Cartography M.Sc.

Master thesis

Automated polygon schematization for thematic maps

Jakob Listabarth

Technical University of Munich

Technische Universität Wien
Vienna University of Technology

Technische Universität Dresden

ITC University of Twente.

2021

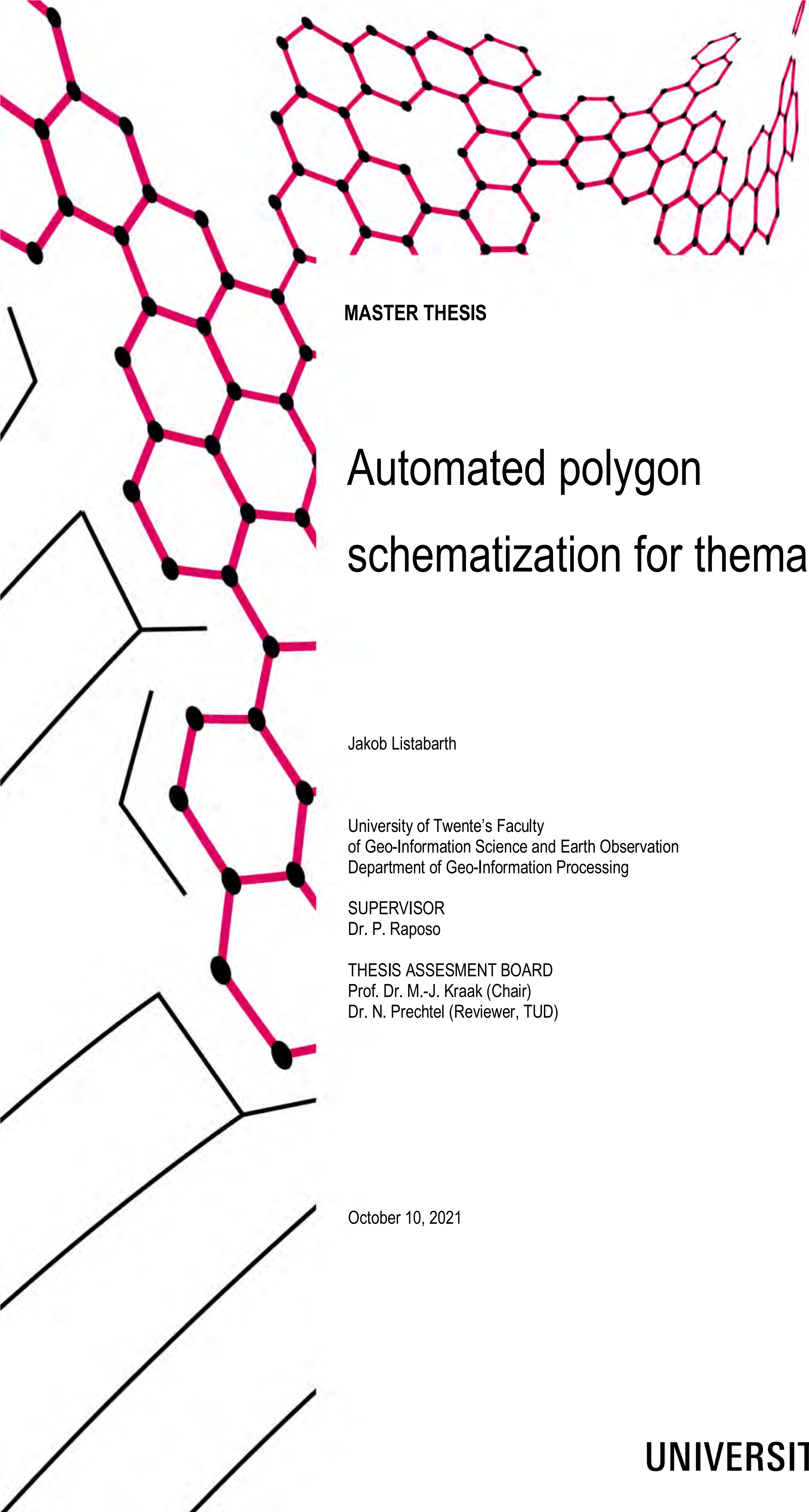
Statement of Authorship

Herewith I declare that I am the sole author of the submitted Master's thesis entitled:

"Automated polygon schematization for thematic maps"

I have fully referenced the ideas and work of others, whether published or unpublished. Literal or analogous citations are clearly marked as such.

Vienna, October 2021 Jakob, Listabarth



MASTER THESIS

Automated polygon schematization for thematic maps

Jakob Listabarth

University of Twente's Faculty
of Geo-Information Science and Earth Observation
Department of Geo-Information Processing

October 10, 2021

UNIVERSITY OF TWENTE.

Automated polygon schematization for thematic maps

Master Thesis

Jakob Listabarth
March—October 2021

STATEMENT OF AUTHORSHIP

Herewith I declare that I am the sole author of the submitted Master's thesis entitled: *"Automated polygon schematization for thematic maps"*. I have fully referenced the ideas and work of others, whether published or unpublished. Literal or analogous citations are clearly marked as such.

Vienna, 2021-10-10
Jakob Listabarth

DISCLAIMER

Abstract

Schematization in cartography is a particular case of abstraction, with the purpose of radically reducing visual complexity. Simplification accompanies such an abstraction, underlining the simplified quality of the map. This thesis aims to build a prototype for an interactive web-based schematization tool that processes vector polygon data, e.g. administrative boundaries, provided by the user. To achieve this goal, I (1) specified software requirements, (2) implemented a schematization algorithm and (3) integrated the algorithm's implementation into a prototype. Lastly (4), I evaluate this prototype based on the specified requirements (step 1). Suitable algorithms exist and such a tool is feasible. Yet, it implicates challenges regarding performance and robustness. Robustness issues mainly originate in user-defined input data. Therefore, they can be addressed by firstly validating and consequently rejecting invalid input, and by secondly improving how the algorithm handles unexpected conditions. Considering the long running times, meaningful feedback as one principle of usability requires attention. The prototype evaluation is based on a requirement verification. Validation-related requirements were met. However, shortcomings regarding the algorithm implementation and the Graphical User Interface (GUI) need to be fixed to allow extensive user tests. Consequently, the prototype can be iteratively improved for release.

Keywords: schematization, generalization, thematic mapping, automated cartography, cartographic web services

Acknowledgements

I would like to say "Thank you" to

Paulo Raposo — supervisor, for being a critical and helpful conversation partner throughout the entire semester, your support is very much appreciated.

Nikolas Prechtel — external reviewer, and *Menno-Jan Kraak* for your feedback as member of the thesis assessment board.

Wouter Meulemans, for the very helpful correspondence about specific issues of the algorithm implementation, thank you so much!

Christoph Listabarth, for endless phone calls, shared coding sessions and lastly, for helping me to not lose the enthusiasm.

Sarah Zeller, for accompanying me through this time in the best way I could think of.

Contents

1. Introduction 1

1.1. Research identification . . . 1

1.1.1. Research Sub-Objectives (RSOs) . . . 2

1.1.2. Research Questions (RQs) . . . 2

1.1.3. A web-based, freely available schematization tool . . . 3

1.2. Thesis outline . . . 3

2. Background and related work 5

2.1. Schematization . . . 5

2.1.1. Schematization in general . . . 5

2.1.2. Schematization in cartography . . . 6

2.1.3. Properties of schematized maps . . . 9

2.1.4. Schematization types . . . 11

2.1.5. Geometric qualities of schematization . . . 14

2.2. Thematic mapping . . . 15

2.3. Schematized regions in thematic maps . . . 20

2.4. Automation in cartography . . . 24

2.5. Cartographic web services . . . 26

3. Methodology 28

3.1. Requirement engineering . . . 29

3.1.1. Business requirements . . . 29

3.1.2. User requirements . . . 30

3.1.3. Functional requirements . . . 30

3.1.4. Requirement verification . . . 31

3.2. Prototyping . . . 32

3.3. Usability studies . . . 34

3.4. Adapting methods . . . 35

4. Implementing the prototype 38

4.1. Comparing schematization approaches . . . 38

4.1.1. Selecting approaches . . . 39

4.1.2. Discussing approaches . . . 39

4.1.3. Limitations of the comparison . . . 41

4.2. Implementing the selected approach 42
4.2.1. Area-preserving simplification and schematization algorithm 42
4.2.2. The Doubly-Connected Edge List (DCEL) 43
4.3. Software requirements 44
4.3.1. Vision and scope 45
4.3.2. User requirements 45
4.3.3. Functional requirements 48
4.3.4. Features 52
4.4. The GUI 54
4.4.1. GUI components 55
4.4.2. User-defined parameters 57

5. Results and discussion **59**
5.1. Algorithm limitations 59
5.2. Implementation limitations 60
5.3. Prototype evaluation 61
5.4. Future research 63

6. Conclusion **65**

7. List of references **67**

A. Pseudocode implementation of DCEL generation **71**

B. Designs for the mock-up prototype **76**

C. Proposed user study **81**
C.1. Study setting 81
C.2. Scenario and user activity 82

D. Requirement verification’s test protocol **84**

E. Functional requirements **90**

List of Figures

2.1. Henry Beck's tube map. 7
2.2. Schema of a T-O-map. 8
2.3. Erntestatistik Winter Weizen für das Jahr 1903. 9
2.4. Transit map with schematized fare zones. 11
2.5. Geometric styles of schematization. 12
2.6. Advantages of map matching approaches for C-oriented schematizations. 13
2.7. Topology preserving schematizations. 14
2.8. Vertex-restriction of schematization. 15
2.9. Graphic density. 19
2.10. Angular separation. 19
2.11. Retinal separation. 20
2.12. Increased angular legibility by schematization. 21
2.13. Schematization and visual complex thematic overlays. 22
2.14. Using schematization to establish contrast by detail variation. 22
2.15. Contrast by shape in combinations of schematization styles and thematic overlay. . 23
2.16. Running times of algorithms. 25
2.17. Screenshot of the simplification tool Mapshaper. 27

3.1. Prototyping lifecycle. 32
3.2. A typical software prototyping process. 33
3.3. Iterative design process. 37

4.1. The DCEL data structure. 44
4.2. Topology issues ought to be detected prior to schematization. 50
4.3. Feature-tree for the schematization tool. 53
4.4. Screenshot showing the GUI components. 55
4.5. Launch screen of the proposed GUI. 56
4.6. The floating panel adapts to the application's context. 57
4.7. User Interface (UI) component to set up C prior to schematization. 58

5.1. Algorithm limitations regarding the schematization's orientations. 59

B.1. Start screen. 76
B.2. Setup of a regular schematization. 77

B.3. Setup of a irregular schematization. 77
B.4. Setup of a k and λ. 78
B.5. Screen showing the process while schematizing. 78
B.6. Screen after a successful schematization. 79
B.7. Screen during the input validation. 79
B.8. Screen after a negative input validation. 80

C.1. Schematized map for the newspaper. 82

D.1. State of the UI while input data is processed. 87
D.2. The schematized region is displayed in the GUI's map-view. 88
D.3. The input region is displayed in the GUI's map-view. 89

List of Tables

4.1. Overview of existing schematization approaches for regions 39
4.2. Functional requirements for the alpha release 52
4.3. The features and their planned implementation 54

5.1. Requirement verification 62

E.1. Functional Requirements for the beta release 90
E.2. Functional Requirements for the first release candidate 91

List of acronyms

API	Application Programming Interface
CLI	Command Line Interface
CRS	Coordinate Reference System
CSS	Cascading Style Sheets
DCEL	Doubly-Connected Edge List
ESRI	Environmental Systems Research Institute
GIS	Geographic Information System
GUI	Graphical User Interface
HTML	HyperText Markup Language
ICA	International Cartographic Association
ISOTYPE	International System of Typographic Picture Education
JS	JavaScript
JSON	JavaScript Object Notation
NACIS	North American Cartographic Information Society
npm	Node.js Package Manager
OFL	Open Font License
RQ	Research Question
RSO	Research Sub-Objective
SVG	Scalable Vector Graphics
Ts	TypeScript
UI	User Interface
URL	Uniform Resource Locator
WFS	Web Feature Service
WMS	Web Map Service

1. Introduction

Schematization is a powerful mean of communication. Therefore, it is widely applied in cartographic practice. Despite their frequent use, schematized maps are usually still drawn by hand: a slow and tedious process, which seems particularly anachronistic considering recent developments in cartography, like real-time maps. Additionally, efficient processes that automatically keep maps up-to-date exemplify the need for generating schematized maps in an automated manner. The lack of accessible tools facilitating automated schematization explains why schematization is usually still a manual process. And yet numerous algorithmic approaches to generate schematized maps have been published over the last two decades. Nevertheless, none of these algorithms have been implemented into an accessible working cartographic service. This thesis aims to contribute to this missing link between the proposed algorithms and a tool which enables the map maker to create such schematized maps.

A considerable amount of research and techniques on generalization, particularly tailored for topographic mapping, originates from cartographic practice and research in this domain over the last decades. Some of these approaches are innovative, e.g. using not only vector data but image processing (Shen et al., 2018) as a starting point. Besides topographic maps, schematized maps (Burghardt et al., 2014), more precisely transit maps, are in the spotlight of research on map generalization: publications concern their usability and techniques to automatize their generation (Roberts, 2014; Roberts et al., 2016; Roberts & Vaeng, 2016; Wu et al., 2020). Cartographers use schematization, an extreme type of simplification, for maps which "are narrow in their function and task" (Burghardt et al., 2014, p. 300).

Despite the amount of research carried out regarding schematized transit maps, white spots on how to generalize for thematic maps may remain on the cartographic research-map (Raposo et al., 2020). Therefor, this research focuses on automated polygon schematization, particularly for thematic mapping.

1.1. Research identification

This section outlines the overarching research objective and its resulting sub-objectives. Each sub-objective relates to a corresponding research question. These are answered in chapter 2 Background and related work, 4 Implementing the prototype, and 5 Results and discussion by employing methods described in chapter 3 Methodology. To conclude the research identification, I describe the contribution of this research to the cartographic community.

1.1.1. Research Sub-Objectives (RSOs)

The main research objective is to develop a prototype of an interactive web-based tool allowing to automatically schematize a set of polygons. The result differs based on input data, geographic vector data, and parameters for the schematization, all interactively defined by the user. The research objective and its corresponding sub-objectives are described as follows.

Main research objective
Create a prototype of an interactive web-based schematization tool for use in thematic mapping.

Research Sub-Objectives

→ **RSO A** Define cartographic requirements for polygon schematization regarding thematic mapping.

→ **RSO B** Compare existing algorithms for the schematization of regions regarding their suitability for the proposed tool and their characteristics (e.g. computational and visual differences).

→ **RSO C** Define software requirements for such a schematization tool based on a requirement engineering framework proposed by Wiegers and Beatty (2013).

→ **RSO D** Implement one schematization approach, based on one or more algorithms, as discussed in *RSO C*.

1.1.2. Research Questions (RQs)

The main research objective is met by providing answers to the following RQs, each of them corresponding to one sub-objective:

→ **RQ A** What are best practices for designing the geographic layer of thematic maps and how are they compatible with the properties of schematized maps?

→ **RQ B** Which types of automated schematization exist and what are their cartographic (visual) and technical (software) characteristics?

→ **RQ C** Considering the characteristics of region schematizing algorithms and the environment of a web-based tool – what are the system features, data requirements, user requirements, quality attributes, and possible other requirements?

→ **RQ D** To which extent does the prototype satisfy the requirements specified in *RSO B*?

1.1.3. A web-based, freely available schematization tool

The popular web-based generalization tool Mapshaper.org is used as best-practice example for an accessible cartographic web service in the scope of this project. Mapshaper is free, open-source and compatible with a series of standard geographic data formats (*.shp, GeoJSON, TopoJSON, DBF, CSV*). The tool is accessible to a broad group of users. This is firstly because it runs directly in any modern browser, without further dependencies. Secondly, it is not only intended for experts but also for lay cartographers. Aiming for such a broad user group, the tool does not require specific skills like coding. Regarding this aspect, the Mapshaper tool serves as a role model for this project.

With Mapshaper, an accessible tool for generalization does exist. However, there seems to be a lack of a comparable tool concerned with the schematization for both the schematization of networks – even though schematization research has focused on these for years – and the schematization of regions. Hence, this research project aims to explore the feasibility of implementing such a tool to schematize regions. It means to identify possible constraints in the implementation and determine limiting factors such as the web environment or underlying algorithms. Furthermore, this research intends to provide an overview of existing schematization algorithms for regions. This overview focuses on their suitability for the objective of implementing them in the described web application context.

1.2. Thesis outline

Introduction. In the first chapter, I outline the motivation for working on automated polygon schematization for thematic maps and the research aim. I also explain the project's contribution to the scientific community in the cartographic domain.

Background and related work. This chapter provides the definitions for the central terms. It introduces the fundamental concepts of schematization, automation in cartography and cartographic web services, on which this thesis is based. Furthermore, I discuss and compare related work regarding these concepts.

Methodology. The third chapter concerns the research design: it describes the methods for each research sub-objective and each method's characteristics. I outline in detail how each of the research phases has been carried out and relate the applied methods to the literature.

Implementing the prototype. The implementation of the proposed prototype included several steps, including the selection of an schematization algorithm based on criteria relevant cartographic purposes, the specification of requirements for such prototype and the visual design of the GUI. In this chapter I describe these preliminary processes.

Results and discussion. The main part of this research is the presentation and discussion of the results. It includes a comparison of existing schematization algorithms and their suitability within an interactive, web-based schematization tool, software requirements for such a tool, the implementation of a schematization algorithm, as well as the implementation of a prototype of a schematization tool. Lastly, I assess the findings within a wider research context to identify open questions which might be relevant for future research.

Conclusion. To conclude this thesis, I summarize the obtained results and draw a conclusion on opportunities and limitations regarding the implementation of a web-based cartographic service which facilitates polygon schematization for thematic maps.

2. Background and related work

Aiming for a schematization prototype, which produces schematized territorial subdivisions for their application in thematic maps, it is necessary to first define underlying concepts. This chapter starts with a general view on schematization – what does it mean and when is it applied? – and continues with narrowing it down to schematization in the context of cartography. Along the definition of cartographic schematization, I outline the relation to cartographic generalization, as well as typical characteristics, types of schematization and use cases. Furthermore, I discuss requirements on the design of the base map in thematic maps. Scholars like, e.g. Imhof (1972) and Bertin (1974) define a set of rules on how a (schematized) base map should be designed to achieve readable thematic maps in interaction with other map elements (e.g., the thematic layer). After that, I demonstrate how these requirements or rules regarding the design of thematic maps can be applied in the context of schematization. Furthermore, I briefly discuss preliminaries for using automation in cartography and its purpose. In the last section of this chapter, I examine the advantages and challenges related to cartographic web services.

2.1. Schematization

Schematization is a mean of communication, which serves the purpose of "emphasizing certain aspects and deemphasizing others" (Klippel et al., 2005, p. 1). As such it is part of verbal and visual communication. In the forthcoming sections I describe the concept of schematization and its role for communication, particularly for visual communication. Later on, I refer to schematization in the context of cartography.

2.1.1. Schematization in general

According to Herskovits (1998), schematization itself consists of three major processes: abstraction, idealization, and selection. More specifically, for the visual domain, Kazmierczak (2003) describes schematization as a strategy applied by designers to provide the audience with nothing more but the most helpful (visual) stimuli, facilitating a prompt interpretation by the perceiver.

In his book *The Visual Display of Quantitative Information*, Tufte (2001) is also concerned with schematization in visual communication. He coins the term of *Graphical excellence*, which can be achieved by conveying "complex ideas [. . .] with clarity, precision and efficiency" (Tufte, 2001, p. 13). Hence, this can be accomplished by using the sum of strategies related to schematization: generalization (simplification), abstraction, idealization and conceptualization (Klippel et al., 2005). As one mean to create graphics of graphical excellence, Tufte suggests applying the *Data : Ink ratio* principle. In line with the argument of Klippel et al. on emphasizing important aspects and deemphasizing others, Tufte defines two kinds of ink, present – to some extent – in every graphic: *data-ink*, which holds information, and therefore cannot be masked out, and *redundant non-data-ink*, which does not add (new) information. The latter type (e.g. grids or frames in certain situations) "can be erased without loss of data-information" (Tufte, 2001, p. 93). The proportion of data-ink to non-data-ink is defined as data-ink ratio. The higher the share of the data-ink, the better the graphic can "help people to reason about quantitative information" (Tufte, 2001, p. 91).

Geometric schematization in the domain of graphic and information design is included in pictograms. Otl Aicher's pictograms for the sport disciplines of the summer Olympics 1972 in Munich pose a landmark for pictogram design. The design, even though comparable to the International System of Typographic Picture Education (ISOTYPE) developed by Otto Neurath and Gerd Arntz, is yet more consistent in regard to geometric aspects. Every icon fits into a square and is constructed on the same grid of 45° (Jansen, 2009). Aicher aimed to create a "comprehensive and universal image" of the "universe of sport" (Folkmann, 2013, p. 170). His pictograms of Olympic sport disciplines exemplify the use of simple geometric shapes with the aim of achieving comprehensive visuals. This concept is formalized by Biederman (1987). He bases this on the work by Marr and Nishihara (1978): to this end, he introduces the concept of *geons*. All authors suggest that drawing objects works similar to perceiving and recognizing them: first, the object is broken down into simple geometric shapes, so-called geons. It can then be brought to paper or be identified respectively.

2.1.2. Schematization in cartography

It is widely acknowledged that generalization is inherent to cartography: as maps depict the world in a smaller scale than 1:1, it is necessary to generalize all data before it is displayed on a map. Monmonier (1991, p. 2) calls this process "the white lies cartographers justify as necessary generalization." Consequently, this can even be phrased the following way: "Not only is it easy to lie with maps, it's essential" (Monmonier, 1991, p. 1). This "power of abstraction" can unfold particularly when it is applied to bring non-tangible objects – like social phenomena – on the map (Fabrikant, 2004, p. 39).

Due to the indispensable process of generalization during the map creation, Klippel et al. (2005) argue that every map can be described as schematic. Nevertheless, not all of them are *schematic maps*, taking into account the cognitive meaning of schematization. The question now arises as to what differentiates a merely generalized map from a *schematic* map and how schematization relates to the concepts generalization and simplification.

Reimer (2010) suggests to revitalize the term *chorematic maps* for highly generalized and abstract maps of thematic nature. Such maps are used to explain complex geographic relations in a visually simplified way. The term *chorème* was coined by *Brunet* and his school of geographic thought (Reimer, 2010). Yet this definition seems too specific regarding it purpose to accommodate all schematic maps (see examples Figure 2.1, Figure 2.2, Figure 2.3, Figure 2.4).

Schematization is an extreme case of generalization, "a process which uses cartographic generalization operators in such a way as to produce maps of a lower graphical complexity compared to maps of the same scale" (Mackaness & Reimer, 2014, p. 301). To achieve such a lower graphical complexity, generalization demands to generalize the underlying model as well as the cartographic output (Mackaness et al., 2014; Reimer & Fohringer, 2010). This definition also implies one constraint of this research project: visual aspects of generalization, and particularly schematization as its extreme case, are considered. However, modeling aspects are not discussed.

In his thesis Meulemans sketches out the relation between generalization and schematization. He also considers how they both adopt the concept of simplification, in different ways due to their contrasting intents. Simplification is the mere reduction of detail, i.e., the process of removing points (vertices) to decrease the level of detail. Yet this process is done without any design aspiration. Solely applying simplification does not necessarily result in useful data for mapping. The essential difference between generalization and schematization is therefore the manner of bounding the simplification process: whereas generalization aims to "maximize the amount of detail" while still preserving legibility, schematization aims "to minimize the complexity of the map" (Meulemans, 2014, p. 11) and is limited by a minimum level of functionality. In a topographic map, the mapmaker usually seeks to render a maximum level of detail, i.e., to include as much and as detailed information as possible. However, at the same time, the mapmaker applies generalization in such a way that the map is not overloaded and still legible. For a schematized map, in contrast, the intent is to drastically reduce the level of detail. The reduction of details stops when a certain threshold of complexity is reached. Such a threshold determines a minimum level of complexity at which the map is still functional.

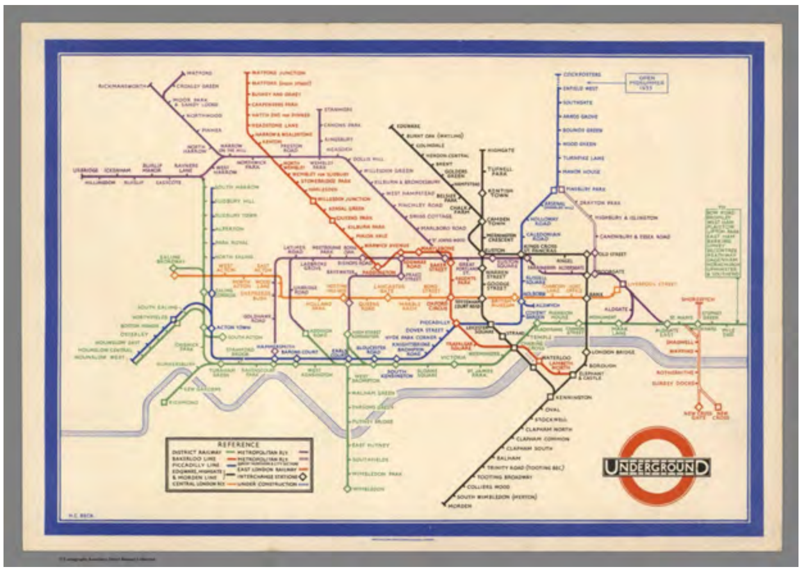


Figure 2.1.: Henry Beck's famous map of London's underground network (Beck, 1933).

Within the cartographic community, schematized maps are closely linked to the characteristic transit maps. *The* transit map, Henry Beck's tube map (Figure 2.1) from 1933, is possibly the most cited and prototypical schematized map. Steadman (2019) states that circuit diagrams, with their highly schematic design, which were invented and standardized only few decades prior, inspired Beck when designing the London tube map. But already centuries ago the idea of schematization was known by mapmakers: an ancient type of maps, which relates to some characteristics of schematization, are the medieval *T-O-maps* (Figure 2.2). A notable example from this kind of maps is also mentioned by Mackaness and Reimer in the context of schematization: Bünting(1581) modifies the traditional way of depicting T-O-maps. He uses a cloverleaf instead of a simple *T* to represent Europe, Asia and Africa.

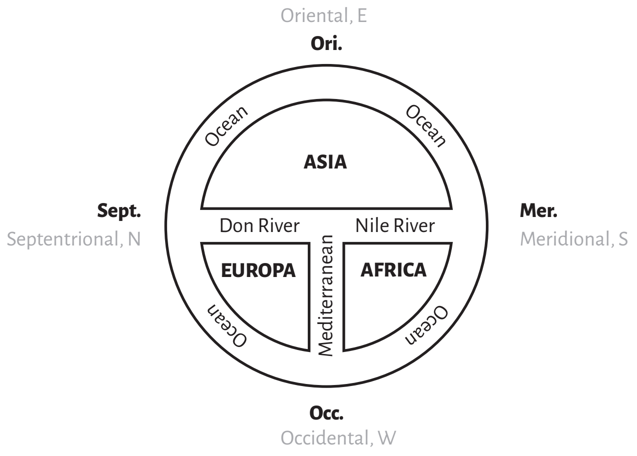


Figure 2.2.: This prototypical T-O-map contains their basic elements, and demonstrate their schematic nature.

The first schematic maps as we define them today were created in the context of the first *cartograms*: Levasseur (1876) replaced administrative boundaries – in this case boundaries of countries – with rectangles. They scaled them by thematic data attributes, e.g. population or budget. Some decades later, Raisz (1944, 1962) continues the cartogram tradition and uses rectangles in a similar manner as Levasseur. However, he shows them in an oblique view. But in contrast to Levasseur he employs the third dimension by extruding the rectangles to blocks. This indicates the thematic data (e.g. population numbers).

