Noname manuscript No.
(will be inserted by the editor)

Applicability of open source web mapping libraries for building massive Web GIS clients

Gábor Farkas

the date of receipt and acceptance should be inserted later

Keywords approximate learning curve for javascript; client side library; comparison; massive client; software metrics; web gis;

Abstract The increasing capabilities of web browsers, and the growing spread of JavaScript has an impact on the development of web based GIS systems. While in traditional Web GIS applications the client side component is only responsible for creating representation models, modern geographically enabled JavaScript libraries have extended capabilities, making them capable of doing extensive tasks, like complex geographical analyses. This paper identifies the most capable libraries for being the basis of a Web GIS client (Cesium, Leaflet, NASA Web World Wind, OpenLayers 2, and OpenLayers 3), and compares them. The libraries are compared by their GIS feature coverage, and some quality metrics. OpenLayers 3 is identified for being the most capable library by supporting nearly 60% of the examined GIS features, its small size, and moderate learning curve. For comparing the learning curves of JavaScript libraries, a new metric named Approximate Learning Curve for JavaScript is proposed, which is based on other software metrics.

1 Introduction

The development of web mapping and Web GIS technologies is a recent trend in geoinformatics. After one and a half decade has passed since Google revolutionized this field with Google Maps (Farkas, 2015), there are matured toolkits or libraries, and frameworks (Ramsey, 2007) to be utilized. Contrary to the slow reaction from the open source segment of geoinformatics to proprietary desktop solutions like SYMAP, or ARC/INFO (Westervelt, 2004), the field of web based GIS in the Web 2.0 era was quickly dominated by open source projects, and initiatives (Haklay et al., 2008).

Gábor Farkas
Doctoral School of Earth Sciences, University of Pécs, 7624 Pécs, Ifjúság street 6, Hungary
E-mail: randal73@gamma.ttk.pte.hu

This phenomenon can be broken down to various factors. Web mapping, and web based GIS is a recent segment of geoinformatics. A more important factor is the nature of the web, and web based applications. Server side components can be written in a compiled language, like C, however client side components must be written in a browser compliant language. The most feasible language for that purpose is JavaScript, although its compilation is rather an optimization process. Consequently, client side codes and libraries can be studied, analyzed, and reused easily (Wang and Wu, 2014). The only way to protect client side source code is via obfuscating, but the result can still be reverse engineered by the decently experienced with appropriate tools, and methods.

Due to the inability of saving browser generated data other than images – which is a valid security restriction – every web based GIS software consist of a server side, and a client side component. There are three types of architecture based on the weight of these components (Doyle, 2000). In a thin client architecture, every operation is done on the server, only the results are sent to the client. The thick client architecture is the contrary. In this architecture, the client is fully responsible of rendering raw spatial data. The medium client is the compromise architecture, where display elements (described features, and rasters) are sent to the client, which is only responsible for drawing them on the screen.

These architectural definitions are useful, although they are outdated, as they only consider the rendering process (Doyle, 2000). This can be resolved with a historical perception. When they were created, rendering was a serious bottleneck. In modern browsers, harnessing the increased capability of modern computers (e.g. PCs, laptops, tablets, smartphones), client side rendering is not a problem anymore. As follows, modern clients can be supplied with more features, like minor geoprocessing algorithms, in brief, client geoprocessing (Hamilton, 2014).

Recently, several general purpose web based GIS frameworks, and libraries arose, both in the open source, and in the commercial segments. Although they vastly differ, there is one thing they mostly have in common: they are built with a thick client architecture without geoprocessing capabilities. ESRI's ArcGIS for Server, and CARTO's CartoDB.js are two of the most popular Web GIS frameworks, and both of them use a robust server side component to do every geoprocessing task (Hamilton, 2014; Esri, 2015). The purpose of this study is to analyze the most matured open source web mapping libraries in order to determine their applicability of being the basis of a massive client architecture.

