web environment, presenting those below.

3.4.1 High-Level Languages

High-level languages denote programming languages which are designed to be on a high abstraction level and thus contain features which are not necessary for a programming language but benefit or speed up the development. Traditional examples of these kind of features are automatic memory allocation (Krueger, 1992) and language constructs such as exceptions (Mitchell, 2003). More modern high-level language features are value type checking systems and abstracted support for parallel operations using futures (Totoo et al., 2012). It should be noted that the high-levelness of a language is a *relative* property, i.e., it is not possible to determine the requirements for a high-level language per se, only high-levelness of languages compared to other languages. For example, Krueger (1992) considers all programming languages above the abstraction level of the assembly language high-level languages, while Carro et al. (2006) consider e.g., lack of automatic memory management or type system a sign of lower-level language.

High-level languages *abstract* frequently used procedures into seemingly uncomplicated operations, thus reducing the work and cognitive capacity needed for developing the application. (Krueger, 1992, chap. 3)

As the number of elementary high-level language constructs is usually relatively low it is possible for programmers to master the use of those constructs with sufficiently little effort, rendering *selection* unproblematic. (Krueger, 1992, chap. 3)

Specialization of high-level language features is usually achieved by parameterizing the constructs, either implicitly or explicitly. For example, when instantiating a class in Java, the only parameterization needed for memory management is the actual object instance. However, e.g., exception handling always requires at least the logic needed for handling the exception. (Krueger, 1992, chap. 3)

Integration of high-level language features is automatically done when compiling the software code. However, due to the nature of high-level languages, it is usually not possible to mix-and-match different programming languages easily in the same program. (Krueger, 1992, chap. 3)

The advantages of high-level languages are mainly related to the decreased need for developing frequently needed procedures manually, such as allocating or deallocating memory, case-by-case. These operations in high-level languages can be mapped into more complex procedures in some lower-level languages, effectively making them reusable software components. In practice, using high-level languages can yield a productivity gain up to 500 %. (Krueger, 1992, chap. 3)

The main disadvantage of using high-level languages is the potential decrease in performance. As with any software reuse, high-level programming languages abstract the supported procedures by making them more generic. This often leads to additional complexity and unnecessary operations on the compiled program. However, the decrease can often be minimized by using additional compile-time optimizations. (Carro et al., 2006)

On the web, the technologies used on the client-side are inherently fixed to descendants of HyperText Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript (World Wide Web Consortium, 2014, 2011; ECMA, 2011). Therefore, web application languages are relatively high-level by definition. However, it is still possible to raise the abstraction level by using e.g., CoffeeScript (Ashkenas, 2009) instead of JavaScript or LESS (Sellier, 2009) instead of CSS.

3.4.2 Design and Code Scavenging

Design and Code scavenging refers to the technique of scavenging pieces of software *ad hoc* from existing software systems and using the pieces as parts for a new software system (Krueger, 1992, chap. 4). The aim of this technique is to reduce the amount of work needed to build the system. For example, when building an user interface (UI) component for choosing a date, the developer may scavenge the code for a calendar from an older software system.

Scavenging can be done without modifications to the code in the target code base (code scavenging) or by modifying the details of the scavenged code (design scavenging) (Krueger, 1992, chap. 4). The *abstraction* gained by scavenging is therefore mostly informal and in some cases even its existence is questionable (Sametinger, 1997, chap. 3). Usually, there is no "hidden part" of the abstraction but the developer must maintain the functionality of all the code himself (Sametinger, 1997, chap. 3).

Usually there is no formal mechanism or support for *selecting* pieces of software to be scavenged. Therefore, the developer must rely on his memory, experience and word-of-mouth in order to find suitable pieces of software. (Sametinger, 1997, chap. 3)

Specialization is done by manually editing the scavenged source code. While it is often the fastest method of acquiring results, this requires the developer to deeply understand the scavenged implementation. It can also lead to fragmentation and maintainability issues in the future. (Krueger, 1992, chap. 4)

Integration of the scavenged code is done by copying and pasting the code to the target source code file. This may lead to namespace collisions between original and scavenged code which may result in the need for refactoring the code. (Krueger, 1992, chap. 4)

The main advantages of design and code scavenging are the ability to quickly include existing functionality to new software systems (Krueger, 1992, chap. 4). As it is usually not needed to prepare the code to be scavenged before scavenging it, the extent of possible pieces of software is often significantly larger than when using any other reuse method.

However, finding suitable pieces of software for scavenging is hard. Moreover, scavenging pieces of software often does not decrease the *cognitive distance* between the target and implementation of the system. It may also create issues with maintainability of the software. (Krueger, 1992, chap. 4)

3.4.3 Source Code Components

Using source code components is a type of reuse that chooses and uses software components from a component repository (Sametinger, 1997, chap. 3). Software component can be any piece of code, but in practice, components usually consist of one or more functions, modules or classes (Sametinger, 1997, chap. 3). An example of a source code component is a trigonometry module which contains functions for sine, cosine and tangent calculations. When a developer needs to calculate sines in her program, she searches a component repository for trigonometry components and utilizes the component found in her own program (Krueger, 1992, chap. 5).

Ideally, source code components *abstract* the implementation details of the component inside. This means that the required cognitive distance between the concept and the implementation of the software system is lower when using source code components instead of e.g., code scavenging.

In order to be *selectable*, source code components should be accompanied by abstract names (function names) and descriptions of the functionality provided (Krueger, 1992, chap. 5). The names should describe *what* the components does instead of *how* it does it (Krueger, 1992, chap. 5). These names can then be used for reasoning about the purpose of the component and finding the component in source code component repositories – in order to use the component, the developer must be able to find it and to know what it does (Krueger, 1992, chap. 5).

Source code components can be *specialized* by modifying the source code Krueger (1992, chap. 5). However, as this technique yields unwanted consequences explained in the previous section, many components support specialization by parameterization. For example, the programmer could provide the sine function the angle in question. Additionally, when integrating the trigonometry module to her software system, the programmer could specify if the functions should use degrees or radians. In some components, specialization can also be achieved via subclassing (Krueger, 1992, chap. 5).

All modern programming languages support *integration* of reusable source code components written in the same language. Usually, the procedure is a very simple addition of source code files, which requires little to no effort on the programmer side. However, all source code components can't be used in the same program due to conflicts e.g., in naming and value types (Krueger, 1992, chap. 5).

The main advantages of using source code components are the abstraction provided and organized nature of the component repositories. Ideally, the repositories provide a search functionality so that even developers with no previous experience on the component domain can find the components needed. Moreover, the abstraction level and the hiding of implementation details decreases the cognitive distance between the concept and the implementation of the system and reduce the source code needed to be written.

The main disadvantages of the source code components lie in the fact that the functionality must be deliberately designed to support reuse. The abstraction of the components is a major challenge (Krueger, 1992, chap. 5) in designing source code components. Additionally, the component repositories need administration and maintenance.

3.4.4 Software Schemas

!FIXME **Add similarly structured content as in the previous subsubsection. Or maybe drop?** FIXME!

3.4.5 Application Generators

Application generators are usually domain-specific generators which take very high level instructions (specifications) as input and then output significantly lower level software code (implementation) (Cleaveland, 1988; Krueger, 1992, chap. 7). On fundamental level, application generators differ from high-level language compilers mainly by being designed to work on a narrow domain and thus being able to support considerably higher-level instructions

(Krueger, 1992, chap. 7). Unlike source code components, the reused components generated by application generators are usually not encapsulated or separated (Sametinger, 1997, chap. 3).

Application generators *abstract* the concept or specification of the software system, hiding the actual implementation completely from the user of the generator (Cleaveland, 1988). However, in some cases it may be necessary to modify the output of the generator which essentially removes the abstraction.

In principle, *selecting* application generations is moderately easy since the abstraction level of application generators is usually very high, rendering reasoning about the purpose of the generator fairly easy (Krueger, 1992, chap. 7). However, since application generators are usually suited for a very narrow domain, it is usually difficult to find a suitable generator (Krueger, 1992, chap. 7).

Typically, software systems generated with application generators consist of variant and invariant parts (Krueger, 1992, chap. 7). Invariant part is the part of the program which the developer using the generator can't modify. The developer *specializes* the program by modifying the variant part. There are several methods of modifying the variant part. One of the simplest may be straightforward parameterization: the developer chooses the parameters of the system from a predefined set of alternatives. This method makes using the generation extraordinarily easy. However, it also limits the resulting application considerably.

On the other end of the spectrum, the application generator may require the variant parts to be inputted using a domain-specific or generic-purpose programming language. This makes the application generator incredibly versatile, but requires both more domain-specific and programming knowledge.

Typically, application generators generate complete applications which do not require further *integration* (Krueger, 1992, chap. 7). However, occasionally, the resulting applications are not independent per se, but require integration to other systems. This may be an issue since often it is not possible to select the integration interfaces freely, but to use the ones provided by the generator.

One of the main advantages of the application generators is the abstraction they provide. In some cases, the application generators may even require no programming language knowledge as long as the user has relevant domain-specific knowledge (Horowitz et al., 1985). Moreover, application generators excel when there is a need for building multiple similar applications (Krueger, 1992, chap. 7).

However, application generators require an unambiguous mapping between the specifications and implementation details (Krueger, 1992, chap. 7). Moreover, building application generators requires a reliable, generic implementation and user interfaces for developers (Cleaveland, 1988). Therefore, building application generators requires comprehensive domain-specific knowledge in addition to extensive software development expertise.

3.4.6 Design Patterns

!FIXME **Add similarly structured content as in the previous subsubsection** FIXME!

3.4.7 Software Frameworks

Software frameworks are a reuse technique which combines the use of software components and programming patterns (Johnson, 1997). Therefore, it can be argued that software frameworks enable creating reusable software design. Another definition of software frameworks is a collection of consolidated components, i.e., components which share the design, interfaces, and, to some degree, implementations (Johnson, 1997). It should also be noted that software frameworks are typically strictly object-oriented reuse technique (Johnson, 1997).

Largely, software frameworks *abstract* the implementation details the same way that components do, i.e., providing a higher-level interface for the low-level operations. In addition to that, frameworks abstract software *design patterns* used to provide a more complete architecture and functionality.

Selection of frameworks can be regarded as straightforward, since typically, the number of applicable frameworks is considerably smaller than, e.g., the number of applicable source code components. However, as frameworks are by definition more complex than single software components (Johnson, 1997), selecting the right framework of a purpose may be considerably more difficult (Fayad and Hamu, 2000).