Even though it seems like most examples of early schematic maps relate to cartograms, there are also some early examples of schematic maps independent of the idea of cartograms: Arnberger (1966) names a schematic map produced in 1905 by the statistical agency of the German Empire Figure 2.3 as example. It displays the administrative outlines of the statistical units (*Regierungsbezirke* or *Kreishauptmannschaften*) in a highly simplified and schematized manner. It employs the method of choropleth mapping, showing the per hectare crop of wheat per administrative unit for the year 1903.

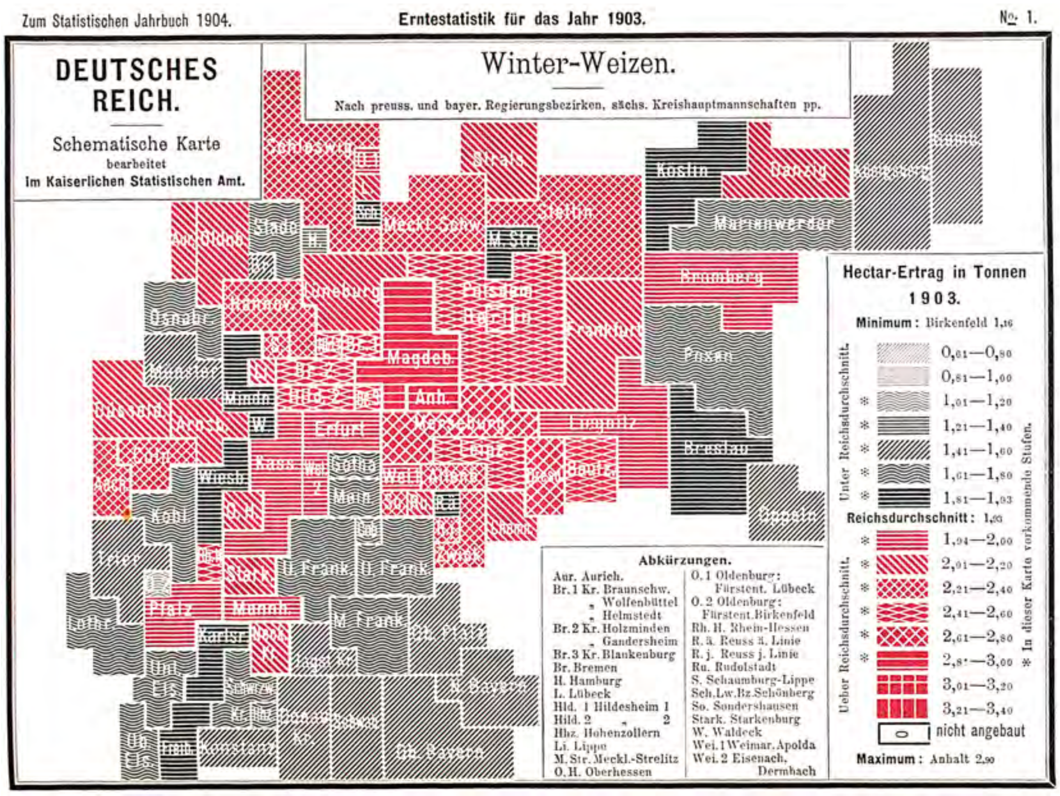


Figure 2.3.: This map was published in the statistical yearbook of the German Empire for the year 1904 and serves as an example for an early schematic map (even titled as such), which is no cartogram. Edited by the Author from *Statistisches Jahrbuch für das Deutsche Reich 1904*.

2.1.3. Properties of schematized maps

Apart from formal definitions of schematized maps, common properties of such maps can be identified to demarcate them from other map types. Properties of schematized maps include low visual complexity (a), use of simple geometric shapes (b), preservation of geographic relations under certain constraints (c) and that the geometric shapes are similar (d) in terms of recognizability, i.e., they match with the map reader's mental picture (Meulemans, 2014). Vujaković (2014) even draws a direct connection between schematic and mental maps: he argues that mental maps or a "person's personal geography" most probably are much more like a highly schematized map than a common topographic map. As usually in cartography, compromises have to be made when aiming for schematization to conform all properties equally, as these properties can contradict each other to a certain degree. This is comparable to the cartographer's dilemma with projections and also poses challenges for implementing schematization algorithms. Nevertheless, these four properties are required to some extent to recognize schematized maps: e.g. a schematized map with a high level of detail, even though the details itself follow strict geometric constraints, can still be received as an exact map (Meulemans, 2014).

The level of detail also leads to possible answers to the question in which cases schematic maps may be better for the reader than conventionally generalized maps. Monmonier (1991, p. 34) recommends, even though not specifically, to use "highly generalized maps" (i.e., schematic maps) whenever "geometric accuracy is less important than linkages, adjacency, and relative position." Even though Monmonier does not explicitly mention schematized territorial outlines here, it is assumed that this is not only true for networks. Later in his book *How to Lie with Maps*, he claims that cartographers can use details to make a map look accurate to sidetrack the map reader. Referring to the example of adding data from a soil map to "a database with more precise information, these data readily acquire a false aura of accuracy" (Monmonier, 1991, p. 38). This phenomenon is identified by Meulemans (2014) as *illusion of accuracy*, to which schematization seems to be a proper remedy: Maps displaying a high level of detail in any of its information layers (base map, auxiliary layers, thematic data) convey a sense of accuracy, somewhat independent of the spatial accuracy of each individual layer. Sometimes, cartographers use this technique "to cover up the use of inadequate source materials or, what is worse, to mask carelessness in the use of adequate sources" (Wright, 1942, p. 9). Furthermore, Wright mentions that accurate and aesthetically beautiful maps are more likely to be trusted (1942). To counteract all these, schematized geographic representations can be purposely used to make it clear to the reader that "the map in question is not a (purely) geographic one" (Meulemans et al., 2010, p. 2). For that reason, Meulemans et al. (2010) even concludes that schematic visualizations are the better choice for all maps which do not require exact boundaries.

Another use case and reason for using schematic maps is of more practical nature than the previously mentioned reasons and closely related to the map making process. It is that certain schematization styles fit well with certain mapping methods: e.g. some schematization approaches generate territorial outlines which match cell-based grid maps (Meulemans, 2016). Grid maps in this context are maps build upon a set of (connected) regular cells – usually triangles, squares, hexagons. All the cells together, which sometimes carry a diagram each, resemble the geographic shape of the depicted area (Eppstein et al., 2013; Slingsby et al., 2010).

Lastly, a widespread application in practice are fare zones of transit maps (see Figure 2.4). However, they are not often mentioned by cartographic literature. It seems that cartographers and graphic designer apply intuitively the principle, mentioned by Meulemans et al. (2010) that polygons of schematic nature complement a network better than a mere generalized subdivision. The latter potentially increase the cognitive load by adding visual clutter.

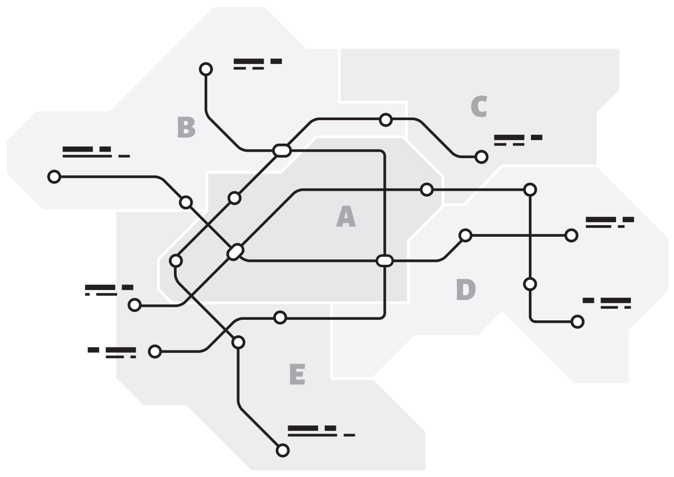


Figure 2.4.: Typical transit map with a C-oriented octilinear schematization of networks (e.g. metro lines) and regions (fare zones).

2.1.4. Schematization types

Meulemans (2014) differentiates several types of schematizations. He distinguishes schematizations based on the type of the schematized object (a) and the style of the resulting schematization (b).

Due to their diverging requirements, it is important to differentiate between the schematizations of networks and the schematization of regions. Hence, two basic schematization types exist, depending on the type of the schematized object: a network in this case is defined as a set of line features, where a set of (usually but not necessarily adjacent) polygon features defines a region. Even though there are maps which combine both, e.g. (Figure 2.4), and the visual output is usually analogical, distinct (algorithmic) schematization approaches need to be applied for networks and regions. For the schematization of networks, the most important topological feature are junctions. These are the intersection of more than two lines. In the typical case of a transit network, the schematization further needs to preserve the order of the stops on a line as well as the junctions (the stops where one can transfer). The length of the line or the relative position of other stops to each other can be heavily distorted. Nevertheless, for the schematization of regions (e.g., the fare zones), the schematization has to be topology- and shape-aware in a much stricter sense. Not only the adjacencies of the regions need to be identical after the schematization. Ideally, also their resulting shapes ought to resemble – to some extent – the original shape. Furthermore, the region area needs to be preserved: significant changes can affect the map's legibility.

Considering the classic geometric categories used in map design – point, line, area – the following question arises: Besides the described schematization of lines (networks) and area (regions), can a point – or point accumulations (Imhof, 1972) – also be schematized? Roth et al. (2011) suggest a typology of 24 generalization operators for multi-scale maps. The following seven operators out of the proposed typology are applicable to point features or result in point features: *collapse, displace, adjust iconicity, rotate, adjust shape, adjust size, typify*. If they are used in a manner that leads to an output of relatively low visual complexity, one could speak of point feature schematization. Widening the scope, regarding two additional categories of dimensions for spatial phenomena as defined by Slocum et al. (2009), 2.5- and three-dimensional map symbols also need to be considered. However, cartographic literature lacks theory on the schematization of a set of points as well as in regard to 2.5- and three-dimensional map symbols.

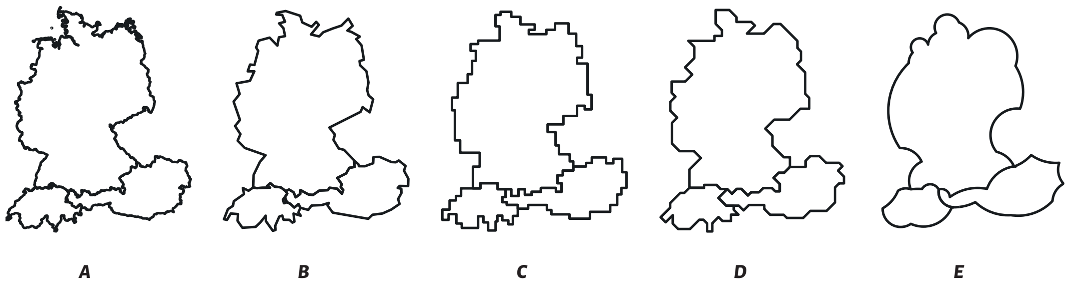


Figure 2.5.: Geometric styles of schematization: ***A*** original, ***B*** irregular schematization (simplification), ***C*** C_4-oriented (rectilinear), ***D*** C_8-oriented octilinear, ***E*** curved. Adapted from Figure 1.5 in Meulemans, 2014 (all four styles were drawn manually).

Another possible typification of schematizations relies on visual appearance, determined through the geometric principles which are applied for the schematization. In literature, the sum of these characteristics is referred to as *style*. Schematization styles can either rest on circular arcs or *Bézier* curves (Heimlich & Held, 2008; van Dijk et al., 2014; van Goethem et al., 2013; van Goethem et al., 2015) or on principles like constraining angles (Buchin et al., 2016), parallelism (Reimer & Meulemans, 2011) or isothetic graphs – a grid, aligned to a small set of focal points (Meulemans, 2014). A prominent and widely used style is the so-called C-oriented schematization, based on constraining the occurring angles. This style is used for the schematization of both networks and regions. However, it is mostly applied to networks in the context of transit maps. A C-oriented schematization only allows certain orientations to occur: C denotes a set of orientations to which all edges of the resulting region or network have to adhere. Typically, this set is regular, i.e., the angles between the individual orientations are equal. Nevertheless, also irregular sets can be used (Buchin et al., 2016). The smaller the number of allowed orientations within C, the stricter is the schematization. Note that the minimum number of orientations is two. The case of a regular C defining only two orientations is called *rectilinear*. Other typical regular sets for C are the *hexilinear* with three and *octilinear* with four orientations (Figure 2.5). Technically, more orientations, regular and irregular, are possible. However, they do not often occur and therefore have no names. Meulemans (2016) states that within C-oriented schematizations, certain approaches trump others: a schematization based on a regular grid benefits the visual appearance by – under certain conditions – preventing visual collapses and enforcing collinear edges as well as edges, whose length is always a multiple of the length(s) of the grid cell (see Figure 2.6). These properties which are visually desired properties for schematization contribute to "a stronger sense of schematization and a more coherent 'look and feel'" (Meulemans, 2016, p. 2).

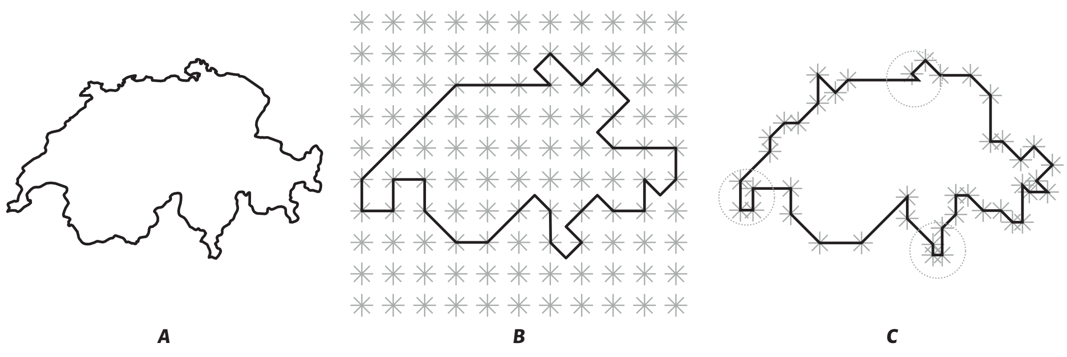


Figure 2.6.: The advantages of map matching over heuristic approaches for C-oriented schematizations are visible when visually comparing them: ***A*** shows the original outline of Switzerland, ***B*** C-oriented schematization using a graph grid, **C** C-oriented schematization, result of heuristic approach. Visual collapses are highlighted with the dashed circle.

A study on the usability of schematization (of networks) has been carried out among others by Roberts and Vaeng (2016). It is assumed that their findings are likely to apply in a similar way for schematization of regions. Nevertheless, the question of which style is appropriate for a specific subdivision depends on the map context: the audience, the task and the input for the resulting schematized map (Buchin et al., 2016).

Having outlined two characteristics, type of the schematized object and schematization style, Meulemans (2014) mentions that there may be different ways of categorizing schematic maps. Reimer (2010) and Mackaness and Reimer (2014) classify schematic maps upon their context of use or the mapmaker's intention. They identify seven types: mental maps or sketches, educational maps, propaganda maps, mass-media maps, schematic or metro maps, chorematic maps, and geodesign maps, which are derived from the map context and intention. Furthermore, they link schematic maps with the concept of persuasive maps, as defined by Muehlenhaus (2010, 2011, 2012, 2013). They describe the most characteristic generalization operators for each map type. Reimer (2010) comments that it is not always possible to allocate schematic maps to one of the proposed categories.

Due to the rather communicational point of view on the map-making process, I consider it of less relevance in the context of this research. Visual and perceptional aspects are this research's focus. This promotes a differentiation of schematizations based on the discussed geometric properties (type of the schematized object, schematization style).

2.1.5. Geometric qualities of schematization

Depending on the specific approach, schematizations have different geometric qualities: they can be either vertex-restricted or non-vertex-restricted, they can preserve area or not, and preserve topology or not (Meulemans, 2014). The principle of vertex-restriction applies in the same extent to the schematization of networks and regions. Area preservation is usually more relevant for regions. However, sometimes, it is also considered in schematization algorithms for transit maps as a particular case of networks. The quality of topology preservation is equally relevant for both networks and regions. However, this applies in different ways: for networks, the vertex adjacencies (e.g. the order of metro stops) is relevant. For regions, adjacencies matter in regard to face-to-face adjacencies. The following section describes each geometric quality in detail.

To *preserve the topology* is probably the most fundamental requirement of schematization approaches and their respective algorithms: in a cartographic context, usually only topologically correct results are useful. In contrast to the area, which in some cases is allowed to change or is even changed intentionally, the topological relations between the spatial entities must not be changed while schematizing. This concerns particularly adjacencies: the original and the schematized region have to represent the same face-to-face adjacencies (Figure 2.7). An edge which is incident to two faces in the original data needs to be incident to the exact same two faces after schematization. Nevertheless, the boundary's shape between such face pairs can be altered. (Meulemans, 2014).

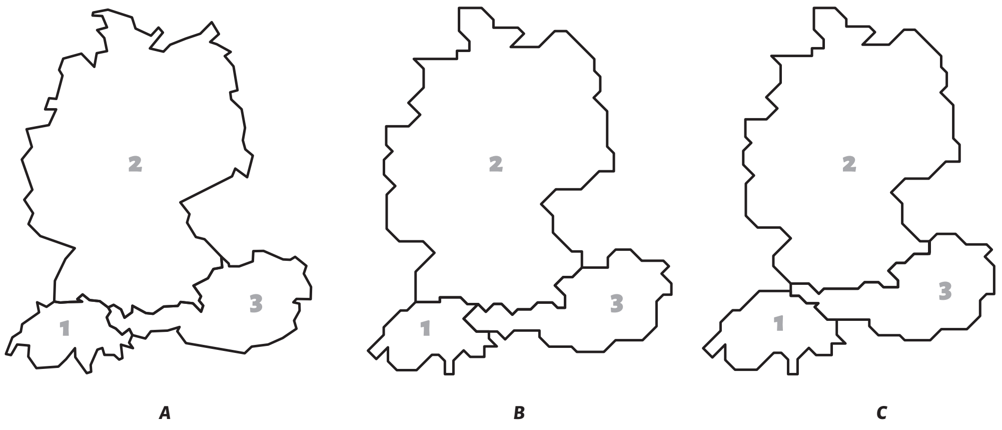


Figure 2.7.: ***A*** In the original input polygon 1 and 2, 1 and 3, and polygon 2 and 3 share one border each. The same adjacencies are present in ***B***, making it a topology preserving schematization of A. ***C*** shows a schematization where polygon 1 and 2 do not have any edge in common, thus a schematization which does not preserve topology.

Not all schematization approaches guarantee an *area-preserving* result. In the case of an area-preserving schematization, the area of each polygon remains the same relatively to the other polygons within its region. Nevertheless, this characteristic is particularly important when data are provided in an area-preserving projection (Buchin et al., 2016). Furthermore, Meulemans (2014) states that schematizations which cause major distortion of the subdivisions can conflict with the map reader's mental image. Schematizations which do no longer resemble the original geographic relations at least to a minimal extent cannot not be considered useful for cartographic purposes.

Every schematization is a simplification process. Therefore, the number of vertices in the resulting subdivisions is always smaller than in the original subdivision. The simplification is called *vertex-restricted* if the vertices of resulting subdivisions are a subset of the original subdivision vertices (Figure 2.8). No new vertices were added or existing vertices moved throughout the schematization process. Instead vertices which were less important in terms of the intended geometric constraints were removed to achieve a lower visual complexity. Meulemans argues that vertex-restricted simplifications are less complex in respect to their implementation because they are less flexible. Well-known examples for vertex-restricted simplification algorithms are the Douglas-Peucker and Visvalingham-Whyatt algorithms (Douglas & Peucker, 1973; Visvalingam & Whyatt, 1993). The concept of vertex-restriction is closely tied to the schematization style, as defined above: certain schematizations, like C-oriented schematizations, require non-vertex restrictions. Meulemans

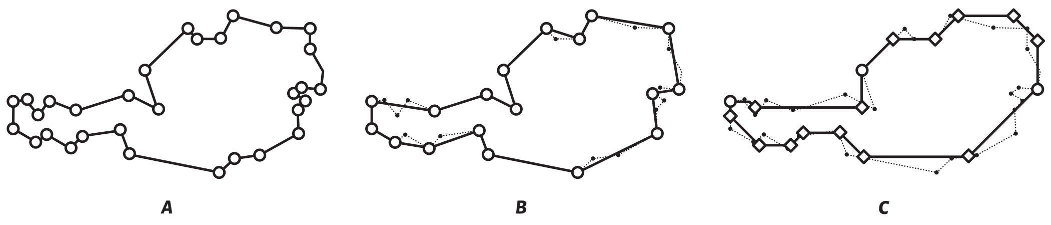


Figure 2.8.: ***A*** shows the input with 32 vertices. ***B*** A vertex-restricted schematization with 18 vertices (deleted vertices are shown as a black dot). ***C*** A non-vertex-restricted C-oriented schematization with 16 vertices (the moved vertices with a new location are shown as diamonds). Adapted from Figure 1.10 in Meulemans, 2014.

2.2. Thematic mapping

Some use cases for schematic maps mentioned in cartographic literature, relate to thematic maps. As the prototype aims to generate schematized regions for the use in thematic mapping, it is necessary to firstly assess the requirements for thematic mapping determined by scholars. These heuristics can be used to evaluate the schematization results.

Different definitions for thematic maps exist (Imhof, 1972; Raisz, 1948; Schulz, 2014; Slocum et al., 2009; Tyner, 2010). Neumann's multilingual encyclopedia, based on the work of International Cartographic Association (ICA)'s former commission II on "Definition, Classification and Standardization of Technical Terms in Cartography", defines a thematic map as "A map designed to demonstrate particular features or concepts. In conventional use this term excludes Topographic(al) Maps [. . .] in the strict sense" (Neumann, 1997, p. 447). This definition reflects a discussion within the cartographic community. It implies to split the continuum of all maps into two halves: thematic and topographic maps. This differentiation is based upon a simple, yet only loosely defined characterization. Such a distinction results from a practical need for a differentiation, which does not necessarily follow a strict logic (Imhof, 1972). Therefore, even such a binary concept cannot categorize all maps (Schulz, 2014); every thematic map draws on topographic elements. Vice versa, every topographic map is to some extent thematic (Imhof, 1972). An example for maps which belong to both thematic and topographic maps or are located somewhere between these poles, according to Imhof (1972), are hiking, city and ski maps. Also, no consistent system further differentiates thematic map types. One example, for the fuzzy terminology regarding types of thematic maps, is the term *statistical maps*. Whereas some scholars (Imhof, 1972; Schulz, 2014) consider *statistical maps* as a subgroup of thematic maps, others (Raisz, 1948; Slocum et al., 2009; Tyner, 2010) use the term as a synonym for thematic maps. Despite the numerous logical constructs on how to categorize map types and how thematic maps and other map types correlate, statements on the content and the purpose of thematic maps are mostly congruent. The purpose of a thematic map is to "to display the spatial pattern of a theme or attribute" (Slocum et al., 2009, p. 1). It contains predominantly non-topographic, but spatial phenomena, which can be tangible but also intangible. Examples are relations and hypotheses (Imhof, 1972). This may be one reason why thematic maps "are the primary map type seen in newspapers, journals, reports, and textbooks" (Tyner, 2010, p. 7).

Ding and Meng (2014) assess the purpose of thematic maps from another perspective. The authors compare them with scientific visualization: they argue that the main aim of a thematic map is "the communication of known information from the map maker to the map user via information abstraction" (Ding & Meng, 2014, p. 25). Thus, the mapmaker needs to abstract using generalization operators, semantic aggregation or thematic classification. This process of abstraction is an important part of thematic map design. As a consequence, the map is — in the best case – easy to comprehend and less complex than scientific visualizations. Scientific visualizations, in contrast, do not intend to convey a certain message, but rather to provide the user with a tool to explore comparatively raw data. In this case, data accuracy or a high level of detail is indispensable. Hence, scientific visualizations usually require more effort from the user (Ding & Meng, 2014).

Having defined the general notion of thematic maps, their content and purpose, in the following, the individual elements composing a thematic map are discussed. Slocum et al. (2009) define the following map components: the frame line, neat line, thematic symbols, base information in the mapped area, inset maps, a title and possibly a subtitle, a map legend, data sources, a bar scale, a north arrow and a graticule. Dent et al. group those components differently, differentiating between "the base map, the thematic overlay, and a set of ancillary map elements" (2009, p. 10). Of the above mentioned components, in the context of this research, the most relevant is the topographic base information. However, there is a mutual effect between the topographic base elements of a thematic map and its thematic elements. Hence, the following ideas and concepts collected by scholars need to be seen in relation to the entire map (including the ancillary map elements), and particularly in the context of the component holding the thematic symbols. The purpose of the base map is to add spatial context to the thematic information, i.e., to set the stage for thematic symbols: it should provide the user with a basic sense of orientation. Still, the geographic accuracy is a fundamental requirement for every thematic map, even though it is less important and the required level of detail much lower than for topographic maps (Schulz, 2014). Otherwise, tables or statistical charts that do not (visually) acknowledge the spatial distribution of phenomena are considered easier to read and therefore more efficient (Imhof, 1972).

Every map and its context call for an individually designed base map. It is generally agreed that creating a well-designed base map is a challenge. It needs to balance between *too much and too few* when selecting relevant content as well as when designing it in a subtle yet comprehensive style. Imhof comments this issue: "Such concurrent *as much as possible and as little as possible* inheres the problem of choice" (Imhof, 1972, translated by the author).

This challenge is closely related to how the reader perceives the map: according to the aforementioned definition by Neumann of thematic maps, communicating particular features or concepts, those features and concepts ought to be designed such that the user can rapidly and easily perceive these features. To this end, it is necessary to design the base map visually inferior to the thematic symbols: "[. . .] the reference points chosen can be represented by very simple, light signs; their visibility should never be equal or, even worse, superior to the content of the information." (Bertin, 2011) Hence, the relation between base map and thematic map layer can be described as follows: the base map should support and complement the depicted topic as much as needed and at the same time pollute and interfere with the thematic component as little as possible (Imhof, 1972).

Typically, the topographic base map contains administrative boundaries and hydrographic features, and in some cases also the relief. For the latter, a simple shaded relief in grey-scale is used, which is more powerful in conveying the terrain than contour lines, particularly in small-scale maps (Imhof, 1972). Returning to the concept of balance, it is important to carefully select the content of the base layer – to show enough to provide spatial context, and at the same time not more than necessary to avoid visual clutter. This aspect relates strongly to generalization. Dent et al. (2009) argue that, as most thematic maps are in small scales, it is particularly important to apply cartographic generalization operators deliberately and in an effective way.

Slocum et al. (2009) and Tyner (2010) formalize what Imhof refers to with the term inferior: every map employs an intellectual – and in the best case a corresponding – visual hierarchy. The latter is important, as it leads the map user's attention and aims to convey the map's main message effectively and efficiently. Slocum et al. (2009) suggest a simple generic intellectual hierarchy, on which the visual hierarchy relies: on top – and therefore most important – are all graphics elements immediately concerned with conveying the thematic information. These are either symbols or labels, followed by title, subtitle and legend, and only then by the base information. On the bottom of the intellectual hierarchy is mostly what Dent et al. defines as "ancillary map elements" of thematic maps. These are – if applicable – scale bar and north arrow, data sources and elements like frames and neat lines. Accordingly, an important aspect of visual hierarchy in regard to thematic mapping is to make the base information salient and to mute topographic information (Dent et al., 2009).

The most effective mean to establish a visual hierarchy is to apply the concept of contrast, which is possible in several ways (Slocum et al., 2009). "Visual contrast leads to perceptual differentiation" (Dent et al., 2009), which makes it an effective mean to establish a visual hierarchy. Types of contrast in cartography are line contrast, texture contrast, value contrast, variation of detail, and color contrast (Dent et al., 2009). Bertin (1974) presents a more general approach – the rules of legibility, including similar but more conceptual ideas of how to establish a visual hierarchy in thematic maps. These three rules rely on the term of visual variables. Therefore, they are discussed and illustrated in detail after briefly introducing visual variables.