2 Materials and methods

2.1 Massive Web GIS clients

Massive Web GIS client (hereafter, massive client) is a term proposed in this paper for identifying web clients, which resemble a general purpose GIS in functionality. These clients are inherently thick clients, as they must receive raw data, which they can not only render, but also analyze. Clearly, the set of features in a general purpose GIS can greatly vary (Maguire, 1991, 2008), and there is a hint of subjectivity in defining the compared features of such a system. On the other hand, the fundamental features of a basic GIS are well-outlined in the "classical" literature (Maguire, 1991; Meaden and Chi, 1996; Thrall and Thrall, 1999). A client does not have to support every feature from this study to be considered a massive client, although it must be able to perform some sort of client side data processing and analysis. This study defines the massiveness of a library by collecting those fundamental features, and by evaluating the coverage of the libraries of most popular web based standards and services (Wendlik et al., 2011; Brackin and Gonçalves, 2014).

The main purpose of the massive client architecture is balancing both traffic, and server resources. Contrary to the thick client architecture, a Web GIS with a massive client can be deployed on weaker servers, or can dedicate the server's resources to more time consuming tasks. To highlight some of the numerous possibilities of a massive client architecture, one can port smaller geoprocessing tasks to the client's computer, while the server executes operations on larger datasets, or one can dedicate a server to only serve data, while statistical analysis (e.g. aggregating, interpolating) is done in the client's browser.

There is an additional benefit for a massive client architecture. Traditional GIS software can be strongly platform dependent as they are usually written in programming languages which has to be recompiled between different hardware architectures and operating systems (e.g. C, C++). On the other hand, a massive client, which can utilize server resources besides local geoprocessing is platform independent. It works in browsers, therefore only cross-browser support has to be achieved. If it can harness the power of desktop applications (e.g. via Web Processing Service), only one code base has to be maintained, and the product can be used not only on different operating systems, but also on completely different devices (e.g. smartphones, tablets, PCs).

2.2 Identifying the candidates

In order to identify the most competent subjects for building a massive client, a few initial criteria must be set. The study only deals with open source libraries capable of web mapping, from which a massive client can be built. The main factor in rejecting proprietary applications is the general interest in having maximum control over the final application. While there are great proprietary web mapping libraries, their license terms can be restrictive, and they cannot be modified for better optimization, or increased extensibility. They act as a black box in the product. On the other hand, FOSS (free and open source software) libraries grant the four basic freedoms of running, studying and adapting, redistributing, and releasing improvements to the public (Steiniger and Hunter, 2013).

The libraries must not be abandoned, which means there should be recent (i.e. less, than six months) releases, or at least recent development activities in their repositories (e.g. bug fixes). Additionally, they should offer an API (Application Programming Interface) for creating interactive maps. As a final criterion, they should be general purpose libraries, thus one can integrate them into different frameworks without modifying their source code.

Prior to this study, web mapping libraries have been already compared for various purposes. In 2014, a big scale comparison took 16 open source libraries in account along some frameworks, and closed APIs. The purpose of that study was identifying the best technologies for teaching and web mapping (Roth et al., 2014). This study extends that list with some new, emerging libraries developed since, and attempts to identify the best of them for building massive clients.

2.3 GIS features

As massive clients have true GIS capabilities (e.g. spatial analysis) (Thrall and Thrall, 1999), the most accurate comparison would match the libraries against the capabilities of a desktop GIS software. However, there are no standards about the mandatory features of a desktop GIS. Luckily, as the field developed, there were various contributions to the definition of Geographical Information Systems (Maguire, 1991), and thus the most characteristic features of such a system are well-outlined.

For better clarity, one can group those features based on their specificity. First, there are non GIS specific features, mostly adapted from computer-aided design, computer cartography, database management, and remote sensing systems (Maguire, 1991). These features are responsible for rendering vector, and raster data, creating representation models (e.g. styling raw spatial data), adding cartographic elements, modifying geometries interactively, and organizing features in a database (DBMS), which can gradually enhance attribute management.

The second group consists of GIS specific features. GIS software need to know how to handle geographic data, and data exchange formats. Reading, and writing these formats, as well as the capability of connecting to spatial databases are GIS specific features. In order to narrow down the tremendous amount of data exchange formats, the most trending ones were selected for comparison according to a recent study (Orlik and Orlikova, 2014). Furthermore, geographic data are in coordinate reference systems, which must be handled by a GIS. Finally, a GIS should be able to pre-process (e.g. validate, transform, warp), manipulate, and analyze geographical data (Meaden and Chi, 1996; Albrecht, 1998; Thrall and Thrall, 1999).