Specializing frameworks is greatly dependent on the purpose and design of the framework. As some frameworks are designed as domain-specific (Johnson, 1997), it is typically not needed to specialize the system extensively. According to Brugali et al. (1997), frameworks are usually specialized in an object-oriented fashion: using parameters and subclasses to fine-tune functionality (Brugali et al., 1997).

Typically, software frameworks are *integrated* to other frameworks and to larger software systems. However, as frameworks are generally designed for adaptation instead of integration (Mattsson et al., 1999), this leads to integration problems. Mattsson et al. (1999) describe several framework integration problems, e.g., architecture, design and pattern mismatches. Nonetheless, most of the problems can be overcome by using a number of solutions, such as separating the concerns cleanly and wrapping the functionality to compliant components (Mattsson et al., 1999).

One of the main advantages of frameworks is that they enable a complete, potentially opinionated approach for reusing software while preserving the possibility for customization (Johnson, 1997). The main disadvantages of using software frameworks consist of occasional steep learning curve and challenges in integration (Fayad and Schmidt, 1997).

Chapter 4

Research Gap

Currently, research on geographic or map visualization is abundant !Fixme **Use a bunch of geoviz references to prove this? Or some other way?** Fixme!. Moreover, according to the visualization definition by Kosara (2007), it may be possible to abstract parts of the visualization implementation in order to achieve visualization process which requires less effort and technical knowledge. However, both research about geovisualization-related software reuse and actual reusable geovisualization components are scant. This implies that there is room for improvement in both research and implementation related to geovisualization software.

To address this shortcoming, we decided to attempt creating a reusable geovisualization tool for the web. We also evaluated the tool in order to obtain knowledge about its benefits when compared to building geographic visualizations from the beginning.

!Fixme **IMHO this is a bit too short for a chapter. How to fix?** Fixme!

Chapter 5

Environment

!Fixme **Maybe drop this chapter, or merge a short version of it to implementation** Fixme!

5.1 Web Applications

Describe HTML5, JS and other related technology, because these are really the things that restrict the implementation.

Visualizations need to be implemented using web technology, because it is crucial for distributability and discoverability that the system works in web browsers... !Fixme **continue...** Fixme!

5.2 Geographic Visualization on the Web

Probably should describe the relevant web technology, some needed HTML5/JS features etc. !Fixme **to subsection of related webtech?** Fixme!

5.3 Current State of Building Map Visualizations on the Web

Describe the current libraries, frameworks and procedures used to build map visualizations on the web. Should these be evaluated separately? Use Find-

Booze, Peruskartta and Ottoapp as examples. Also, reason why it is unnecessarily laborious to write map visualizations every time from scratch.

Chapter 6

Methods

6.1 Evaluating Software Reuse Effectiveness

In section 3.2, we concluded that it is critical to analyze and measure software reuse. Several methods for analyzing software reuse exist. Frakes and Terry (1996) present six general types of software reuse analysis models: cost-benefit analysis models, maturity assessment models, amount-of-reuse models, failure modes analysis, reusability assessment models and reuse library metrics.

Cost-benefit analysis models consider both development costs for the reusable component and the reuse productivity and quality benefits. Maturity assessment models categorize the matureness of a systematic software reuse program. Amount of reuse metrics assess the proportion of reused software in the software system created. Software reuse failure modes model introduces a set of reuse failures which are then used to assess a systematic reuse program. Reusability assessment models aim to analyze various software attributes to assess the reusability of a piece of code. Reuse library metrics concentrate on the reusability of software component libraries instead of single components.

Not all the models discussed above are applicable to this type of research. For instance, as maturity assessment models assess a reuse program as a whole instead of single reusable piece of software, it can not be used for this type of work. Moreover, it should be noted that these models are high-level analysis tools which do not take a stance on how to measure the detailed data required by the measurements.

In their review of multiple case studies, Mohagheghi and Conradi (2007) list several methods as alternatives for analyzing software reuse. Of them, notable ones include *controlled experiments*, *case studies*, *surveys* and *experience reports*. However, as the scope of this work does not allow for large sample sizes required by controlled experiment method, it is not a feasible alternative for this study. Moreover, using experience reports requires a considerable experience on creating and using reusable software.

According to Kitchenham and Pickard (1998), one method for conducting a case study is using a *sister project* comparison. In sister project comparison, a minimum of two different, but sufficiently similar software projects observed. In at least one of the projects, a new method is employed, while in at least one project, the old method is in use. All other practices and aspects should be left unchanged. Mohagheghi and Conradi (2007) argue that this kind of comparison is applicable for analyzing software reuse effectiveness for a specific piece of software. The sister project case study is appropriate when no systematic reuse program exists or when there are other barriers impeding the use of models presented by Frakes and Terry (1996). Moreover, it is possible to create the sister project synthetically by building the same kind of application twice, once with and once without the reusable component.

As presented above, in the higher abstraction level, analyzing projects is fairly straightforward. However, the actual low-level measurements for reusing software are much more complex. Mohagheghi and Conradi (2007) argue that measuring software reuse effectiveness precisely is difficult due to a number of factors: 1) Metrics are difficult to validate since there is no universally accepted definition of "quality" in software products, and 2) the productivity of development is difficult to measure and therefore highly subjective, vague or even erroneous metrics are utilized.

Both Frakes and Terry (1996) and Mohagheghi and Conradi (2007) agree that the size of the code needed to be written correlates inversely to the development productivity. Moreover, research by Banker et al. (1993); Gill and Kemerer (1991) show that software code complexity correlates inversely to the software maintenance productivity. Due to the nature of software maintenance and the fact that it is often hard to distinguish small-scale software development and maintenance (Chapin et al., 2001), it is likely that this principle applies to developing new software to some degree as well. Therefore, it seems evident that in order to enable productive use of reusable software, the new code to be written (outside the reusable components) should be made simple and concise.

The conciseness of the code can be measured by several different metrics. According to Fenton and Pfleeger (1998), the number of lines in program source code is only one perspective. It can be complemented by measuring the functionality and complexity of the program.

According to Fenton and Pfleeger (1998), the most commonly used metric for program size is its length, i.e., the number of lines of source code. The metric can be refined by only considering effective lines, ignoring lines consisting of comments and whitespace. However, Fenton and Pfleeger (1998) encourages the use of both effective and physical line counts to determine the size of a program.

The functionality of the program can be determined with several different metrics. Fenton and Pfleeger (1998) presents the function point approach, which uses the number and complexity of external inputs and outputs, object point approach which uses the number of different screens and reports involved in the application, and so-called "bang metrics" which use the total number of primitives in the data-flow diagram of the program. It should be noted that all of these methods are highly subjective and only provide speculative metrics.

Also the complexity of the program can be determined with several different techniques. According to Fenton and Pfleeger (1998), the complexity of the program (solution) should ideally be not higher than the problem complexity. According to them, in the ideal case it is possible to determine the complexity of the program by determining the complexity of the program !FIXME **Fix! Complexity of something else? Consult Fenton. complexity of the problem?** FIXME!. However, this is not usually the case in real-life applications. McCabe (1976) presents a computational approach for determining the complexity of an application implementation. His approach is used by counting the number of cycles in the program flow graph. The advantage of this approach when compared to the method presented by Fenton and Pfleeger (1998) is that it can be used to compute complexity differences of multiple implementations of the problem.

In addition to the metrics presented above Halstead (1977) presents a number of complexity measures. These measures can be used to estimate software size, difficulty level and the needed effort solely based on the code. These measures do not conflict with the metrics presented above, and when determining software reuse effectiveness, the measures – especially needed effort – facilitate the process considerably. !Fixme **Elaborate this. How are these determined? Why are they good?** Fixme!

!Fixme **Use Mendes et al. (2001) for HTML metrics, or argue why it is not relevant. Also Mendes and Kitchenham maybe.** Fixme!

6.2 Evaluating the Effectiveness of a Visualization

We decided to examine whether the reusable visualization tool is advantageous to the effeciveness of the visualization, i.e., if visualizations built with the tool are likely to be more effective in conveying the information than visualizations built without the tool. This can be done with several different methods.

The principles by Tufte (1986), such as data-ink ratio presented in section 2.2 can be used to evaluate the visualization. Another method for evaluation is presented by Azzam and Evergreen (2013). In this method, data representation truthfulness if emphasized. In other words, the visualization is evaluated based on how truthfully the visualization represents the data. Third method for evaluation is presented by Kraak (1998), concentrating on the phrase "*how* do I say *what to whom* and *is it effective*". The phrase encourages the evaluator to evaluate the selected method and its relation to the data and the target audience.

Visualization heuristics by Zuk et al. (2006) presented in section 2.2 and objectives by Schlichtmann (2002) presented in section 2.3.2 can also be used to evaluate the visualization. When compared to methods mentioned in the previous paragraph, there are arguably more concrete and thus enable easier evaluation.

It is notable that none of the methods presented above provide concrete, computable means for determining the effectiveness. Indeed, Kraak (1998) states that evaluating map visualization effectiveness is predominantly done by estimating the visualization subjectively in relation to its context. However, the methods presented above can still be used as a basis when examining the visualizations.

6.3 Research Methods Chosen for the Analysis

For examining the success of the reusable component, we considered both the methods presented by Frakes and Terry (1996) and Mohagheghi and Conradi (2007). As stated in the previous section, several of the methods are only applicable for assessing large-scale reuse programs or cases with large sample size and therefore were not considered for this work. Of the remaining methods, a cost-benefit analysis was selected. For studying costs and benefits of the reuse, a case study with sister project comparison approach was deemed most applicable.

In order to measure the effort needed as reliably as possible, we decided to use a diverse sets of different metrics. Software size and complexity metrics by Fenton and Pfleeger (1998) were used, with the exception that measuring software complexity is done with the method by McCabe (1976) as it allows comparing different implementations of similar applications. The metrics were complemented with Halstead measurements for difficulty and effort.

For examining the visualization effectiveness, all the methods presented above could be used in combination. However, to keep the scope of this work manageable, we decided to concentrate on the concrete data visualization heuristics by Zuk et al. (2006), complementing the evaluation with geovisualization-specific perspective by using thematic mapping objectives by Schlichtmann (2002). The visualization effectiveness cannot be evaluated by comparing the implementation to sister projects, because the resulting visualizations are planned to be as similar as possible. Therefore, we decided to evaluate the implementation qualitatively by examining whether the implementation benefits the visualizer in terms of conforming to the heuristics and achieving the objectives.

