A design pattern approach to map design with big geospatial data

Serena Coetzee[1] and Victoria Rautenbach[1,*]

[1]Department of Geography, Geoinformatics and Meteorology, University of Pretoria, Pretoria, South Africa

*Correspondence to: victoria.rautenbach@up.ac.za

Abstract: A design pattern approach for conceptualizing the cartographic design process is presented. A design pattern presents a solution to a problem by describing solutions for the commonalities in problems to be solved. The commonalities of the cartographic design process are identified and MapDesign is described, a design pattern for generating a set of maps from big geospatial data. The MapDesign pattern allows increased map throughput through automation and distributed processing, which frees up time for cartographers and geographic information scientists to make sense of big geospatial data and to select and refine map designs. The description of the MapDesign pattern can aid software developers in understanding how to build tools that allow automation and distributed processing in parts of the map design process for visualising big geospatial data. Conversely, the visual nature of design patterns can assist cartographers with understanding how software components interact to produce maps.

Keywords: cartography, map design, design pattern, big data, big geospatial data

1. Introduction

Today, ever increasing volumes of data are generated continuously by a large variety of sensors, including smartphones, social media users, global positioning systems (GPSs) and radio-frequency identification (RFID) tags. The resulting huge volumes of complex datasets have become known as ‘big data’. Four characteristics distinguish big data from other data: volume, variety, velocity, and veracity (4Vs) (Saha and Srivastava 2014; Tsou 2015; Ward and Barker 2013). In many cases, big data includes a direct or indirect reference to a location on the Earth and can then be referred to as ‘big geospatial data'.

Big (geospatial) data presents many challenges (Lee and Kang 2015; Robinson et al. 2017), amongst others, for the analysis and understanding of large volumes of data. Visualisations can assist with understanding such large volumes of data. In the case of large volumes of geospatial data, geovisualisations (maps) can assist with understanding the data, e.g. by showing spatial patterns in the data. A classic example is the map used by Dr John Snow to detect clusters of cholera cases in London in 1854 (Johnson 2006). Today, maps successfully show spatial patterns in much larger volumes of data (MacEachren 1995; Slocum et al. 2009). In some cases, the pattern can be detected by just mapping the data (MacEachren and Ganter 1990); in other cases, a cartographic technique, such as dot density maps or choropleth maps, make the spatial pattern visible (MacEachren 1982; Brewer et al. 1997).

A design pattern presents a solution to a problem by describing solutions for the commonalities in problems to be solved. In this article, we show how design patterns can be used to describe solutions to the commonalities and variations in map design, and discuss how such patterns can contribute to map design challenges with big geospatial data. Firstly, the cartographic design process is presented and extended into a design pattern – MapDesign – for designing and generating a set of maps. Secondly, software design patterns are used to conceptualise MapDesign. Opportunities for automation and parallel processing are pointed out and discussed.

The paper is structured as follows: Section 2 provides background on the cartographic design process and software design patterns. In Section 3, we present MapDesign, a design pattern for map design with big geospatial data. The structure of MapDesign is presented in terms of well-known software design patterns. An evaluation and discussion of the design pattern approach follows in section 4 and shows that benefits are achievable. Concluding remarks are presented in section 5.

2. Background

2.1. Cartographic design process

Cartography is generally defined as the art, science and technology of making and using maps (ICA 2003). As a discipline, cartography deals with the conception, production, dissemination and study of maps (ICA 2011). Before the 1960s, cartography was focused on the manufacturing of maps (Kraak and Ormeling 2003). Due to advancements in technologies, as well as the emergence of social media and crowdsourcing, the focus in cartography has shifted to conveying spatial information by means of a map (Kraak and Ormeling 2003; MacEachren 1995). Cartography encapsulates all the tools and processes involved in the production of all types of maps (Bolstad 2012; Slocum et al. 2009). Tyner (2010) extends the definition to include the design, compilation, construction, projection, reproduction, use, and distribution of maps.

The primary aim of cartography is to communicate geospatial information by means of a map. In order to achieve this aim, one needs to identify the intended audience, the information to be communicated, and the area of interest (Bolstad 2012; Kraak and Ormeling 2003; Slocum et al. 2009; Tyner 2010). These aspects impact on other cartographic design decisions, such as the cartographic technique, the scale and symbols. The cartographic technique can contribute to, and aid in, the interpretation of the map, or it can hinder and obstruct the interpretation thereof. It is thus an important consideration in cartographic design for big geospatial data (Rita et al. 2010). The cartographic design process in Figure 1 is adapted from various researchers

(MacEachren 1995; Slocum et al. 2009; Tyner 2010). The tree in Figure 2 illustrates dependencies of design choices in the cartographic design process. Depending on the selected cartographic technique, certain classifications are appropriate; and depending on the selected classification method, a different numbers of classes are possible.

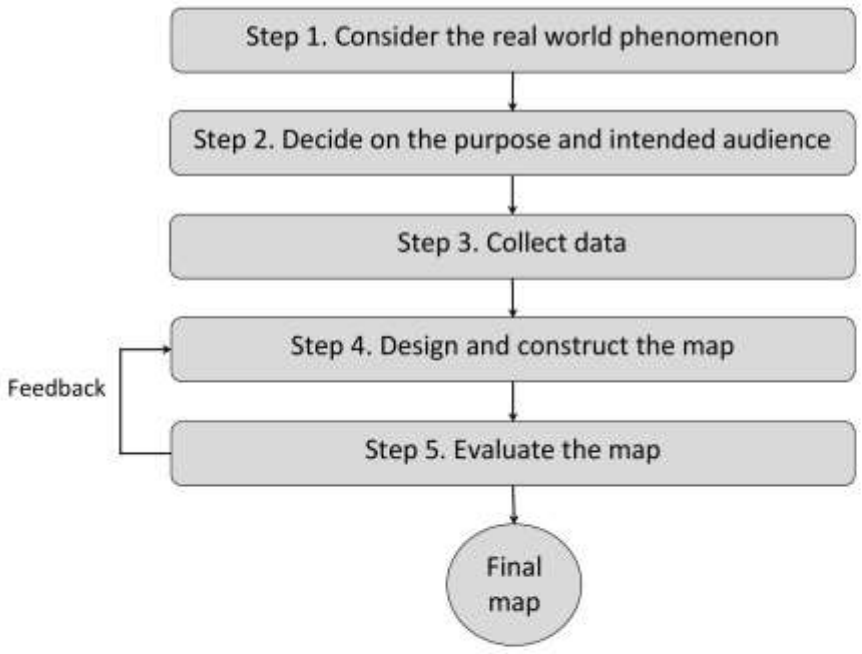
Step 1. Consider the real world phenomenon
Step 2. Decide on the purpose and intended audience
Step 3. Collect data
Step 4. Design and construct the map
Step 5. Evaluate the map
Feedback
Final map

Figure 1. The cartographic design process (adopted from MacEachren 1995; Slocum et al. 2009; Tyner 2010)