While the two groups discussed above should cover the minimal capabilities of a desktop GIS software, a Web GIS software should also implement some web mapping specific features. As spatial databases occasionally hold voluminous amount of data (Agrawal and Gupta, 2014), in some cases it is more convenient to convert, or filter that data on the server side, and send only a simplified subset to the client. For this purpose, several standards were proposed. These standards include interfaces for client-server communication, from which the implementation of the most popular ones are checked as features. These features include some of the Open Geospatial Consortium service standards collaboratively named OGC Web Services (Percivall, 2010; Brackin and Gonçalves, 2014), and some of the most popular tiling services. These tiling services include open source specifications from OSGeo projects (e.g. OpenStreetMap's slippy map), as well as proprietary REST APIs (Wendlik et al., 2011).

Although these services (e.g. WMS, WFS) are not considered as formats in the literal sense, they behave similarly in practice. They fetch data from a source based on some parameters provided by the user or the developer. As from the perspective of this study there is no real difference between services, and traditional formats, this study groups services in the formats category. Following this analogy, despite images and rasters can be grouped in a single category in a desktop GIS, in a Web GIS they should be distinguished. The visualization of images (RGBA matrices) are natively supported in browsers, while rasters (matrices with arbitrary values) are not recognized by them. Finally, despite tile services and tiling schemes technically differ, their support in a client is similar. They only differ on the server side, as tile services usually require some kind of software to provide tiles (e.g. WMTS), while tiling schemes describe a template on accessing tiles directly from the server's file system (e.g. slippy map or MapBox's vector tiles). Due to the similarities in their client side implementation, tiling schemes are discussed in the tile service category.

The implementation of the final set of features (Figure 1) is examined for the candidate libraries. For practical reasons, the grouping scheme was altered to fit the modularity of a Web GIS client better, although it was striven to preserve the original grouping schemes of the literature mentioned above[1]. The ratings are moving on a scale between 0 and 1. If a library natively supports a feature, it gets a rating of 1, while if a plugin is required, or only partial support is available, a score of 0.5 is given. If a library does not support a given feature, therefore a workaround is required, it gets a score of 0. This way, the final rating represents a percent of coverage at the time of the analysis. Since the required features always depend on the purpose of a given project, no weighting has been applied to the scoring scheme. It shall be noted, as web mapping technologies are under rapid development, this score is rather a snapshot, than an absolute value (Roth et al., 2014).

In qualitative terms, the achieved score is inversely proportional with the necessary work to create a fully functional GIS with a library. The full support (100%) is a theoretical optimum, which cannot be achieved with current web technologies. If a library achieves this score in the future, it can be used as a basic standalone GIS without, or with minimal server side support. The same rule applies to groups and categories (e.g. rendering, data manipulation). The closer a library is to the total coverage, the less work is needed for a complete implementation of the category as a GIS module. There are 75 examined features, thus complete support of a single feature increases the overall coverage by 1.3%, while partial support increases it by 0.6%.

2.4 Static software metrics

As Roth et al. (2014) have pointed out, a pure capability matrix cannot characterize a library alone. There are other important factors, like platform dependency, test coverage (stability), documentation, examples, support, or difficulty to use a library. Some of these attributes can be quantified easily, while others can be ascertained by coding. One of the latter attributes (difficulty) is where an empirical study is superior, however the included subjectiveness should not be underestimated. In order to minimize this subjectivity, this study tries to make assumptions about the difficulty to use a library based on some of its quality metrics (Table 1).

These simple, sometimes addressed as crude metrics, which are around for nearly half a decade are often derived from LOC (Lines of Code), McCabe's cyclomatic complexity, and Halstead's software science metrics (Fenton and Neil, 1999).

[1] Of course the items can be restructured or modified based on other aspects until they give an overall picture of a functional GIS.