Because all thematic maps use a geographic or topographic layer, the base map, in the most simple case a thematic map has two components: the geographic layer and a thematic overlay. The geographic layer already uses both of the planar dimensions. Visual variables are applied to make the thematic component stand out from the geographic layer (Bertin, 1974). Bertin coins the term of visual variables. Many scholars have referred to this concept, but improved or clarified the individual variables. Some even added new variables (Tyner, 2010). Whereas not all of these variables are equally accepted in literature, the original graphic variables of Bertin are still common. These eight variables include: the two dimensions of the plane and in addition size, value, texture, color, orientation, and shape. With these variables, cartographers can display (additional) thematic information via map symbols (Dent et al., 2009).

The visual variables depend on the geometric categories to which they are applied. Point, line, and area (or polygon) are the geometric categories, or map symbols, available for map design (Bertin, 1974; Imhof, 1972; Monmonier, 1991; Tyner, 2010). Other scholars also consider volume (Kraak & Ormeling, 2010) and time (Dent et al., 2009) as dimensions of map symbols. The dependence on the symbol's geometric category becomes evident when comparing how to apply the variable of size to a point and a line symbol. For a point symbol, this variable is applied by simply scaling the symbol in the two available dimensions on the plane. In contrast, for the line symbol, the same variable is applied by changing the line width.

Moreover, the visual variables differ in their capacity to convey different types of information. To this end, Bertin (1974) defines three component levels as complement to the *invariant*. They can have three levels of organization: qualitative (or nominal), ordered, and quantitative. A similar concept, defining measurement scales for information, is commonly adopted by scholars in the field of cartography: the nominal scale (values of equal importance), ordinal scale (values of different importance), interval scale (values of different importance, e.g., they can be ordered and the distance between single values can be determined) and the ratio scale (values of different importance, and all values can be related to each other) (Kraak & Ormeling, 2010).

To explain how the visual variables are applied effectively, Bertin determines so-called rules of legibility. They can be interpreted as guidelines on choosing the combination and application of visual variables to provide contrast (Bertin, 2011). He compares the rules of legibility with the way a speech is delivered to the audience. It does not matter whether the speak is well-written, i.e., understandable in terms of logic and grammar, or not. It will be hard to grasp if e.g. the speaker's pronunciation is poor or their voice too low. The three rules of legibility for diagrams, networks and maps include:

1. The rule of *graphic density*, defined as "an optimum number of marks per cm^2" (Bertin, 2011), which should neither be too dense nor too sparse (see Figure 2.9).

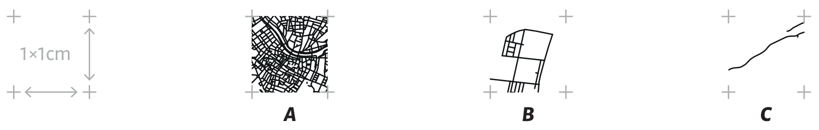


Figure 2.9.: ***B*** shows an optimum number of marks per cm^2, whereas ***A*** and ***C*** are either too dense or too sparse.

2. *Angular separation*, which allows to identify angles depending on the angle itself and on the length of the two lines enclosing the angle (see Figure 2.10). i.e., angles close to 0° or 180° are less legible, as are those which are enclosed by short lines. It can be described as contrast within the two-dimensional plane.

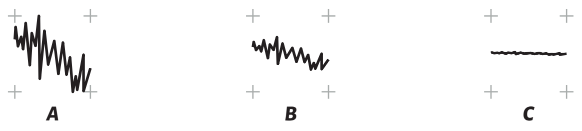


Figure 2.10.: ***B*** shows a line with a high angular legibility as the lines enclosing the angles are long and the angles are not as close to 0° or 180° as in ***A*** and ***C***.

3. Lastly, the rule of *retinal separation* denotes the idea that a minimum of contrast needs to be established. This enables the reader to differentiate between important and less important or between figure and ground. Bertin denominates this the "elevation 'above' the plane" (Bertin, 2011, p. 62). One among several aspects of retinal separation is the relative amount of black used for background and foreground (see Figure 2.11).

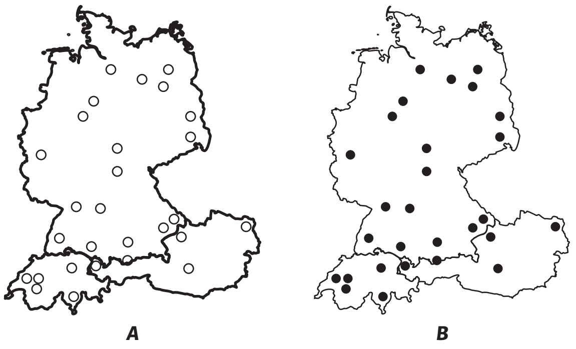


Figure 2.11.: One aspect of retinal separation is the amount of black used for background and foreground: in ***A***, the majority of black is applied to the actual background, making it more salient than the foreground information. The relation of black use for fore- and background is correct in ***B***.

For none of these rules a simple threshold can be given, as legibility depends on context. As an example, in respect to the rule of graphic density, the optimum number of signs per cm^2 depends, according to Bertin on "the number of different images (length of the component), the utilization of differences in implantation, the retinal variables employed, and the reading habits of the individual" (2011, p. 176).

How schematization of polygons for thematic mapping can be used to achieve legibility, according to Bertin, is discussed in the following.

2.3. Schematized regions in thematic maps

This section aims to answer **RQ A**. The underlying concepts are properties of schematizations and best practices, or heuristics, for designing a thematic map's base layer. In this section, use cases for schematization in the context of thematic mapping are illustrated and discussed based on the rules of legibility. I combine schematized regions with thematic overlays, taking into account different schematization styles and properties as well as thematic mapping methods.

To this end, I created a set of mock-ups for thematic maps: the polygons in these maps resemble a fictitious set of territorial boundaries, e.g. a city with its districts. The mock-ups feature alike fictitious thematic data of different types, which are symbolized using several thematic mapping methods: sparklines (Figure 2.14) and bar charts (Figure 2.15) represent a time series; a dot density map (Figure 2.13) and proportionally scaled symbols (Figure 2.15) show spatial distribution. The maps aim to implement and demonstrate recommendations of scholars (e.g. Bertin; Imhof; Slocum et al.) regarding thematic map design. Originally, I planned to create the C-oriented schematized layer of these mock-ups based on results obtained with the discussed proof-of-concept prototype. However, this is impossible due to implementation shortcomings. Instead, manually generated schematizations were used for both the C-oriented and curved schematized territorial outlines.

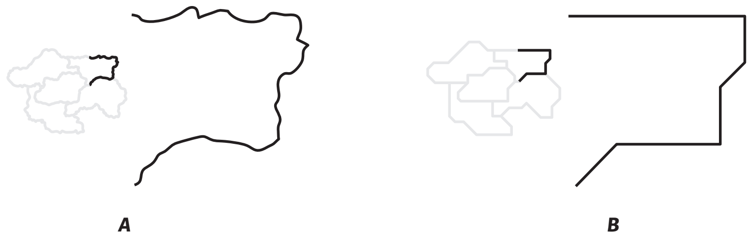


Figure 2.12.: Increased angular legibility by schematization: the magnified polygon boundary of the schematized region ***B*** expose fewer, but longer edges than the generalized region ***A***. Furthermore, angles close to 0° or 180° do not occur.

Generally, cartographic schematization can increase legibility, as defined by Bertin: the rules of graphic density, angular separation and retinal separation. Figure 2.12 illustrates how schematization can affect angular legibility: the C-oriented schematization on the right exposes fewer but longer edges, which enclose angles relatively close to 90°, than the conventionally generalized region on the right. Bertin (1974) claims that these qualities improve angular legibility. Note that this is less applicable for more flexible C-oriented schematizations.

In the following, I illustrate the effect of schematized regions on legibility aspects. These illustrations consist of three maps each: the first map is based on a merely generalized region, the second on a C-oriented and the third on a curved schematized region. They are visually compared and set into the context of Bertin's rules of legibility. This allows to identify situations where schematization is beneficial or unfavorable when seeking maximum legibility. Nonetheless, it is important to note that though schematization can help to avoid certain pitfalls, by itself it does not guarantee higher legibility. Considering the map context, establishing a set of rules which are universally valid is impossible. Legibility always depends on several factors: the content and design of the thematic overlay, the applied thematic mapping methods, the application of graphical variables, the map's scale, and altogether the map's context, defined by the map maker's intent or the purpose of the map. To reduce these factors' influence on the following examples and to show the effect as isolated as possible, fewer visual variables are applied (using e.g. monochrome depiction, no hues, and no labels). Despite this limitation, the following examples illustrate relations between schematization and legibility.

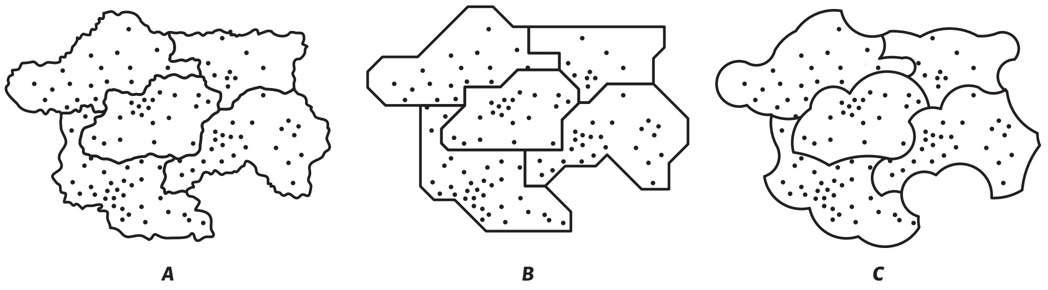


Figure 2.13.: Schematization can contribute to a relatively low visual complexity, particularly in the case of a thematic overlay of high visual complexity. The overall visual complexity of the generalized region in ***A*** is proportional to the graphic density. ***A*** exceeds the C-oriented ***B*** and the curved schematization ***C*** in complexity.

The first example, Figure 2.13, illustrates how schematization can be used to reduce visual complexity (van Goethem et al., 2014) within a thematic map: the dot map, as thematic overlay, intentionally exposes a high level of detail. As such, it benefits particularly from a highly simplified or even schematized geographic layer, as compared to other mapping methods. Nevertheless, this approach may be problematic in an automatic schematization, which does not take into account the position of additionally provided point features: through schematization, these point features could eventually lie on different sides of the face-to-face boundaries, i.e., they are located within other polygons than they originally were. This is particular for dot density maps (Tyner, 2010), where it is statistically relevant to which feature a dot belongs.

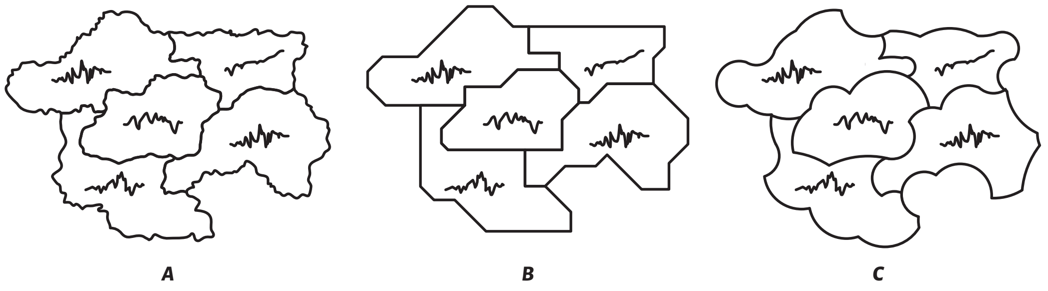


Figure 2.14.: The contrast based on the detail variation is increased by schematizing the region's boundaries, as in ***B*** and ***C***, compared to the simply generalized region in ***A***.

Seeking for contrast is "a major goal of the designer" (Dent et al., 2009, p. 213) as it enables perceptual differentiation. Scholars (Dent et al., 2009; Tyner, 2010) mention detail variation as one mean to establish contrast. Attention is always attracted by detail: "the reader's eye will be attracted to those areas of the map with the most detail" (Dent et al., 2009, p. 214). For thematic maps, it is desirable to guide the user's attention towards the "unpredictable" (the thematic overlay), and not the "predictable" (the geographic layer) (Bertin, 1974, 180f). This principle is applied in Figure 2.14 by using sparklines. Whereas for the generalized region, the contrast in detail to the thematic overlay is low, both the C-oriented and the curved schematization exhibit a contrast between the highly geometrically simplified region boundaries and the area-related sparklines.

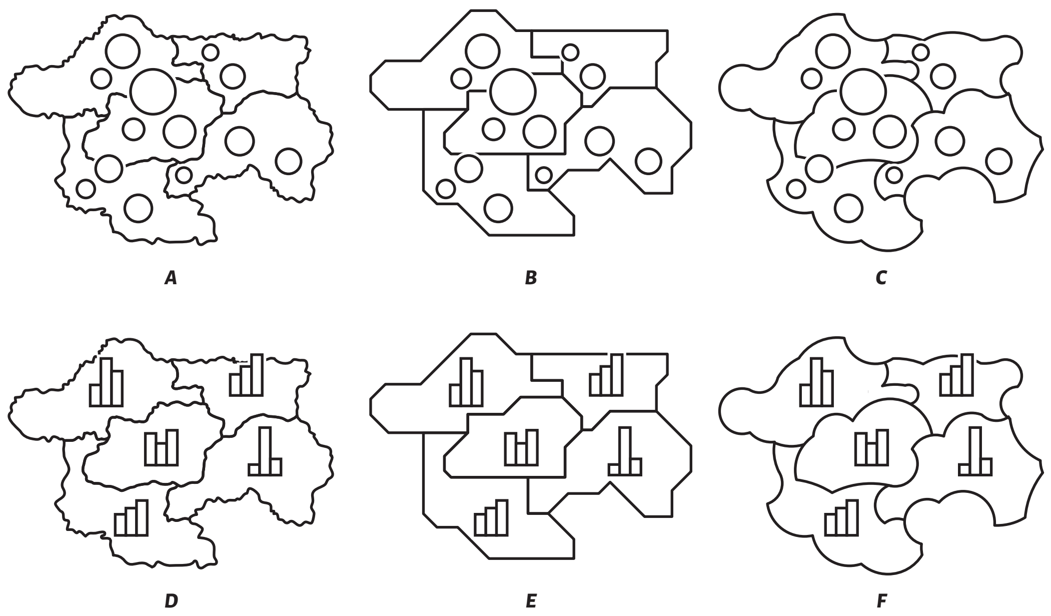


Figure 2.15.: Combinations of generalized and schematized regions with distinct mapping methods result in distinct visual contrast. The straight lines of the C-oriented schematization (***B, E***) contrast well with the curved overlay shapes. Likewise, the arched boundaries of the curved schematization (***C, F***) contrast with the straight bar charts lines.

Regarding the retinal separation, shape differences are another mean to establish contrast. Leaning on the perceptual grouping of elements that exhibit a similar shape (Dent et al., 2009), the schematization style and the shapes used in the thematic overlay impact the visual contrast between the base layer and the thematic overlay. This is illustrated in Figure 2.15: the first map series shows the same point-related proportional symbol layer using circles. The second series employs the thematic mapping method of area-related bar-charts. Whereas the circular proportional symbols contrast well to the straight lines of the C-oriented schematization (***B***), they do not exhibit a high shape contrast against the arched boundaries of the curved schematization (***C***). Similarly, the rectilinear bar charts contrast the curved schematized region boundaries (***F***) better than the octilinear schematization (***E***). At the same time, this figure demonstrates that a schematization does not always increase map legibility: the visual contrast of the circular proportional symbol against the generalized region boundaries (***A***) seems higher than against the curved region boundaries (***C***). Nevertheless, in the case of a detailed rendered geographic layer (***A***) or (***D***), the map reader's attention will be, according to Dent et al. (2009), attracted by the high level of detail applied to the regions' boundaries. However, the boundaries pose *the predictable* visualization component. Such a contrast might result in muting the in fact more relevant thematic overlay, *the unpredictable*.

2.4. Automation in cartography

Generally, automation in the field of cartography means the "the operation or control of equipment, a process, or a system by a machine rather than by hand" (Buckley & Watkins, 2007, p. 1). More specifically, for schematization, automation aims to reduce the amount of human intervention in the process of schematizing geographic data.

Scholars argue that automatic generation of schematized maps facilitates speeding the design process up (Burghardt et al., 2014). Thus, a great number of schematizations, even based on different types of input, can be generated in comparatively little time. Tobler (1959, p. 1) was proved right when he forecasted on cartography that "Automation, it would seem, is here to stay." The aspect of automation to schematization gained even more relevance with the rise of digital, interactive maps, which regularly need updates. The efforts put into the development of automated schematized transit maps show the importance of automation for such maps (Dubrau & Bagel, 2016). Avelar (2002) and Meulemans (2014) state that most schematized maps are still drawn by hand, which requires a skilled cartographer or designer and a certain amount of time. Automated generation would not only increase the speed of creation, but may also result in maps of higher quality. On the other hand, Tobler seems to still be correct with his assumption that automated cartography cannot overcome certain problems when the artistic aspect of cartography is concerned (Tobler, 1959). This is particularly true for schematization. For example, it is not easy to define a good schematization: even though the design criteria for schematization can be defined precisely, the result is largely a compromise on one of the constraints (Mackaness & Reimer, 2014).

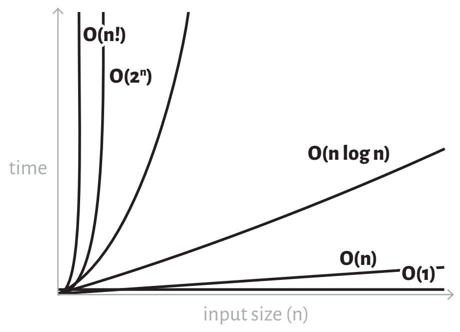


Figure 2.16.: Running time of algorithms described depending on input size.

Not all automation processes are equally efficient. To evaluate how efficient algorithms solve problems and to make them comparable, their efficiency (running time) needs to be described independent from hardware constraints. One possibility to make them comparable is therefore to compare input and the resulting run time. In the case of region schematization, the input varies in the number of edges (and vertices). The complexity is formalized based on how much the computational cost of the algorithm grows depending on the input. If the input considered exceeds a certain size, only the order of growth of the running time is crucial for determining whether an algorithmic approach is more efficient than another. This formalization, called *asymptotic efficiency*, is implemented by defining a function. On its X axis, the input size can be found, and on the Y axis, the algorithm's running time. The steeper the curve, the larger is the growth of running time, i.e., the less efficient is the algorithmic approach in respect to the input size. There are different notations used in the domain of computer science to the describe this function. It indicates an algorithm's computational complexity in respective to the input: the Θ notation indicates the upper and the lower asymptotic efficiency for a function. The Ω notation deploys the asymptotic lower bound for the best-case running time, whereas the O notation uses the upper bound for the worst-case running time. It is important to note that, considering e.g. the O notation, defining the upper bound for the worst-case is not equal to the actual running time. In practice, the algorithm's running time could be theoretically – depending on the input characteristics – significantly shorter. Nevertheless, using the worst-case running time has a series of advantages over examining the average or the best-case running time. Firstly, certainly the algorithm does not take longer. Furthermore, depending on the algorithm and the application context, the worst-case running time may occur regularly. Lastly, empiricism shows that for many algorithms, the running time of the average case is close to the worst-case running time (Cormen, 2009). Hence, the O notation, indicating the worst-case running time, is a meaningful criterion in the context of this research. It is therefore used in Table 4.1. Figure 2.16 shows algorithm running times depending on the input size.

2.5. Cartographic web services

Mapshaper is a web-based generalization service and guides parts of this research project. Reasons for creating such a tool as a web service and not as a desktop app are manifold and root in the nature of web environments. A global approach (Djordjevic, 2017) is inherent to web services: theoretically, anyone with access to the internet is a potential user. Typically, due this global take related to web services, they are easy to use, as the targeted audience is broad. Thus, it is not meaningful to create a highly sophisticated tool for experts only. Another advantage of web services is that maintenance is relatively simple: they are upgraded in a centralized way, i.e., the individual user does not trigger the update. Just by deploying the new version, it is immediately accessed when a user requests the respective Uniform Resource Locator (URL). In addition, the costs in respect to the average user number are usually lower when compared to desktop apps. Nevertheless, one of the largest advantages and another reason why the development and maintenance of such services is relatively simple and cheap is the following: normally, they run on several browsers, independently from the operating system the browser is installed on (Djordjevic, 2017).

At the same time, the web environment also poses challenges to the development of services. They are evident when considering their characteristics as defined by Murugesan (2008): the requirements are considered volatile. The broad user group also implies that this group is inhomogeneous in terms of requirements, goals, and needs, e.g. languages. Another challenge is the management of content provided by the user. This is even the core element for some cartographic services. Moreover, Murugesan mentions the demand for high aesthetic appeal, security, privacy needs, as well as a variety of hardware (e.g. different screen sizes, internet coverage and bandwidth) and software (e.g. different browser clients) setups. Particular to the web environment is also the quick dissemination and adaption of new technologies. This leads to generally short development time frames (Murugesan, 2008).

(2006), the initiators of the Mapshaper project, refer implicitly to some of the mentioned characteristics when arguing why they developed Mapshaper as a web service. By using a web service, they do not require certain hardware capacity at the user's end, but can rather rely on servers. They tried to reduced the therefore necessary data transmission: only already compressed files are transmitted. Furthermore, they implemented a remote file storage, accessible by for registered user. This allows users to access the generalized data remotely. Additionally, the service is always live and easy to upgrade. They also mention the advantage of platform independence over conventional Geographic Information System (GIS) and design software. To meet the users' expectations in regard to the GUI, they mention the aim to display the map as prominently as possible. In contrast, they use a minimal amount of space for the tool itself. Also, they explicitly advise to decrease the amount of visual clutter to "create a minimal, uncluttered work environment" (Harrower & Bloch, 2006, p. 26).

As the web and corresponding technologies have undergone rapid developments over the last decades, Mapshaper as a service has significantly changed. At this writing (2021), the tier-model, i.e., differentiation between regular and registered users, was discarded. Currently, every user can use the software without constraints. The most important changes, beside the apparent revision of the

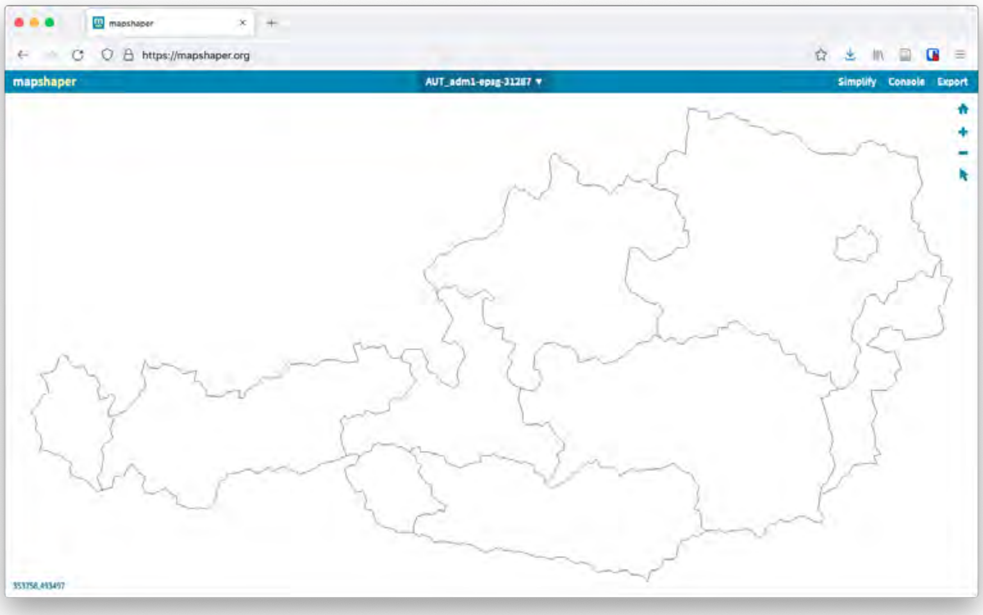


Figure 2.17.: This screenshot shows the interface of the simplification tool Mapshaper's current version.

GUI (see Figure 2.17), are from a technical point of view the move from a *Flash*-based implementation towards an implantation relying on pure JavaScript (JS) and is that in the current version, no data is transferred. The entire computation occurs in the user's browser, which does not only satisfy security concerns but also allows for generalizing files of significantly larger size. To meet requirements of different users, they now provide two interfaces, the original GUI, even though it also has changed since the first release, and supplementarily a Command Line Interface (CLI). Yet, the underlying motivation and reasoning for using a web service over a standalone application remain the same.

3. Methodology

The following methods are applied to meet the research objectives: literature review, requirement engineering, prototyping and user testing. Firstly, I offer an overview on the specific methods applied to answer the respective research questions. Then, each method is outlined in detail. Lastly, I describe how each method was adopted in the context of this thesis.

Review literature for **RQ A, B**. A literature review has been conducted to determine best practices on designing the geographic layer for thematic maps and their relationship to characteristics of schematized maps. Likewise, comparing approaches to automated region schematization is based on a literature review. To decide which of these algorithm to implement, each is briefly discussed. This discussion is based on the following criteria: geometric characteristics, computational costs, and flexibility. The chosen algorithm is then implemented in the prototype.

Assess requirements of such a tool for **RQ C**. To systematically plan and implement the schematization tool prototype, a requirement engineering process was performed. The requirement engineering framework proposed by Wiegers and Beatty (2013) serves as compass. The reasoning for using such a framework is to come up with a precise idea of the requirements already early in the process. These should consider both the user's as well as the developer's needs and account for technical constraints given by the application's environment. The framework is extensive, and therefore – for this comparably small project (low number of stakeholders involved) – simplified and adapted to the projects needs.

Implement a prototype based on an existing algorithm for **RQ D**. A prototyping approach is adopted to implement a first version of the outlined product. The resulting prototype is needed for further evaluation: the specified requirements are iteration validated based on this proof-of-concept prototype. Additionally, possible technical constraints or conceptual shortcomings can be detected early with such prototypes.

Requirement Verification **RQ D**. Ultimately, the prototype is evaluated with the help of the specified software requirements, as identified in RQ C. A requirement verification, determines to which extent the prototype meets the specified requirements. Likewise, the in the scope of this thesis created mock-up prototype needs to be evaluated in a qualitative user study, based on the *thinking-aloud-method*. However, it was not yet conducted due to time constraints. This suggested study will examine the usability of the prototype. It ought to reveal shortcomings, particularly of the GUI. With that, it is considered to be part of the requirement verification. See Appendix C for the proposed user study.

3.1. Requirement engineering

Requirement engineering is a crucial process of software development. Software which undergoes this process before developing starts is, according to Thayer and Dorfman (1997), more likely to meet the product expectations. It is possible that needs of the user and other stakeholders are not considered throughout the development process. This can result in irrelevant or unsatisfying software systems. Requirements engineering helps to identify problems at an early stage. It can thus prevent high costs for fixing errors at a later stage (Paetsch et al., 2003). As every product and its development context are different, requirement engineering works differently as well: there are multiple ways to conduct it.