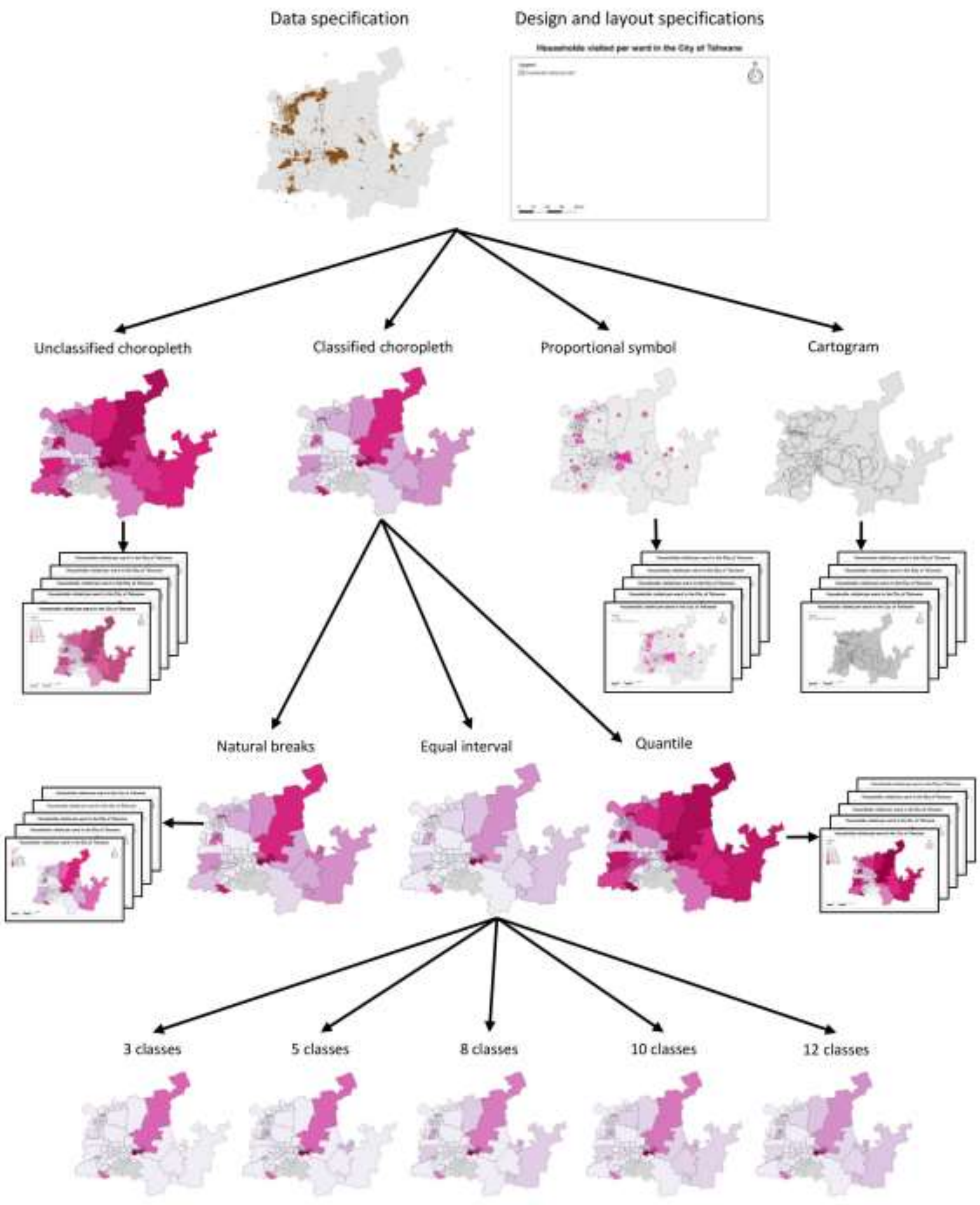

Figure 2. Design choice dependencies

The cartographic design process is generally represented in a linear fashion (even though some parts of the process may happen in parallel) and designed with an individual cartographer in mind who produces a single map. This can be contributed to the fact that map making used to be a manual and paper-based process, where any changes to a hardcopy map implied starting the process anew. With the rapid increase in the volumes of geospatial data and parallel processing opportunities, the cartographic design process may need to be adapted. Also, this representation of the cartographic design process does not identify design choices and dependencies between them; this information is required for software developers who want to automate (parts of) the process.

2.2. Software design patterns

In software engineering, a design pattern describes a reusable solution for a problem that occurs commonly (Shalloway and Trott 2004). Shalloway and Trott (2004) define a pattern as 'a solution to a problem in a context'. Design patterns are developed by examining solutions to common problems in existing software, and in best practices developed by programmers. A design pattern specifies a solution to a common problem at a next level of abstraction compared to object-oriented design. Design patterns have the same advantages as object-orientated programming, including abstraction (to use the object, a programmer has to only know its interface), reuse (once created, objects can be reused by programmers in other programs), and encapsulation (hiding implementation details from users of the objects).

Software design patterns are grouped into the following types: creational, behavioural and structural patterns. Creational patterns facilitate the work of creating, initializing and configuring objects and classes. They are useful when multiple instances of an object need to be rendered, stored or duplicated. Behavioural patterns facilitate the work of algorithmic calculations and communication between classes. Lastly, structural patterns enable the modification of the structure of the code, such as the structural associations of classes, class associations and hierarchies of class structures. In the subsections, we briefly introduce those software design patterns included later in this article. The descriptions of the design patterns are based on Gamma et al. (1994) and Shalloway and Trott (2004).

The Object Management Group (OMG) defines Unified Modeling Language (UML) as a graphical language for visualising, specifying, constructing, and documenting the artifacts of a software-intensive system (OMG 2015). UML, specifically class diagrams, is typically used to visually describe the structure of design patterns. UML class diagrams describe the structure of a software system, i.e. classes with their attributes and operations, and the relationships between classes. UML sequence diagrams (also called interaction diagrams) illustrate how objects (instances of classes) interact with each other.

2.2.1 Factory method

Table 1 provides and overview of the Factory method and the structure is depicted in Figure 3.

Table 1. Overview: Factory method design pattern (based on Gamma et al. 1994; Shalloway and Trott 2004)

Item	**Description**
Intent	Define an interface for creating an object, but let subclasses decide which class to instantiate.
Problem	Use the Factory method when it cannot be anticipated which derived class needs to be instantiated.
Solution	The derived class makes the decision on which class to instantiate and how to instantiate it.
Participants	Product, ConcreteProduct, Creator, ConcreteCreator
Collaborations	Creator relies on its subclasses to define the factory method so that it returns an instance of the appropriate ConcreteProduct.
Consequences	Clients have to subclass the Creator class to make a particular ConcreteProduct.

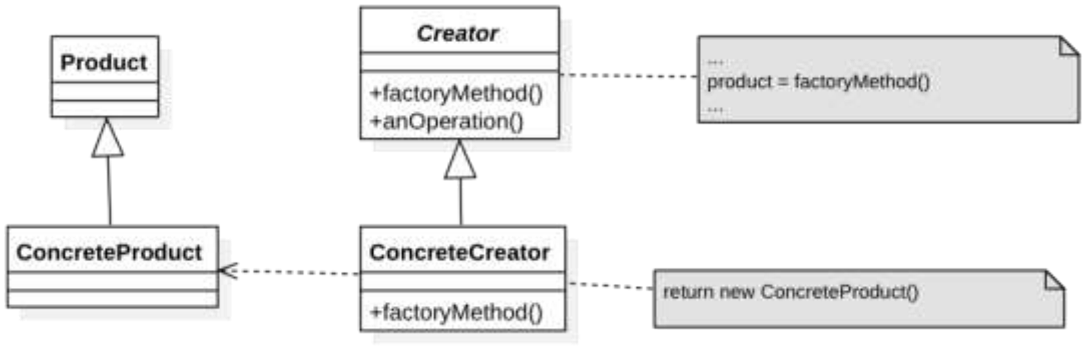

Figure 3. The structure of the Factory method design pattern (redrawn from Gamma et al. 1994)

2.2.2 Strategy

Table 2 provides and overview of the Strategy design pattern and the structure is depicted in Figure 4.