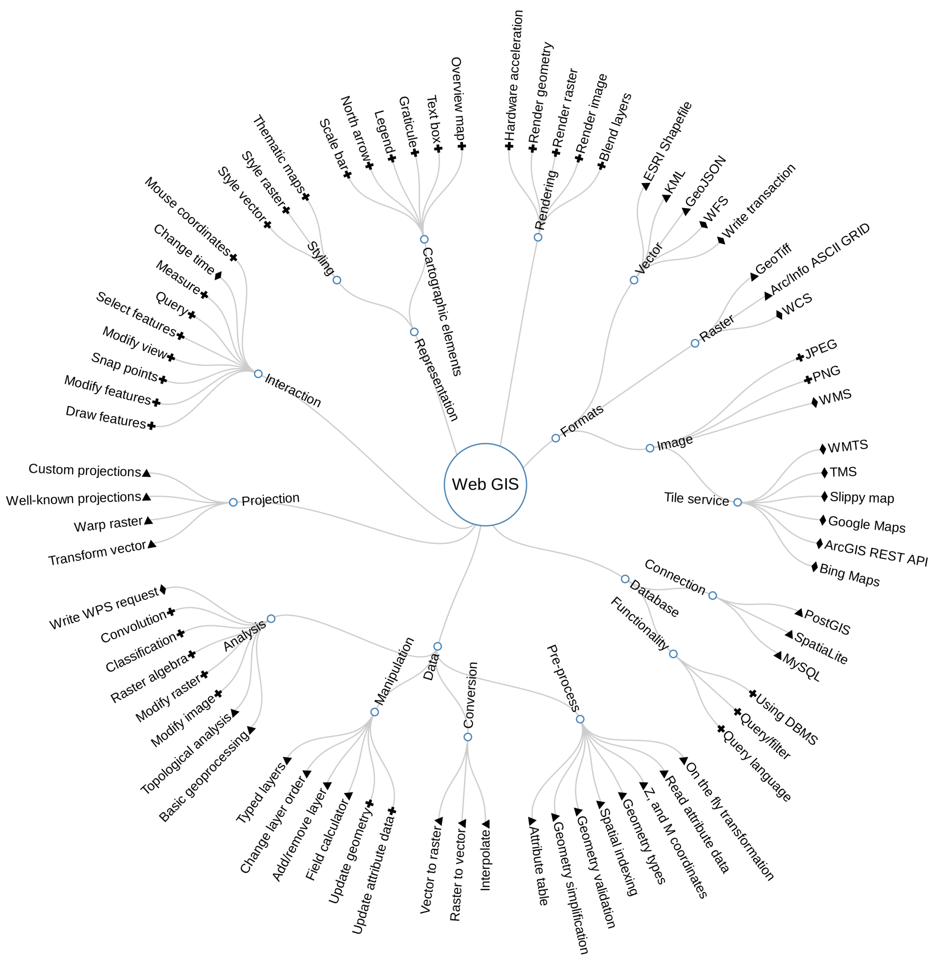

Fig. 1 Possible features of a basic Web GIS application grouped by functionality. Triangles, crosses, and diamonds denote GIS specific features, non GIS specific features, and web mapping specific features respectively.

From the LOC family, LLOC (Logical LOC) seems to be the most suitable, as new lines counted in PLOC (Physical LOC) can be used for formatting, guiding the reader's eye, or separating sections (Nguyen et al., 2007), which creates a distortion over the number of imperative statements. Besides LLOC, and physical size, the cyclomatic complexity is also calculated. McCabe's cyclomatic number ($v(G)$) is a graph-theoretic complexity measure, which looks at a code as a control flow. It decomposes a program to a graph G by breaking it to blocks (n vertices) delimited by statements (e edges) affecting control flow (e.g. if-else clause, for loop). It also respects the number of subroutines encountered (p connected components) (McCabe, 1976):

$$v(G) = e - n + p \tag{1}$$

As a final metric, the number of exposed functions are collected. These are constructors and functions accessible from the namespace of the analyzed library. Furthermore, as the size of the library is already accounted for by introducing the LLOC value, the latter two metrics are normalized to be independent from the size of the project. As cyclomatic complexity is used as a weighting factor for the overall complexity of the project, it is normalized with the number of functions F in the given library. On the other hand, as the user only encounters exposed functions EF, which can be directly invoked from the namespace of the library, the number of logical lines per exposed function seems to have the most impact on the learning curve of a project from the perspective of users (Fowler et al., 1999). Multiplying the metrics so far would give a greatly biased result towards the size of a library, thus a base ten logarithmic transform is performed on the LLOC metric. As the new number would be slightly biased towards the weighted average size of exposed functions, a base two logarithmic transform is performed on that metric. As a final result, a novel metric named $\mathrm{ALC_{JS}}$ (Approximate Learning Curve for JavaScript) is introduced, which still only gives a rough approximation on the difficulty of learning a library:

$$ALC_{JS} = \log_{10} LLOC \times \log_2(\frac{v(G)}{F} \times \frac{LLOC}{EF}) \tag{2}$$