For gathering data, we implemented several typical map visualizations with and without the framework. The types of visualizations were selected to obtain data about a wide variety of different map visualizations while still concentrating to the most frequently used methods.

Chapter 7

Thematic.js

!FIXME **...or "Implementation"?** FIXME!

As discussed in the chapter 3, reusing software typically leads to increased productivity and better quality. Therefore, to achieve the targets of this thesis, we decided to implement a reusable visualization tool. As the tool is designed to benefit building geographic visualizations, or thematic maps, on the web, we decided to name the tool Thematic.js.

7.1 Problem Setting

Currently, when building a visualization for geographical data, it is unnecessarily laborious to develop the visualization from the beginning using low abstraction level APIs provided by mapping libraries. This leads to the situation when using especially more complicated visualization methods such as isarithmic maps, it is not feasible to create an effective visualization, encouraging to use a simpler, yet more ineffective methods such as dot maps.

Second, when building web-based geographical visualizations, it is typically needed to build the whole visualization architecture using web technology such as HTML (World Wide Web Consortium, 2014) and JavaScript (ECMA, 2011). Therefore, an astonishing amount of knowledge of such technology is required to even develop a simple map visualization.

!FIXME **Do these require some concrete or literature proof?** FIXME!

7.2 Application Requirements and Design

We started the implementation process by analyzing the requirements of the different geographic visualization methods presented in chapter 2.3.1. Specifically, we analyzed the underlying structure of the visualizations in order to abstract the applicable parts as reusable components. For this, we adopted the "hot spot" method by Schmid (1997) for detecting similarities and dissimilarities in software.

In order to solve both problems presented in the previous section, we decided to implement a dual approach for visualization. First, as one of the problems related to visualizations is the amount of application architecture work needed, the tool should provide a so-called whole-page scaffold architecture (Jazayeri, 2007) which contains the needed page-specific architecture. We call this part of the system *framework*. Second, we decided to implement an independent visualization component in order to support embedding the visualizations in existing web pages. We call this part *library*. When referring to both of the parts of the system, we use the term *application*.

7.2.1 Reuse Methods

We gathered the analyzed data about the requirements in addition to problems discussed in the previous chapter, and determined the forms of reuse applicable in this case. Since the techniques are not mutually exclusive and each has its own benefits, we decided to use a combination of multiple techniques.

The application was developed, and can be used with, the JavaScript language, which is relatively high-level programming language. The scaffold architecture uses the framework method for enabling the visualizer to get started with the visualization quickly while allowing thorough customization later if needed. The visualization library is built as a collection of software components, which allow versatile functionality and composability while abstracting the implementation details. Moreover, the tool contains example visualizations which can be used as a starting point for building visualizations using the design and code scavenging method.

!FIXME **Maybe why we dropped the rest of the methods?** FIXME!

7.2.2 Supported visualization methods

As discussed in the chapter 2.3.1, several different thematic mapping methods exist. As some of these methods are fundamentally different in implementation, it was needed to explicitly consider the requirements of each method. It was also necessary to decide whether to implement support for each method, as support for some of the methods might have been needed to be dropped in order to manage the application complexity and the scope of this work. In order to support the most frequent use cases, we decided to implement support for the following visualization methods:

- Choropleth maps
- Dasymetric maps
- Isarithmic maps
- Dot maps
- Proportional Symbol maps

Of the visualization methods presented in the chapter 2.3.1, we decided to exclude explicit multivariate maps, cartograms and flow maps. The decision was made primarily due to the fact that of the methods presented in chapter 2.3.1, these are the least frequently used. Moreover, since there are several fundamentally different design options for those methods, it is considerably more difficult to abstract the implementation details to provide a general-purpose visualization module. However, it should be noted that the modular architecture of the application enables easy extendability to support these type of visualizations in the future. Additionally, multivariate maps can be achieved to some degree by using several of the map methods simultaneously.

7.3 Application Architecture

In the highest level, the application architecture consists of two parts: the visualization framework and the visualization library. While the framework uses the library for visualization, it also consists of other functionality and the library can be used separately of the framework. The architecture of the framework is presented in figure 7.1.

Framework consists of a Single-Page Web Application (SPA) which embeds the visualization library along with other functionality necessary or beneficial for user experience. Notably, the application uses CSS for displaying the page correctly and HTML5 Application Cache[1] for offline availability and faster loading times. !FIXME **Any other?** FIXME! Application also contains functionality for displaying the visualization correctly on devices of different sizes and capabilities.

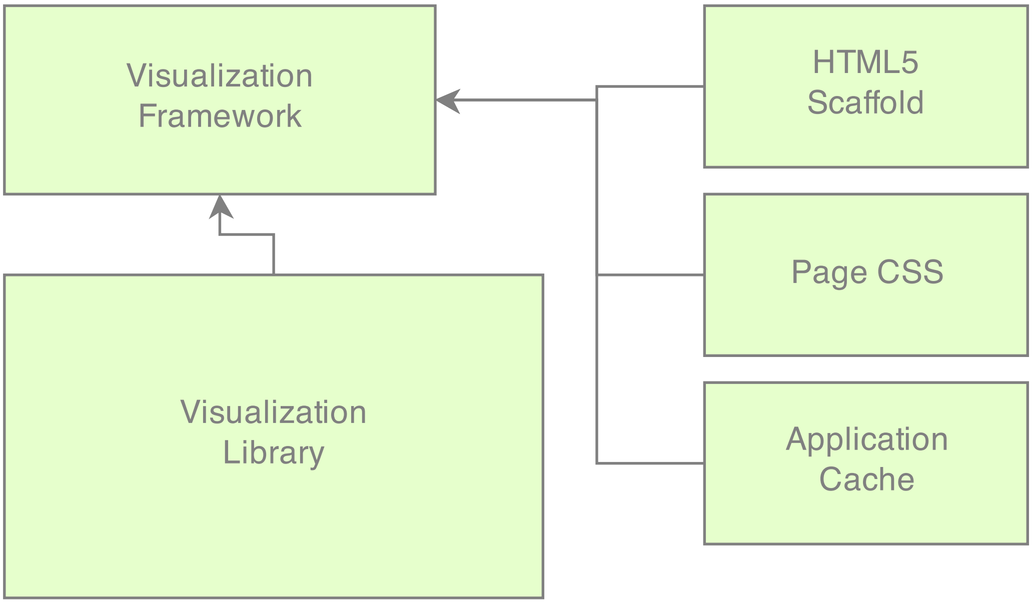

Figure 7.1: The architecture of the Thematic.js framework.

The architecture of the library is described in figure 7.2. The library consists of a map component, mapping modules, data aggregators and data converters. The map component is used for displaying the map layer and for managing mapping modules. The component can be added to any block-level element on a web page. Internally, the map is displayed using Leaflet[2].

Individual mapping modules are added to the map as Leaflet layers. For convenience and maintainability, we created an abstract mapping module for

[1] `http://diveintohtml5.info/offline.html`
[2] `http://leafletjs.com/`

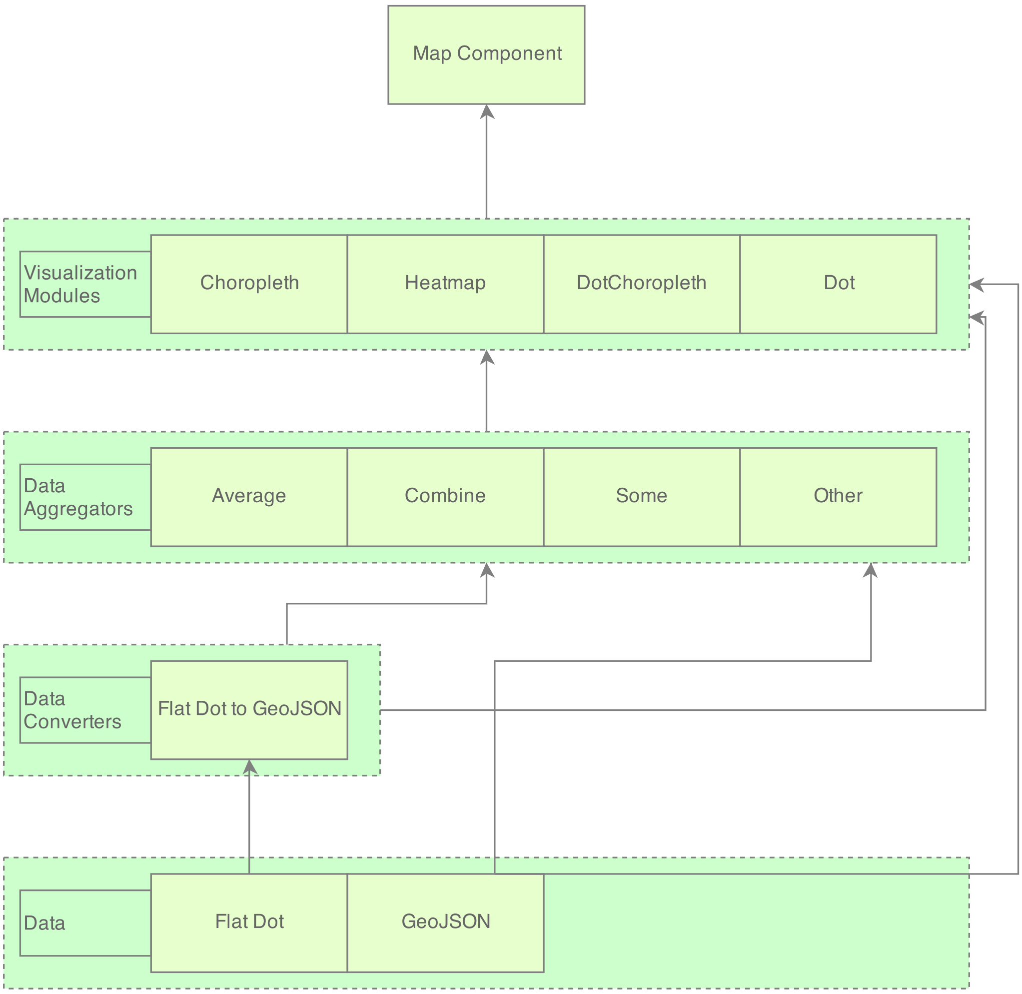

Figure 7.2: The architecture of the Thematic.js library. Arrows denote the flow of data.

handling system-specific procedures such as keeping module status. The abstract module should be used as an object prototype for all mapping modules. Individual mapping modules each support a mapping method, but all can be customized to a degree. Currently, the modules only support GeoJSON[3] data, but there are no technical restrictions on using any format of data.