Table 2. Overview: Strategy design pattern (based on Gamma et al. 1994; Shalloway and Trott 2004)

Item	**Description**
Intent	Define a family of algorithms, encapsulate each one, and make them interchangeable.
Problem	Use this pattern if the selection of an algorithm to be applied depends on the client making the request or the data being acted on.
Solution	Separate the selection of the algorithm from the implementation of the algorithm so that the selection can be made based on the context.
Participants	Strategy, ConcreteStrategy, Context
Collaborations	Strategy and Context interact to implement the chosen algorithm. A context forwards requests from its clients to its strategy.
Consequences	The pattern defines a family of algorithms; conditional logic is eliminated; and each algorithm must be invoked in the same way.

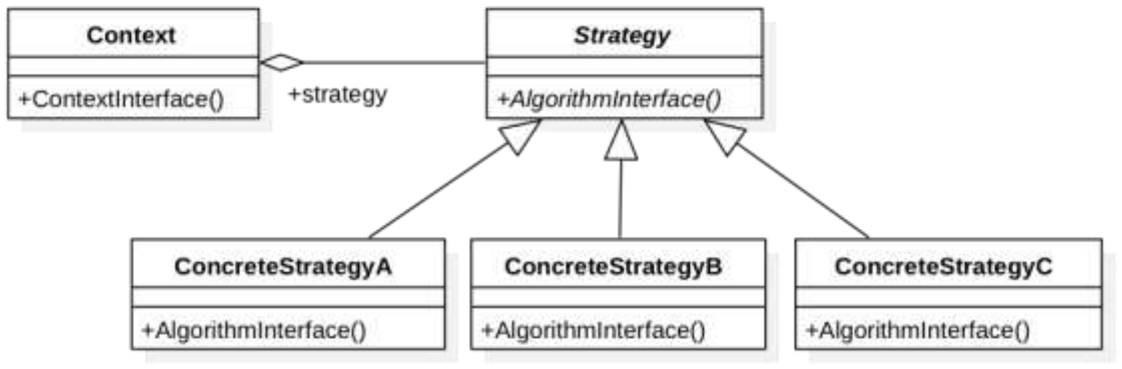

Figure 4. The structure of the Strategy design pattern (redrawn from Gamma et al. 1994)

2.2.3 Builder

Table 3 provides and overview of the Builder design pattern and the structure is depicted in Figure 5.

Table 3. Overview: Builder design pattern (based on Gamma et al. 1994)

Item	**Description**
Intent	Separate the construction of a complex object from its representation so that the same construction process can create different representations.
Problem	Use this pattern if the algorithm for creating a complex object should be independent of the parts that make up the object. The construction process must allow different representations for the object that is constructed.
Solution	Defines an abstract operation for the construction of each component that a director may want to create. By default, the operation does nothing by default. Each ConcreteBuilder overrides those operations it's interested in creating.
Participants	Builder, ConcreteBuilder, Director, Product
Collaborations	The client creates the Director object and configures it with the desired Builder object. The Director notifies the builder whenever a part of the product should be built. The Builder handles requests from the director and adds parts to the product. The client retrieves the product from the builder.
Consequences	With this pattern, one can vary a product's internal representation; isolate the code for construction and representation; and one has finer control over the construction process.

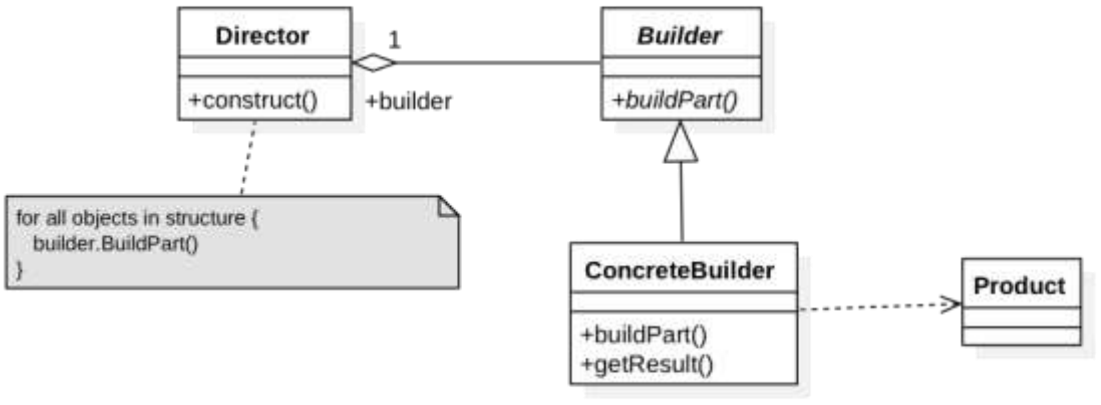

Figure 5. The structure of the Builder design pattern (redrawn from Gamma et al. 1994)

2.3 Related work

Manually designing a map can be very time consuming, even if software is used, and various parts of the map design process have been automated, for example, the generalisation of features and symbol placement (Foerster et al. 2010; Harrie and Revell 2007; Kumar 2000; Li 2015). However, there has also been the need to find new approaches to design and generate maps from big geospatial data.

Traditional cartographic tools and processes are designed for small well-defined datasets that produce a single map. Cartographers and geographic information scientists have to find new ways of designing and generating maps from big geospatial data. Researchers have approached this challenge in various ways (Krimbacher 2014; Moncrieff et al. 2016; Rautenbach et al. 2013). For example, Rautenbach et al. (2013) present ThematicWS, a web service that produces choropleth and proportional symbol maps by orchestrating distributed web services: a central controlling process executes web services in a specified order and manner to produce the maps; it also handles any interactions between the web services. Similarly, Krimbacher (2014) designed and developed a service-orientated architecture for thematic map production. To deal with rapidly increasing volumes of data, Moncrieff et al. (2016) developed a platform for the exploration and analysis of large dynamic geospatial datasets. Mapping functionality was encapsulated in standard web services. These approaches are typically based on the cartographic design process and exhibit commonality (e.g. the order of map design tasks) and variation (e.g. different cartographic techniques).

A design pattern presents a solution to a problem by describing solutions for the commonalities in problems to be solved. Software design patterns arose, amongst others, from architects who argued that judging the beauty of a building is not only a matter of taste; they argued that the quality of the design can also be objectively assessed. This idea was transferred to software design in the 1990s (Gamma et al. 1994) and many software design patterns have been described since then (Millet and Tune 2015). It is important to note that even if the pattern is used repeatedly to solve a common problem, the outcome, the building or software artifact, may be totally different. Visualisation design patterns have been described (Chen 2004, Heer and Agrawala 2006, Stolte et al. 2002), some of them for maps, but none of them describe the map design process.