There are two mathematical limits to this metric. Due to the logarithmic transformation, the LLOC value must be greater than, or equal with 1 (i.e. empty scripts cannot be measured for zero complexity). Furthermore, only libraries can be measured with this metric, as there must be at least one function in the measured code to prevent dividing by zero.

The candidates are measured with a tool named complexity-report, written for measuring JavaScript software. The values are recored for the release version of the libraries. The LLOC value, and the per function cyclomatic number are recorded with complexity-report, while a small script was written for obtaining the number of exposed functions (Appendix A). It could have been obtained in a static style (i.e. from the outside), however it is more convenient to collect it with JavaScript.

2.5 Other metrics

While static software metrics have the clear advantages of exact methodology and producing values on a ratio scale (e.g. nor LLOC, neither cyclomatic complexity can have negative values), it's hard – if not impossible – to describe usability or stability with them. On the other hand, those aspects of a software are also very important in an assessment, and are often described with ordinal values evaluating documentation, community, and support among other characteristics (Ramsey, 2007; Steiniger and Bocher, 2009; Poorazizi and Hunter, 2015).

To assess usability, the documentation and the community support are used. The documentation quality of a library is evaluated from the existence of an API documentation, the number of tutorials, and the number of examples. For evaluating the community support, the number of answered questions are aggregated from two well-known forums: GIS Stack Exchange and Stack Overflow.

The stability metrics (Table 1) are derived from statistics of the projects' VCS (version control system). From the numerous metrics, the number of contributors,

Table 1 Summary of the considered metrics for this study.

Metric	Description
Feature matrix	Support of basic GIS features which can act as indicators for the massiveness of the library.
Size	Physical size of the production version of the library.
LLOC	Logical lines of code in the production version of the library.
CC/F	Cyclomatic complexity of the library normalized with its number of functions.
Exposed functions	Functions which can be directly invoked from the namespace of the library.
$\mathrm{ALC_{JS}}$	Approximate learning curve derived from static metrics.
Documentation	Documentation quality derived from the library's API, tutorials, and examples.
Community support	Quality of community support derived from number of answered questions on StackExchange forums related to the library.
Contributors	The number of contributors and major contributors of the library.
Open issues	Number of open issues and their ratio to total issues in the library's VCS.
Release frequency	Average number of days between releases.

the number of open issues, and the release frequency are collected. To narrow down those numbers, an arbitrary threshold of 1 000 lines contributed is set to identify major contributors, while for open issues, their ratio to the total number of issues is also calculated. The release frequency RF – in order to be comparable across finished and active projects – is calculated from the days passed between the first release D_{FR} and the last release D_{LR}, and the number of releases n:

$$RF = \frac{D_{LR} - D_{FR}}{n - 1} \tag{3}$$

3 Results & Discussion

3.1 Subjects of comparison

Some of the libraries (Table 2), such as ka-Map, Modest Maps, or Polymaps were cutting-edge technologies in their times, but their support is discontinued. OpenScales is a special library, as its development is not yet completely abandoned, but it uses Flash, a technology whose support will not be discontinued on desktop platforms in the near future, but isn't available on mobile platforms since 2012 (Adobe, 2016). Some of them like D3, Raphaël, or Processing.js allow to create maps and other vector based graphics, but they are graphics libraries built for advanced data visualization, rather than web mapping (Bostock et al., 2011). This makes them exceptionally useful and convenient for rendering various vector graphics, like interactive charts, or static maps (Roth et al., 2014), but it would need too much work to shape them into a web mapping library (e.g. projection, layer, interaction, format support).

Since the study only considers open source libraries, proprietary APIs, like Google Maps API, and ArcGIS API for JavaScript were also sorted out. Although

Table 2 Well-known libraries used for web mapping.