More complex projects, e.g. with an increasing number of involved stakeholders or features, increase the importance of a proper requirements engineering process. Dependent on the scale of the project, several stakeholders can be involved: manager, product manager, business analyst, developer, and tester (Wiegers & Beatty, 2013). Independent of its complexity or its context, typical steps of requirements engineering include elicitation (the process of discovering requirements), analysis and negotiation (the process of sorting, evaluating and deriving new requirements), documentation (the process of organizing and presenting requirements), validation (the process of conforming whether a system build upon these requirements still meets the expectations towards the system), and management (Paetsch et al., 2003). Wiegers and Beatty (2013) suggest to consider requirement management as a separate subdiscipline of requirement engineering at the same level as requirement development. In this case, the development itself embodies the first four of these steps: elicitation, analysis, documentation, and validation. Considering the resulting information of these steps, they can be structure roughly into three levels. These are typical for a software development process: business requirements, user requirements, and functional requirements. Each level of requirements holds a different kind of requirement information, usually ordered from general to specific (Wiegers & Beatty, 2013). These three levels are discussed in the following.

3.1.1. Business requirements

The topmost level holds the *business requirements*. Their main purpose is to point out the *why* – What is the motivation for creating the piece of software? Wiegers and Beatty suggest to record this level's information in the *vision and scope document*. This document does not only describe the business requirements itself, but can also specify the project's scope and limitations further. It also provides the business context. Thus, creating such a document usually involves economic and marketing concerns.

The business requirements depict, amongst others, opportunities, objectives, vision risks as well as assumptions and dependencies. Another important part is the vision statement. A concise way of generating a vision statement was suggested by (Moore, 2014). It was adapted by Wiegers and Beatty: this version uses the keywords *for, who, the, is, that, unlike, our product* to phrase a paragraph that includes all relevant information.

3.1.2. User requirements

The next level, the *user requirements*, concerns most user aspects and holds more specific requirement information than the business requirements. Its requirement information are based on the *vision and scope document* and summarized in the *user requirements document*. In contrast to the business requirements, the user requirements describe the *what*: the features the system offers to the user, or "product attributes or characteristics that are important to user satisfaction" (Wiegers & Beatty, 2013, p. 9). Use cases and user stories are among the most popular techniques to show user requirements. Both need a preliminary definition of possible user classes. Different user types may have different requirements. In the best case, this information is based on interviews or workshops conducted with actual (or future) users of the product.

3.1.3. Functional requirements

Functional requirements pose the lowest and with that the most specific level. The requirement information of this level are based on the *user requirements document*: this level contains the *functional requirements*, describing how the product operates in specific situations. If these requirements are met, the high level business objective is met. Furthermore, they explicitly define task packages for the developer, which have to be implemented so that the business requirements are satisfied. Usually, they are listed in the form of "shall"- and "if"-statements.

Nonfunctional requirements

The *functional requirements* are influenced and limited by *nonfunctional requirements* and *system requirements*. Thus, both of them need to be considered beforehand. System requirements are only relevant for projects which consist of several sub-components, which can be either hard- or software components. As this is not the case for the proposed schematization tool, system requirements are not further examined. The *nonfunctional requirements* complement the functional requirements by describing to which extent a product can perform certain tasks. They can be considered "important characteristics or properties of the system" (Wiegers & Beatty, 2013, p. 10). Nonfunctional requirements can be distinguished into internal and external qualities. Internal requirements are apparent while the system is being used and therefore most relevant for users. External qualities are mostly relevant for software developers. It is noteworthy that these qualities can be interrelated: they can either have a positive relation, meaning that meeting one requirement also helps to met the other, or a negative relation, for requirements which are to some extent opposed. Wiegers and Beatty mention as an example for opposing requirements that systems aiming towards a high interoperability or reusability, usually do not score well regarding their performance.

Wiegers and Beatty (2013, p. 263) consider the following quality attributes crucial for internet applications: "availability, integrity, interoperability, performance, scalability, security, usability." The first four are considered external, and the last two internal qualities.

Availability "is a measure of the planned up time during which the system's services are available for use and fully operational" (Wiegers & Beatty, 2013, p. 267). It can be operationalized e.g. by a minimum up time (in minutes or seconds) of the system during a week, month or year. The quality of *integrity* "deals with preventing information loss and preserving the correctness of the data entered into the system" (Wiegers & Beatty, 2013, p. 270). As such, integrity is concerned on the one hand with security matters in terms of access rights and on the other hand with the modification of data. The demand for integrity increases if data is distributed or needs to be accessed from several points in the network. A resulting integrity requirement is e.g. that the system should compare a derived data set to the original for unwanted divergences. Next, the system quality attribute of *interoperability* "indicates how readily the system can exchange data and services with other software systems and how easily it can integrate with external hardware devices" (Wiegers & Beatty, 2013, p. 271) Typical issues regarding interoperability are file formats or file format versions (e.g. standard data formats). A possible interoperability requirement can therefore be a concise list of data formats accepted as input data, and a similar list for possible data formats for an eventual output. The *performance* indicates "the responsiveness of the system to various user inquiries and action" (Wiegers & Beatty, 2013, p. 272). But the definition also incorporates other aspects of performance: e.g., a maximum number of simultaneous users, maximum file sizes for the input or processed data, or a maximum number of certain operations. As such, a typical performance requirement related to web services could be how long, in the maximum case and given a certain bandwidth, it should take to completely load the data, i.e., display the site. The requirements of *security* target towards access, authorization processes and encryption. An example for a specific security requirement regarding the authorization process is, e.g., that a user who has not changed the password for six months is preliminarily blocked. Another nonfunctional requirement is *usability*. Requirements of this type aim to specify "the effort required to prepare input data for a system, operate it, and interpret its output" (Wiegers & Beatty, 2013, p. 279). In particular, such requirements can specify the amount of clicks, or the average time, needed to perform a certain action. The last nonfunctional requirements relevant for web applications, according to Wiegers and Beatty, is the requirement of *scalability*. It refers to the "ability of the application to grow to accommodate more users, data, servers, geographic locations, transactions, network traffic, searches, and other services without compromising performance or correctness" (Wiegers & Beatty, 2013, p. 285). Therefore, scalability is related to hardware and software constraints. Again in the context of web applications, a scalability requirement could be to specify the number of page visits a system can sustain without noticeably decreasing performance.

3.1.4. Requirement verification

While eliciting, analyzing and documenting the requirements, it is necessary to validate them throughout the process. Validation in this context means to evaluate whether the specified requirements are in line with expectations towards the system or the product (Wiegers & Beatty, 2013). One possibility to validate a particular set of requirements are usability tests. Only validated requirements should be verified. A verification of not validated requirements does not allow a meaningful interpretation of the results. Once the requirements are validated, several methods can verify that the system meets them. Possible test methods include testing, e.g. by unit testing – executing portions of the code to assert expected behavior or to discover unexpected behavior. Other methods are inspecting, i.e., reviewing the code in regard to its ability to meet the requirements, or demonstration (Wiegers & Beatty, 2013). In chapter 6 Conclusion, I verify the discussed schematization tool prototype against the specified requirements.

3.2. Prototyping

"A prototype is the partial implementation of a system built expressly to learn more about a problem or a solution to a problem" (Davis, 1992). In this sense, prototypes were originally used in a manufacturing and engineering context to avoid manufacturing risks. One example is building a production plant just to find out that the developed product does not work as intended. Similarly, prototypes in the development context avoid development risks. Such a risk is mainly the time-consuming development of a "software that satisfies an incorrect set of requirements" (Davis, 1992, p. 71). This implies a close relation to software requirements. According to this rationale, a *software* prototype "is an executable model of a proposed software system that accurately reflects chosen aspects of the system" (Berzins, 2003, p. 1636). Without the need for a complete implementation of a proposed product, a prototype can already support in collecting, defining and validating requirements. It also assists the developer in gathering information about the product's characteristics.

Different types of prototypes exist in terms of their purpose and of their respective integration into the development process. Thayer and Dorfman Usually, a prototype cannot be used for operational tasks, as it only implements certain vital parts and is not built for operational use. It is rather meant to define basic aspects and concepts of the proposed product. These underlying concepts, in contrast, are then the base not only for the next prototype, but eventually for the final product (Berzins, 2003).

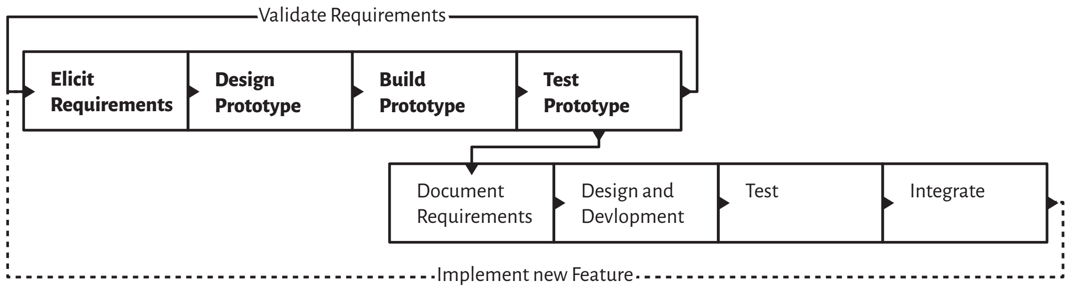


Figure 3.1.: The prototyping lifecycle shows, in a schematic way, how a prototype can be iteratively used to validate a set of requirements and, with that, to build the basis for the operational system. Adapted from Figure 2 in Thayer and Dorfman, 1997.

As already shortly mentioned, different prototyping approaches (rapid or "throwaway" prototyping, incremental or "evolutionary" prototyping and "operational" prototyping) (Davis, 1992, p. 71) exist. They concern different stages of the prototype within the development process. Wiegers and Beatty define three dimensions along which prototypes can be characterized: scope, future use and form. A prototype aims – regarding its scope – either for evaluating the usability, for which a mock-up prototype is enough, or for examining the technical feasibility of the chosen approach, which requires for an proof-of-concept prototype. Considering the future use of a prototype it can either be a throwaway prototype or an evolutionary, i.e., incremental, prototype. Whereas the first only serves once to gather certain responses by the user, the latter "grows into the final product through a series of iterations" (Wiegers & Beatty, 2013, p. 297). The last dimension along which prototypes ca be differentiated is the form: prototypes can bei either of a very simple nature, like a drawing, or a actual partial technical implementation of the proposed system.

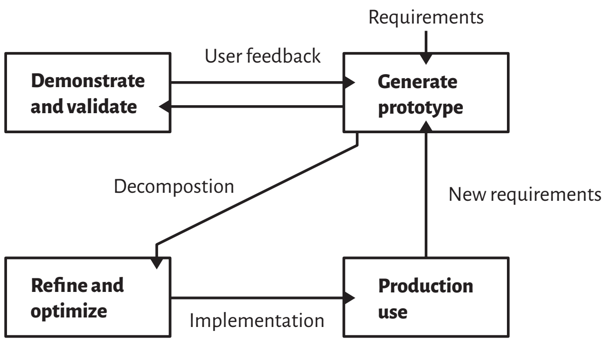


Figure 3.2.: A typical software prototyping process. Adapted from Figure 1 in Berzins, 2003.

Nevertheless, all these concepts have an iterative notion in common: the prototypes pose only the first step of an iterative process (see Figure 3.1). The limited scope of this thesis project does not allow for such an iterative process, which is therefore not conducted as suggested by literature (Berzins, 2003). In that sense, the presented prototype rather marks a starting point for an iterative process. Figure 3.2 shows the role a software prototype can play in the development process: it is built based on the requirements, expose a certain functionality to the user, who can then give feedback. This feedback is used to potentially adjust the requirements and respectively the prototype. Furthermore, results and parts of the prototype are iteratively refined and optimized and, in case of a evolutionary prototype, eventually implemented into the final product. This feedback driven iterations justify the need to incorporate the user into this process. Usability tests are one possibility of how to include the user into such a process.

3.3. Usability studies

The usability of UIs is a relevant aspect of cartographic web services, which needs attention during the development of such systems (Swan et al., 2007). Empirical user testing is one component of usability engineering aiming to improve the usability of a UI.

Nielsen, a noted author on usability regarding software systems, mentions five usability goals. These are of difference priority, depending on the context of the product – particularly on the frequency and the situations in which it is used. These goals, which can be also considered fundamental aspects of usability, are: learnability, i.e., how difficult it is to learn how to use the product; efficiency, i.e., how much effort is needed by the user when using the system to their the goal; memorability; the frequency and gravity of errors; and the user's satisfaction.

In the paper "The Usability Engineering Life Cycle" (1992a), the author outlines the individual steps of usability engineering. The complete list contains the following eleven steps (Nielsen, 1992b):

0. Consider the larger context
1. Know the user [. . .]
2. Competitive analysis
3. Setting usability goals
4. Participatory design
5. Coordinated design of the total interface [. . .]
6. Guidelines and heuristic analysis
7. Prototyping
8. Empirical testing
9. Iterative design
10. Collect feedback from field use

Nevertheless, having evaluated the use of all these methods in practice, he concludes that in line with the "Discount Usability Engineering" approach, it is not necessary to conduct all methods in every development process. Throughout his work Nielsen promotes this simple and cost-extensive attitude towards usability testing (Nielsen, 1992a, 1995; Nielsen, 1993). He argues that it is preferable to implement any user testing, even if it is not as professional due to limited time or financial resources and therefore potentially misses usability errors, than no testing at all. According to this notion, he suggests to implement a simpler process compared to these eleven steps, using only the four following methods: "user and task observation, scenarios, simplified thinking aloud, heuristic evaluation" (Nielsen, 1993).

For the user task observation, Nielsen recommends to visit potential system users and observe them without intervention. He describes scenarios as an "especially cheap kind of prototype" (Nielsen, 1993, p. 18), due to their flexibility. A scenarios in this context is e.g., a mock-up prototype. Such a prototype mimics the functionality of the proposed system but only following a predefined series of interactions. It is possible to implement them as simple paper-prototypes or in a technically more sophisticated way in digital prototyping environments (Nielsen, 1992a). The simplified thinking-aloud-method demands a user who tries to complete a given task with the product. While doing so, users inform the experimenter on "not just what they are doing with the interface, but also why they are doing it", exposing major shortcomings or misunderstandings regarding the UI (Nielsen, 1993, p. 18). In the mean time, the experimenter takes notes. A simple thinking-aloud-method does not use advanced and time-consuming technologies like video recording. An advantage of the heuristic analysis, as suggested by (Nielsen, 1993), is that the designer can directly implement it without any user. An heuristic analysis is a systematic evaluation of an user-interface with the help of general principles applying to different kinds of UIs (Nielsen, 1993). The applied heuristic should be of different levels, from generally well-known guidelines to more specific rules particular to the concerned system. Examples of general heuristics are (Nielsen, 1992a, p. 16):

→ Use simple and natural dialogue
→ Speak the user's language
→ Minimize user memory load
→ Be consistent
→ Provide feedback
→ Provide clearly marked exits
→ Provide shortcuts
→ Provide good error messages
→ Prevent errors

3.4. Adapting methods

A simple yet comprehensive software requirement engineering process accompanied the implementation of the two prototypes: the technical evolutionary proof-of-concept prototype and the mock-up prototype. The software requirement provided focus during the implementation: to prioritize core over secondary features, to set up tests within the testing framework and to make design decisions regarding the technical implementation as well as visual aspects of the GUI. The requirement engineering process started with generally considering the schematization tool. I used a broad perspective for the vision statement, gradually increasing detail by specifying user requirements, system qualities and eventually defining the functional requirements. For this project, I choose the technique of *user stories* combined with one *use case* to represent the user requirements. Due to the limited scope of this thesis, it is infeasible to consider either economic or marketing aspects of the developed software. Hence, I implemented only basic aspects of business requirement information. This included a vision statement and limitations based on external dependencies. Furthermore, due to time constraints, no interviews or workshops have been conducted, which ideally serve as a base for user requirements. Instead I used literature on schematization algorithms and cartographic web services in general as well as personal experience – considering myself as a future product user – and a conversation with a graphic designer who is creating an extensive series of schematized maps (M. Alfaro, personal communication, March 2, 2021). Based on the vision statement and the user requirements the scope, in terms of features implemented by the time of the different releases, was defined. To this end, core features were identified and the extent of their implementation at different times during the development and the release-circle. During the implementation, the specified requirements and the developed prototypes influenced each other: e.g. while designing individual GUI components – and lastly entire screens and flows –, requirements related to the user's input and its validation became evident. Such requirements were then either added to the requirement documents, or existing requirements were upgraded or adjusted accordingly. At the same time, the changing requirements influenced the visual design. Figure 3.3 represents this iterative process with the help of a UI component. It shows, even though only subtle, changes in the visual design. After the first iteration (***A***), a separated header element was added (***B***) to preserve the consistency between different states of this component. Furthermore, the slider at the top area of the component, which enables the user to switch between a regular and an irregular set of orientations C, was simplified. In further iterations, various input element styles were integrated (***C*,*D***). The last iteration (***D***) reintroduces the label for the component "set of orientations". A further example in which the requirements directly influenced the implementation is using the specified requirements to deduce concise specifications for evaluating the implementation's reliability. Additionally, pre-processing steps, necessary to to preserve topology were prioritized. Another example are considerations on how to handle file formats and input data in geographic coordinate systems.

I favored a proof-of-concept prototype, which I combined with an incremental mock-up prototype, due to the main research objective of this thesis: an incrementally developed prototype focusing on the core features can reveal technical constraints. If these constraints are taken into account an iteratively improved prototype could eventually lead to a first release, ready to be evaluated in practice. Such a release-ready stage can never be the achieved when using a throwaway prototype (Wiegers & Beatty, 2013). This poses a reason why evolutionary proof-of-concept prototypes are common practice in web design (Wiegers & Beatty, 2013).

The planned usability tests based on the thinking-aloud method were not conducted due to time constraints. However, a suggestion for the setup of a test to evaluate the prototype's usability is presented in Appendix C. The shortcomings of the GUI design, which are identified with such a study, need to be analyzed to prioritize severe usability issues. These issues can then be addressed by developing specific design solutions, which again are required to be tested. According to Nielsen (1994), it is not necessary to conduct a large number of usability tests using the thinking-aloud-

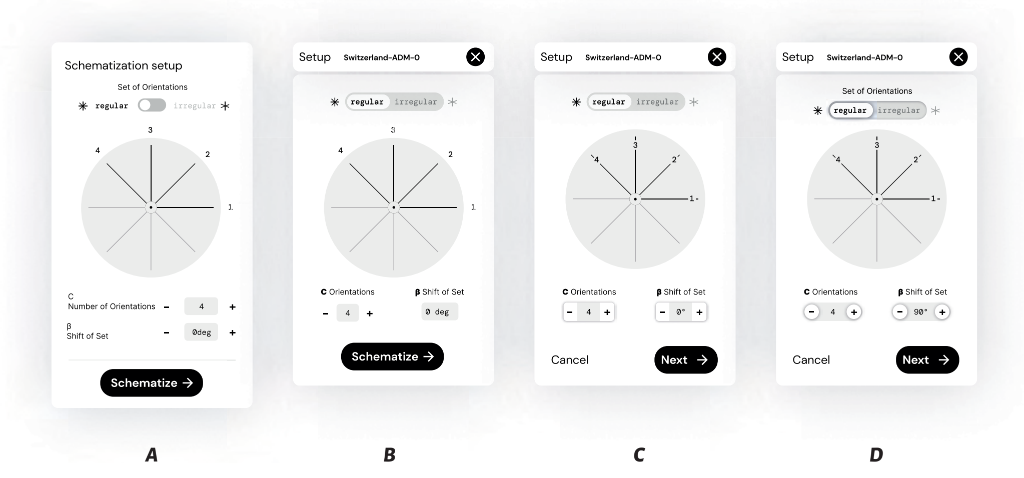


Figure 3.3.: A component of the mock-up prototype is gradually adjusted in an iterative process.

method: the number of newly identified usability problems for every additionally conducted test does not increase significantly when conducting more than five tests. Nielsen refers to software systems in general, and new web technologies have emerged since the publication of the mentioned papers. Hence, for the practical and specific guidelines on creating usability tests for web sites – or web services in the case of this project – I drew on works by Krug (2009, 2013). He particularly targets web developers and UI designers in these works, referring as well to Nielsen for the underlying concepts.

Adopting these methods implies an iterative implementation process, driven by requirements and incorporating feedback into the prototype. This process and its intermediate results are discussed in the following chapters.

4. Implementing the prototype

This section describes the implementation of the proposed schematization tool prototype. As such, it addresses significant milestones of the implementation phase. Firstly, a suitable schematization algorithm was identified. Secondly, the algorithm itself was implemented. Therefore, I briefly discuss the working principle of the chosen algorithm in including preliminary steps. Software requirements guided this implementation process. Furthermore, I describe the UI I designed in parallel to the implementation based on the same specifications. This section also presents an overview on the schematization parameters and how they can be adjusted by the user.

The implementation relies vastly on JS, more particularly on TypeScript (Ts), an open-source language based on JS which allows for typing and validating code. It also provides support for consistent documentation. Furthermore, the following libraries are used: leaflet.js for the preliminary and final visualization of the schematized data and uuid to create unique identifiers for DCEL entities. The (no-longer maintained) geojsonhint by *mapbox* was used to validate the geoJSON (an extension of JavaScript Object Notation (JSON) for representing geographic features) files. The graham_scan package calculates the convex hull of irregular polygons.

For development, the testing library jasmine was used as well as browser-sync for live-reload. For compiling from typified JS to plain JS code, nodemon, esbuild and ts-node as well as jasmine-ts were employed.

A prerequisite for implementing the schematization approach by Buchin et al. (Buchin et al., 2016) was the system's capability to transform the geographic data provided by the user into the DCEL structure. This facilitates a topology-preserving schematization. Without this structure, the system can schematize neither regions nor individual polygon features with islands or lakes while preserving topology. Instead, only a single outline of a geographic feature could be schematized. This transformation is not part of the algorithm described by Buchin et al. (2016). Therefore, converting geographic data in the common, standardized geoJSON format to the DCEL data structure and back is examined in pseudocode in Appendix A.

4.1. Comparing schematization approaches

The following section aims to provide an overview on algorithmic approaches tackling the schematization of regions, posing an answer to **RQ B**. The systematic comparison aims to evaluate existing approaches regarding their suitability for the implementation in the prototype of a web-based schematization tool. Firstly, I outline the approaches I selected. Then, I explain the tabular overview and briefly discuss it, concluding with a section on the limitations of this literature-based comparison.

4.1.1. Selecting approaches

Note that *only* algorithmic approaches concerning polygon schematization (either particularly tailored to polygon schematization or applicable to them) were included in the overview. I am aware that there are numerous algorithms and strategies regarding line and network schematization. Nevertheless, the requirements for algorithms differ significantly depending on the intended input type, i.e., depending on whether networks or regions are schematized. Therefore, approaches towards the automated generation of schematized network maps are not considered in this overview. Furthermore, schematization is more than mere simplification. Consequently, algorithms and approaches which aim to simplify but not impose geometric constraints during the schematization are likewise excluded. If similar versions of the same approach were published, the most recent version was chosen.

4.1.2. Discussing approaches

Table 4.1 allows to systematically compare approaches towards polygon schematization, as it features four properties related to geometric qualities of the schematization, and one property concerning the computational complexity of the approach, i.e., running time of the algorithm for each approach. The table's rows are sorted by year of publication in descending order.

Table 4.1.: Overview of existing schematization approaches for regions

	Author(s)	**Year**	**Style**	***A***	***T***	***V***	**Complexity**
[S1]	Meulemans et al.	2021	C-Oriented	-	+	-	p, O(n^2 + mn)
[S2]	Buchin et al.	2016	C-Oriented	+	+	-	p, O(n^2)
[S3]	Meulemans	2016	C-Oriented	+	+	-	np-complete
[S4]	Meulemans	2016	C-Oriented	+	+	-	np-complete
[S5]	van Goethem et al.	2015	Arcs	+	+	-	p, O(n^2)
[S6]	van Dijk et al.	2014	Arcs	-	+	-	p, O(n^2h log n)
[S7]	van Goethem et al.	2013	Arcs, Bezier Curves	+	+	+	p, O(n^3 k)
[S8]	Cicerone and Cermignani	2012	C-Oriented	-	-	-	p, O(mn^2 log n)
[S9]	Reimer and Meulemanns	2011	Parallelism	-	-	-	np-hard
[S10]	Meulemans et al.	2010	C-Oriented	+	+	-	p, O(n^2)
[S11]	Heimlich and Held	2008	Simplification only	-	+	-	p, O(n log n)

On the first glance, schematization approaches differ visually regarding the geometric style which they are aiming for. Schematization has additional geometric qualities; not all are relevant for every application. Yet, for cartographic applications, certain limitations apply: "Schematic maps in the context of geography are very different from schematic maps in the context of circuit board design, or architectural layouts. This is because geography is inherently hierarchical in nature." (Dykes et al., 2010, p. 5) The geometric qualities used in Table 4.1 are relevant for cartographic applications: the first one, *area preserving* (***A***), indicates whether (+) or not (-) the approach considers area. If the area of the input polygons is relatively equivalent to the are of resulting regions, the approach preserves the area. *Topology preserving* (***T***), the second quality, points out whether (+) or not (-) the algorithm is topology aware: in the first case (+), geographic relations are maintained throughout the schematization; in the latter (-), topological constraints are loose. Lastly, the *vertex-restricted* (***V***) specifies whether (+) or not (-) the output includes only input vertices. Non-vertex-restricted schematization are considered more flexible. Lastly, column *complexity* indicates the complexity formalized as time-space costs of each algorithm, employing the Ω notation.

To determine which of the considered algorithm is suitable, the geometric qualities (***T,A,V***) and their combinations were considered. Nevertheless, it is necessary to also examine peculiarities of certain strategies, which are deployed in some of the approaches. To this end, firstly, the combinations of geometric qualities and their consequences in respect to each approach's suitability are discussed. Thus, some approaches can be discarded based on these three variables. Consequently, more specific characteristics of the remaining individual approaches are discussed. This process illustrates that no perfect schematization exists, but every approach has its limits. The cartographer's task is to evaluate schematization approaches and choose the best among them in respect to map requirements and context: project and data constraints as well as the intended audience.

Starting with geometric characteristics, for topology (T), the schematization approach creates valid output only for a singular *outline*, i.e., for an individual spatial unit, if topology is not considered. Territorial outlines with holes (lakes), islands or adjacent outlines cannot be schematized with such approaches: they are not intended for subdivisions, which are relevant for thematic mapping, but for circuit boards, [S8], or for generating map-like visualizations such as *chorematic diagrams* (Reimer & Meulemans, 2011) as in approach [S9]. Therefore, such algorithms are not suitable for implementing the prototype; this characteristic would pose severe limitations to applying them in the thematic mapping context. Numerous algorithms concerned with simplifying, smoothing and schematizing polygons (territorial outlines) do not necessarily preserve topology. Therefore, the list of examples in Table 4.1 – Table 4.1 is incomplete in this respect. Further research may examine possibilities of implementing such algorithms so that they are 'topology aware': many line simplification algorithms already consider individual edges and their characteristics (Barkowsky et al., 2000; Douglas & Peucker, 1973; Latecki & Lakämper, 1999; Tutić & Lapaine, 2009; Visvalingam & Whyatt, 1990). This poses a possible starting point for implementing data structures like a DCEL to explicitly introduce topological relations.

Area preserving (A) is not as fundamental as maintaining topological relations for all thematic maps, e.g. for flow maps. Still, for thematic mapping methods, directly concerned with area such as point density mapping or *choropleth* mapping methods, and generally when using an area-equivalent projection, preserving area is a *de facto* required schematization quality. Therefore, approaches [S1], [S6], [S8], [S9], and [S11] were discarded for implementing the prototype.

The remaining geometric quality, vertex-restriction (V), shows a homogeneous picture across the selected algorithms. Only [S7], the topologically safe curved schematization by van Goethem et al. (2013), is vertex-restricted. This deficiency is also mentioned by the authors, who reason that "vertex restriction poses problems for very low complexities and subdivisions" (van Goethem et al., 2013, p. 11). Accordingly, [S7] is less suitable.