Typically, mapping modules are used with some external data. The data is often in a non-standard format as presented in section 8.1. For this, data converters are used. Data converters are straightforward stateless components which transform data from a non-standard format to format supported by the mapping modules. For example, the library provides a converter from "flat dot" JSON format (see appendix A) to standard GeoJSON FeatureCollections.

Aggregators are similar to converters in the sense that both transform data. However, while converters do 1-to-1 conversions, aggregators combine several grouped data sets into one by, e.g., calculating an average of the values in sets.

The use of converters and aggregators is not required when creating a visualizations if the data is in the correct format, but the components help bring a structured way of transforming the data into appropriate format.

7.4 Supported Platforms

The application is developed using standard web technology. Theoretically, this means that the app supports all HTML5, CSS3 and ECMAScript 6 compliant browsers. However, in practice, none of the widely used browsers support the standards completely (Manian et al., 2011). Therefore, we have tested the application on the most widely used modern web browsers, i.e., the latest versions of Google Chrome (38), Mozilla Firefox (33), Safari (7.1), Opera (25), Internet Explorer (11), Mobile Safari (8) and Mobile Chrome (38). In total, this represents the browsers of NN % of the Internet users (StatCounter, 2014). !FIXME **Check the browsers and versions actually supported and the resulting percentage** FIXME!.

[3] Geographic JavaScript Object Notation, `http://www.geojson.org`

While the application is most naturally run in a web environment, it is also possible to embed the system to various native applications using a web view component. Web view components are available on at least Windows (Small, 2012), Mac OS X (Hunter, 2014), Android (Google, 2014) and iOS (Apple, 2014) platforms.

Due to the dual approach described in chapter 7.2, the application can be used in almost any existing web application, using almost any framework and library. However, due to a possibility of a namespace collision, other libraries using global namespaces `L` (Leaflet), `_` (underscore), or `thematic` may cause an incompatibility with the application as described in Osmani (2011).

7.5 Implemented Functionality

The following sections present the most important application functionality, namely supported mapping methods, managing input formats and values, and modularity for supporting future extensions.

7.5.1 Choropleth Maps

Choropleth maps are used for visualizing enumerated or areally aggregated data (Dent et al., 2008, chap. 6). According to Slocum and McMaster (2014, chap. 14), it is the most frequently used mapping method. Therefore, implementing choropleth mapping functionality is essential for a successful mapping tool.

To use choropleth mapping, visualizer uses the choropleth mapping module of the application. If the user already has the relevant data in GeoJSON format, nothing else is required. However, if the relevant data is stored separately of the designated area definitions, the user needs to use the *combine* aggregator provided by the application. The aggregator associates the data with the area definition in question and outputs the data in GeoJSON format supported by the mapping module.

!FIXME **add screenshot of a choropleth map** FIXME!

7.5.2 Dasymetric Maps

Dasymetric maps are supported in an approximated fashion: the dasymetric mapping module approximates dasymetric data by using the floating grid method as presented by Langford and Unwin (1994). The module is provided the grid of dots in GeoJSON format as data, and the data is used to generate appropriate dasymetric map approximation. The data can also be aggregated and converted using any of the supplied aggregators and converters.

!FIXME **add screenshot of a dasymetric map** FIXME!

7.5.3 Isarithmic Maps

The application provides two isarithmic mapping modules: the Isarithmic module enables approximated isarithmic mapping while the Heatmap module enables heat map visualizations. Heatmap method is more accurate than the approximated isarithmic method, but requires a dot-like data set in order to provide best results. The approximated isarithmic module employs the floating grid method of Langford and Unwin (1994).

!FIXME **add screenshot of an isarithmic map** FIXME!

7.5.4 Dot Maps and Proportional Symbol Maps

The application supports producing dot and proportional symbol maps by providing the Symbol mapping module. With default configuration, the module produces dot maps, but it is possible to provide an option for calculating and using proportional symbol values. For enabling the user getting more information about data points, it is possible enable information bubbles which are activated by click the symbol. The module also supports using customized symbols for data points, the default being a simple Leaflet marker.

!FIXME **add screenshot of a dot map** FIXME!

7.5.5 Input Formats

All implemented mapping modules use GeoJSON as their input format. GeoJSON is the *de facto* format for transmitting geographical data on the web

(Bostock and Davies, 2013). There is also great support for GeoJSON data in the existing software, for example in the Leaflet map library used by the application. However, due to its verboseness, GeoJSON may be unsuitable for simpler data sets and visualizations such as dot maps. Therefore, we have implemented a number of converters for transforming data to GeoJSON format. Currently, there is support for converting "flat dot" (see appendix A) and X formats !FIXME **check which formats are supported.** FIXME!

Typically, the data is fetched from an external resource (external API or a separate JSON file) asynchronously. Therefore, the modules support using ECMAScript Promises[4] to pass visualized data. However, also synchronous data (such as using data defined in the source code file) is supported by wrapping the values in Promise objects.

7.5.6 Value Normalization

When visualizing a metric such as average temperature of an area, the scale of values is completely different from when visualizing, say, population density. Therefore, in order to provide general-purpose map visualization tools, it is necessary to support displaying a wide variety of values and scales.

For this application, we decided to implement a highly versatile normalization functionality which allows the visualizer to work on virtually any scale. Instead of transforming the input values into a predefined value set, the application transforms the input values directly to a visualizable value, such as "red" on a choropleth map, or "10px" on a proportional symbol map. Moreover, this mechanism is compatible with, e.g., scaling functionality of D3.js[5] visualization library, so the visualizer can leverage the sophisticated scaling functionality of external libraries.

7.5.7 Modularity and Extendability

It is hardly possible to cover the whole area of geographic visualizations. Therefore, instead of trying to support every visualization method possible,

[4] `https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Promise`

[5] `http://d3js.org/`, `https://github.com/mbostock/d3/wiki/Scales`

we implemented the architecture of the application so that it is as straightforward as possible to extend the functionality.

As a result of this, visualization methods can be easily extended by adding tailored mapping modules to the application. Additionally, it is possible to create customized aggregators, converters and scales and bundle these as an extension to the application.

!FIXME **Would the implementation chapter need more practical examples, be it code or screenshots?** FIXME!

Chapter 8

Evaluation

In this chapter, we describe the evaluation of the tool built. The evaluation is performed by using two separate methods: we evaluate the efficiency of development process with the software effort and complexity metrics presented in chapter 6, and visualization effectiveness with heuristics by Zuk et al. (2006) complemented by mapping objectives by Schlichtmann (2002) presented in chapter 2.2. As the first of the methods requires a baseline project, we decided to implement a number of sister projects as defined by Kitchenham and Pickard (1998). In this chapter, the visualizations built during sister projects are referred to with the term “reference visualization” while the visualizations built with Thematic.js are referred to with “Thematic.js visualization”. !FIXME **Are these terms clear enough?** FIXME!

8.1 Defining the Evaluated Cases

The visualization tool should be able to visualize a large variety of data. Moreover, the benefits of reusable software are typically emphasized when examining a large number of relatively similar cases (Frakes and Terry, 1996). However, in order to keep the scope of this work manageable, we decided to evaluate a set of visualization cases listed below.

Alko stores in Finland Alko provides an unsupported representational state transfer (REST) API[1] for fetching data of Alko stores. The data is in a non-standard "flat dot" format (see appendix A). Therefore, we decided to visualize Alko store locations using a dot map. The map should display all Alko stores in an effective fashion, with clustering support for markers in order to avoid map cluttering. This case is later referred to as "store map".

Earthquakes in California Earthquakes have two fundamental data axes: location and magnitude. Therefore, earthquakes are best visualized using a proportional symbol map with the size of the symbol representing magnitude. United States Geological Survey provides historical earthquake data[2], and we decided to visualize earthquakes in the state of California since January 1, 1900. The data is available in a Comma-Separated Values (CSV) format which is can be trivially transformed to "flat dot" JSON format. This case is later referred to as "earthquake map".

Circulation of the Biggest Finnish Newspapers !FIXME **Describe this** FIXME!

Voter Turnout in Finnish Presidential Election of 2012 The Finnish Ministry of Interior provides regional voter turnout data of the presidential election of 2012[3]. This data is provided in electoral district and municipality level. We decided to visualize the turnout in municipality level, using municipality data by the Finnish Land Survey[4]. The municipality data is provided in GeoJSON format by Teemu Tiilikainen[5]. These data can be combined to create an effective choropleth visualization of regional turnout. The data should be normalized in quantized fashion, i.e., using thresholds to create a discrete color range. This case is later referred to as "election map".

[1] `http://www.alko.fi/api/store/mapmarkers?language=fi`
[2] `http://earthquake.usgs.gov/earthquakes/search/`
[3] `http://tulospalvelu.vaalit.fi/TP2012K2/s/aanaktiivisuus/aanestys1.htm`
[4] `http://www.maanmittauslaitos.fi/en/opendata`
[5] `https://github.com/varmais/maakunnat`

Share of People with No Secondary Education in Finland Statistics Finland[6] provides provincial data on the education of the population of Finland in CSV format. This can be combined with province data by the Finnish Land Survey[7] to create an effective choropleth visualization. The data should be normalized in linear fashion, i.e., using a continuous color range. This case is later referred to as “education map”.

Travel Times to a Single Destination Travel times to a destination can be visualized using an isarithmic map. We decided to visualize travel times to Futurice headquarters[8] using public transport. The travel times can be obtained by using Travel Time Visualization Utility for HSL Reittiopas[9] which provides the data in an approximated “flat dot” format. The data should be normalized in a quantized fashion to emphasize isarithmic contours. This case is later referred to as “simple travel times map”.

Travel Times to Multiple Destinations In addition to visualizing travel times to a single destination, we decided to evaluate a case for displaying travel times to multiple destinations. The travel times are obtained with the method defined in the previous paragraph, and combined using a weighted average method. Like in the previous case, the data should be normalized in a quantized fashion. This case is later referred to as “complex travel times map”.