Name	Version	Type	License[a]	Dependency	Last release[b]	Last activity[b]	Classification[c]
ArcGIS API for JavaScript*	4.0	Web mapping	Commercial	N/A	< 6 months	Unknown	Proprietary
Bing Maps AJAX Control*	7.0	Web mapping	Commercial	N/A	> Year	Unknown	Proprietary
CartoDB.js	3.15.10	Web mapping	BSD 3-Clause	Leaflet	< Month	< Month	Specific purpose
Cesium	1.23	Virtual globe	Apache 2.0	N/A	< Month	< Week	Candidate
D3	4.1.1	Data visualization	BSD 3-Clause	N/A	< Month	< Week	Vector graphics
Google Maps JavaScript API*	3.24	Web mapping	Commercial	N/A	< 6 months	Unknown	Proprietary
HERE Maps API for JavaScript*	3.0.12.4	Web mapping	Commercial	N/A	< Year	Unknown	Proprietary
ka-Map	1.0	Web mapping	MIT	N/A	> Year	Unknown	Abandoned
Kartograph	0.8.2	Web mapping	GNU LGPL	Raphaël	> Year	> Year	Abandoned
Leaflet	1.0.0-rc2	Web mapping	BSD 2-Clause	N/A	< Month	< Week	Candidate
Mapbox JS*	2.4.0	Web mapping	BSD 3-Clause	Leaflet	< 6 months	< Month	Specific purpose
Mapbox GL JS*	0.21.0	Web mapping	BSD 3-Clause	N/A	< Month	< Day	Specific purpose
MapQuery	0.1	Web mapping	MIT	OpenLayers 2	> Year	> Year	Abandoned
MapQuest JavaScript Maps API*	7.2	Web mapping	Commercial	N/A	> Year	Unknown	Proprietary
Modest Maps	3.3.6	Web mapping	BSD	N/A	> Year	> Year	Abandoned
NASA Web World Wind	0.0.1	Virtual globe	NOSA	N/A	> 6 months	< Week	Candidate
OpenLayers 2	2.13.1	Web mapping	BSD 2-Clause	N/A	> Year	< Week	Candidate
OpenLayers 3	3.17.1	Web mapping	BSD 2-Clause	N/A	< Month	< Week	Candidate
OpenScales	2.2	Web mapping	GNU LGPL	N/A	> Year	< 6 months	Other
OpenStreetMap iD	1.9.7	Web mapping	ISC	D3	< Month	< Day	Specific purpose
OpenWebGlobe	Unknown	Virtual globe	MIT	N/A	No release	< Year	Abandoned
Polymaps	2.5.1	Web mapping	BSD 3-Clause	N/A	> Year	> Year	Abandoned
Processing.js	1.6.0	Data visualization	MIT	N/A	< Month	< Month	Vector graphics
Raphaël	2.2.0	Data visualization	MIT	N/A	< 6 months	< Month	Vector graphics
WebGL Earth	2.4.2	Virtual globe	GNU GPLv3	Cesium	< Month	< Month	Specific purpose

[a]Every JavaScript library listed above is free to use. The main purpose of this column is to differentiate between open– and closed source projects.
[b]Compared to the date of the survey (28th July, 2016).
[c]If a library could be rejected based on multiple criteria, the strongest one was chosen in the following order: proprietary, abandoned, vector graphics, specific purpose, other.
*Requires an API key.

some of the libraries, like CartoDB.js, Mapbox JS, and WebGL Earth[2] extend their dependencies with valuable features, they are tailored to their respective frameworks with an adapter pattern (Shalloway and Trott, 2002). OpenStreetMap's iD is a preliminary example how D3 can be used for web mapping, however it is also tailored to the OpenStreetMap ecosystem, making it a special purpose library. The last rejected project was Mapbox GL JS. It has a great potential, as it uses hardware acceleration via WebGL (Eriksson and Rydkvist, 2015). However, similarly to Mapbox JS, it is developed to be compliant with other Mapbox services, thus it is not a general purpose library, rather a very fast vector tile renderer.