The majority of the compared algorithms aim for a C-oriented style. This style is relatively strict, compared to a mere simplification [S11], which is arguably not even a proper schematization in its narrow sense. Even though the outcomes of the C-oriented schematization approaches are similar, the underlying algorithmic strategies (e.g. simple map-matching, heuristic, simulated annealing) diverge fundamentally. Consequently, their computational complexity differ as well. Some of these algorithm strategies are, again, peculiar. Map-matching uses a planar grid-graph and optimizes subdivisions so that all edges align with this grid. This method positively affects the schematized result. Among the C-oriented approaches, some are more flexible, enabling a definition of orientation sets. For example, [S2] even allows the definition of irregular sets (sets of orientation which are not evenly spread over the unit circle), whereas other approaches only allow a limited number of orientations: these are usually rectilinear, octilinear or both, ([S1], [S3], [S8], [S10]).

Another criterion for suitability are the possibilities for user interaction offered by the approach. Some of the approaches need the user to select important vertices, or they improve significantly with such selection ([S9]) Others mention specific parameters like (*flat*, *regular*, *curvy*) for the arc style approach ([S2]). Most of the examined approaches do not explicitly mention parameters which the user can alter to modify the output.

For the purpose of this cartographic research project, [S5], the *Curved Schematization* (van Goethem et al., 2015), and [S2], the *Area-Preserving Simplification and Schematization of Polygonal Subdivisions* (Buchin et al., 2016), seem most appropriate for implementing the proposed prototype. This is firstly because they meet the desired schematization qualities – they preserve area as well as topology and are non-vertex-restricted. Secondly, this is due to their computational complexity, which is comparatively low with a quadratic running time. For example, approach [S7] is discarded because of the long running time, given a complexity of O(n^3k), due to the high complexity of required preprocessing steps. Nevertheless, only one algorithm could be implemented in the scope of this thesis. It remains unclear if the C-oriented approach [S2] (Buchin et al., 2016) was the best choice.

4.1.3. Limitations of the comparison

The overview of approaches for region schematization is based on a mere literature review. For a more systematical comparison of such schematization approaches, implementing these algorithms is necessary. This would allow a systematical comparison. It would not be limited to the approaches' theoretical properties, but facilitates visual comparison and evaluation of each approach, along with an evaluation of computational costs in practice. Hence, testing several approaches using the same input set would provide insights and could reveal important practical aspects. Nonetheless, implementing further algorithmic approaches to region schematization exceeds the limited scope of this thesis. Also, widening the scope and including such line schematization approaches, which can be modified to be topology preserving, into the systematic review can expose focus points for further research. Lastly, there are technical characteristics which are not yet included in Table 4.1 but may be relevant for evaluating such approaches.

4.2. Implementing the selected approach

It is necessary to outline how the selected schematization algorithm works to understand limitations of this schematization approach. Such characterization further shows the need to preliminarily implement a particular data structure, called DCEL, to facilitate a schematization which preserves topological relations.

4.2.1. Area-preserving simplification and schematization algorithm

The simplification and schematization algorithm for polygonal regions proposed by Buchin et al. (2016) consists of two algorithms: an orientation-restriction and a simplification algorithm, in the following also referred to as algorithm 1 and algorithm 2 respectively. The first algorithm introduces geometric constraints to the region, whereas the second is concerned with reducing the visual complexity of the intermediate orientation-restricted result.

Algorithm 1 employs a C-oriented schematization approach: all edges of the resulting schematized region adhere to C, i.e., a set of preliminarily defined orientations. The algorithm allows to define a regular and irregular C. This embodies rectilinear, hexilinear, and octilinear Cs and any other possible combination of regular and irregular orientations. Hence, it is in this regard more flexible than other approaches. Furthermore, they introduce the possibility of a β-shift, allowing to rotate C. For that reason, C is part of a set of parameters which user can modify in the schematization prototype.

Before the angles are constrained, a preprocessing step is necessary to harmonize the edge lengths within the region. According to Buchin et al., it is important to subdivide edges so that the upcoming step, creating staircases, results in a shape which resembles the original. To this end, the region's edges are subdivided so that they do not exceed a defined threshold. This threshold, ϵ, is calculated by dividing the input diameter by fraction λ. This fraction λ poses the second parameter which the user can manipulate. Buchin et al. suggest to use a λ of 0.05.

The adherence of edges to a specific angle set is achieved by turning unaligned edges, i.e., edges which do not already adhere to C by incidence, into staircases. This staircase is built by a certain number of steps, whose edges are all aligned to C. For this part, computing the number of steps is essential: if the number is too small and the area which the individual steps span consequently too large, they run the risk of intersecting with other staircases or edges. Such intersections violate the requirement of topology preservation. Therefore, they are avoided by computing a minimum number of steps for each staircase.

The second algorithm simplifies the resulting geometrically constrained region of algorithm 1: to meet the requirement of low visual complexity, the region is simplified in this second step. The complexity of a region (or subdivision) S is defined in the context of this schematization approach as the edge number, $|S|$, within the region. To this end, the region is considered as a planar graph, consisting of edges and vertices. Since by introducing the staircases, the complexity $|S|$ is increased compared to the input's complexity, a simplification process is required. This even applies when input data exhibits low complexity. The so-called edge-move can perform this simplification without introducing new orientations. Thus, it guarantees a C-oriented result. At the same time, the total region area is preserved during the simplification. Hence, it guarantees an area-preserving result. This is because edge-moves are always implemented in pairs: one move adds area to the region and another subtracts the same area from the incident face. To this end, for each edge-move, a complementary pair of configurations is determined. A configuration consists of three consecutive edges, two outer edges and one inner edge. Configurations are complementary if one of them allows a positive and the other a negative contraction. A contraction reduces the amount of vertices in the region (and therefore also $|S|$) by sliding the inner edge along straights through the outer edges so that at least two vertices coincide. Such contractions either increase the area of the incident face (positive contraction) or decrease it (negative contraction). The simplification stops when either no further valid edge-move is possible or a minimum complexity $|S|$ in the region is reached. This minimum, defined by the number of edges, k, poses the third parameter which the user can adjust. All three parameters are detailed in section 4.4.2 User-defined parameters.

4.2.2. The DCEL

The working principle of the area-preserving simplification and schematization algorithm by Buchin et al. demonstrates that a series of measures is needed to preserve topology during automated schematization. Yet, a fundamental measure from the development perspective, is not yet discussed. The following section concerns the DCEL data structure and how it is implemented to facilitate a topology-preserving schematization.

To explicitly store which edge bounds which polygon of the region, the input region is converted into a DCEL in a first step. A DCEL is a list-based data structure to describe geometry, including "structural, topological information" (de Berg et al., 2008, p. 30). It can be used in the two- as well as in the three-dimensional domain (de Berg et al., 2008; Karmakar et al., 2014; Karmakar et al., 2012; Saha & Biswas, 2021). However, after the schematization, the tool needs to convert the DCEL back to a common geographic data format.

The concept of a DCEL is loose. Several ways of implementation exist, adjusted to the specific use case. Nevertheless, every DCEL needs three fundamental entities, namely vertices, edges, and faces. A characteristic feature of a DCEL is the use of half-edges. Every edge of the original planar graph consists of two directed half-edges, running in opposite directions. Such a pair of half-edges is called

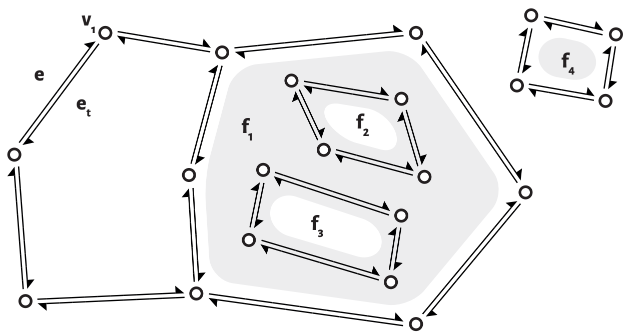


Figure 4.1.: The DCEL data structure employs vertices (**v_1**), edges (**e, e_t**) and faces (**f_1, f_2, f_3**). This example demonstrates the importance of inner (**f_2, f_3**) and outer (**f_1**) components to allow modelling e.g. lakes. Furthermore, **f_1, f_2, f_3** and **f_4** need to store the same feature ID (a unique identifier, usually an integer) as they are all part of the same multipolygon. A multipolygon according to the current geoJSON specifications is defined as "an array of Polygon coordinate arrays" (Butler et al., 2016, p. 25), i.e. words such a multipolygon can consist of several polygons. The polygon by its own consists at least of an external ring and optional internal rings.

twins. A half-edge stores a pointer to its corresponding twin, and furthermore to the previous and the next half-edge. Every half-edge is incident to one face on its left side: incident in this context means that one half-edge within a DCEL bounds only one face. A pointer to this face is stored for each half-edge. Therefore, all half-edges bounding one face run in counterclockwise direction. Every vertex needs to store at least one incident half-edge. An edge is incident to a vertex if the vertex is the edge's endpoint. Nevertheless, in the case of the discussed implementation, it was beneficial for algorithm 1 to record all incident edges of a vertex. Every face except the unbounded face, which is defined by not pointing to any half-edge, is linked to one edge. This edge is part of the boundary describing that face. Thus, all entities are linked to each other. This results in an effective structure for accessing certain data parts, e.g. all edges bounding one face or the adjacent face of one given a shared edge, i.e, a pair of half-edges (de Berg et al., 2008).

4.3. Software requirements

Three levels of requirements, are discussed in the following section. Due to the constrained scope of this research, no requirement management was implemented and requirement development was limited to a narrow elicitation and documentation process. Nevertheless, the resulting requirements answer **RQC**. They are presented in the following section. In the following, the term *product* is used. It refers to the carried-out schematization tool.

4.3.1. Vision and scope

I built the vision statement based on a proposed structure, using keywords. Those keywords are *highlighted*.

> *For* cartographers (and those working in the field of data visualization) *who* need to create schematized maps (of thematic nature), *the schematization product* for subdivision *is* a *web app that* enables the user to schematize any valid vector geographic data containing polygons. The product can load a common GIS file format, namely the shapefile format by ESRI) and outputs a schematized subdivision, again as a shapefile. The user can modify schematization characteristics by altering parameters. *Unlike* already existing approaches, this product does not need any software dependencies (Java, python). It runs in any modern browser on any operating system. It helps the user save time when generating schematized maps and therefore facilitates more flexibility in the cartographic design process.

Limitations and dependencies

The product is limited by several external factors. They originate from industry standards, which need to be considered when implementing the product, as well as from constraints based on the available algorithms for map schematization.

→ *LI-1* The product depends on the underlying schematization algorithm by Buchin et al.: it determines the running time as well as the resulting schematizations, which are limited by the algorithmic approach. See section 5.1 Algorithm limitations.

→ *LI-2* The product depends on the specification of those input formats it uses. Each of those industry standard geographic data formats (shape file, geoJSON and topoJSON) has its own specification. The latter two are specific versions of the generic JSON format. Neither for the geoJSON format nor for the topoJSON format, the specifications are concerned with projections (Bostock & Metcalf, 2021; Butler et al., 2016). The current geoJSON specification drops the previous support of a Coordinate Reference System (CRS) object. But projections are important when dealing with area-preserving schematizations. Hence, the safest and most practical current format which can deal with projections is the shape format. It is the intended input and output format for the product. Nevertheless, also for the visualizations of the (intermediate) results, the geoJSON format is used internally.

4.3.2. User requirements

Due to the product's unique outcome, a schematized map, there is no need to specify either more than one user class or more than one use case. The single use case of this product can be described as follows:

Schematize a subdivision.

Despite this simple setup, various user types with slightly different needs exist. In the following, two of them are described and used to create three user stories.

- **Cartographer** (professionals with a GIS background): The cartographer wants to schematize already processed and modified geographic data. They are familiar with GIS tools and their underlying concepts. They may focus on geographic aspects of the resulting schematization. Because of possible advanced data edits, which need GIS, they may want to further process the resulting schematization using GIS software.

  User story A As a cartographer, I want to schematize given geographic data according to the envisioned map's needs so that I can load the result back into a GIS for further edits. Examples for further edits are adding labels or charts on top of the geographic layer.

  User story B As a cartographer, I want to generate a consistent series of schematized subdivisions for an atlas so that I can integrate this step into my semi-automatized workflow.

- **Information designer** (professionals with a design or data visualization background): The designer wants to schematize geographic data to load it into the preferred vector design software, where they compile the map. The designer may not be familiar with GIS tools and their related concepts. The visual characteristics of the resulting map is the most important aspect.

  User story C As a graphic designer, I want to receive a visually simplified version of a shape file with boundaries that I found online. Thus, I can use the schematized boundaries as a base layer for a print map which I want to compile in a vector-based design software.

User stories A and *B* are used as a basis for the use case *UC-1*. The use case is outlined consecutively in a comprehensive way, indicating normal and alternative flows, and in particular possible exceptions. For the use case, either of the two identified user types take the *Mapmaker* role.

UC-1 Schematize a subdivision.
Primary Actor Mapmaker

Description
The Mapmaker specifies the input data, which are then processed by the system. The Mapmaker is informed about the process. After a successful schematization, the result is visualized and the Mapmaker can download the result in different formats.

Preconditions

→ The Mapmaker has a modern browser and an internet connection.

→ The geographic data the Mapmaker wants to schematize is valid.

Normal flow **1.0 Schematize a subdivision using the GUI**

1. The Mapmaker specifies the input data.
2. The system validates the input data and visualizes it.
3. The system offers to change parameters as well as the option to start the schematization process.
4. The system informs the Mapmaker about the process (the time passed and the current step) throughout the schematization.
5. The system stores the intermediate results of the single schematization steps.
6. During the schematization, the Mapmaker can stop the schematization at any time.
7. After the successful schematization, the system shows the Mapmaker the intermediate results and allows the user to inspect them visually.
8. After successful schematization, the system offers the Mapmaker the option to adjust the parameters and restart the schematization.
9. After successful schematization, the system offers the Mapmaker the option to download the result in a specific format.

Alternative flow **1.1 Schematize a subdivision using the CLI**

1. The Mapmaker specifies the directory with the input data (possibly several input files at the time), optional parameters and the output directory for the schematized subdivisions within the CLI.
2. The system validates the input data.
3. The system informs the Mapmaker about the process (the time passed and the current step) throughout the schematization.

4. During the schematization, the Mapmaker can stop the schematization at any time.
5. After successful schematization, the system prints a short log to the CLI.

4.3.3. Functional requirements

In the following, I specify functional and the preliminary nonfunctional requirements for the schematization tool. The nonfunctional requirements refer to the first release candidate of the schematization tool. The functional requirements on the other hand are prioritized: those ending with *a* are intended to be implemented in the first alpha release for internal testing. These requirements build the basis for the requirement verification. Requirements ending with *b* refer to the beta release, intended for public testing. Those ending with *rc* refer to the release candidate of the tool.

Nonfunctional requirements

Certain system qualities are more important depending on the project's context. As the prototype is a web service, I define in the following nonfunctional requirements regarding integrity, interoperability, performance, and usability as recommended. Additionally, I specify robustness and scalability requirements. I discarded the quality of security because the web service will neither transfer nor store data. Similarly, since the web service processes data on the client side, I do not specify any availability requirement. The web service is only requested once. Afterwards, the computationally intensive processes run decentralized on the user's machine. Therefore, availability is not as crucial as for other systems – e.g. the original implementation of Mapshaper where the processing ran on one remote server. However, I decided to define requirements for *robustness*, which express the system's ability to cope with unexpected conditions. This is because during development, I identified robustness issues related to invalid or unexpected input data. As this aspect seems crucial for the intended system, I specified robustness requirements, which may be extended while development continues.

Installability. Note that this requirement is only related to the planned Node.js Package Manager (npm) package and the CLI, as the web service does not require any installation. The CLI targets advanced users. Therefore, it seems adequate to provide the system as an npm module, which allows a straightforward installation, which includes the installation of dependencies, and an upgrade or downgrade via the CLI.

→ *ISL-1* The advanced user shall be able to install the system as an npm module via the command line within 60.0 seconds.

→ *ISL-2* The advanced user shall be able to upgrade the system via the command line.

Integrity. To guarantee that no information, neither geometric nor attribute information, disappears, the system needs to compare certain data aspects before and after the schematization.

→ *INT-1* The system shall confirm that the resulting schematization preserved area and therefore has the same total area as the input.

→ *INT-2* The system shall confirm that the resulting schematization preserved topology and therefore has the same face-to-face boundaries.

Interoperability. To offer the schematization tool as a practical option in a map-making process, it is important that it fits into a typical cartographic workflow. This can happen either with the intention of creating a static (for print or screen) or an interactive map (e.g. for web maps). Such a map is usually performed on a desktop machine using GIS and graphic design software. Thus, the tool should be able to handle the most common data formats in respect to these two software types. In this context the Mapshaper software serves as reference regarding its input and output formats. This results in the following requirements:

→ *IOP-1* The system shall be able to import the shapefile format by Environmental Systems Research Institute (ESRI).

→ *IOP-1* The system shall be able to export the schematized region as shapefile, for further use in GIS applications, and as Scalable Vector Graphics (SVG) for further use in any vector-based design software (e.g., Inkscape or Adobe Illustrator).

Performance. Performance in the schematization tool context is most relevant regarding the algorithm's implementation: due to its quadratic-time complexity, it is dependent on the user input. Therefore, it is sensible to specify performance requirements in relation to user input. Nevertheless, the short response time which the average user expects may differ from the running time of the of the considered algorithmic approach, particularly its simplification algorithm. This algorithm depends not only on the number of input edges, but as well on their relative length, and the user parameters λ and k (see section 4.4.2 User-defined parameters). Thus, it would not be meaningful to specify related performance requirements merely based on the number of edges (e.g. *PER-2*). The number of seconds specified for *PER-2* is based on the computational benchmarks of the algorithm's original implementation by Buchin et al. (2016).

→ *PER-1* The system shall be able to load and display the input (including the conversion to a DCEL) within a maximum of 3.0 seconds.

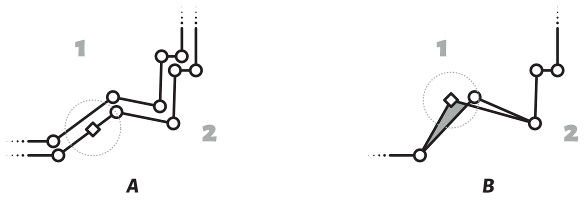


Figure 4.2.: ***A*** shows a typical situation for a missing vertex on the face-to-face boundaries: the common boundary between polygon 1 and 2 has one superfluous vertex on the boundary of polygon 2. To increase legibility, the two boundaries are shown with a small offset in this sub-figure. ***B*** illustrates a situation where two vertices with two incident edges are misaligned. A topologically correct solution is to merge these two vertices to one with three incident edges.

→ *PER-2* The system shall be able to perform an octilinear schematization of a region consisting of less than 1,300 edges (given a reasonable value for λ and typical distribution of edge lengths over the region) in no more than 28.00 seconds.

Robustness. Similar to performance, the most crucial aspects of robustness are directly related to the input data. One important issue is how the system copes with input regions which exceed a certain level of complexity. However, it seems relatively easy to resolve, as it requires a simple check of whether the edge number in the specified input data, in combination with the schematization parameters (see performance requirements), is inferior to a certain complexity $|S|$. The main error source during the schematization, nevertheless, may originate from topological issues in the input data specified by the user. This poses a great challenge, as it seems out of the project's scope to implement a thorough topology check. However, such a check could run when loading the data, already before the schematization starts. In the context of region schematization, this validation only concerns topological relations regarding polygons. Topology errors which may cause problems and were also faced during the development include edges between two adjacent features, which are described with a different amount of vertices, and misaligned vertices with two incident edges, which ought to have three or more incident edges (see Figure 4.2). Therefore, the resulting requirements regarding the system's robustness are the following:

→ *ROB-1* If the input data is too complex, the system stops the import to prevent a system failure.

→ *ROB-2* All configuration parameters for the schematization (C, β, λ and k) are validated before the schematization.

Usability. One example for creating memorable and intuitive interaction modes for the user is a drag-and-drop component. This element is used in applications specific to cartographic purposes like Mapshaper, but also in popular applications outside the cartographic community like wetransfer.com.

- *USE-1* The user shall be able to perform a schematization within 10 mouse clicks (excluding the clicks necessary to select the input in the GUI of the user's operating system).
- *USE-2* The average user shall be able to set up the parameters and start a schematization within 20.0 seconds.

Scalability. Two aspects concerning the scalability of the schematization tool seem relevant: firstly, the level of the input data's complexity $|S|$, see section 4.2.1 Area-preserving simplification and schematization algorithm) at which the tool handles the processing without running the risk of a system failure. This aspect ties into the consideration which level of detail is needed in the input data for an efficient yet meaningful schematization in relation to the intended complexity of the resulting schematized region. This underlying question needs to be answered before specifying a measurable requirement for this aspect. Secondly, such a tool could be a platform for implementing other schematization approaches. These approaches may include not only different algorithms for C-oriented schematizations, but also other schematization styles. Therefore, it seems desireable to isolate certain functionalities into stand-alone software components or libraries which are reused when implementing further schematization approaches. One example for such a functionality is the conversion from the common geoJSON file format into the DCEL data structure. As this is beyond the scope of this research project, no scalability requirements were specified.

Functional requirements for alpha release

After specifying the underlying requirements, the following section specifies the functional requirements. They mostly follow an 'if - shall' schema. The schematization tool is denoted as system. The requirements are grouped by priority, i.e., the software version in which they are implemented: functional requirements with a very high priority are meant to be implemented in the system's alpha release, allowing basic testing. Requirements of medium priority are needed to perform at least basic user testing. The requirements with low prioritization are not as vital for the system to meet a defined use case. They rather focus on improving the user experience, e.g. by integrating external services which are not directly related to the schematization process. Table 4.2 shows the function requirements related to the alpha release. The remaining functional requirements can be found in Appendix E.

Table 4.2.: Functional requirements for the alpha release

FR	Requirement
2-a	The system shall be able to parse geoJSON as input data.
3-a	If the input data is not a region (if it contains features of type other than “polygon” or “multipolygon”) the program shall exit and the user shall be informed.
4-a	If the input data is not a valid geoJSON the program shall exit and the user shall be informed.
5-a	If the input data is too detailed, i.e., if it exceeds a maximum number of edges or vertices, the program shall exit and the user shall be informed.
6-a	The system shall preserve potential attributes attached to the inputs features in the output.
7-a	The system shall preserve the number of features of the input in the output.
8-a	The system shall be able to generate a DCEL from a geoJSON.
9-a	The system shall be able to generate a geoJSON from a DCEL.
10-a	While the data is being processed, the user shall be informed that the application is processing.
11-a	The user shall be able to specify a regular set of directions (without β-shift) of the schematization.
17-a	The system shall display the schematized region in the map-view after the schematization is finished.
22-a	The user shall be able to track the progress of the schematization.

4.3.4. Features

Five main features of the proposed schematization tool and their respective sub-features are identified. A feature consists, according to the definition by Wiegers and Beatty, "of one or more logically related system capabilities that provide value to a user and are described by a set of functional requirements" (2013, p. 11). Figure 4.3 shows the features as a feature-tree (Wiegers & Beatty, 2013). The feature-tree is one method of visualizing a system's capability. It offers a visual overview on the planned product features. The branching structures the features into different levels. Due to this characteristics a feature-tree poses the "ideal model to show to executives who want a quick glance at the project scope" (Wiegers & Beatty, 2013, p. 95).

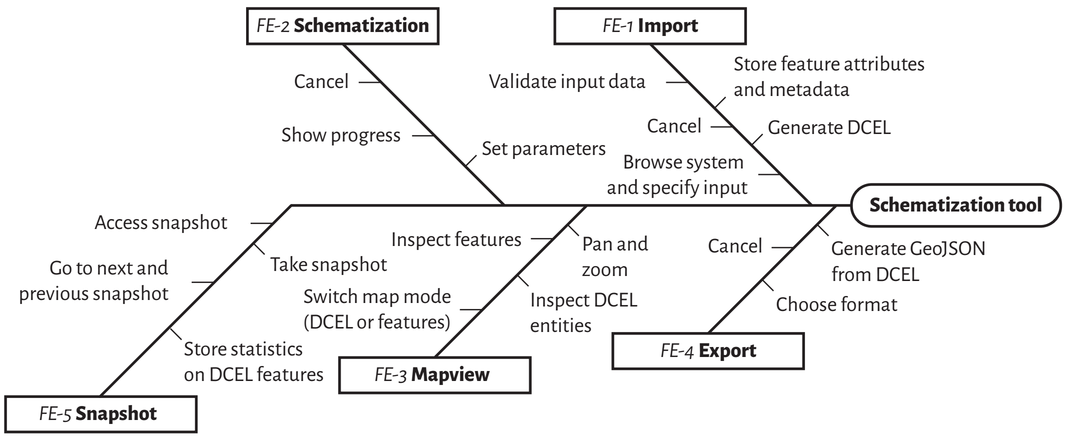


Figure 4.3.: The feature-tree for the schematization tool shows the five main features, including their lower-level features.

The features correspond to the user requirements. Every feature consists of sub-features. These sub-features are interlinked with the functional requirements. See Appendix E for a list, which illustrates the relations between features – including their corresponding sub-features – and related functional requirements.

Release scope

Not all planned system capabilities are implemented in the proof-of-concept prototype. This type of prototype serves mainly to evaluate the technical feasibility. Therefore, primarily capabilities which seemed critical in terms of performance were implemented in the prototype. Nevertheless, due to the incremental prototype nature, other features were also considered. Table 4.3 shows to which degree the main features are planned to be implemented in the product releases.

Table 4.3.: The features and their planned implementation

	Feature	Alpha release	Beta release	Release candidate
FE-1	Import	import of geojson files only	supports import of shp files only	completely implemented
FE-2	Schematization	no cancel, no summary, no track of process	no summary, simple track of process only	completely implemented
FE-3	Map-view	only zoom/pan, basic inspection of dcel features and switch dcel/feature mode	completely implemented	completely implemented
FE-4	Export	not implemented	one file format (.shp) only	completely implemented
FE-5	Snapshot	only basic visual overview of snapshots	completely implemented	completely implemented

4.4. The GUI

In parallel to the proof-of-concept prototype, I designed screens and components for the schematization tool's GUI. This was implemented in the software figma. By adding interactions and transitions to the static visuals, figma allows to turn the design into a clickable mock-up prototype (Wiegers & Beatty, 2013). Static screens of this prototype are included in Appendix B. In this section, I outline the most important design aspects. Some components are shown in figures. The GUI is – in its fundamental aspects – inspired by Mapshaper: it shares the Mapshaper GUI's goal of a simplistic impression which allows intuitive interaction.

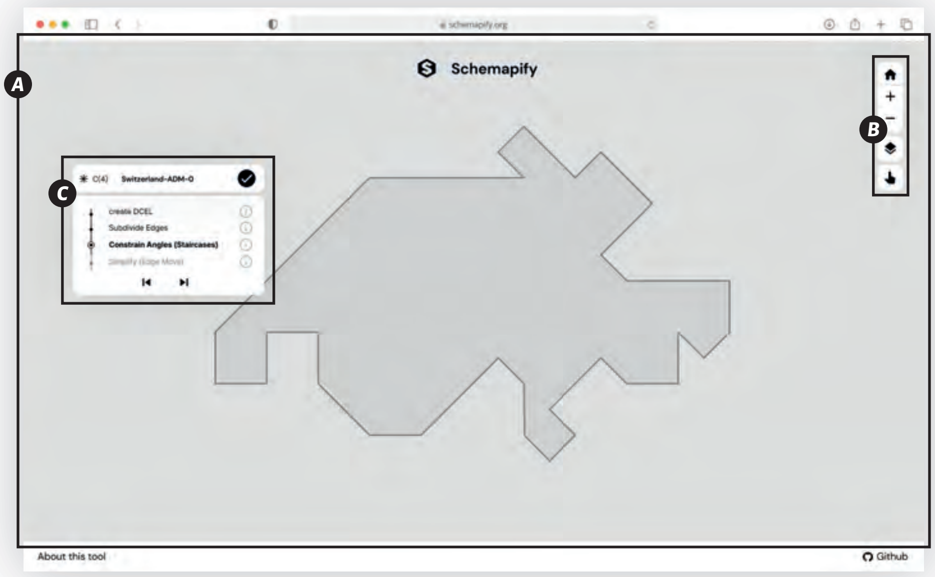


Figure 4.4.: The GUI components in a typical product view: the map-view ***A***, the map control panel ***B***, the floating panel for schematization functionalities ***C***.