While the visualized data is arbitrarily selected, the cases are picked to reflect the typical usage of visualizations. Choropleth map and isarithmic map are the most frequenly used thematic mapping methods (Slocum and McMaster, 2014, chap. 14-15) and therefore it is beneficial to the evaluation to examine multiple visualizations with those methods.

In order to better model typical real-life use cases, and to be usable on the web, the visualization cases include also a generic application structure

[6] `http://www.tilastokeskus.fi/`
[7] `http://www.maanmittauslaitos.fi/en/opendata`
[8] `http://futurice.com/contact#helsinki`
[9] `https://github.com/pyryk/reittiopas-travel-times`

and HTML features such as application caching and bookmarking support which are required for web applications.

8.2 Implementing Sister Projects

We implemented seven separate sister project visualizations with no visualization library to compare to visualization cases as defined in the previous section. The functionality of the visualizations was designed to reflect the functionality of the evaluated visualizations as accurately as possible. Sister visualizations were implemented using HTML, CSS and JavaScript to enable straightforward comparison to the evaluated visualization cases. While we did not use any visualization library for the sister projects, we deemed using a generic mapping library such as Leaflet.js appropriate, because typically, creating map visualizations is not feasible without using one. Moreover, also Thematic.js uses Leaflet.js as a mapping library.

In order to better reflect the actual situations involving building visualizations, the sister projects were implemented in *ad hoc* fashion, meaning that the design or architecture of the applications were not planned extensively beforehand. Also, no reuse of any form between visualizations was planned. However, during implementation, some design and code scavenging was done in order to speed up the development process. !Fixme **The sister project code can be found in ...** Fixme!

8.3 Evaluating Efficiency of Development

We evaluated the efficiency of development by several metrics: software code length (number of physical (LOC) and logical (LLOC) lines of code), cyclomatic complexity (CC), Halstead difficulty (HD) and Halstead effort (HE). For measurements, we used ESComplex[10] for analyzing JavaScript programs. !Fixme **What about HTML and CSS? Any way to measure these? Or is the effort for these considered equal in sister projects?** Fixme!

We began the evaluation by measuring the aforementioned metrics for

[10] `https://github.com/philbooth/escomplex`

the visualizations. It should be noted that for these measurements, we did not include code from Thematic.js or other third party libraries. The measurements are shown in table 8.1.

Visualization	**LOC**	**LLOC**	**CC**	**HD**	**HE**
Thematic.js store	12	12	1	7.31	4600
Reference store	126	85	10	23.6	96500
Thematic.js earthquake	15	14	1	8.38	6280
Reference earthquake	79	121	10	23.5	87400
Thematic.js election	26	29	2	10.9	12800
Reference election	141	101	14	23.8	107000
Thematic.js education	17	17	1	10.2	10200
Reference education	129	87	10	27.8	109000
Thematic.js travel times simple	27	28	2	10.8	9840
Reference travel times simple	184	129	12	29.4	182000
Thematic.js travel times complex	30	30	2	11.6	13300
Reference travel times complex	193	144	12	34.5	255000

Table 8.1: Measurements for developed visualizations, including only visualization-specific code. !FIXME **Add the last missing visualization** FIXME!

According to the results, using Thematic.js yields significantly lower complexity, difficulty and effort values when compared to using no visualization library. This is likely a direct result of Thematic.js providing an extensive map-specific visualization functionality, allowing the visualizer to concentrate on the visualized data. In practice, this means that when creating map visualizations, it is significantly more effective to use a library such as Thematic.js than to write the visualization from the ground up, given that the visualizer possesses – or is able to achieve – a general knowledge of the library functionality.

However, it is likely that the results do not describe the most typical real-life scenarios completely accurately. It can be assumed that typically, visualizers do not possess knowledge of Thematic.js functionality beforehand

and therefore effort for each line of code is considerably higher than when building the visualization from the ground up. In the results, this is reflected in rather high values for relative difficulty for Thematic.js visualizations as seen in figure 8.1.

!FIXME **Qualitative analysis - don't repeat yourself etc.** FIXME!

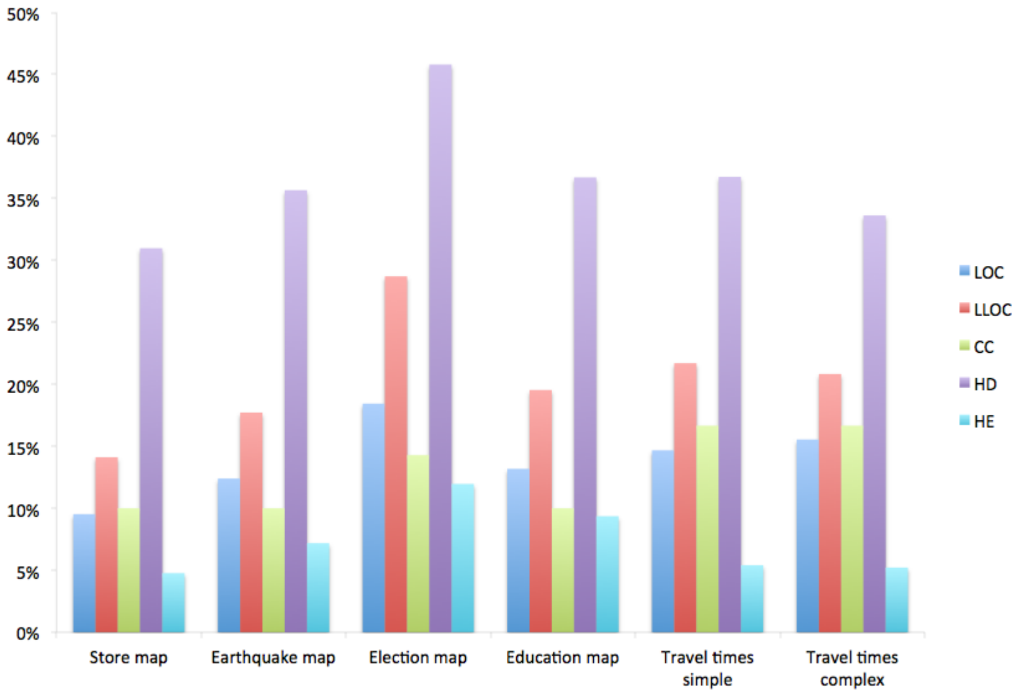

Figure 8.1: Thematic.js visualization metrics for each case in relation to a corresponding reference visualization. !FIXME **Is this the correct way of saying "metric value in Thematic.js visualization / metric value in reference visualization"?** FIXME! !FIXME **Polish the chart** FIXME!

Figure 8.1 displays the ratio of Thematic.js visualization metrics for each visualization case to reference visualizations. The resulting metrics provide a good overview of the effort needed for Thematic.js visualizations compared to the using no visualization library.

Surprisingly, according to the results, using Thematic.js yields relatively high benefits for all metrics for the dot map visualization (store map). The reason for this may be that, even though dot map is an inherently simple visualization method, e.g., error handling and marker clustering support add complexity to the reference implementation. Moreover, as the Thematic.js visualization implementation for dot map visualization consists of only 12 lines of relatively simple code, the boilerplate code needed in the reference implementation causes the relative metrics to be notably low.

Relative metrics of proportional symbol (earthquake) map are approximately similar to the metrics of dot map. However, as the proportional symbol map is missing clustering support but involves symbol scaling functionality which is also needed in the Thematic.js implementation, the relative metrics are slightly more favorable to the reference implementation than in the dot map case. Nevertheless, also in proportional symbol case, using Thematic.js yields significant benefits compared to using no visualization framework, with Halstead effort measurement in Thematic.js implementation being under 10 % of the corresponding measurement in the reference implementation.

!FIXME **Add newspaper map** FIXME!

Election map case yields the least benefits of all the evaluated cases, with Halstead difficulty being almost 50 % and Halstead effort over 10 % of the values for the reference implementation. This is likely due to the fact that the Leaflet.js mapping library provides a comprehensive GeoJSON polygon support used with choropleth visualizations. Therefore, the additional manual implementation needed for the reference case is relatively straightforward, the most effort needed being related to the coloring and showing values, the features that need to be implemented manually also in the Thematic.js implementation. That being said, even a simple choropleth map like this benefits considerably from the Thematic.js library.

In the education map case, using Thematic.js yields slightly greater benefits than in the election map. Like the election map, the education map uses choropleth mapping technique for visualization. However, the data in this case is already combined in GeoJSON format, so no manual combining is required. In both Thematic.js and reference implementations, an external coloring functionality is used, which reduces the size of the code bases. Especially the use of external coloring functionality reduces the Thematic.js relative complexity considerably, which leads to the more significant gains when using the library than in the election map case.

Both travel time maps yield highly similar results in measurements. This is likely due to the fact that both maps use isarithmic mapping method with relatively similar data, the only difference being the need for combining several data sources in the complex case. The data combining likely results in slightly lower values in Halstead difficulty and effort. However, in these cases, the Halstead effort metric is notably low, indicating that the effort for visualizing with Thematic.js taking only 5 % of the effort of the reference implementation. This is probably due to the fact that even approximated isarithmic visualization requires a complex graphics implementation not supported directly by any mapping library.

Across all cases, the Thematic.js measurements indicate more significant differences in physical lines of code than logical lines of code. This is likely due to the fact that Thematic.js API is designed to encourage functional-style, chainable operations while traditional JavaScript APIs typically are imperative and non-chainable. This is demonstrated in listing 8.1. The chainable version can be used without line breaks, resulting in only 1 physical line of code, while with the API format in second example, it is not customary to combine the lines using only source code line. However, in practice, this has little effect on the actual effort needed as the underlying functionality stays largely similar.

```
// chainable API supported by Thematic.js
map.addModule('voting', new Choropleth('percentage')
        .setScale(scale)
        .setData(data));

// non-chainable API typical for traditional
// JavaScript libraries
var module = new Choropleth('percentage');
module.setScale(scale);
module.setData(data);
```

```
map.addModule('voting', module);
```

Listing 8.1: Thematic.js API format. The code has been simplified to increase readability.

Additionally, in all the cases, Halstead difficulty measurements in Thematic.js implementations are 30 to 50 % of the reference implementations while corresponding Halstead effort measurements are 4 to 12 % of the reference implementations. This reflects the fact that the reference implementations consist largely of straightforward but laborious boilerplate code such as initializing the map. Thematic.js implementations consist mostly of data-specific initialization of the visualization, which is typically less straightforward but considerably more concise.