Among the benefits attributed to studying design patterns are that design patterns provide a higher perspective on analysis and design, and that design patterns improve communication and individual learning. On the practical side, design patterns improve the quality of software by simplifying the code and making it easier to construct and maintain the code.

Various design pattern approaches for information visualisation and geographic information science have been proposed. For example, Stolte et al. (2002) described four patterns that capture the zoom structures in their system, one of them for thematic maps. Carral et al. (2013) propose a scale ontology design pattern which can be used to document and publish knowledge about map scaling applications on the web.

According to Heer and Agrawala (2006), despite a diversity of software architectures supporting information visualisation, it is often difficult to identify, evaluate, and re-apply the design solutions implemented within such frameworks. To overcome this, they captured successful solutions as design patterns. These abstract descriptions of interacting software components can be customised by programmers to solve visualisation design problems within a particular context.

Chen (2004) described visualisation design patterns that summarise common practices and techniques applied in the process of dynamic, analytical data visualisation, including maps. They distinguish between visualisation design patterns for users of visualisation systems to model, design and perform visualisation tasks, and software design patterns for developers to design and implement a visualisation system. They point out that visualisation design patterns could impact software development and become special software design patterns used by developers of visualisation systems.

Gordillo et al. (1999) describe design patterns for the most common design problems that developers of geographic information system (GIS) applications must face. Camara et al. (2001) implemented a number of well-known software design patterns in the open source GIS software library, TerraLib. They argue that design patterns are well suited to capture the complexity of the components of a GIS. Others have also followed this approach, e.g. GeoTools[1].

While others have focussed on GIS software, visualisation tasks and architectures that support information visualisation, this article expands the cartographic design process into a design pattern for map design of a set of maps. The structure of this pattern is described in terms of a number of well-known software design patterns.

---

[1] http://docs.geotools.org/latest/userguide/tutorial/factory.html

3. A design pattern approach for map design

In this section, we follow the method proposed by Shalloway and Trott (2004) for describing a design pattern. In 3.1, the concepts (commonalities) and concrete implementations (variations) in the problem domain (map design) are identified. Then, after the concept for the required functionality (map design) has been identified, the interfaces for the abstractions that encapsulate this are specified in terms of a number of well-known software design patterns.

The first commonality in the cartographic design process is the order of the map design tasks (or steps). In Figure 6, we fill in details for each step in the process presented earlier in Figure 1. These sub-steps present commonalities in the map design process, such as standardising the data and selecting colour schemas for styling. The details also represent the variation, such as different ways of standardisation or classification of the data.

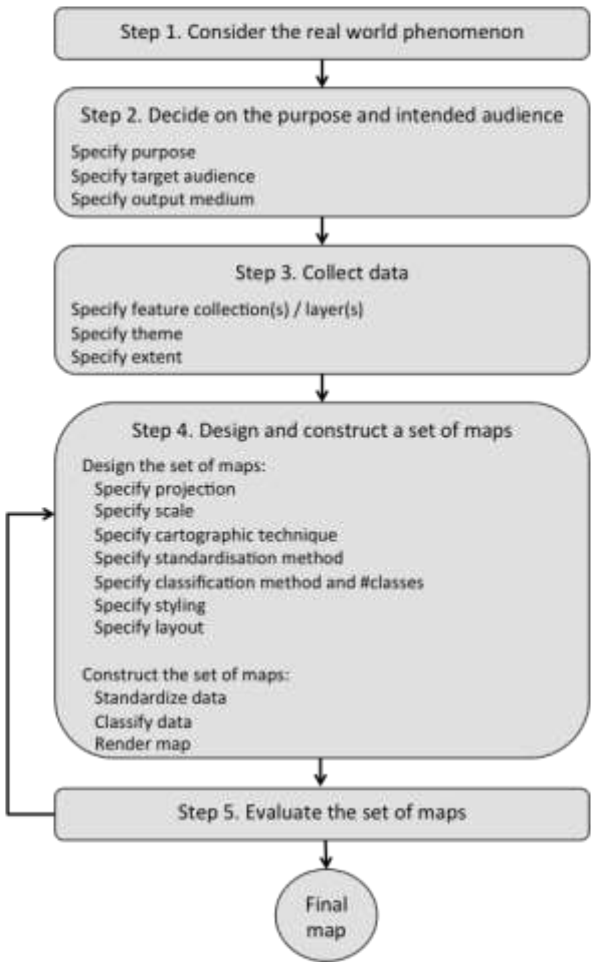
Step 1. Consider the real world phenomenon
Step 2. Decide on the purpose and intended audience
Specify purpose
Specify target audience
Specify output medium
Step 3. Collect data
Specify feature collection(s) / layer(s)
Specify theme
Specify extent
Step 4. Design and construct a set of maps
Design the set of maps:
Specify projection
Specify scale
Specify cartographic technique
Specify standardisation method
Specify classification method and #classes
Specify styling
Specify layout
Construct the set of maps:
Standardize data
Classify data
Render map
Step 5. Evaluate the set of maps
Final map

Figure 6. Commonalities and variations in the cartographic design process

Next, the commonalities and variations for each step are identified with reference to Figure 6. Opportunities for automation and parallelisation are identified.

1) **Consider the real world phenomenon**

The real world phenomenon to be mapped is a commonality with an endless list of variations; theoretically, any real world phenomenon for which big geospatial data exists can be mapped. This step cannot be automated.

2) **Decide on the purpose and intended audience**

The cartographer establishes the intent or message of the map that needs to be produced. This is the most important step of the process, as all design and implementation choices that follow depend on the outcome of this step. The purpose, target audience and output medium are commonalities. Variations for the purpose and target audience are open-ended and cannot be automated. Many variations in the output media are possible, but any specific implementation is likely to implement a finite number of variations.

3) Collect data

The cartographer reviews data and identifies one or more datasets suitable for the purpose. The theme (attribute/s) in the dataset(s) and the extent (e.g. geographic and/or temporal) of interest for the map are specified. The dataset identifier, theme and extent are commonalities. Variations of these depend on the outcome of earlier steps in the process, e.g. appropriate extents for the data depends on the purpose of the map.

4) Design and construct a set of maps

During this step, design choices are specified and then the set of maps is generated. In some cases, there are dependencies between design choices, e.g. the cartographic technique determines which classification methods are applicable (if any). Different implementations (variations) of projection, standardisation and classification methods are possible. For the other design choices – scale, styling and layout – a wide range of options are possible. A cartographer typically plays around with different combinations of design choices until satisfied that a map meets all the requirements. A map design tool could propose a first set of combinations of design choices, which the cartographer adjusts and refines. To facilitate automation and parallelisation, it must be possible to generate a map from a combination of pre-specified design choices. This will also make it possible to generate a set of maps in parallel. Constructing (rendering) a map based on a combination of design choices is another commonality.

5) **Evaluate the set of maps**

The set of maps is evaluated by the cartographer. The final map is selected from the subset of map that meets the purpose, i.e. successfully communicates the intended message. The evaluation is subjective, relying on the cartographer's experience and preferences. This step cannot be automated. If necessary, the previous step is repeated with revised combinations of design choices to produce a revised set of maps.

Final map

The cartographer selects the map, which, according to him/her, fulfils the purpose by successfully communicating the message to the intended target audience. The process ends.

Table 4 describes MapDesign, the design pattern for the above process.