4.4.1. GUI components

The full-screen map-view takes most of the screen's space and continues in the background below the other floating UI components. This can be seen in Figure 4.4: ***A*** extends below ***B*** and ***C***. As usual, the tools to interact with the map are on right side. Most of the interaction regarding the schematization process is implemented in a floating side panel. Both the prominent full-screen map-view and the floating panel are borrowed from web services like wetransfer.com and major providers of interactive maps like openStreetMap, Google maps, Bing maps and here maps. All three main GUI components are shown in Figure 4.4.

As main font, the free and open-source font *Inter* by Rasmus Andersson was used. It was particularly designed for screen use. It is complemented by the alike free display font *DM Sans* designed by Colophon Foundry, Jonny Pinhorn and the Indian Type Foundry, released under the Open Font License (OFL). DM Sans is only used for headlines and buttons; for longer text and inputs, Inter is used. For the icons, I use the free and open-source *Material design icons*, released by Google under the Apache-2.0 license, as they pair well with the two other sans-serif interface fonts. No colors, only different shades of gray are used in the proposed UI design. Nevertheless, I aimed for the

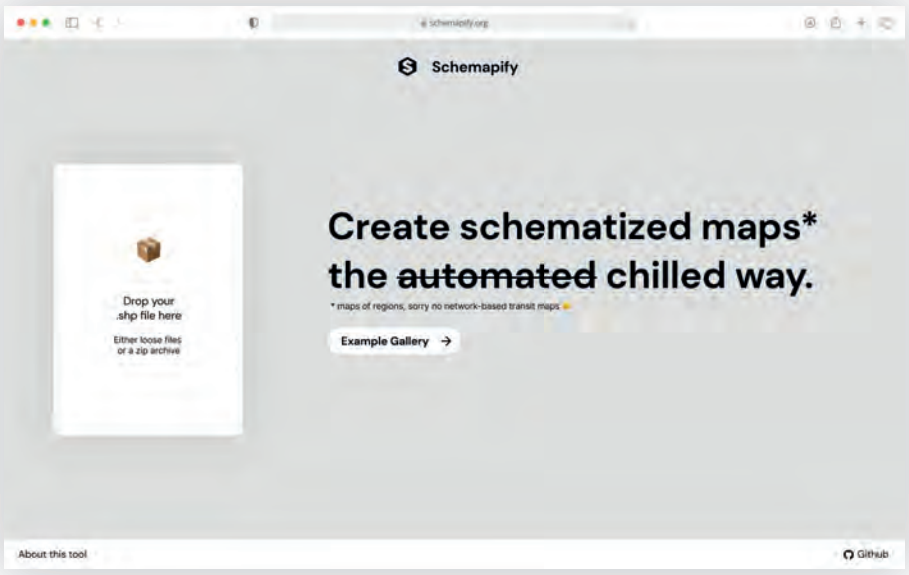

Schemapify
Create schematized maps*
the ~~automated~~ chilled way.
* maps of regions, sorry no network-based transit maps
Example Gallery →
Drop your .shp file here
Either loose files or a zip archive
About this tool
Github

Figure 4.5.: The proposed design for the screen of the web service when opened.

necessary contrast for the salient UI elements. The GUI also ought to adapt to the system's GUI by implementing a dark mode via Cascading Style Sheets (CSS) queries.

Considering the identified user types, the start screen (Figure 4.5) aims to appeal to GIS experts as well as professionals with a graphic design background. It uses simple words, i.e., no technical terms. Furthermore, a link to an example gallery is planned to show examples of schematized regions created with the tool or even finished maps which draw on schematized regions generated with the tool. Note that at this stage, the above-mentioned map-view including the control panel is hidden as they are only initialized once data is imported into the tool.

This floating panel adapts depending on the context: when the application starts, it offers to import data. Afterwards, it facilitates the configuration of the schematization parameters and finally, it tracks the schematization process. Figure 4.6 shows this state of the floating panel.

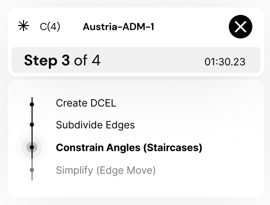


Figure 4.6.: The floating panel shows the advance in the schematization process while the algorithm is running.

4.4.2. User-defined parameters

To set the parameters of the C-oriented schematization, the floating panel adapts, showing the defined orientations of C. This component ought to redraw immediately after the user changes the values in the input field directly underneath the component. It is planned to be implemented with an SVG, which is partially implemented and successfully tested in the proof-of-concept prototype. Figure 4.7 shows this state of the floating panel.

The set of orientations C is the parameter with the visually most characteristic effect on the resulting schematization. The configuration of C is therefore visually a prominent part of the setup component (see Figure 4.7). The component displays the set of orientations. This interactive visual is drawn and updated instantly to provide the user with a prompt feedback and an intuitive approach for conveying the concept of the C-oriented schematization style. Likewise to the implementation of C, the directions are defined, assuming a horizontal line of 0°. Accordingly, a rectilinear schematization consists of two orientations: 0° and 90°. Also, the user interface limits the value for the number of orientations for both cases, regular and irregular Cs, to a logical minimum of two and a defined maximum of twelve. In the case of a regular C, the number of orientations is provided by a simple HyperText Markup Language (HTML) input field of type number. The interactive visual is updated respectively. For a regular C, the user can also specify a β-shift. With that, the user can offset the orientations. In the case of an irregular C, the user needs to specify the precise angle of the orientations. These angles can be either set by dragging the corresponding line within the interactive visual element or by adjusting the values within the respective HTML text input. Default values are defined for the setup of C, assuming that an average user chooses an octilinear schematization (see Figure 2.5. With these values, the user shall be able to quickly use the schematization tool.

The parameter λ determines the maximum edge length of the subdivision prior to algorithm 1. It is assumed with a default value of 0.05. It can be adjusted by the user via an HTML input field of type number.

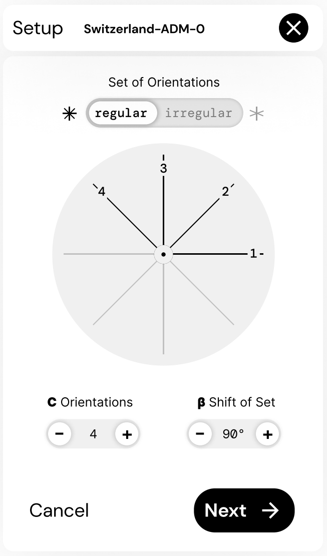


Figure 4.7.: The floating panel allows to define C, providing instant feedback for the user via its interactive SVG element.

Similarly k, determining $|S|$, the minimum complexity of the resulting region, can be set by the user. A reasonable default value for k remains an unresolved issue (Buchin et al., 2016). It may be a certain ratio of the input's complexity $|S|$, which is possibly dependent on the amount of junctions, i.e., the graph's density. Further research is needed to identify a method to determine an acceptable default value for k based on input data.

The UI component shown in Figure 4.6 enables the user to select an intermediate algorithm step to be rendered in the map-view. The visual inspection of the intermediate results can assist in understanding how each of the three discussed parameters affects the schematization result.

5. Results and discussion

The core of this research, the main research objective, is the prototype implementation for a web-based schematization tool. In this chapter, I discuss the prototype as main result, evaluate it with specified requirements and sketch possible directions for further research and open questions.

The prototype has, by definition, a limited functionality. It consists of a basic UI based on the envisioned design for the first product release and on a preliminary implementation of the schematization approach "Area-Preserving Simplification and Schematization of Polygonal Subdivisions" by Buchin et al. (2016). Therefore, following Wiegers and Beatty (2013), the prototype is an evolutionary, electronic proof-of-concept prototype. It is exclusively built with and based on open-source software. A repository containing the code is publicly available on github: github.com/jakoblistabarth/area-preserving-polygon-schematization

5.1. Algorithm limitations

The algorithm, implemented in the proof-of-concept prototype, is limited by the following factors. Firstly, the edge-move cannot consider configurations containing an inner vertex with four or more incident edges. Furthermore, Buchin et al. mention another, though minor drawback based on geometric properties of the input data: a small angle between two edges which are incident to the same vertex can require consideration when these two edges are converted into staircases to prevent interference. This situation of deviating edges increases the number of staircase steps, resulting in more edges to be simplified. This significantly affects the entire schematization's running time, as algorithm 2 has a quadratic running time. However, such a "situation rarely occurs for territorial outlines" (Buchin et al., 2016, p. 19).

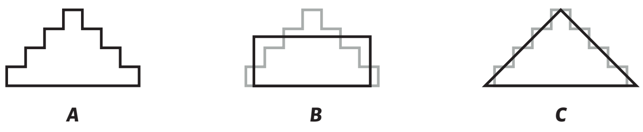


Figure 5.1.: This figure shows the algorithmic approach's limitation regarding the introduction of orientations given in C. ***A*** shows a rectilinear "triangle" as a potential result of an octilinear orientation-restriction. The result obtained with the discussed algorithmic approach is always rectilinear ***B***. Nevertheless, in certain cases, an octilinear result ***C*** may be preferred. Adapted from Figure 22 in Buchin et al., 2016.

The C-oriented schematization by (Buchin et al., 2016) is a heuristic approach. It has drawbacks regarding the obtained geometry when compared to C-oriented schematizations employing a map-matching approach. This drawbacks are namely less uniform edge lengths and potential visual collapses. Nevertheless, particular to this specific heuristic approach of Buchin et al. is that the simplification algorithm, by design, does not allow introducing new orientations. Yet, in certain situations, it may be beneficial to introduce orientations which are contained in C yet do not occur in the affected staircases (see Figure 5.1). Buchin et al. mention a triangle as example: with an octilinear schematization, the intermediate result after the orientation-restriction can be a rectilinear "triangle". Such a shape always remains rectilinear, even though an octilinear representation would enable a lower visual complexity. Thus, it is preferable over the rectilinear representation.

5.2. Implementation limitations

The following section lists the major limitations in the current implementation of the C-oriented schematization algorithm proposed by Buchin et al. These limitations were so severe that they did not allow a meaningful visual interpretation of the obtained results. Therefore, computational benchmarks could not have revealed relevant insights. Hence, a section on a visual evaluation of the results as well as a section on details regarding the implementation's running time based on different input was discarded.

The first shortcoming regards algorithm 1: the number of steps for the staircases of unaligned deviating edges is not calculated correctly. Such unaligned deviating edges pose a special case within the orientation-restriction algorithm. The bug results in interfering staircases, which alter the topology of the schematized region.

Secondly, the algorithm is not robust enough regarding misaligned vertices in algorithm 1. It creates a nearly infinite amount of steps for staircases where two vertices, which normally are one junction, are incongruent. This can increase the running time significantly or even lead to a system failure. This seems to be a relevant issue, as it has occurred during the development using geographic data for tests from the widely used data source Natural Earth released among others by the North American Cartographic Information Society (NACIS). An open question is how such cases can be identified in a preprocessing step prior to algorithm 1. Additionally, if it proves insufficient to validate data beforehand and accordingly reject data exhibiting certain geometric particularities, possibilities to avoid generating superfluously detailed staircases during algorithm 1 need to be found.

A minor drawback of the specific implementation relates to the algorithm for the orientation assignment of the edges around significant vertices in algorithm 1. This assignment poses a preliminary step to construct staircases in algorithm 1. At this writing it is implemented using a simple brute-force algorithm. However, a branch-and-bound-approach is preferable (W. Meulemans, personal communication, June 29, 2021). Nevertheless, this may not affect the running time under average conditions (C_2 to C_8). Furthermore, algorithm 2 is the bottleneck in terms of complexity and resulting running time. Hence, decreased performance due to this simple implementation may be irrelevant once the product is in use.

Another shortcoming, which is even more severe than the already mentioned shortcomings related to algorithm 1, is the very limited implementation of algorithm 2. At this writing, the simplification algorithm's implementation only allows the simplification of simple outlines. Schematization of regions with adjacent faces and shared boundaries is therefore not yet possible. Nevertheless, even for outlines certain configuration combinations can cause either an infinite loop or introduce new orientations. This is particularly the case in situations when the two configurations' inner edges are separated by one edge.

Also, handling visually superfluous vertices after algorithm 1 is not thoroughly considered in this implementation: vertices whose incident edges point exactly in opposite directions do not increase the intermediate result's visual complexity. Nevertheless, they significantly increase the number of edges and vertices to be examined in algorithm 2. The implementation removes such vertices by adapting the edge-move algorithm. Nevertheless, other approaches may increase the algorithmic approach's overall performance. One option is to eliminate such points based on a second edge-classification, implemented after constructing the staircases.

Furthermore, the current implementation uses a simple approach when determining configurations' feasibility for edge-moves in algorithm 2. At this writing, a configuration is valid if no blocking point exists, i.e., the blocking number equals zero. A more advanced implementation would allow to consider blocking points instead of a simple blocking number. Considering the blocking point's position, compensations which only shift so far that they do not cross the (closest) blocking point are also valid. Such an approach would increase the implementation's flexibility in algorithm 2.

5.3. Prototype evaluation

The requirements, structured in several levels, serve as the basis for the prototype evaluation. According to the releases' scope of the schematization tool, the conducted requirement verification draws on the functional requirements targeted for the alpha release, i.e., the proof-of-concept prototype (see Table 4.2). Since the prototype starts an iterative process, this evaluation poses only an intermediate result. For conducting the verification I used unit tests, where possible, and demonstration or code inspection in the remaining cases. For the unit tests, a test for each functional requirement was created. The specifications used in this automated verification are also included in the public github repository. See Appendix D for the test protocol which was creating during the verification. The verification result is discussed in the following: I outline the most critical shortcomings and possible solutions.

Table 5.1.: Requirement verification

FR	Requirement	met
2-a	The system shall be able to parse geoJSON as input data.	**y**
3-a	If the input data is not a region (if it contains features of type other than "polygon" or "multipolygon") the program shall exit and the user shall be informed.	**y**
4-a	If the input data is not a valid geoJSON the program shall exit and the user shall be informed.	**n**
5-a	If the input data is too detailed, i.e., if it exceeds a maximum number of edges or vertices, the program shall exit and the user shall be informed.	**y**
6-a	The system shall preserve potential attributes attached to the inputs features in the output.	**n**
7-a	The system shall preserve the number of features of the input in the output.	**n**
8-a	The system shall be able to generate a DCEL from a geoJSON.	**y**
9-a	The system shall be able to generate a geoJSON from a DCEL.	**y**
10-a	While the data is being processed, the user shall be informed that the application is processing.	**n**
11-a	The user shall be able to specify a regular set of directions (without β-shift) of the schematization.	**n**
17-a	The system shall display the schematized region in the map-view after the schematization is finished.	**y**
22-a	The user shall be able to track the progress of the schematization.	**n**

Table 5.1 shows the requirement verification's results. A **y** indicates that the requirement was met, a **n** that it was not met. Considering the listed requirements, they roughly structure into requirements related to the input (*FR 2-a* to *5-a*), those related to ensuring a valid result of the schematization process (*FR 6-a* to *9-a*) and those related to providing the user with useful feedback (*FR 10-a* and 11-a, *17-a* and *FR-22-a*). *FR 2-a, 3-a, 4-a* and *5-a* concern the parsing and the validation of the input data. The majority of these requirements were met. Only those requirements which premise the schematization of regions – and not only outlines – failed: their schematization is not supported at this writing. In addition, the validation of the input's level of detail requires further attention: in the current implementation, the number of vertices is used with a threshold of 5,000 vertices. Yet, this number does not take into account the region's density, considering the region as a planar graph. To account for that, an improved requirement implementation may draw on the number of edges in relation to the number of vertices of the created DCEL. This would factor in the graph's characteristics, but at the same time increase the time for user feedback. This is due to the generation of the DCEL itself, which is preliminary to such an implementation: the running time for such a DCEL generation may significantly increase for highly complex, i.e. detailed, regions. The verification of the requirements regarding the conversion to the DCEL data structure and back to the geoJSON data

structure is positive. However, further iterations concerning these conversions may be necessary to thoroughly test for failures in particular region cases, and to detect and solve performance issues. *FR-6a* and *7a* touch the integrity of the schematized results: even though the integrity requirements are moderate in respect to further product releases, these two requirements could not be verified. Further effort in this directions is needed to achieve a reliable prototype.

Yet, this verification-process setup based is limited. This is mainly because of the constrained implementation of the schematization algorithm. Consequently, requirements regarding the validity of the results and also regarding certain events related to the GUI can only be verified to a very limited extent. This concerns *FR 6-a*, *7-a* and *FR 10-a*, *11-a* and *22-a*. Tests using a headless browser setup would enable an automated verification of such requirements, which are directly related to the GUI. Nevertheless, verifying these requirements with demonstration and code inspection shows that, for the current stage of the GUI implementation, user feedback is not yet implemented in a required manner: at this writing, error handling and feedback on the data–processing progress is limited to (error) logs in the browser's console. Therefore, *FR 10-a*, *11-a* and 22-a fail.

The requirement verification reveals that the requirements on validating user-defined input data and on the integrity of the resulting schematized region are mainly covered. In contrast, those requirements targeting feedback to the user are hardly met. This demonstrates the need to put more effort into the implementation and design of the GUI. For now, the web service including the GUI is intended to be implemented using facebook's JS library *react*. Due to its component-based approach, combined with the possibility of maintaining an application-wide or component-wide state, it is widely used to build interactive GUIs. For further verification, particularly after implementing the GUI, advanced testing frameworks will improve the validity of the requirement verification results.

5.4. Future research

To answer RQ A in more detail, further research may compare implementations of approaches to polygonal schematization. The comparison of such implementations may generate insights into the algorithm's time and space costs. Furthermore, such a comparison may reveal geometric characteristics of each approach regarding the visual output.

The proof-of-concept prototype reveals stability issues, which such a tool faces in day-to-day use: further effort is needed to increase the implementation's robustness. This can either be implemented by adapting or extending corresponding algorithms in the approach itself or by thoroughly validating data prior to schematization. Related to the stability concern, but also an important factor for the tool's performance, is the input's complexity, i.e., the input's data level of detail. The accuracy level which is reasonable for input data remains unclear in two instances: the minimum level of detail needed to create schematic shapes which resemble a geographic feature's known shape and the maximum complexity level which should not be exceeded to quickly process. Further exploration on integrating the simplification prior to the actual schematization into the tool may be beneficial, either by adopting an existing solution or by implementing a tailored one.

In respect to the tool's usability, further research may focus on increasing usability by determining appropriate default settings and parameters. This depends on the input data and on the intended schematization. Automatically suggested schematization setups are based on geometric characteristics of the input data which are combined with the intended map context. They could only determine individual parameters within a user-defined schematization style, but could also already suggest a certain schematization style. Related to the algorithm implemented within this research, further research may be carried out regarding the parameters k and C (Buchin et al., 2016), but also regarding λ. To answer the underlying questions on automatically deciding on parameters or schematization styles, it may help to research beneficial styles from a map user's point of view, regarding the map's legibility. Another aspect concerning usability is how to ideally implement such a tool in a map-making workflow. Research focusing on this aspect, may benefit from eliciting further software requirements, which consequently require new interfaces, e.g. Application Programming Interfaces (APIs), or Web Feature Services (WFSs) and Web Map Services (WMSs) capabilities. Concerning the integration into map-making workflows, future research may explore possibilities of further stylize the resulting schematized polygons, e.g. by rounding corners at vertices of degree two.

6. Conclusion

Based on implementing a proof-of-concept prototype for a web-based schematization tool, the aim of the thesis is to evaluate the practical feasibility of such a tool. To this end, cartographic requirements for using schematized regions in thematic maps were examined. This examination was led by design principles which scholars mention regarding the base map of thematic maps (RQ A). Then, a suitable schematization approach was chosen, based on comparing existing algorithmic approaches (RQ B). Furthermore, crucial software requirements regarding a web-based schematizations tool were specified (RQ C). Finally, the prototype was evaluated on the basis of these requirements (RQ D).

To obtain these results, a literature review was conducted to identify characteristics of polygon schematization. By means of these characteristics, existing approaches were systematically compared. Furthermore, the prototyping method was applied in combination with a simplified software-requirement-engineering process. Using prototyping as a method implies an iterative development. The visual design revisions of a singular component of the GUI are an example for this iterative design process. The chosen algorithm was implemented within a technical and incremental proof-of-concept prototype. For this, requirements were used to outline aspects which need to be considered from a technical but particularly from a cartographic point of view, i.e., the design principles. These requirements contain several levels of requirement information and build in addition the basis for the concluding prototype evaluation in the requirement verification.

The design principles for applying schematized regions in thematic maps show that visual characteristics of schematic outlines are in line with design principles for thematic mapping. Nevertheless, literature shows as well that schematization is only one mean of designing maps, which on its own cannot guarantee a legible, efficiently designed map. The comparison demonstrate that existing algorithmic approaches cover various schematization styles and exhibit heterogeneous geometric properties. Nevertheless, only few preserve area and topology, which is indispensable for most cartographic purposes. Furthermore, the computational complexity resulting in long running times for complex input data poses a technical constraint. The C-oriented schematization approach by Buchin et al. (2016) was chosen due to its flexibility regarding visual customization of the schematization style and its comparatively low complexity. The specified software requirements expose that validating input data for geometric particularities is a crucial preliminary for a stable system. Furthermore, they reflect the need to consider error sources, particularly for the orientation-restriction algorithm, originating from the input data's geometry. The remaining requirements concern handling projections (and respectively handling data which is not projected but in a geographic CRS), and the design of an interface which provides meaningful feedback to the user and allows an efficient setup of the schematization parameters. The evaluation, conducted with the requirement verification, particularly reveals shortcomings on system capabilities and the current GUI implementation. Therefore, upcoming development and prototyping steps will focus on these aspects.

The results of the requirement verification, together with the release scope and the functional requirements in Appendix E, pose the basis for a road map towards a release of the schematization tool. The ideally iterative nature of such a software development process requires further work on the carried-out prototype. This includes thoroughly validating the specified software requirements, conducting the proposed user study (see Appendix C), and iteratively implementing the gained feedback into the prototype.

Implementing this algorithm in an accessible web-based cartographic service embodies a series of challenges regarding robustness, performance and usability. This thesis points out these challenges. However, it exceeds this thesis' scope to provide ideas on how to overcome such constraints. Therefore, further research on the intersection between the theoretical design of schematization algorithms and their practical implementation is needed. Moreover, further research may include a systematical comparison of existing schematization approaches based on computational benchmarks and a visual analysis of the results obtained in comparable implementations.

Given the growing interest for automated schematizations concerning network as well as polygon schematization on the one hand and the time-consuming process of manual schematization on the other, the need for a practical application of published algorithms is evident. Throughout this research project, aspects of this implementation process were addressed. The carried-out results aim to fill this gap in cartographic literature on map schematization. This thesis outlines the process of implementing such an algorithm and shows that the proposed web-based schematization tool is feasible. Nevertheless, computational time constraints decreased performance and, above all, the tool's robustness needs further attention. Only then can the approach be transferred into a useful tool, which blends in smoothly into common map-making workflows.

7. List of references

Arnberger, E. (1966). *Handbuch der thematischen Kartographie*. Deuticke
Open Library ID: OL5574992M.

Avelar, S. (2002). *Schematic maps on demand: Design, modeling and visualization* (Doctoral dissertation). ETH Zurich. https://doi.org/10.3929/ETHZ-A-004443857

Barkowsky, T., Latecki, L. J., & Richter, K. .-. (2000). Schematizing Maps: Simplification of Geographic Shape by Discrete Curve Evolution. In C. Freksa, C. Habel, W. Brauer, & K. F. Wender (Eds.), *Spatial Cognition II: Integrating Abstract Theories, Empirical Studies, Formal Methods, and Practical Applications* (pp. 41–53). Springer. https://doi.org/10.1007/3-540-45460-8_4

Beck, H. C. (1933). Map of London's Underground Railways. Underground. A new design for an old map. We should welcome your comments. Please write to Publicity Manager, 55, Broadway, Westminster, S.W.1 Waterlow & Sons Limited, London, Dunstable & Watford. (750M - 1-33).

Bertin, J. (1974). *Graphische Semiologie: Diagramme, Netze, Karten* (W. Scharfe, Ed.; G. Jensch & D. Schade, Trans.; Translated and rev. from the 2nd french ed. edition). de Gruyter.

Bertin, J. (2011). *Semiology of graphics: Diagrams, networks, maps* (W. J. Berg, Trans.). ESRI Press.

Berzins, V. (2003). Software prototyping. *Encyclopedia of Computer Science* (pp. 1636–1638). John Wiley and Sons Ltd.

Biederman, I. (1987). Recognition-by-components: A theory of human image understanding. *Psychological review*, *94*(2), 115.

Bostock, M., & Metcalf, C. (2021). The TopoJSON Format Specification.

Buchin, K., Meulemans, W., Renssen, A. V., & Speckmann, B. (2016). Area-Preserving Simplification and Schematization of Polygonal Subdivisions. *ACM Transactions on Spatial Algorithms and Systems*, *2*(1), 1–36. https://doi.org/10.1145/2818373

Buckley, A., & Watkins, D. (2007). Automated map production workflows. *24th International Cartographic Conference*.

Bünting, H. (1581). Die ganze Welt in einem Kleberblatt welches ist der Stadt Hannover meines lieben Vaterlandes Wapen.

Burghardt, D., Duchene, C., & Mackaness, W. (Eds.). (2014). *Abstracting Geographic Information in a Data Rich World: Methodologies and Applications of Map Generalisation*. Springer International Publishing. https://doi.org/10.1007/978-3-319-00203-3

Butler, H., Daly, M., Doyle, A., Gillies, S., Schaub, T., & Schaub, T. (2016). *The GeoJSON Format* (Request for Comments RFC 7946). Internet Engineering Task Force. https://doi.org/10.17487/RFC7946

Cormen, T. H. (Ed.). (2009). *Introduction to algorithms* (3rd ed). MIT Press
OCLC: ocn311310321.

Davis, A. (1992). Operational prototyping: A new development approach. *IEEE Software*, *9*(5), 70–78. https://doi.org/10.1109/52.156899

de Berg, M., Cheong, O., van Kreveld, M., & Overmars, M. (2008). *Computational Geometry: Algorithms and Applications* (Third Edition). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-77974-2

Dent, B. D., Torguson, J., & Hodler, T. W. (2009). *Cartography: Thematic map design* (6th ed). McGraw-Hill Higher Education
OCLC: ocn184827987.

Ding, L., & Meng, L. (2014). A comparative study of thematic mapping and scientific visualization. *Annals of GIS*, *20*(1), 23–37. https://doi.org/10.1080/19475683.2013.862856

Djordjevic, N. (2017). Evaluation of the usability of web-based applications. *Vojnotehnicki glasnik*, *65*(3), 785–802. https://doi.org/10.5937/vojtehg65-11319

Douglas, D. H., & Peucker, T. K. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization*, *10*(2), 112–122.

Dubrau, A., & Bagel, J. (2016). Transit Maps: Apple vs. Google vs. Us.

Dykes, J., Muller-Hannemann, M., & Wolff, A. (2010). Schematization in Cartography, Visualization, and Computational Geometry. *Schematization in Cartogra- Phy, Visualization, and Computational Geometry*, 36.

Eppstein, D., van Kreveld, M., Speckmann, B., & Staals, F. (2013). Improved grid map layout by point set matching. *2013 IEEE Pacific Visualization Symposium (PacificVis)*, 25–32. https://doi.org/10.1109/PacificVis.2013.6596124

Fabrikant, S. I. (2004). Abstraction and Scale in Spatialisation, 9.