Lastly, it should be noted that while the measurements are suitable for comparing different cases, as absolute metrics they are approximate at best. In practice, this means that it is not sensible to assume that using Thematic.js reduces the effort needed to 10 % of the original. However, in light of these results, it seems extremely likely that using the library for visualizations similar to the evaluated cases yields considerable benefits over building the visualizations from the ground up. For more details about the measurements, see appendix B.

!FIXME **Should we try the evaluation with library code as part of Thematic.js numbers? If yes, add this here. If not, discuss this in discussion.** FIXME!

8.4 Evaluating Effectiveness of Visualizations

In order to allow as reliable effort comparison as possible, we decided to implement the same functionality to reference visualizations as in Thematic.js visualizations. In practice, this results in the reference visualization being as similar feature-wise and visually to the Thematic.js visualization as possible. Therefore, it is not reasonable to compare the effectiveness of the corresponding reference and Thematic.js visualizations. Instead, we decided to evaluate the Thematic.js visualizations qualitatively, concentrating on how the library encourages the visualizer to create effective visualizations.

8.4.1 Visualization Heuristics

Zuk et al. (2006) provide a list of heuristics for data visualizations. These heuristics are described in more detail in section 2.2. We decided to use the heuristics as a basis for evaluating the created visualizations and Thematic.js functionality. We evaluated Thematic.js using a three-step scale: positive if the system has a positive effect (encourages creating effective visualizations) when compared to using no visualization library, neutral if the system has no effect, and negative if the system encourages creating ineffective visualizations. The evaluation results are outlined here. The full results, along with the reasoning, can be seen in appendix C.

Almost all heuristics yield a nonnegative result. According to the results, Thematic.js encourages the visualizer to conform to the heuristic in 12 of the 25 cases, such as preserving action history and displaying details of the data on demand using popups. Using Thematic.js has no effect on conforming to the heuristic in another 12 cases. Only one case was deemed negative: in some cases, Thematic.js encourages the visualizer to increase graphical dimensionality by visualizing scalar data in a non-scalar fashion, such as when using the proportional symbol method. The overview of the heuristics evaluation can be seen in table 8.2. !Fixme **Any insight on why the results are like this?** Fixme!

The heuristics evaluation indicates that using Thematic.js may be beneficial to the effectiveness of the visualization. However, this is likely dependent on the visualizer. An experienced visualizer will probably conform to the heuristics as well or even better without the library. However, for inexperienced visualizers, using the library is likely beneficial for building effective visualizations.

	Positive	**Neutral**	**Negative**	**Total**
Count	12	12	1	25

Table 8.2: The overview of evaluation based on heuristics presented by Zuk et al. (2006).

8.4.2 Thematic Mapping Objectives

Schlichtmann (2002) provide a list of objectives for thematic mapping. These objectives are covered in more detail in section 2.4. We evaluated the Thematic.js library by examining whether the library encourages the visualizer to achieve the objectives or not. Like in the previous section with heuristics, we employed a three-step scale for evaluation. Result for each objective is regarded as *positive* if the library encourages achieving the objectives better than typical non-visualization mapping library does. Result is regarded as *neutral* if using Thematic.js has no effect on achieving the objective, and *negative* if Thematic.js discourages the visualizer to achieve the objective. The results are presented in full detail in appendix D, with overview below.

Table 8.3 displays the number of positive, neutral and negative results related to the objectives. Most of the results are regarded as neutral. This is likely due to the fact that many mapping libraries, such as Leaflet.js, already provide satisfactory level of support for many of the objectives. Therefore, using Thematic.js provides no additional benefit related to these objectives. Nevertheless, Thematic.js achieves a positive result for 4 out of 10 objectives. These are mostly due to providing explicit support for features encouraging effective visualizations, such as defining different symbols for different topeme types. It is also notable that none of the results are regarded as negative.

	Positive	**Neutral**	**Negative**	**Total**
Count	4	6	0	10

Table 8.3: The overview of evaluation based on objectives presented by Schlichtmann (2002).

The evaluation results hint that using Thematic.js may be beneficial to the effectiveness of resulting visualizations. However, as with the heuristics results, this does not imply that Thematic.js is beneficial for every visualizer. An experienced visualizer may not benefit from the library in terms of effectiveness. However, especially for less experienced visualizers who might not recognize the objectives by heart, Thematic.js is probably beneficial.

Chapter 9

Discussion

In this chapter, we discuss the validity of the thesis along with its potential shortcomings. By internal validity, we mean the validity and appropriateness of the used methods for evaluating the use cases. By external validity, we mean the generalizability of the results, i.e., whether the evaluation results can be generalized for other use cases. We also define a number of relevant aspects not covered by this research to be further studied in the future.

9.1 Interpretation of Results

- What has been found? (A library for building visualizations can be created?)
- Interpretation of the findings: it makes sense to use the reusable thingy in certain situations
- Are the findings in line with the literature? (Yes, the literature also hints that the findings are appropriate) !Fixme **How about literature about effective visualizations? Does using a library benefit the effectiveness? Maybe use some GIS studies related to the effectiveness?** Fixme!

9.2 Internal Validity of the Study

According to most of the literature presented in chapter 3, measuring software reuse is extraordinarily difficult. We identified several aspects potentially hindering the reusability evaluation and its validity.

The evaluation metrics used assume that the visualizer is equally acquainted with all the libraries and APIs used. However, in practice, this is unlikely. Typically, visualizers creating web geovisualizations possess at least an elementary knowledge of JavaScript APIs. Some visualizers also have experience on using Leaflet or other mapping libraries. On the contrary, it can safely be assumed that most developers do not possess knowledge of Thematic.js beforehand. Therefore, it is likely that for typical visualizer, difference in effort between using and not using Thematic.js is smaller than what is indicated in the evaluation results.

- Evaluation assumes that all APIs are equally known for the visualizer
  - This likely not true
  - The difference between known and unknown APIs – if JS (and Leaflet) is known, but Thematic.js is not? How does this affect the true effort?
  - Would need research on the average visualizer – does she know JS or Leaflet?
- Different evaluation methods yield different results (should the framework be included in calculations? how?) What measurements are used? Etc.
  - Building the library incurred costs
  - The library itself likely needs maintenance etc.
  - From the visualizer perspective, these do not affect much
- Framework is reusable - how is that taken into account when calculating results (can be used in the future infinity times)
- At the moment not comparing HTML, CSS etc.
- Now all the implementations were done by the same guy – the one who nows the library API and its capabilities

  - Better option would be to try this with external developers – some with the tool and some without
  - However, external developers likely have varying experience on JS, mapping etc.
  - In order for this kind of study (with ext devs) to be reliable, sample size should be several (dozen) people (reference for this page 34 of this thesis?). This was not feasible in the scope of the thesis.
- How about using this long term? If the visualizer gets to know the library, he will be more productive with it.

9.3 External Validity of the Study

- The library was built using the same literature as the evaluated cases defined – this likely makes it so that the library suits particularly well the cases.
- The nature of cases picked for evaluation affects the results considerably - e.g. choosing really similar cases yields more positive results, really different cases yield more negative results - a note from one of the reuse sources that mention reuse evaluation being really hard
- These are inherently simple visualizations - with more complex data or need, benefits are likely to diminish.

9.4 Further Research

- Framework likely affects other properties of visualizations - quality, maintainability etc. These could be studied in the future.
- Interaction not covered. Software does not provide any complex interaction etc. These still need some manual coding.
- Analysis on average visualizers – are they cartographers, web developers, laymen? These qualities likely affect the library requirements (and most suitable metrics)

At this point, you will have some insightful thoughts on your implementation and you may have ideas on what could be done in the future. This chapter is a good place to discuss your thesis as a whole and to show your professor that you have really understood some non-trivial aspects of the methods you used…

Chapter 10

Conclusions

Time to wrap it up! Write down the most important findings from your work. Like the introduction, this chapter is not very long. Two to four pages might be a good limit.

!FIXME **Hand-check references, remove google books urls, consolidate accessed dates, etc.** FIXME!

Bibliography

Vladimir Agafonkin. Leaflet, May 2011. URL `http://www.leafletjs.com`. Accessed 13.10.2014.

R. Amar and J. Stasko. A knowledge task-based framework for design and evaluation of information visualizations. In *IEEE Symposium on Information Visualization, 2004. INFOVIS 2004*, pages 143–150, 2004. doi: 10.1109/INFVIS.2004.10.

Gennady L. Andrienko and Natalia V. Andrienko. Interactive maps for visual data exploration. *International Journal of Geographical Information Science*, 13(4):355–374, 1999. ISSN 1365-8816. doi: 10.1080/136588199241247. URL `http://dx.doi.org/10.1080/136588199241247`.

Apple. UIWebView class reference, 2014. URL `https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIWebView_Class/index.html`. Accessed 21.10.2014.

Jeremy Ashkenas. CoffeeScript, December 2009. URL `http://coffeescript.org`. Accessed 7.9.2014.

Tarek Azzam and Stephanie Evergreen. *J-B PE Single Issue (Program) Evaluation, Volume 139 : Data Visualization, Part 1 : New Directions for Evaluation.* John Wiley & Sons, Somerset, NJ, USA, 2013. ISBN 9781118793374. URL `http://site.ebrary.com/lib/aalto/docDetail.action?docID=10768989`.

Rajiv D. Banker, Srikant M. Datar, Chris F. Kemerer, and Dani Zweig. Software complexity and maintenance costs. *Commun. ACM*, 36(11):81–

94, November 1993. ISSN 0001-0782. doi: 10.1145/163359.163375. URL `http://doi.acm.org/10.1145/163359.163375`.

Tim Berners-Lee. Information management: A proposal, March 1989. URL `http://www.w3.org/History/1989/proposal.html`.

Tim Berners-Lee, Robert Cailliau, Jean-François Groff, and Bernd Pollermann. World-wide web: The information universe. *Internet Research*, 2(1):52–58, December 1992. ISSN 1066-2243. doi: 10.1108/eb047254. URL `http://www.emeraldinsight.com/journals.htm?articleid=1670698&show=abstract`.

B. Boehm. Managing software productivity and reuse. *Computer*, 32(9): 111–113, September 1999. ISSN 0018-9162. doi: 10.1109/2.789755.