As a result of the analysis, five candidates were chosen for further research. Although virtual globes are built for different purposes than web mapping libraries, Cesium (Amato and Ring, 2015) and NASA Web World Wind not only have powerful 2D rendering engines, but they also have numerous GIS specific features, making them appropriate for the comparison. OpenLayers 2 has a completed status, however it is not abandoned. Minor updates are constantly delivered to this mature library with rich functionality. Moreover, based on OpenLayers 2's maturity and OpenLayers 3's young age, it should be assessed, if OpenLayers 3 is competent enough to completely replace its predecessor. OpenLayers 3 is the successor of OpenLayers 2, but it is a totally different library including cutting-edge features to meet modern demands (e.g. TopoJSON, vector tile support). Last, but not least, Leaflet is the lightweight solution for creating web mapping applications. Contrary to its small size, it is highly capable due to its extensibility, and vast amount of third party extensions developed.

3.2 Competitive analysis

As the result of the competitive analysis (Table 3 and Appendix B) shows, there are no major differences between the GIS feature support of the libraries. The OpenLayers libraries achieved the highest scores, showing that they were built with considerations to provide a basic GIS structure. Additionally, OpenLayers 3 gaining the lead is a clear indicator for the maturity of the library. It has reached a development level, where it should be the obvious choice amongst the two, if backward compatibility is not a criterion. The other libraries are on a similar coverage level, however the support of different feature groups does differ, thus in order to gain better insight, a more detailed analysis is presented.

For rendering tasks, the engine of Cesium is the best. It does everything but rendering raster data. From the rest of the libraries, NASA Web World Wind, and OpenLayers 3 are prominent, as they both support hardware acceleration through WebGL. As a downside, OpenLayers 3's support is only partial, as it cannot render lines, and polygons with its WebGL renderer, which is inevitable for a massive Web GIS client.

Regarding the format handling, all of the libraries have a similar coverage. They support an adequate number of formats, which can be broken down to specific subgroups (Table 4). Most of their differences are in their vector handling. The support for vector formats are outstanding in the OpenLayers libraries. They can

[2] Whether WebGL Earth is more of a façade due to its dependence on Cesium, than an adapter is debatable.

Table 3 GIS feature coverage of the candidate libraries.

Feature group	Cesium	Leaflet	NASA WWW[a]	OpenLayers 2	OpenLayers 3
Rendering	80%	40%	60%	40%	60%
Formats	65%	62%	53%	82%	76%
Database	0%	8%	0%	17%	0%
Data	32%	30%	18%	34%	44%
Projection	63%	50%	75%	63%	88%
Interaction	33%	50%	33%	83%	72%
Representation	22%	44%	33%	56%	56%
Average	41%	41%	34%	54%	56%

[a]Web World Wind.

Table 4 Format support coverage of the candidate libraries.

Format	Cesium	Leaflet	NASA WWW[a]	OpenLayers 2	OpenLayers 3
Vector	50%	60%	50%	90%	90%
Raster	17%	17%	17%	17%	17%
Image	100%	100%	100%	100%	100%
Tile service	67%	32%	50%	100%	83%
Average	65%	62%	53%	82%	76%

[a]Web World Wind.

read and write almost every examined format. Leaflet also has a rich vector format support, although it achieves reading from most of them via third party extensions, and only has the capability to write features in GeoJSON format natively, and WFS transactions with an extension. The raster and image support of the libraries are uniform, as they do not support any of those formats. From the specified raster formats, GeoTiff can be processed with a third party library, and thus drawn as a plain image on the map canvas. Image formats do not have to be supported, as they are supported by the browsers themselves (except from WMS, which is supported by all of the candidates). Finally, the coverage of the most popular tile services also differ in the libraries. Both versions of OpenLayers offer a rich support for tile providers even beyond the scope of the analysis. The other libraries have more limited support, however all of them implement the most basic ones, like the slippy map tiling scheme, or WMTS.