Folkmann, M. N. (2013). *The aesthetics of imagination in design*. MIT Press.

Harrower, M., & Bloch, M. (2006). MapShaper.org: A map generalization Web service. *IEEE Computer Graphics and Applications*, *26*(4), 22–27. https://doi.org/10.1109/MCG.2006.85

Heimlich, M., & Held, M. (2008). Biarc approximation, simplification and smoothing of polygonal curves by means of voronoi-based tolerance bands. *International Journal of Computational Geometry & Applications*, *18*(03), 221–250. https://doi.org/10.1142/S0218195908002593

Herskovits, A. (1998). Schematization. *Representation and processing of spatial expressions* (pp. 149–162). Lawrence Erlbaum Associates Publishers.

Imhof, E. (1972). *Thematische Kartographie* (1st edition). de Gruyter.

Jansen, W. (2009). Neurath, Arntz and ISOTYPE: The Legacy in Art, Design and Statistics. *Journal of Design History*, *22*(3), 227–242. https://doi.org/10.1093/jdh/epp015

Karmakar, N., Bhowmick, P., & Biswas, A. (2014). Segmentation of 3D Articulated Components by Slice-Based Vertex-Weighted Reeb Graph. In E. Barcucci, A. Frosini, & S. Rinaldi (Eds.), *Discrete Geometry for Computer Imagery* (pp. 370–383). Springer International Publishing. https://doi.org/10.1007/978-3-319-09955-2_31

Karmakar, N., Biswas, A., & Bhowmick, P. (2012). Fast Slicing of Orthogonal Covers Using DCEL. In R. P. Barneva, V. E. Brimkov, & J. K. Aggarwal (Eds.), *Combinatorial Image Analaysis* (pp. 16–30). Springer. https://doi.org/10.1007/978-3-642-34732-0_2

Kazmierczak, E. T. (2003). Design as Meaning Making: From Making Things to the Design of Thinking. *Design Issues*, *19*(2), 45–59.

Klippel, A., Richter, K.-F., Barkowsky, T., & Freksa, C. (2005). The Cognitive Reality of Schematic Maps. In L. Meng, T. Reichenbacher, & A. Zipf (Eds.), *Map-based Mobile Services* (pp. 55–71). Springer-Verlag. https://doi.org/10.1007/3-540-26982-7_5

Kraak, M. J., & Ormeling, F. (2010). *Cartography: Visualization of geospatial data* (3rd ed). Prentice Hall
OCLC: ocn477062041.

Krug, S. (2009). *Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability Problems* (1st edition). New Riders.

Krug, S. (2013). *Don't Make Me Think: A Common Sense Approach to Web Usability* (revised edition). New Riders.

Latecki, L. J., & Lakämper, R. (1999). Convexity Rule for Shape Decomposition Based on Discrete Contour Evolution. *Computer Vision and Image Understanding*, *73*(3), 441–454. https://doi.org/10.1006/cviu.1998.0738

Levasseur, P. É. (1876). Budget. 1 millim. c. représente 1.156.000 fr. de dépense nette.

Mackaness, W., Burghardt, D., & Duchêne, C. (2014). Map Generalisation: Fundamental to the Modelling and Understanding of Geographic Space. In D. Burghardt, C. Duchêne, & W. Mackaness (Eds.), *Abstracting Geographic Information in a Data Rich World: Methodologies and Applications of Map Generalisation* (pp. 1–15). Springer International Publishing. https://doi.org/10.1007/978-3-319-00203-3_1

Mackaness, W., & Reimer, A. (2014). Generalisation in the Context of Schematised Maps. In D. Burghardt, C. Duchêne, & W. Mackaness (Eds.), *Abstracting Geographic Information in a Data Rich World: Methodologies and Applications of Map Generalisation* (pp. 299–328). Springer International Publishing. https://doi.org/10.1007/978-3-319-00203-3_10

Marr, D., & Nishihara, H. K. (1978). Representation and recognition of the spatial organization of three-dimensional shapes. *Proceedings of the Royal Society of London. Series B, Biological Sciences*, *200*(1140), 269–294. https://doi.org/10.1098/rspb.1978.0020

Meulemans, W. (2014). *Similarity measures and algorithms for cartographic schematization* (Doctoral dissertation). Technische Universiteit Eindhoven. Eindhoven.

Meulemans, W. (2016). Discretized Approaches to Schematization. *arXiv:1606.06488 [cs]*.

Meulemans, W., van Renssen, A., & Speckmann, B. (2010). Area-Preserving Subdivision Schematization. In D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, S. I. Fabrikant, T. Reichenbacher, M. van Kreveld, & C. Schlieder (Eds.), *Geographic Information Science* (pp. 160–174). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-15300-6_12

Monmonier, M. S. (1991). *How to lie with maps*. University of Chicago Press.

Moore, G. A. (2014). *Crossing the Chasm, 3rd Edition: Marketing and Selling Disruptive Products to Mainstream Customers* (3rd edition). Harper Business.

Muehlenhaus, I. (2010). *Lost in visualization: Using quantitative content analysis to identify, measure, and categorize political cartographic manipulations.* (Doctoral dissertation)
Accepted: 2010-03-18T17:03:42Z.

Muehlenhaus, I. (2011). Genealogy That Counts: Using Content Analysis to Explore the Evolution of Persuasive Cartography. *Cartographica: The International Journal for Geographic Information and Geovisualization*, *46*(1), 28–40. https://doi.org/10.3138/carto.46.1.28

Muehlenhaus, I. (2012). If Looks Could Kill: The Impact of Different Rhetorical Styles on Persuasive Geocommunication. *The Cartographic Journal*, *49*(4), 361–375. https://doi.org/10.1179/1743277412Y.0000000032
_eprint: https://doi.org/10.1179/1743277412Y.0000000032

Muehlenhaus, I. (2013). The design and composition of persuasive maps. *Cartography and Geographic Information Science*, *40*(5), 401–414. https://doi.org/10.1080/15230406.2013.783450

Murugesan, S. (2008). Web Application Development: Challenges And The Role Of Web Engineering. In G. Rossi, O. Pastor, D. Schwabe, & L. Olsina (Eds.), *Web Engineering: Modelling and Implementing Web Applications* (pp. 7–32). Springer London. https://doi.org/10.1007/978-1-84628-923-1_2

Neumann, J. (1997). *Enzyklopädisches Wörterbuch Kartographie in 25 Sprachen* (2., erw. Ausg.). K.H. Saur Verlag GmbH.

Nielsen, J. (1992a). The usability engineering life cycle. *Computer*, *25*(3), 12–22. https://doi.org/10.1109/2.121503

Nielsen, J. (1995). Applying discount usability engineering. *IEEE Software*, *12*(1), 98–100. https://doi.org/10.1109/52.363161

Nielsen, J. (1992b). Finding usability problems through heuristic evaluation. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '92*, 373–380. https://doi.org/10.1145/142750.142834

Nielsen, J. (1993). *Usability Engineering* (1st). Morgan Kaufmann.

Nielsen, J. (1994). Estimating the number of subjects needed for a thinking aloud test. *International Journal of Human-Computer Studies*, *41*(3), 385–397. https://doi.org/10.1006/ijhc.1994.1065

Paetsch, F., Eberlein, A., & Maurer, F. (2003). Requirements engineering and agile software development. *WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.*, 308–313. https://doi.org/10.1109/ENABL.2003.1231428

Raisz, E. (1944). Population Problems.

Raisz, E. (1948). *General cartography* (Second). McGraw-Hill Book Co.

Raisz, E. (1962). *Principles of cartography*. McGraw-Hill.

Raposo, P., Touya, G., & Bereuter, P. (2020). A Change of Theme: The Role of Generalization in Thematic Mapping. *ISPRS International Journal of Geo-Information*, *9*(6), 371. https://doi.org/10.3390/ijgi9060371

Reimer, A. (2010). Understanding Chorematic Diagrams: Towards a Taxonomy. *The Cartographic Journal*, *47*(4), 330–350. https://doi.org/10.1179/000870410X12825500202896

Reimer, A., & Fohringer, J. (2010). Towards constraint formulation for chorematic schematisation tasks - work in progress, 18.

Reimer, A., & Meulemans, W. (2011). Parallelity in Chorematic Territorial Outlines. https://doi.org/10.13140/2.1.2189.0566

Roberts, M. J. (2014). What's Your Theory of Effective Schematic Map Design? *Schematic Mapping Workshop*, 9.

Roberts, M. J., Newton, E. J., & Canals, M. (2016). Radi(c)al departures Comparing conventional octolinear versus concentric circles schematic maps for the Berlin U-Bahn/S-Bahn networks using objective and subjective measures of effectiveness. *Information Design Journal*, *22*(2), 92–115.

Roberts, M. J., & Vaeng, C. N. (2016). Expectations and prejudices usurp judgements of schematic map effectiveness. *Design Research Society Conference 2016*. https://doi.org/10.21606/drs.2016.123

Roth, R. E., Brewer, C. A., & Stryker, M. S. (2011). A typology of operators for maintaining legible map designs at multiple scales. *Cartographic Perspectives*, (68), 29–64. https://doi.org/10.14714/CP68.7

Saha, S., & Biswas, A. (2021). Surface polygonization of 3D objects using norm similarity. *Journal of Combinatorial Optimization*. https://doi.org/10.1007/s10878-021-00786-2

Schulz, T. (2014). *Der Statistische Atlas. Untersuchungen zu klassifikatorischen, inhaltlichen, gestalterischen, technischen und kommunikativen Aspekten.* (Doctoral dissertation). Technische Universität Dresden. Dresden.

Shen, Y., Ai, T., & He, Y. (2018). A New Approach to Line Simplification Based on Image Processing: A Case Study of Water Area Boundaries. *ISPRS International Journal of Geo-Information*, *7*(2), 41. https://doi.org/10.3390/ijgi7020041

Slingsby, A., Dykes, J., & Wood, J. (2010). Rectangular Hierarchical Cartograms for Socio-Economic Data. *Journal of Maps*, *6*(1), 330–345. https://doi.org/10.4113/jom.2010.1090

Slocum, T. A., McMaster, R. B., Kessler, F. C., & Howard, H. H. (2009). *Thematic cartography and geovisualization* / (3rd Edition). Pearson Prentice Hall.

Statistisches Jahrbuch für das Deutsche Reich 1904. (1905). Statistischen Reichsamt.

Steadman, P. (2019). Abstraction and Schematization in the Repeated Copying of Designs. *Social Analysis*, *63*(4), 131–148. https://doi.org/10.3167/sa.2019.630407

Swan, J., Anand, S., Ware, M., & Jackson, M. (2007). Automated Schematization for Web Service Applications. In J. M. Ware & G. E. Taylor (Eds.), *Web and Wireless Geographical Information Systems* (pp. 216–226). Springer. https://doi.org/10.1007/978-3-540-76925-5_16

Thayer, R. H., & Dorfman, M. (Eds.). (1997). *Software Requirements Engineering, 2nd Edition* (2nd edition). Wiley-IEEE Computer Society Pr.

Tobler, W. R. (1959). Automation and Cartography. *Geographical Review*, *49*(4), 526–534. https://doi.org/10.2307/212211

Tufte, E. R. (2001). *The Visual Display of Quantitative Information* (Second). Graphics Press.

Tutić, D., & Lapaine, M. (2009). Area Preserving Cartographic Line Generalization. *Kartografija i geoinformacije (Cartography and Geoinformation)*, *8*(11), 84–100.

Tyner, J. A. (2010). *Principles of map design*. Guilford Press OCLC: ocn437300476.

van Dijk, T. C., van Goethem, A., Haunert, J.-H., Meulemans, W., & Speckmann, B. (2014). Map Schematization with Circular Arcs. In M. Duckham, E. Pebesma, K. Stewart, & A. U. Frank (Eds.), *Geographic Information Science* (pp. 1–17). Springer International Publishing. https://doi.org/10.1007/978-3-319-11593-1_1

van Goethem, A., Meulemans, W., Reimer, A., Haverkort, H., & Speckmann, B. (2013). Topologically Safe Curved Schematisation. *The Cartographic Journal*, *50*(3), 276–285. https://doi.org/10.1179/1743277413Y.0000000066

van Goethem, A., Meulemans, W., Speckmann, B., & Wood, J. (2014). Exploring Curved Schematization. *2014 IEEE Pacific Visualization Symposium*, 1–8. https://doi.org/10.1109/PacificVis.2014.11

van Goethem, A., Meulemans, W., Speckmann, B., & Wood, J. (2015). Exploring Curved Schematization. *IEEE Transactions on Visualization and Computer Graphics*, 15.

Visvalingam, M., & Whyatt, J. D. (1990). The Douglas-Peucker Algorithm for Line Simplification: Re-evaluation through Visualization. *Computer Graphics Forum*, *9*(3), 213–225. https://doi.org/10.1111/j.1467-8659.1990.tb00398.x

Visvalingam, M., & Whyatt, J. D. (1993). Line generalisation by repeated elimination of points. *The Cartographic Journal*, *30*(1), 46–51. https://doi.org/10.1179/000870493786962263

Vosatka, M. (2021). Deutsche Fregatte Bayern: Ein Schiff auf umstrittener Mission.

Vujaković, P. (2014). The State as a 'Power Container': The Role of News Media Cartography in Contemporary Geopolitical Discourse. *The Cartographic Journal*, *51*(1), 11–24. https://doi.org/10.1179/1743277413Y.0000000043

Wiegers, K. E., & Beatty, J. (2013). *Software Requirements* (3rd edition). Microsoft Press.

Wright, J. K. (1942). Map Makers Are Human: Comments on the Subjective in Maps. *Geographical Review*, *32*(4), 527–544. https://doi.org/10.2307/209994

Wu, H.-Y., Niedermann, B., Takahashi, S., Roberts, M. J., & Nöllenburg, M. (2020). A Survey on Transit Map Layout – from Design, Machine, and Human Perspectives. *Computer Graphics Forum*, *39*(3), 619–646. https://doi.org/10.1111/cgf.14030

A. Pseudocode implementation of DCEL generation

This appendix describes the approach for creating a Doubly-Connected Edge List data structure out of the common geoJSON format, particularly in the web context, and of converting a DCEL back to geoJSON for displaying it within the tool and for further use or conversion to the desired output format. The necessary conversion of other input formats to geoJSON and finally from the geoJSON back, e.g., to the shp format is done by external libraries.

Preliminaries

S ....GEOJSON, input subdivision

s ...geoJSON feature of S

D ...output DCEL

v ...vertex

V ...list of vertices

e ...half-edge

E ...list of half-edges

f ...face

F ...list of faces

r ...ring

P ...list of feature properties

Conversion of a geoJSON to DCEL

For this approach it is necessary to introduce a new ID (FID) for every feature s of S, and use it to assign every face f of the DCEL D to one (or more) features of S. A face f has two FIDs pointing to a feature s of S when it demarcates an inner border of a feature s which at the same time is an outer border for another feature (e.g., the border of an enclave). Potential feature properties (e.g. thematic data) can be saved separately and merge later on, after converting the DCEL back to a geoJSON, using the respective FID.

A half-edge e is defined as incident to a vertex v if the tail endpoint of the directed edge e, is v. For territorial outlines a vertex v usually does not have more than four incident edges. And at least two incident edges, as all polygons are closed.

Algorithm 1 geoJSON features to DCEL – Initializing of lists

```
1: procedure DCEL GENERATION($S$)
2:     Create an empty DCEL $D$
3:     Create empty vertex list $V$ in $D$
4:     Create empty edge list $E$ in $D$
5:     Create empty face list $F$ in $D$
6:     Create empty list of feature properties $P$ in $D$
7:     for all features $s$ in $S$ do
8:         Add the properties to $P$
9:     end for
```

After the preprocessing step the individual DCEL entities are added to the respective lists. The first entity is the vertex. Every combination of coordinate pairs (position) can occur only once in a DCEL.

Algorithm 2 geoJSON features to DCEL – Add vertices

```
10:     for all features of $S$ do
11:         Add coordinate pairs as vertex $v$ to $V$, unless it does already exist in $V$
12:     end for
```

After that, the half-edges – including their twins – are created and added to the DCEL. The pair of half-edges, making up twins, need to be link to each other. Likewise every half-edge needs to be linked to a previous and a next half-edge.

Algorithm 3 geoJSON features to DCEL – Add half-edges

```
13: for all features $s$ of $S$ do
14:     for all rings $r$ of $s$ do
15:         Discard the last point        ▷ the first and the last point of a geojson's polygon ring is the same
16:         Sort all coordinate pairs of $r$ clockwise
17:         for all coordinate pairs of $r$ do
18:             Get the coordinate pair and the subsequent pair in clockwise direction
19:             Create a pair of $e$, for these two coordinate pairs.        ▷ a pair for $e$ and its corresponding twin
20:             Add each pair of half-edges to $E$
21:             Assign each pair of $e$s as twins
22:             Assign $e$ (and its twin) as incident half-edge to to the corresponding $v$
23:         end for
24:     end for
25: end for
26: for all vertices $v$ in $V$ do
27:     Sort all incident $e$ clockwise
28:     for all pairs of neighboring $e$ incident edges do
29:         Assign next and prev $e$
30:     end for
31: end for
```

As last entity the faces are created and added to the DCEL. Note that one unbounded face needs to be created which has no edge pointer itself (that is how it is identified), but is still needed as every edge needs to be incident to exactly one face. Lastly the entire DCEL is returned.

Algorithm 4 geoJSON features to DCEL – Add faces

```
32:     for all features $s$ of $S$ do
33:         Store current loop index as FID
34:         for all rings $r$ of $s$ do
35:             Sort all coordinate pairs of $r$ clockwise
36:             Get the first two coordinate pairs
37:             Find the $e$ in $E$ which goes from the first to the second point.
38:             if this $e$ has a $f$ assigned then
39:                 Add FID to the ID list of $f$
40:             else
41:                 Create a new face $f$
42:                 Add $f$ to $F$
43:                 Add FID to the ID list of $f$
44:                 Assign $f$ to $e$ and its subsequent edges
45:                 Assign $e$ to $f$
46:                 if $f$ is not the outer most ring then
47:                     Add $e$ to the list of inner edges of the outer ring
48:                     Assign the outer ring as face to the twins of $e$ and its subsequent edges
49:                 end if
50:             end if
51:         end for
52:     end for
53:     Create an unbounded face $u$
54:     while there is an edge $e$ without a face do
55:         Assign $u$ as face to $e$ and it subsequent edges
56:     end while
57:     return $D$                         ▷ The DCEL consisting of vertices, half-edges and faces
58: end procedure
```

Conversion of an DCEL to geoJSON

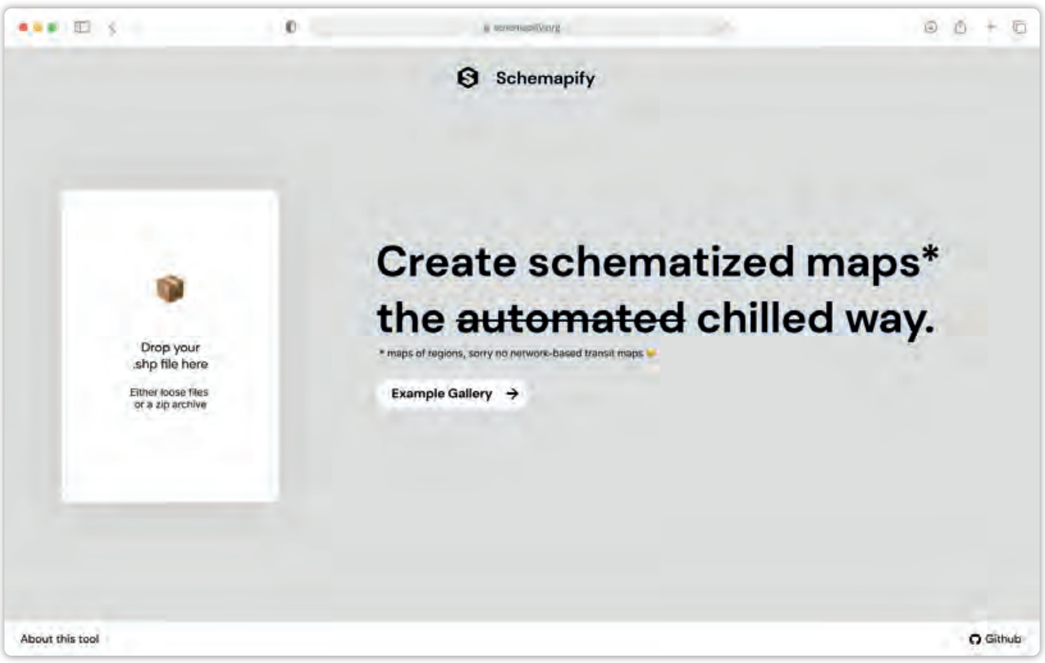
In order to export the schematized region in different industry standard formats, it is necessary to convert it from a DCEL data structure, in which it is schematized, to a geoJSON.

Algorithm 5 DCEL to geoJSON – Group external rings by FID

```
procedure GEOJSON FROM DCEL($D$)
    Create an empty $S$
    Create a list $I$ containing all faces $f$ of $F$ in $D$ which are external rings, grouped by FID
    for all IDs $i$ in $I$ do
        Create a geoJSON feature $s$
        Add properties to $s$ using the FID    ▷ feature properties need to be stored separately ($P$)
        for all Rings $r$ in $i$ do
            Create an empty list $C$ of coordinate pairs
            for all edges $e$ of $r$ in counterclockwise direction do
                Add the tail vertex of $e$ to $C$
            end for
            Add the first vertex of $C$ at the end of $C$
            Add $C$ as external ring to $S$
            if $r$ has inner edges then
                for all inner edges $i$ in $r$ do
                    Create an empty list $C$ of coordinate pairs
                    for all edges $e$ of $r$ in clockwise direction do
                        Add the tail vertex of $e$ to $C$
                    end for
                    Add the first vertex of $C$ at the end of $C$
                    Add $C$ to $r$
                end for
            end if
        end for
        Add $s$ to $S$
    end for
    return $S$    ▷ The geoJSON as a list of multipolygon features
end procedure
```

B. Designs for the mock-up prototype

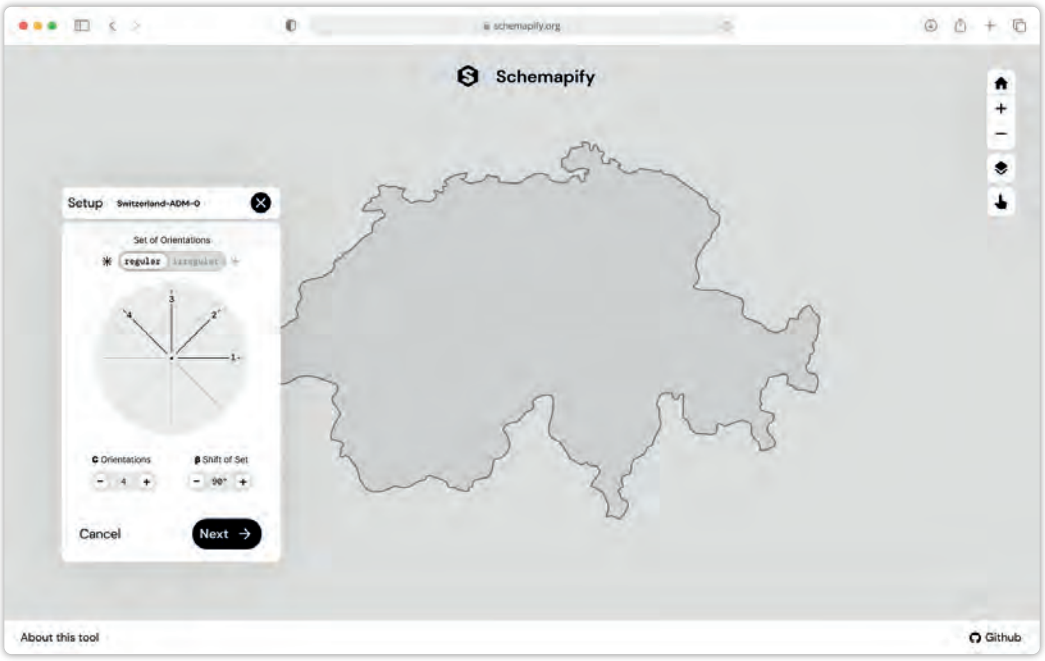
This appendix shows the screens I designed for the mock-up prototype. A clickable, interactive version, created in figma, can be accessed online.



Create schematized maps*
the ~~automated~~ chilled way.

* maps of regions, sorry no network-based transit maps

Figure B.1.: Start screen.



Setup Switzerland-ADM-0

Set of Orientations

regular irregular

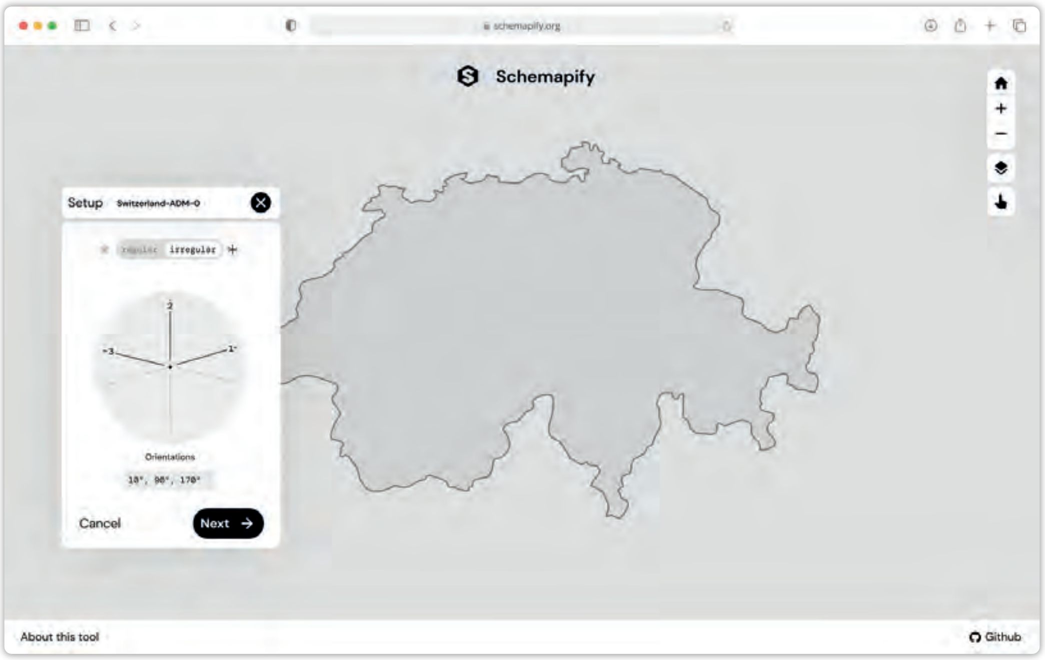
1 2 3 4

C Orientations: 4

β Shift of Set: 90°

Cancel Next

Figure B.2.: Setup of a regular schematization.