Michael Bostock and Jason Davies. Code as cartography. *The Cartographic Journal*, 50(2):129–135, May 2013. ISSN 0008-7041. doi: 10.1179/0008704113Z.00000000078. URL `http://www.maneyonline.com/doi/abs/10.1179/0008704113Z.00000000078`.

Davide Brugali, Giuseppe Menga, and Amund Aarsten. The framework life span. *Commun. ACM*, 40(10):65–68, October 1997. ISSN 0001-0782. doi: 10.1145/262793.262806. URL `http://doi.acm.org/10.1145/262793.262806`.

Manuel Carro, José F. Morales, Henk L. Muller, G. Puebla, and M. Hermenegildo. High-level languages for small devices: A case study. In *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '06, pages 271–281, New York, NY, USA, 2006. ACM. ISBN 1-59593-543-6. doi: 10.1145/1176760.1176794. URL `http://doi.acm.org/10.1145/1176760.1176794`.

Ned Chapin, Joanne E. Hale, Khaled Md. Kham, Juan F. Ramil, and Wui-Gee Tan. Types of software evolution and software maintenance. *Journal of Software Maintenance*, 13(1):3–30, January 2001. ISSN 1040-550X. URL `http://dl.acm.org/citation.cfm?id=371697.371701`.

J.C. Cleaveland. Building application generators. *IEEE Software*, 5(4):25–33, July 1988. ISSN 0740-7459. doi: 10.1109/52.17799.

Borden Dent, Jeff Torguson, and Thomas Hodler. *Cartography: Thematic Map Design*. McGraw-Hill Science/Engineering/Math, New York, 6 edition edition, August 2008. ISBN 9780072943825.

ECMA. ECMAScript® language specification, June 2011. URL `http://www.ecma-international.org/ecma-262/5.1/`. Accessed 7.9.2014.

Mohamed Fayad and Douglas C. Schmidt. Object-oriented application frameworks. *Commun. ACM*, 40(10):32–38, October 1997. ISSN 0001-0782. doi: 10.1145/262793.262798. URL `http://doi.acm.org/10.1145/262793.262798`.

Mohamed E. Fayad and David S. Hamu. Enterprise frameworks: Guidelines for selection. *ACM Comput. Surv.*, 32(1es), March 2000. ISSN 0360-0300. doi: 10.1145/351936.351940. URL `http://doi.acm.org/10.1145/351936.351940`.

Norman E. Fenton and Shari Lawrence Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Co., Boston, MA, USA, 2nd edition, 1998. ISBN 0534954251.

Eric Fischer. Generalized residential land use plan by density and building type (1971), November 2012. URL `https://www.flickr.com/photos/walkingsf/8225020729/`.

W.B. Frakes and S. Isoda. Success factors of systematic reuse. *IEEE Software*, 11(5):14–19, September 1994. ISSN 0740-7459. doi: 10.1109/52.311045.

William Frakes and Carol Terry. Software reuse: Metrics and models. *ACM Comput. Surv.*, 28(2):415–435, June 1996. ISSN 0360-0300. doi: 10.1145/234528.234531. URL `http://doi.acm.org/10.1145/234528.234531`.

G.K. Gill and C.F. Kemerer. Cyclomatic complexity density and software maintenance productivity. *IEEE Transactions on Software Engineering*, 17 (12):1284–1288, December 1991. ISSN 0098-5589. doi: 10.1109/32.106988.

Google. Maps API, June 2005a. URL `http://maps.google.com`. Accessed 23.7.2014.

Google. Maps, February 2005b. URL `http://maps.google.com`. Accessed 25.7.2014.

Google. Building web apps in WebView, 2014. URL `http://developer.android.com/guide/webapps/webview.html`. Accessed 21.10.2014.

Maurice H. Halstead. Elements of software science, 1977. URL `http://cds.cern.ch/record/281413`.

Benjamin D. Hennig. Population of the world, 2014. URL `http://www.worldmapper.org/svg/map2/index.html`. Copyright Benjamin D. Hennig (Worldmapper Project). Licensed under Creative Commons CC-BY-ND license. Accessed 29.10.2014.

E. Horowitz, A Kemper, and B. Narasimhan. A survey of application generators. *IEEE Software*, 2(1):40–54, January 1985. ISSN 0740-7459. doi: 10.1109/MS.1985.230048.

Leah Hunter. Why the WebView is the future of mac OS x apps, April 2014. URL `http://www.fastcolabs.com/3029292/why-the-webview-is-the-future-of-mac-os-x-apps`. Accessed 21.10.2014.

Ohad Inbar, Noam Tractinsky, and Joachim Meyer. Minimalism in information visualization: Attitudes towards maximizing the data-ink ratio. In *Proceedings of the 14th European Conference on Cognitive Ergonomics: Invent! Explore!*, ECCE '07, pages 185–188, New York, NY, USA, 2007. ACM. ISBN 978-1-84799-849-1. doi: 10.1145/1362550.1362587. URL `http://doi.acm.org/10.1145/1362550.1362587`.

M. Jazayeri. Some trends in web application development. In *Future of Software Engineering, 2007. FOSE '07*, pages 199–213, May 2007. doi: 10.1109/FOSE.2007.26.

Ralph E. Johnson. Frameworks=(components+ patterns). *Communications of the ACM*, 40(10):39–42, 1997. URL `http://dl.acm.org/citation.cfm?id=262799`.

Barbara Ann Kitchenham and Lesley M. Pickard. Evaluating software eng. methods and tools part 10: Designing and running a quantitative case study. *SIGSOFT Softw. Eng. Notes*, 23(3):20–22, May 1998. ISSN 0163-5948. doi: 10.1145/279437.279445. URL `http://doi.acm.org/10.1145/279437.279445`.

Cornelis Koeman. *Het beginsel van communicatie in de kartografie*. Theatrum Orbis Terrarum, 1969.

R. Kosara. Visualization criticism - the missing link between information visualization and art. In *Information Visualization, 2007. IV '07. 11th International Conference*, pages 631–636, July 2007. doi: 10.1109/IV.2007.130.

Menno-Jan Kraak. The cartographic visualization process: From presentation to exploration. *The Cartographic Journal*, 35(1):11–15, June 1998. ISSN 0008-7041. doi: 10.1179/caj.1998.35.1.11. URL `http://www.maneyonline.com/doi/abs/10.1179/caj.1998.35.1.11`.

Menno-Jan Kraak and Alan MacEachren. Visualization for exploration of spatial data. *International Journal of Geographical Information Science*, 13(4):285–287, 1999. ISSN 1365-8816. doi: 10.1080/136588199241201. URL `http://dx.doi.org/10.1080/136588199241201`.

Menno-Jan Kraak and Ferjan Ormeling. *Cartography, Third Edition: Visualization of Spatial Data*. Guilford Press, June 2011. ISBN 9781609181949.

Charles W. Krueger. Software reuse. *ACM Comput. Surv.*, 24(2):131–183, June 1992. ISSN 0360-0300. doi: 10.1145/130844.130856. URL `http://doi.acm.org/10.1145/130844.130856`.

M. Langford and D. J. Unwin. Generating and mapping population density surfaces within a geographical information system. *The Cartographic Journal*, 31(1):21–26, June 1994. ISSN 0008-7041. doi: 10.1179/000870494787073718. URL `http://www.maneyonline.com/doi/abs/10.1179/000870494787073718`.

Divya Manian, Paul Irish, Tim Branyen, Connor Montgomery, and Jake Verbaten. HTML5 please, July 2011. URL `http://html5please.com/`. Accessed 21.10.2014.

Michael Mattsson, Jan Bosch, and Mohamed E. Fayad. Framework integration problems, causes, solutions. *Commun. ACM*, 42(10):80–87, October 1999. ISSN 0001-0782. doi: 10.1145/317665.317679. URL `http://doi.acm.org/10.1145/317665.317679`.

T.J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320, December 1976. ISSN 0098-5589. doi: 10.1109/TSE.1976.233837.

Doug Mcilroy. Mass-produced software components. pages 138–155, January 1969. URL `http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF`.

Emilia Mendes, Nile Mosley, and Steve Counsell. Web metrics - estimating design and authoring effort. *IEEE MultiMedia*, 8(1):50–57, 2001. URL `https://www.cs.auckland.ac.nz/~emilia/publications/IEEEMM2001.pdf`.

MetaCarta. OpenLayers, June 2006. URL `http://www.openlayers.org`. Accessed 13.10.2014.

Christopher C. Miller. A beast in the field: The google maps mashup as GIS/2. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 41(3):187–199, September 2006. doi: 10.3138/J0L0-5301-2262-N779. URL `http://dx.doi.org/10.3138/J0L0-5301-2262-N779`.

Charles Joseph Minard. Carte figurative et approximative des quantités de vin français exportés par mer en 1864, 1865. URL `http://commons.wikimedia.org/wiki/File:Minard%E2%80%99s_map_of_French_wine_exports_for_1864.jpg`.

John C. Mitchell. *Concepts in Programming Languages*. Cambridge University Press, 2003. ISBN 9780521780988.

Parastoo Mohagheghi and Reidar Conradi. Quality, productivity and economic benefits of software reuse: a review of industrial studies. *Empirical Software Engineering*, 12(5):471–516, October 2007. ISSN 1382-3256, 1573-7616. doi: 10.1007/s10664-007-9040-x. URL `http://link.springer.com/article/10.1007/s10664-007-9040-x`.

Parastoo Mohagheghi and Reidar Conradi. An empirical investigation of software reuse benefits in a large telecom product. *ACM Trans. Softw. Eng. Methodol.*, 17(3):13:1–13:31, June 2008. ISSN 1049-331X. doi: 10.1145/1363102.1363104. URL `http://doi.acm.org/10.1145/1363102.1363104`.

Addy Osmani. Essential JavaScript namespacing patterns, September 2011. URL `http://addyosmani.com/blog/essential-js-namespacing/`. Accessed 21.10.2014.

Johannes Sametinger. *Software engineering with reusable components*. Springer, 1997. URL `http://www.google.com/books?hl=en&lr=&id=AxPQvGRs2wUC&oi=fnd&pg=PA1&dq=code+scavenging+reuse&ots=K85JD3_HCN&sig=66nJKlXrtJc3bHAtYMe4SCc2tTc`.

Hansgeorg Schlichtmann. Visualization in thematic cartography: towards a framework. *The Selected Problems of Theoretical Cartography*, pages 49–61, 2002. URL `http://rcswww.urz.tu-dresden.de/~wolodt/tc-com/pdf/sch_2000.pdf`. Accessed 21.7.2014.