Table 4. The MapDesign design pattern

Item	**Description**
Pattern name	MapDesign
Intent	Generate a set of maps from big geospatial data
Problem	Maps are generally created by repeating the same steps iteratively until a desired design solution is obtained. This process can be time consuming and user interaction intensive. In the case of big geospatial data, executing parts of the process in parallel increases throughput.
Solution	Allows the cartographer to specify a dataset(s), a theme (attributes) and extent from which a set of maps is generated simultaneously, each with a different combination of design choices. Maps in this set are evaluated, and if necessary, design choices are adjusted and a new set of maps is generated until the cartographer selects the final map.
Participants	MapCreator, ConcreteMapCreator, MapDesign, ConcreteMapDesign, ClassificationMethod, ConcreteClassificationMethod, StandardisationMethod, ConcreteStandardisationMethod, MapBuilder, ConcreteMapBuilder
Collaborations	The MapCreator relies on its subclasses to define the createMap method so that it returns an instance of the appropriate ConcreteMapDesign. StandardisationMethod and ClassificationMethod interact with MapDesign to execute the chosen algorithm MapDesign configures the map. It notifies MapBuilder whenever an element of the map should be built. MapBuilder handles these requests and adds different elements to the map. MapDesign retrieves the rendered map from the ConcreteMapBuilder.
Consequences	Selected steps in the map design process can be automated and generating a set of maps in parallel reduces the need for user intervention and increases throughput. This allows the cartographer to focus on selecting and refining design choices. The implementation can be distributed in a cloud environment so that different parts of the process, e.g. standardisation, classification and rendering, can be executed in parallel.

To demonstrate the intent of the design pattern, Figure 7 shows the set of maps for a specific combination of design choices.

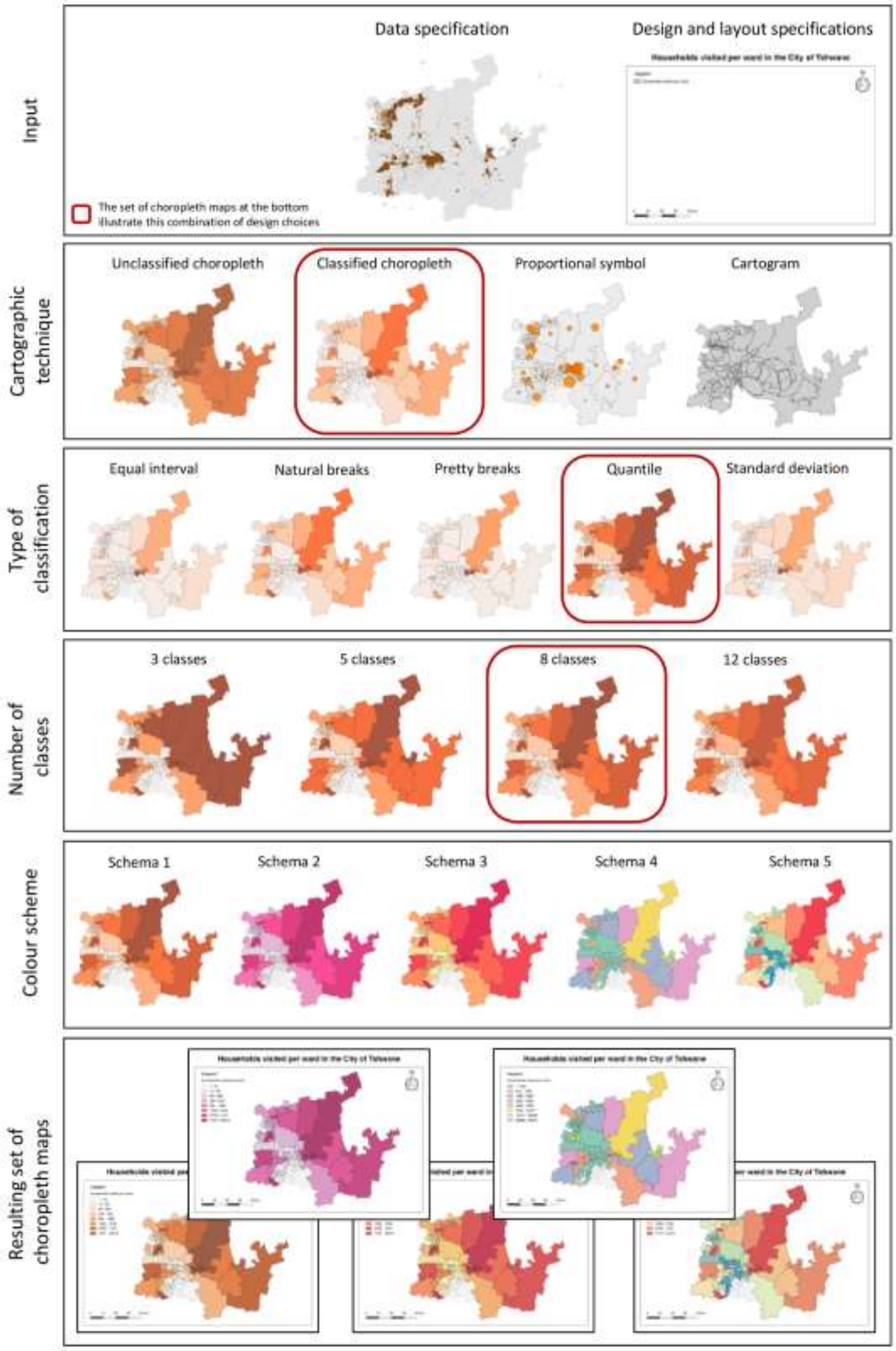

Figure 7. Sets of maps generated for a specific combination of design choices.

The diagram in Figure 8 describes the generic structure of the MapDesign pattern in terms of well-known software design patterns.