Setup Switzerland-ADM-0

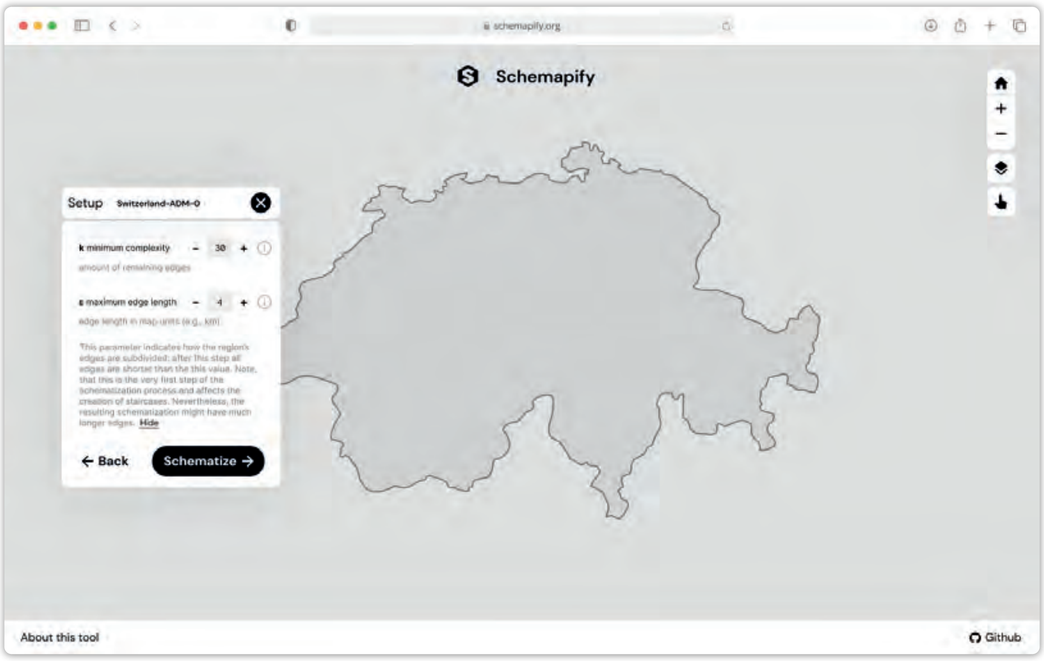
regular irregular

1 2 3

Orientations: 10°, 90°, 170°

Cancel Next

Figure B.3.: Setup of a irregular schematization.



Setup Switzerland-ADM-0

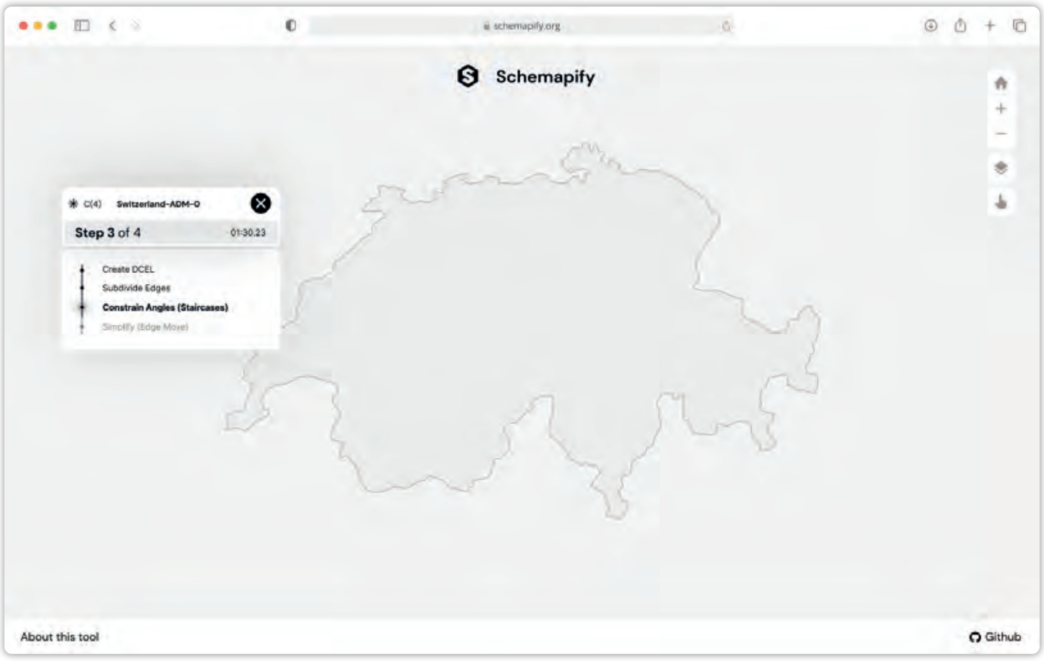
k minimum complexity – 30 +
amount of remaining edges

ε maximum edge length – 4 +
edge length in map units (e.g., km)

This parameter indicates how the region's edges are subdivided: after this step all edges are shorter than the this value. Note, that this is the very first step of the schematization process and affects the creation of staircases. Nevertheless, the resulting schematization might have much longer edges. Hide

← Back Schematize →

Figure B.4.: Setup of a k and λ.



C(4) Switzerland-ADM-0

Step 3 of 4 01:30.23

- Create DCEL
- Subdivide Edges
- **Constrain Angles (Staircases)**
- Simplify (Edge Move)

Figure B.5.: Screen showing the process while schematizing.

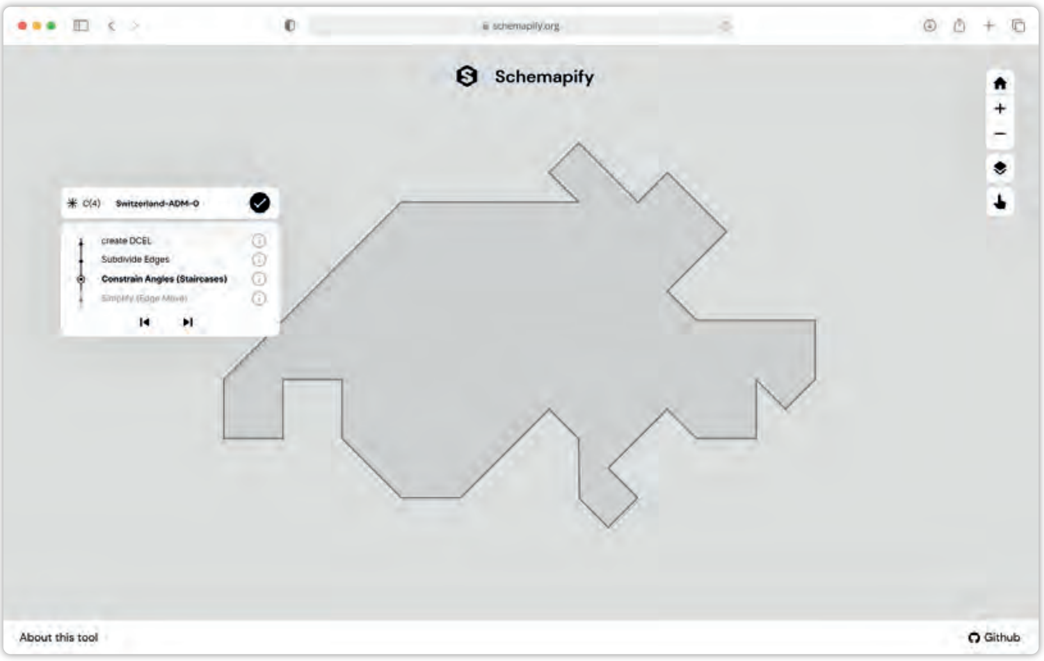


Figure B.6.: Screen after a successful schematization.

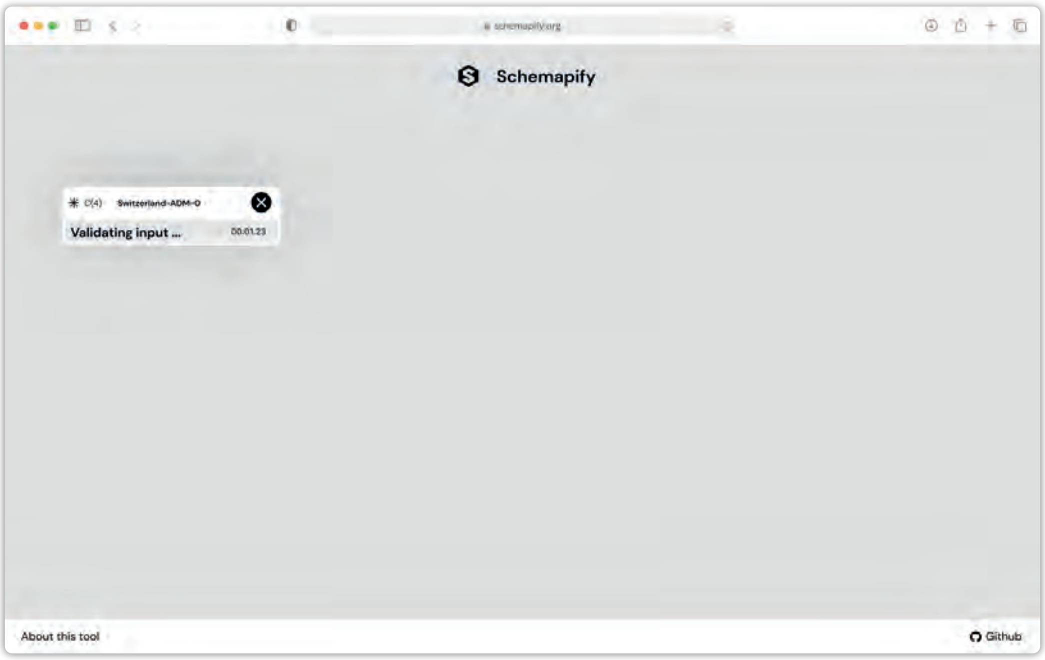
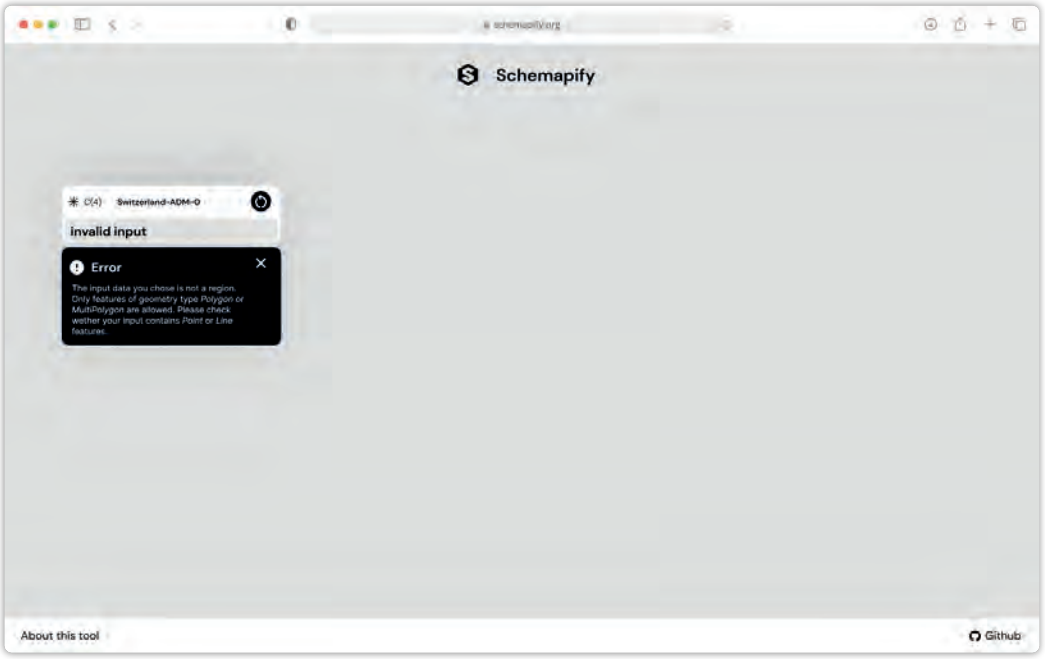


Figure B.7.: Screen during the input validation.



invalid input

Error

The input data you chose is not a region. Only features of geometry type *Polygon* or *MultiPolygon* are allowed. Please check wether your input contains *Point* or *Line* features.

Figure B.8.: Screen after a negative input validation.

C. Proposed user study

While iteratively developing and improving a software product it is necessary prioritize certain methods over others (see section Usability studies). In the case of this project, e.g. neither a competitive analysis nor participatory design methods are planned. However, this appendix briefly outlines a possible usability study, which aims to reveal the most critical usability issues. It can be conducted remotely using the thinking-aloud-method, based on a task (or activity) embedded into a scenario.

C.1. Study setting

The user study is planned to be carried out remotely, with four to six participants. The number is based on the rule of thumb regarding the number of participants in usability testing: Nielsen (1994) estimates this as a sufficient number, where it is likely that all major shortcomings are identified (see section Usability studies). The participants correspond preferably to one of the identified user types (see section User requirements): cartographer and information designer. Moreover, they should be proficient in their English, as the tool will be most likely only available in English at this point.

The study will be conducted in the form of a video call, to reduce the time and effort on the participants side. To this end, the participants will be able to access the prototype through any modern browser. Furthermore, such study setting allows the user to work on the operating system and the hardware devices (screen, keyboard, etc.) they are used to. At the beginning the experimenter will explain the purpose of the study and will ask for consent: they need to agree that their screen and voice is recorded. Additionally, to the participants screen the webcam could be recorded as well, as the facial expressions of the participants can sometimes help to interpret the users behavior or reasoning. After that, the experimenter will outline the procedure of the study and above all the concept of thinking-aloud. In the following, the experimenter will present the scenario and user activity. After that the participant will start to perform the activity. During the activity the experimenter should be passive, but can remind the participant to think-aloud or support the participant in that by asking questions correspondingly. Also the experimenter can give important hints in order to continue the test in the case of a severe usability failure which otherwise would prevent the test's continuation. Furthermore, the experimenter will take notes to while the participants carries out the activity, of important comments or decisions by the participants. This notes mainly serve the purpose to ask the participants for further explanation, on e.g., what particularly was confusing, when the participant was expressing verbally confusion. Afterwards, the experimenter will analyze the screen and voice recording based on the notes taken during the test. Finally, the experimenter creates a

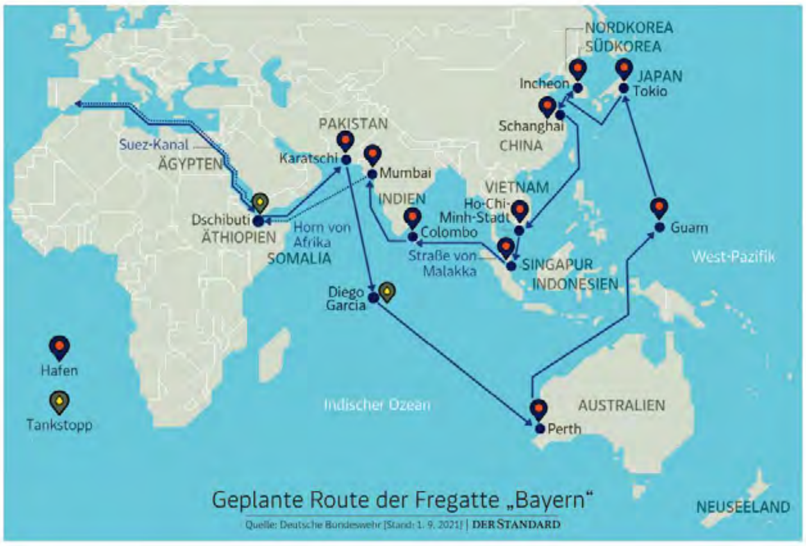


Figure C.1.: This schematized map, published in an Austrian Newspaper, serves as reference in the project brief (Vosatka, 2021).

brief document featuring a description of the most critical identified usability issues, an explanation why they occur alongside possible ways to fix these issues.

C.2. Scenario and user activity

The user scenario and and activities will be the same for both user types and is presented in the following section. The scenario aims to provide the context for the participant on the *why*: the reason why the user activity needs to be carried out to engage the participant with the UI.

Scenario

You are asked to create a schematized map for a newspaper showing routes of migration, as well as rescue and coast guards mission in the Mediterranean. The project brief includes Figure C.1 as style reference. The resulting map of the Mediterranean should match in style. As a first step your task is to create the background layer featuring landmasses and country borders.

The scenario is accompanied by an image of the existing network map. It shows a map, published in the Austrian Newspaper derStandard, and shows the route of a german mission in the Indian

Ocean and the Western Pacific. It is used to, first of all, explain the idea of schematization in cartography, and furthermore to demonstrate the intended schematization style (orientations and degree of simplification). Additionally, the participant is provided with pre-processed geographic data, from Natural Earth, containing all countries in the world, to avoid pitfalls described in section 5.2 Implementation limitations.

User activity

Create a schematized map for the Mediterranean, which matches the map included in the brief regarding its style (contains only angles of a multiple of 45° degrees) and the level of detail. Download the generated map as an SVG file, to finish the map in a vector graphic design software.

D. Requirement verification's test protocol

This test protocol poses the basis for the functional user requirements' verification, for the prototype's alpha release. These specifications include *FR 2-a* to *FR 11-a*, *FR 17-a*, and *FR 22-a*. These operational qualifications are tested employing different kind of validation methods: unit tests, black box testing (testing the behavior of the application) and white box testing (code inspection).

Test protocol, schematization tool alpha-release

Date October 09, 2021
Tester Jakob Listabarth
Hardware Specifications

→ MacBook Pro, Mid 2012

→ 2,5 GHz Dual-Core Intel Core i5 Processor

→ 8 GB 1600 MHz DDR3 Memory

→ Intel HD Graphics 4000 1536 MB Graphics

Software Specifications

→ Operating System: macOS Catalina (10.15.7)

→ Browser: Firefox 93.0 (64-bit)

FR 2-a

The system shall be able to parse geoJSON as input data.

→ *Steps to reproduce* Run unit test 2-a. This test fetches a geoJSON file, containing simplified boundaries of countries in Europe (source: Natural Earth), from the system and converts it into a dcel.

→ *Expected Result* Does not throw an error.

→ *Actual Result* Does not throw an error.

→ *Status* **Pass**

FR 3-a

If the input data is not a region i.e., it contains features of type other than polygon or multi-polygon – the program shall exit and the user shall be informed.

- *Steps to reproduce* Run unit test 3-a. This test fetches a geoJSON file containing geometry of type "LineString" from the system and converts it into a DCEL.
- *Expected Result* Does throw the error "invalid input".
- *Actual Result* Does not throw an error.
- *Status* **Pass**

FR 4-a

If the input data is not a valid geoJSON the program shall exit and the user shall be informed.

- *Steps to reproduce* Run unit test 4-a. This test fetches a series of invalid geoJSON file, e.g. containing geometry validating geoJSON's specification of the right-hand rule, from the system and converts it into a DCEL.
- *Expected Result* Does throw the error "invalid input".
- *Actual Result* Does not throw an error.
- *Status* **Fail**
- *Comment* An validation of the geoJSON is implemented but not working as intended due the library in use (geoJSON hint from *mapbox*. It is not fully compatible with the current *node.js* setup.

FR 5-a

If the input data is too detailed, i.e., if it exceeds a maximum number of edges or vertices, the program shall exit and the user shall be informed.

- *Steps to reproduce* Run unit test 5-a. This test fetches a geoJSON file, containing the states of Austria, using a number of edges exceeding the defined threshold.
- *Expected Result* Does throw the error "invalid input".
- *Actual Result* Does throw the error.
- *Status* **Pass**
- *Comment* The threshold needs to be refined according to performance of the prototype in practice.

FR 6-a

The system shall preserve potential attributes attached to the inputs features in the output.

- *Steps to reproduce* Run unit test 6-a. This test fetches a geoJSON file, containing the states of Austria. This data is converted to a DCEL, schematized, and converted back to a geoJSON. The geoJSON's feature attributes before and after the schematization are compared.
- *Expected Result* The number of feature attributes is the same before and after the schematization. The 4th feature's attribute is equivalent before and after the schematization.
- *Actual Result* A subdivision cannot be schematized as the simplification of subdivisions is not yet implemented. Therefore, the feature attributes cannot be verified.
- *Status* **Fail**

FR 7-a

The system shall preserve the number of features of the input in the output.

- *Steps to reproduce* Run unit test 7-a. This test fetches a geoJSON file, containing the states of Austria. This data is converted to a DCEL, schematized, and converted back to a geoJSON. The geoJSON's feature attributes before and after the schematization are compared.
- *Expected Result* The number of features is the same before and after the schematization.
- *Actual Result* A subdivision cannot be schematized as the simplification of subdivisions is not yet implemented. Therefore, the number of features cannot be verified.
- *Status* **Fail**

FR 8-a

The system shall be able to generate a DCEL from a geoJSON.

- *Steps to reproduce* Run unit test 8-a. This test fetches a geoJSON file, containing the states of Austria. This data is converted to a DCEL, schematized, and converted back to a geoJSON. The number of faces within the DCEL is examined.
- *Expected Result* The DCEL consists of one unbounded and ten bounded faces.
- *Actual Result* The DCEL consists of one unbounded and ten bounded faces.
- *Status* **Pass**

FR 9-a

The system shall be able to generate a geoJSON from a DCEL.

→ *Steps to reproduce* Run unit test 9-a. This test fetches a geoJSON file, containing the states of Austria. This data is converted to a DCEL, and converted back to a geoJSON. The resulting geoJSON is then validated using *mapbox*'s library *geoJSON hint*.

→ *Expected Result* The

→ *Actual Result*

→ *Status* **Pass**

FR 10-a

While the data is being processed, the user shall be informed that the application is processing.

→ *Steps to reproduce* Run the prototype in the browser. Take screenshots during the schematization is done.

→ *Expected Result* The system displays that it is processing data.

→ *Actual Result* The UI is only updated once, after the schematization of the (selected input) has finished. Figure D.1 shows the state of the UI while processing.

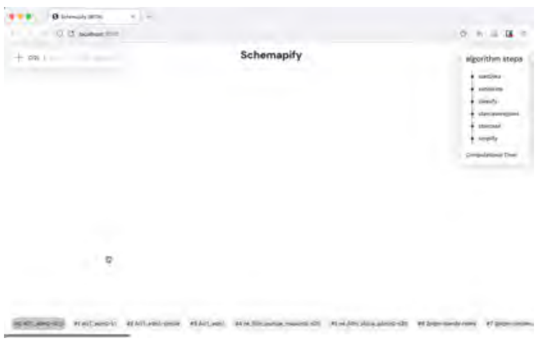


Figure D.1.: State of the UI while input data is processed.

→ *Status* **Fail**

FR 11-a

The user shall be able to specify a regular set of directions (without β-shift) of the schematization.

- *Steps to reproduce* Simple code inspection of the related UI components (*cOutput.ts, selectData.ts, algorithm-navigator.ts*).
- *Expected Result* A GUI's components enables the user to adjust the parameters fo the schematization.
- *Actual Result* Such a component is not yet implemented.
- *Status* **Fail**
- *Comment* Furthermore, the system's ability to schematize regions with different Cs needs to be verified.

FR 17-a

The system shall display the schematized region in the map-view after the schematization is finished.

- *Steps to reproduce* Run the prototype in the browser. Take screenshots once the schematization is done (2567ms).
- *Expected Result* The schematized region is displayed in the map-view after the schematization.
- *Actual Result* The schematized region is displayed in the map-view after the schematization, see Figure D.2.

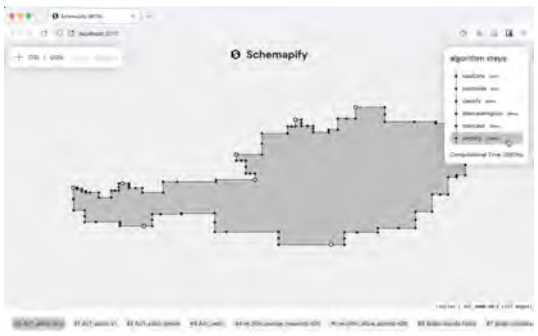


Figure D.2.: The schematized region is displayed in the GUI's map-view.

- *Status* **Pass**
- *Comment* Additionally, it can be visually compared to the input data (see Figure D.3), using the UI component showing the individual steps of the schematization algorithm.

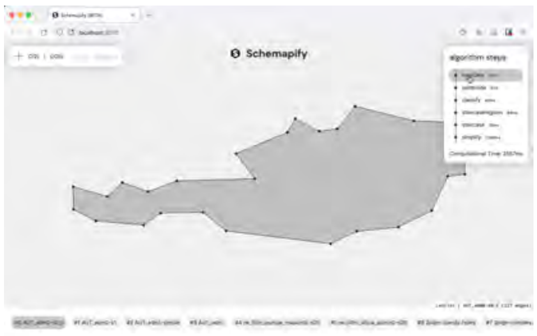


Figure D.3.: The input region is displayed in the GUI's map-view.

Note that the schematization is not working as intended; the simplification only works to a very limited extent.

FR 22-a

The user shall be able to track the progress of the schematization.

- *Steps to reproduce* Run prototype in the browser.
- *Expected Result* The user is informed about the processing steps during the schematization process.
- *Actual Result* The UI is only updated once, after the schematization of the (selected input) has finished. Figure D.1 shows the state of the UI while processing.
- *Status* **Fail**

E. Functional requirements

This appendix completes the list of functional requirements for the alpha release as discussed in section 4.3.3 Functional requirements. The following tables show the functional requirements beyond the alpha release.

Table E.1.: Functional Requirements for the beta release

FR	Requirement
1-b	The user shall be able to load geographic data from the local storage into the system.
2-b	The system shall be able to parse shape files as input data.
11-b	The user shall be able to specify a regular (including a β-shift) as well as an irregular set of directions, and further parameters like ϵ, and k as parameters for the schematization.
12-b	The system shall validate C to consist of at least two and not more than twelve orientations.
13-b	The system shall validate the values of C to be floats between 0 and 180.
14-b	The system shall validate the value of β to be a float – depending on C – between the sector's angle α, $\frac{\alpha}{-2}$ and $\frac{\alpha}{2}$.
15-b	The system shall validate the value of λ to a float between 0.05 and 1.
16-b	The system shall validate the value of k to be an integer greater than 3.
18-b	The system shall be able to parse the CRS of the input data and store it.
19-b	The system shall preserve projected coordinate system given by the input data for the output.
20-b	If the input data does not specify a projected geographic coordinate system the program shall exit and the user shall be informed that only data in a projected coordinate system will be treated by this product.
21-b	While the data is being imported, the user shall be able to cancel the process.
23-b	While the data is being schematized, the user shall be able to cancel the process.
24-b	The user shall be to download the schematized region, alternatively either as .shp file or as SVG file.
25-b	If the input data is too detailed the program shall inform the user that it needs to be simplified beforehand.
28-b	The user shall be able to cancel the export of the schematized region.
29-b	The user shall be able to inspect schematized features in the map-view.

Table E.2.: Functional Requirements for the first release candidate

FR	Requirement
26-rc	The user shall be able to "scrub" through intermediate steps of the schematization, supporting the understanding of how the parameters affect the resulting schematization.
27-rc	The user shall be able to define various values for k, for which the system stores intermediate results during the simplification process.
30-rc	The user shall be able to inspect DCEL entities in the map-view.
31-rc	The system should store basic statistics (number of DCEL entities and computation time) for each snapshot.

The following list outlines the relation between the product's features (FEs), described in section 4.3.4 Features a the functional requirements (FRs).

- *FE-1* Import
  - Generate DCEL relates to *FR 1-b, 2-a, 2-b, 8-a*
  - Store feature attributes and metadata relates to *FR 6-a, 7-a, 18-b, 19-b*
  - Cancel relates to *FR 21-b*
  - Browse system and specify input relates to *FR 1-b*
  - Validate input data relates to *3-a, 4-a, 5-a, 20-b, 25-b*
- *FE-2* Schematization
  - Set parameters relates to *FR 11-a, 11-b, 12-b, 13-b, 14-b, 15-b, 16-b*
  - Show progress relates to *FR 10-a, 22-a*
  - Cancel relates to *FR 23-b*
- *FE-3* Map-view
  - Pan and zoom relates to *FR 17-a*
  - Inspect features relates to *FR 9-a, 29-b*
  - Inspect DCEL entities relates to *FR 30-rc*
  - Switch map mode relates to *FR 29-b, 30-rc*
- *FE-4* Export
  - Generate geoJSON from DCEL relates to *FR 9-a*

- → Choose format relates to *FR 24-b*
- → Cancel relates to *FR 28-b*

→ *FE-5* Snapshots

- → Access snapshot relates to *FR 26-rc, 27-rc*
- → Take snapshot relates to *FR 26-rc, 27-rc*
- → Go to next and previous snapshot relates to *FR 26-rc*
- → Store statistics on DCEL features relates to *FR 2-a, 2-b, 6-a, 8-a*