Han Albrecht Schmid. Systematic framework design by generalization. *Commun. ACM*, 40(10):48–51, October 1997. ISSN 0001-0782. doi: 10.1145/262793.262803. URL `http://doi.acm.org/10.1145/262793.262803`.

Alexis Sellier. LESS, 2009. URL `http://lesscss.org/`. Accessed 7.9.2014.

B. Shneiderman. The eyes have it: a task by data type taxonomy for information visualizations. In *, IEEE Symposium on Visual Languages, 1996. Proceedings*, pages 336–343, September 1996. doi: 10.1109/VL.1996.545307.

Terry A. Slocum and Robert B. McMaster. *Thematic Cartography and Geovisualization: Pearson New International Edition*. Pearson Education, 3rd edition, 2014. ISBN 9781292055442. URL `https://www.dawsonera.com/abstract/9781292055442`.

Matt Small. Ten things you need to know about WebView, October 2012. URL `http://blogs.msdn.com/b/wsdevsol/archive/2012/10/18/nine-things-you-need-to-know-about-webview.aspx`. Accessed 21.10.2014.

John Snow. Cholera cases in london epidemic of 1854, 1854. URL `http://commons.wikimedia.org/wiki/File:Snow-cholera-map.jpg`.

StatCounter. GlobalStats, 2014. URL `http://gs.statcounter.com/`. Accessed 21.10.2014.

Prabhat Totoo, Pantazis Deligiannis, and Hans-Wolfgang Loidl. Haskell vs. f# vs. scala: A high-level language features and parallelism support comparison. In *Proceedings of the 1st ACM SIGPLAN Workshop on Functional High-performance Computing*, FHPC '12, pages 49–60, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1577-7. doi: 10.1145/2364474.2364483. URL `http://doi.acm.org/10.1145/2364474.2364483`.

Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, CT, USA, 1986. ISBN 0-9613921-0-X.

J.J. van Wijk. The value of visualization. In *IEEE Visualization, 2005. VIS 05*, pages 79–86, October 2005. doi: 10.1109/VISUAL.2005.1532781.

World Wide Web Consortium. Cascading style sheets level 2 revision 1 (CSS 2.1) specification, June 2011. URL `http://www.w3.org/TR/CSS2/`. Accessed 7.9.2014.

World Wide Web Consortium. HTML5, September 2014. URL `http://www.w3.org/html/wg/drafts/html/CR/`. Accessed 7.9.2014.

Torre Zuk and Sheelagh Carpendale. Theoretical analysis of uncertainty visualizations. volume 6060, pages 606007–606007–14, 2006. doi: 10.1117/12.643631. URL `http://dx.doi.org/10.1117/12.643631`.

Torre Zuk, Lothar Schlesier, Petra Neumann, Mark S. Hancock, and Sheelagh Carpendale. Heuristics for information visualization evaluation. In *Proceedings of the 2006 AVI Workshop on BEyond Time and Errors: Novel Evaluation Methods for Information Visualization*, BELIV '06, pages 1–6, New York, NY, USA, 2006. ACM. ISBN 1-59593-562-2. doi: 10.1145/1168149.1168162. URL `http://doi.acm.org/10.1145/1168149.1168162`.

Appendix A

Flat Dot Format

Flat dot format is a simple, but non-standard format used by a number of web mapping applications such as the Store Finder of Alko[1]. Format consists of a JSON file representing an array of zero or more objects. The objects must contain latitude and longitude properties, and may contain a number of other properties. An example of the format is depicted below.

```
[
  {
    "_id": "51b9f04d27eaa49f4a0001cc",
    "number": 2,
    "name": "Destination",
    "latitude": 60.314322,
    "longitude": 24.554067
  },
  {
    "_id": "51b9f0f3127ceac8db000263",
    "number": 0,
    "name": "Departure",
    "latitude": 60.314041,
    "longitude": 24.551678
  },
```

[1] http://www.alko.fi/myymalat/

```
  {
    "_id": "51b9fa600752f83f720003c3",
    "number": 1,
    "name": "Pit stop",
    "latitude": 60.316474,
    "longitude": 24.556554
  }
]
```

Appendix B

ESComplex Results for Visualizations

!FIXME **Add the final results for ESComplex.** FIXME!

Appendix C

Visualization Heuristics Evaluation

Thematic.js evaluation results based on the visualization heuristics presented by Zuk et al. (2006) are displayed in table C.1. The evaluation is done using a tree-step scale. Heuristic is evaluated *positive* if Thematic.js encourages conforming to the heuristic when compared to using no visualization library, *neutral* if using Thematic.js has no effect, and *negative* if Thematic.js discourages conforming to the heuristic.

Heuristic	**Evaluation**	**Reasoning**
Visual variable	Neutral	Of map visualizations, this concerns mostly choropleth maps. Thematic.js choropleth maps do not ensure minimum geographical size for areas. However, using the default line weight ensures a minimum screen size of several pixels.

Heuristic	**Evaluation**	**Reasoning**
Color order	Neutral	Thematic.js choropleth, dasymetric and isarithmic maps are primarily based on coloring the map. Moreover, the visualizer is given the possibility of freely choosing the colors. This may lead to situations when the visualizer chooses the colors inappropriately for displaying order. However, this situation is not different from the alternative situation of the visualizer creating the visualization without using a visualization library.
Color size	Neutral	Thematic.js does not provide any color-adjusting mechanisms based on the size of the element.
Local contrast	Neutral	Thematic.js does not provide any color-adjusting mechanisms based on contrast.
Color blindness	Neutral	Thematic.js does not provide any advice regarding color blindness.
Preattentive benefits	Neutral	Thematic.js provides and enforces spacial positioning of the data. However, this is fundamental to any geovisualization, and therefore, cannot be considered a positive trait of the library.
Size variation	Positive	Thematic.js provides size variation in proportional symbol mapping to encourage the visualizer to emphasize quantitative variation in data.

Heuristic	**Evaluation**	**Reasoning**
Graphic dimensionality	Negative	Thematic.js does not enforce preserving dimensionality of the data, and in some cases, such as when using a proportional symbol map, it encourages the visualizer to increase dimensionality by displaying scalar values using proportional symbols.
Most data	Positive	Thematic.js encourages the visualizer to maximize data shown by providing support for several different mapping methods suitable for different kind of data.
No extra ink	Positive	Thematic.js provides data aggregation functionality to combine the relevant data.
Gestalt laws	Positive	Thematic.js provides functionality to support Gestalt laws of grouping, such as using different symbols and sizes for different data points. However, not all Gestalt laws are considered.
Levels of detail	Positive	Thematic.js provides clustering functionality of dots and symbols. While currently there is no support for levels of detail for other mapping methods, the library does not prevent implementing this in the future.
Integrate text	Positive	Thematic.js supports attaching popups with textual content to data points, such as markers or choropleth areas.

Heuristic	**Evaluation**	**Reasoning**
Overview first	Positive	Thematic.js supports overview-first approach in most of the mapping methods. Dot and proportional symbol maps support marker clustering and choropleth, isarithmic and dasymetric maps support zooming in to show the details.
Zoom and filter	Neutral	Thematic.js supports zooming of the map. However, support for filtering data on view-level is not provided.
Details on demand	Positive	Thematic.js supports attaching popups to data points for displaying additional details.
Relate	Neutral	Thematic.js does not support any method of emphasizing relationships between entries other than spacial distribution.
Extract	Positive	While Thematic.js does not support physical saving of data subsets, it provides bookmarking and linking support which effectively provide similar benefits.
History	Positive	Thematic.js supports using the back and forward buttons of the browser to undo and redo actions.
Uncertainty	Neutral	Thematic.js does not encourage the visualizer to display the uncertainties in data.
Relationships	Neutral	Thematic.js does not encourage concretizing relations between data points.
Domain Parameters	Neutral	While Thematic.js modules require explicitly stating the used parameters, there is no guarantee about the importance of selected parameters.

Heuristic	**Evaluation**	**Reasoning**
Multivariate	Positive	Thematic.js provides aggregation functionality in order enable easy experimenting about relationships between variables. Moreover, the modular structure of the library results in the possibility to easily combine several visualization methods to highlight different aspects of the data.
Cause & effect	Neutral	Thematic.js does not provide additional means for determining or displaying cause and effect.
Hypotheses	Positive	The availability of several different mapping methods of Thematic.js encourage the visualizer to better display and evaluate hypotheses.

Table C.1: Evaluation of Thematic.js according to heuristics presented by Zuk et al. (2006).

Appendix D

Mapping Objectives Evaluation

Evaluation results of Thematic.js regarding thematic mapping objectives of Schlichtmann (2002) are presented in table D.1. The evaluation was performed with a three-step scale. Result for each objective is regarded as *positive* if the library encourages achieving the objectives better than typical mapping library does. Result is regarded as *neutral* if using Thematic.js has no effect on achieving the objective, and *negative* if Thematic.js discourages the visualizer to achieve the objective.

Name	**Evaluation**	**Reasoning**
Clarification	Positive	Thematic.js benefits the clarification of the visualization by, e.g., providing clustering functionality of the markers
Emphasis	Neutral	Thematic.js uses visual markers in dot maps. However, this is typically achieved with any mapping library even with no visualization library.
Types of Entries	Positive	Thematic.js provides support for using different markers for different types of topemes.
Sets of Types	Neutral	Thematic.js does neither encourage nor discourage consolidating types of topemes according to mutual similarities.

Name	**Evaluation**	**Reasoning**
Cross-Relations	Positive	Several of the mapping methods, especially proportional symbol method, support indicating similarities between different types of entries.
Local Syntax	Neutral	Thematic.js pays no special attention to managing lower-order units within topemes.
Local Ensembles	Neutral	Local ensembles are not supported in any of the current mapping methods of Thematic.js.
Multilocal Ensembles	Neutral	Multilocal ensembles are not supported in any of the current mapping methods of Thematic.js.
Addable and Non-Addable Quantities	Positive	Thematic.js separates between addable and non-addable quantities by separating between different mapping methods.
The Surface Illusion	Neutral	Thematic.js provides no additional means of achieving the surface illusion when compared to, e.g. the underlying Leaflet.js mapping library.

Table D.1: Evaluation of map visualization objectives of Schlichtmann (2002).