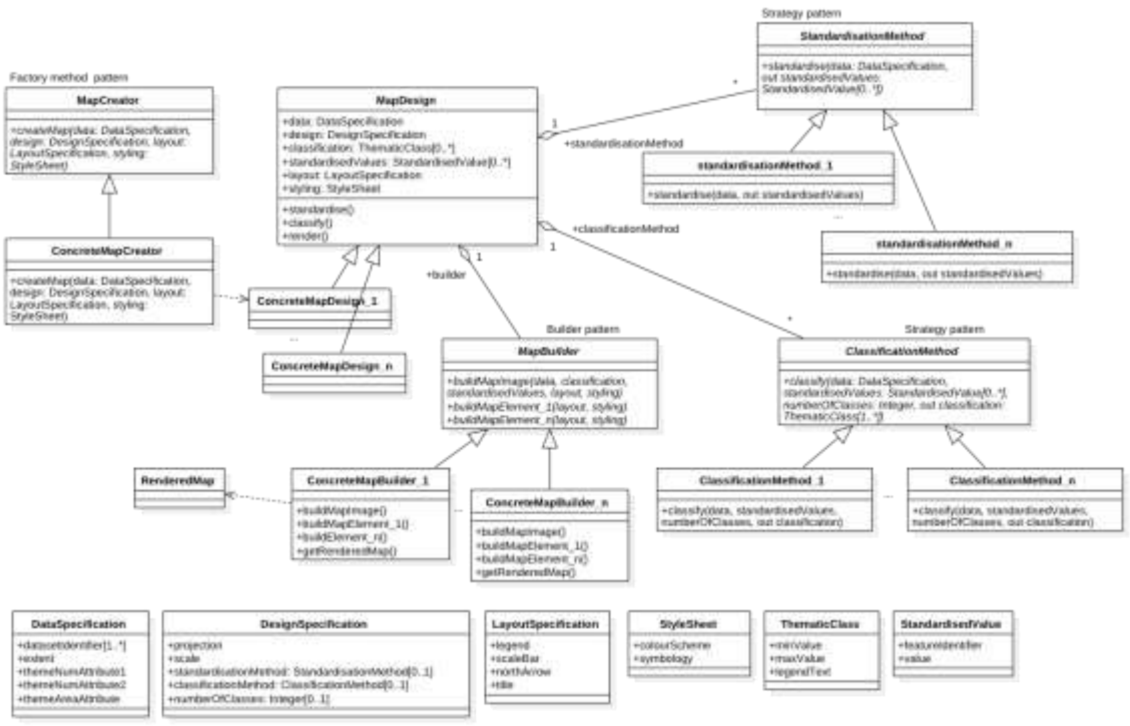
Factory method pattern
MapCreator
+createMap(data: DataSpecification, design: DesignSpecification, layout: LayoutSpecification, styling: StyleSheet)
ConcreteMapCreator
+createMap(data: DataSpecification, design: DesignSpecification, layout: LayoutSpecification, styling: StyleSheet)
MapDesign
+data: DataSpecification
+design: DesignSpecification
+classification: ThematicClass[0..*]
+standardisedValues: StandardisedValue[0..*]
+layout: LayoutSpecification
+styling: StyleSheet
+standardise()
+classify()
+render()
ConcreteMapDesign_1
...
ConcreteMapDesign_n
Strategy pattern
StandardisationMethod
+standardise(data: DataSpecification, out standardisedValues: StandardisedValue[0..*])
1
*
+standardisationMethod
standardisationMethod_1
+standardise(data, out standardisedValues)
...
standardisationMethod_n
+standardise(data, out standardisedValues)
+classificationMethod
1
*
+builder
1
Builder pattern
MapBuilder
+buildMapImage(data, classification, standardisedValues, layout, styling)
+buildMapElement_1(layout, styling)
+buildMapElement_n(layout, styling)
Strategy pattern
ClassificationMethod
+classify(data: DataSpecification, standardisedValues: StandardisedValue[0..*], numberOfClasses: integer, out classification: ThematicClass[1..*])
RenderedMap
ConcreteMapBuilder_1
+buildMapImage()
+buildMapElement_1()
+buildElement_n()
+getRenderedMap()
...
ConcreteMapBuilder_n
+buildMapImage()
+buildMapElement_1()
+buildMapElement_n()
+getRenderedMap()
ClassificationMethod_1
+classify(data, standardisedValues, numberOfClasses, out classification)
...
ClassificationMethod_n
+classify(data, standardisedValues, numberOfClasses, out classification)
DataSpecification
+datasetIdentifier[1..*]
+extent
+themeNumAttribute1
+themeNumAttribute2
+themeAreaAttribute
DesignSpecification
+projection
+scale
+standardisationMethod: StandardisationMethod[0..1]
+classificationMethod: ClassificationMethod[0..1]
+numberOfClasses: Integer[0..1]
LayoutSpecification
+legend
+scaleBar
+northArrow
+title
StyleSheet
+colourScheme
+symbology
ThematicClass
+minValue
+maxValue
+legendText
StandardisedValue
+featureIdentifier
+value

Figure 8. The generic structure of the MapDesign design pattern

```
class MapCreator {

  DataSpecification data;
  DesignSpecification design;
  LayoutSpecification layout;
  Stylesheet styling;

  // Fill in data specification
  data.datasetIdentifier =
  data.extent =
  // etc.

  // Fill in design specification
  design.projection =
  design.scale =
  // etc.

  // Fill in layout specification
  layout.border =
  layout.legend =
  // etc.

  // Fill in styling specification
  styling.colourScheme =
  styling.symbology =
  // etc.
  MapDesign map = createMap(data, design, layout, styling);

  StandardisedValue standardisedValues = map.standardise(data);
  ThematicClass classification = map.classify(data,
                                  standardisedValues,
                                  design.numberOfClasses);

  map.render(data, classification, standardisedValues,
             layout, styling);

  return;
}

class ChoroplethMapCreator {

  public Map createMap(DataSpecification data,
                 DesignSpecification design,
                 LayoutSpecification layout,
                 Stylesheet styling ){

    return new ChoroplethMap(data, design, layout, styling);
  }
}
```

Figure 9. Pseudo code for the MapCreator and one kind of ConcreteMapCreator

We implemented the MapDesign design pattern in Java to evaluate and verify it. The implementation was done with NetBeans 8.2[2] on a personal computer running MacOS Sierra. All the classes and methods of the MapDesign pattern were implemented for choropleth and proportional symbol maps with three different standardisation (area-based, ratio and density) and classification (natural breaks, quantiles and equal intervals) methods. The pseudo code to create a choropleth map is shown in Figure 9. The sequence diagram in Figure 10 shows how the classes in the MapDesign pattern interact with each other to produce the map.

2 https://netbeans.org

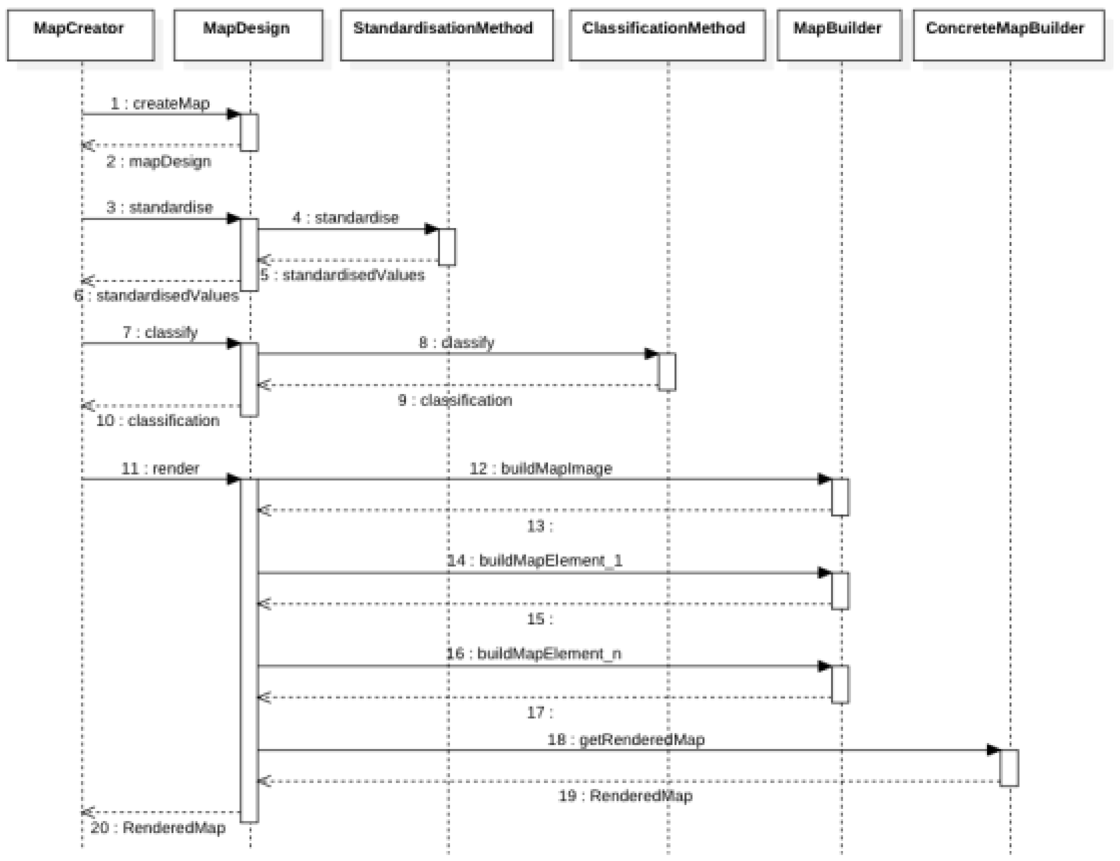

Figure 10. Sequence diagram for the MapDesign pattern

4. Discussion

In this article, we presented MapDesign, a design pattern for generating a set of maps, each with a different combination of design choices. MapDesign is based on commonalities (e.g. the order of tasks or steps in the design process and a set of design choices) and variations (e.g. specific design choices, such as a specific cartographic technique). We pointed out those parts of the design pattern that can be automated. The structure of MapDesign was illustrated in UML.

The design pattern approach has a number of benefits: 1) it makes it possible to specify and produce a set of maps from the same input data; 2) a distributed implementation with parallel processing, e.g. in a cloud environment, is possible and would reduce throughput time; 3) the design pattern is extendible, i.e. additional design choices, such as cartographic techniques and classification methods, can be added; 4) using software design patterns that are well known to computer scientists reduces programmers' cognitive load when developing or reading code that includes cartographic concepts novel to them; and finally, 5) producing a set of maps from the same input data frees up time for the cartographer to evaluate, refine and select a map design. The results contribute to understanding how tools for the automated design and generation of maps from big geospatial data can be developed to assist cartographers and geographic information scientists with map design.

MapDesign can be implemented in a distributed fashion, taking advantage of parallel processing and/or (elastic) processing resources available in a cloud environment. These are important considerations when generating maps from big geospatial data and/or when there is a need to create large volumes of maps. For example, MapDesign could be implemented as web services that access distributed big geospatial data sources on the web, similar to the architecture proposed by Moncrieff (2016) for visual analytics; or by orchestrating a number of web services, an architecture proposed by Rautenbach et al. (2013).

MapDesign differs from the 'traditional' cartographic design process or cartographic communication model (MacEachren 1995; Slocum et al. 2009; Tyner 2010) because it generates a set of maps (as opposed to a single map). Instead of iteratively fiddling around with a single map design, a set of maps is generated simultaneously and the resulting map designs are evaluated and compared by a (human) cartographer. If necessary, a revised set of maps is generated for further evaluation until the final map is selected. In this way, time-consuming trial-and-error design refinements are done in parallel, allowing the cartographer can evaluate more combinations of design choices in a shorter period of time. MapDesign is not aimed at removing the human from the map design process, but rather to assist the human with map design.

The cartographic technique can contribute to, and aid in, the interpretation of a map, or it can hinder and obstruct the interpretation thereof. The choice of cartographic technique is thus an important consideration in cartographic design (MacEachren and Ganter 1990; Brewer et al. 1997; MacEachren 1995; Slocum et al. 2009). MapDesign assists the cartographer with choosing an appropriate cartographic technique because it reduces the time required to explore and compare different cartographic techniques. In the same way, MapDesign is conducive to education because a student or learner can readily compare the effects of different combinations of design choices (reference about challenges of teaching which cartographic technique is best).

Krimbacher (2014) suggests that a map can be automatically generated from a comprehensive map description, i.e. each map description is a unique combination of map design choices. Such map descriptions could be stored for future use, e.g. to generate the same map on a new version of the data. The MapDesign design pattern allows this use case.

Communication is a significant challenge in software design (Parnas 2009; Capretz and Ahmed 2010; Ahmed et al. 2012). One has to bridge the semantic gap between users' understanding of the business or real world, and the software developers' design of a system that will represent the real world. When designing software for mapping, there is a semantic gap between cartographers (who typically do not understand code in a programming language) and software developers (who are not familiar with map design). The visual nature of design patterns presented in UML facilitates communication between stakeholders from different disciplines (Shalloway and Trott 2005, Pilone 2005, Petre 2013).

The description of MapDesign in this article can aid software developers in understanding how to build mapping tools for visualising big geospatial data into a set of maps. By using software design patterns that are well known to software developers, such as the Factory method, Strategy and Builder, the software developers' cognitive load is reduced when developing or reading code that includes cartographic concepts novel to them (Kolfschoten et al. 2010). Design patterns simplify the code because abstract classes represent commonalities, e.g. the commonality standardising data is represented in the abstract class, StandardisationMethod. Similarly, the description of MapDesign can assist cartographers with understanding the software components involved, how they are related to each other, and how they interact with each other.

From a software engineering perspective, the MapDesign design pattern has all the advantages that come with object-oriented design, such as:

- abstraction: to use a specific object (e.g. MapDesign or StandardisationMethod), a programmer has to only know its interface;
- reuse: once objects, such as the different ConcreteMapBuilders, are created, they can be reused by programmers in other programs;
- encapsulation: implementation details are hidden from users of the objects (e.g. the details of the algorithms that implement different standardisation methods); and
- extendibility: objects, e.g. ClassificationMethod, can be extended by adding specialisations that inherit the attributes and functionality of the base class.

The MapDesign design pattern can contribute to overcoming the challenge of analysing and understanding large volumes of data. For example, studying different visualisations (map) of the same big geospatial data can assist with understanding the data; also, generating a set of maps in parallel can speed up the analysis of big geospatial data.

5. Conclusions

In this paper, we presented, MapDesign, a design pattern for incorporating automation and parallelisation into the cartographic design process. This allows increased map throughput, which is essential in the age of big geospatial data. The pattern is based on commonalities (e.g. a set of design choices) and variations (e.g. specific design choices, such as the cartographic technique). MapDesign is conceptualized in terms of a number of well-known software design patterns, namely Factory method, Strategy and Builder.

‘Converting’ the cartographic design process into a design pattern that can be understood and used by software developers, facilitates communication between cartographers and software developers. The mapping design pattern helps software developers to understand what map making entails, so that they are in a position to develop tools for automating and parallelising map making from big geospatial data.

As a next step, we want to focus on implementing serialization for a comprehensive set of design choices. Such a serialization could contribute towards map design interoperability: given the same set of design choices, different applications should be able to render identical maps. Another interesting direction would be intelligent map design, i.e. software that proposes an appropriate set of design choices, based on the characteristics of the input data.

The MapDesign pattern can guide software developers in building the tools that are required to automate and parallelise map design with big geospatial data, thus allowing cartographers and others to make sense of big geospatial data.

References

Ahmed, F., Capretz, L. F., and Campbell, P. 2012. Evaluating the Demand for Soft Skills in Software Development. *IT Professional*, 14(1), pp.44-49.

Bolstad, P. 2012. *GIS Fundamentals: A First text on Geographic Information Systems*. 4th ed. St. White Bear, United States of America: Eider Press.

Brewer, C. A., MacEachren, A. M., Pickle, L. W., and Herrmann, D. 1997. Mapping mortality: Evaluating color schemes for choropleth maps. *Annals of the Association of American Geographers*, 87(3), pp.411-438.

Camara, G., Souza, R. C. M., Pedrosa, B. M., Vinhas, L., Monteiro, A. M. V., Paiva, J. A. C., Carvalho, M. T., Raoult, B. 2001. Design patterns in GIS development: the TerraLib experience. *III Simposio Brasileiro de GeoInformatica*, Rio de Janeiro, RJ.

Capretz, L. F., and Ahmed, F. 2010. Making sense of software development and personality types. *IT Professional*, 12(1), pp.6-13.

Carral, D., Scheider, S., Janowicz, K., Vardeman, C., Krisnadhi, A. A., and Hitzler, P. 2013. An Ontology Design Pattern for Cartographic Map Scaling. *Lecture Notes in Computer Science*, 7882, pp.76-93.

Chen, H. 2004. Towards Design Patterns for Dynamic Analytical Data Visualization. *Proceedings of SPIE Visualization and Data Analysis*, 18 January 2004, San Jose, United States of America.

Foerster, T., Stoter, J., and Kraak, M-J. 2010. Challenges for Automated Generalisation at European Mapping Agencies: A Qualitative and Quantitative Analysis. *The Cartographic Journal*, 47(1), pp.41-54.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. Indianapolis, United States of America: Addison-Wesley Professional.

Gordillo, S., Balaguer, F., Mostaccio, C., Das Neves, F. 1999. Developing GIS Applications with Objects: A Design Patterns Approach. *GeoInformatica*, 3(1), pp.7-32.

Harrie, L., and Revell, P. 2007. Automation of Vegetation Symbol Placement on Ordnance Survey 1:50 000 Scale Maps. *The Cartographic Journal*, 44(3), pp.258-267.

Heer, J., and Agrawala, M. 2006. Software Design Patterns for Information Visualization. *IEEE Transactions on visualization and computer graphics,* 12(5), pp.853–860.

International Cartographic Association (ICA). 2003. *A Strategic Plan for the International Cartographic Association 2003-2011*. Available online at http://icaci.org/files/documents/reference_docs/ICA_Strategic_Plan_2003-2011.pdf, accessed 4 April 2017.

International Cartographic Association (ICA). 2011. *Strategic Plan for the International Cartographic Association 2011-2019*. available online at http://icaci.org/files/documents/reference_docs/ICA_Strategic_Plan_2011-2019.pdf, accessed 4 April 2017.

Johnson, S. 2006. *The Ghost Map: The Story of London's Most Terrifying Epidemic - and How It Changed Science, Cities, and the Modern World*. London, England: Penguin Books.

Kolfschoten, G., Lukosch, S., Verbraeck, A., Valentin, E., and de Vreede, G-J. 2010. Cognitive learning efficiency through the use of design patterns in teaching. *Computers and Education*, 54, pp.652-660.

Kraak, M-J., and Ormeling, F. 2003. *Cartography: visualization of geospatial data*. Essex, England: Pearson Education.

Krimbacher, A. 2014. *Service-oriented Architecture for Thematic Cartography on the Web*. MSc thesis, Eidgenossische Technische Hochschule Zurich, Switzerland.

Kumar, N. 2000. Automation and Democratization of Cartography: An Example of a Mapping System at CEM, University of Durham. *The Cartographic Journal*, 37(1), pp.65-77.

Lee, J-G., and Kang, M. 2015. Geospatial Big Data: Challenges and Opportunities. *Big Data Research*, 2(2), pp.74-81.

Li, Z. 2015. General Principles for Automated Generation of Schematic Network Maps. *The Cartographic Journal*, 52(4), pp.356-360.

MacEachren, A. M. 1982. The role of complexity and symbolization method in thematic map effectiveness. *Annals of the Association of American Geographers*, 72(4), pp.495-513.

MacEachren, A. M., and Ganter, J. H. 1990. A pattern identification approach to cartographic visualization. *Cartographica*, 27(2), pp.64-81.

MacEachren, A. M. 1995. *How Maps Work: Representation, Visualization and Design*. New York, United States of America: The Guilford Press.

Millet, S., and Tune, S. 2015. *Patterns, Principles, and Practices of Domain-Driven Design*. Indianapolis, United States of America: John Wiley & Sons.

Moncrieff, S., Turdukulov, U., and Gulland, E. 2016. Integrating geo web services for a user driven exploratory analysis. *ISPRS Journal of Photogrammetry and Remote Sensing*, In Press, http://dx.doi.org/10.1016/j.isprsjprs.2016.01.015.

Object Management Group (OMG). 2015. *OMG Unified Modeling Language Specification*, version 2.5. Available online at http://www.omg.org/, accessed 28 April 2017.

Parnas, D. 2006. Agile methods and GSD: The wrong solution to an old but real problem. *Communication of the ACM*, 49(10), pp.29.

Petre, M. 2013. UML in practice. In *35th International Conference on Software Engineering (ICSE 2013)*, 18-26 May 2013, San Francisco, CA, USA, pp.722–731.

Pilone, D. 2005. *UML 2.0 in a nutshell. Sebastopol*, CA, United States of America: O-Reilly.

Rautenbach, V., Coetzee, S., and Iwaniak, A. 2013. Orchestrating OGC web services to produce thematic maps in a spatial information infrastructure. *Computers, Environment and Urban Systems*, 37(1), pp.107–120.

Rita, E., Borbinha, J., and Martins, B. 2010. Extending SLD and SE for cartograms. In *FOSS4G 2010*. 6–9 September 2010, Barcelona, Spain.

Robinson, A.C., Demšar, U., Moore, A.B., Buckley, A., Jiang, B., Field, K., Kraak, M-J., Camboim, S.P., and Sluter, C.R. 2017. Geospatial big data and cartography: research challenges and opportunities for making maps that matter, *International Journal of Cartography*, DOI: 10.1080/23729333.2016.1278151.

Saha, B., and Srivastava, D. 2014. Data quality: The other face of Big Data. *Proceedings - International Conference on Data Engineering*, pp.1294–1297.

Shalloway, A., and Trott, J. 2004. *Design Patterns Explained: A New Perspective on Object-Oriented Design*. Boston, United States of America: Addison-Wesley Professional.

Slocum, T. A., McMaster, R. B., Kessler, F. C., and Howard, H. H. 2009. *Thematic cartography and geovisualization*. Upper Saddle River, United States of America: Prentice Hall.

Stolte, C., Tang, D., and Hanrahan, P. 2002. Multiscale visualization using data cubes. *IEEE Symposium on Information Visualization (InfoVis)*, 7-14 January 2002, Boston, United States of America.

Tsou, M-H. 2015. Research challenges and opportunities in mapping social media and Big Data. *Cartography and Geographic Information Science*, 42(sup1), pp.70–74.

Tyner, J. 2010. *Principles of Map Design*. New York, United States of America: The Guilford Press.

Ward, J.S., and Barker, A. 2013. *Undefined by Data: A Survey of Big Data Definitions*. arXiv:1309.5821; 2013.