There are two categories in the database section (Table 5). Similarly to image and raster formats, the libraries have a uniform support in connecting to spatially enabled databases. This is understandable, as supporting direct connection with server side DBMSs is undesirable even with modern web technologies, due to the involved risk with removing a layer of security. For example, every access information would be exposed to attackers, who could gather sensitive data from an incautiously configured database more easily. Features grouped in the database functionality category have a higher variance amongst the candidates due to the small number of inspected features, and their varying support, or semi-support. Although it is a small group, implementing an internal DBMS is a key step in building a GIS software. It has a key role in efficient information processing, and querying (Revesz, 2008). A good DBMS implementation can also help in concatenating the data of a layer in an attribute table, calculating layer-wise statistics, and maintaining the consistency of the data. None of the candidate libraries have implemented an internal DBMS, however OpenLayers 2 offer a solution for fil-

Table 5 Database support coverage of the candidate libraries.

Database	Cesium	Leaflet	NASA WWW[a]	OpenLayers 2	OpenLayers 3
Connection	0%	0%	0%	0%	0%
Functionality	0%	17%	0%	33%	0%
Average	0%	8%	0%	17%	0%

[a]Web World Wind.

Table 6 Data support coverage of the candidate libraries.

Data	Cesium	Leaflet	NASA WWW[a]	OpenLayers 2	OpenLayers 3
Pre-process	50%	25%	25%	19%	63%
Conversion	0%	0%	0%	0%	0%
Manipulation	50%	67%	25%	83%	67%
Analysis	13%	19%	13%	25%	25%
Average	34%	32%	20%	34%	46%

[a]Web World Wind.

tering, and querying attribute data layer-wise, while Leaflet supports filtering a layer.

The data section also comprises numerous features, and can be broken down to various subgroups (Table 6). In pre-processing data, OpenLayers 3 is outstanding with its native support for spatial indexing, four dimensional coordinates, and geometry simplification. On the other hand, it is only capable of warping raster layers in an on the fly approach, although this capability is still unique amongst two dimensional web mapping libraries. The selected virtual globes, however have a complete implementation of on the fly transformation. Conversions, similarly to other raster related algorithms are not supported by any of the libraries, although an immature form of interpolation is present in some of them, which can create heat maps from point data. In data manipulation, the web mapping libraries excel, while the virtual globes fall behind, as they do not offer methods for updating geometries, or even attribute data programmatically. Typed layers has two different interpretations, and the classification examines both of them. The first one can be considered a weak criterion, as all of the compared libraries use constructors to create layers, thus their instances can be queried via native JavaScript. However, these queries have to be made against every layer type, thus storing the type in the layer object is more convenient. The second one is a stronger one, as it examines if vector layers can be constrained to a single geometry type, which is a very important feature in a GIS, as there are geoprocessing tools, which can only operate on a single type. Data analysis has a poor overall coverage, because only writing WPS requests is natively supported, and only in OpenLayers 2. Raster based geoprocessing algorithms are not present in any of the libraries, while vector based ones are uniform, as they are provided via third party libraries, like Turf, or JSTS. Although none of the examined web mapping libraries utilize those modules, both Turf and JSTS offer a common format: GeoJSON. Furthermore, JSTS has an interface for reading and writing the native data model of OpenLayers 3.

Correct projection handling is another key aspect of a GIS. In this field, OpenLayers 3 provides the best coverage, as it can transform vector features, and warp rasters to any projection known by the JavaScript port of the PROJ.4 projection library. OpenLayers 2, and Leaflet also have decent capabilities, however warping rasters is not implemented in them. Virtual globes have a natural support for on

Table 7 Representation support coverage of the candidate libraries.

Representation	Cesium	Leaflet	NASA WWW[a]	OpenLayers 2	OpenLayers 3
Styling	67%	67%	67%	67%	83%
Carto. e.[b]	0%	33%	17%	50%	42%
Average	22%	44%	33%	56%	56%

[a]Web World Wind.
[b]Cartographic elements.

the fly transformations, vector transformations, and raster warping. On the other hand, they have a major weakness, as their projection support is limited, and the supported projections are hard coded in their source code.

The group of interactions contains features (e.g. draw, modify, snap), which make interacting with the application made with a library more convenient. They are functions mapped to GUI (Graphical User Interface) elements giving feedback to–, or taking instructions from the user, and are mostly derived from CAD systems. OpenLayers 2 excels in these features, as it not only supports every examined feature, but provides the richest collection of native interactions amongst the candidates. The other library worth highlighting is Leaflet. Although it does not have a native support for any of the features, the huge number of third party extensions make its possible capabilities even wider, than OpenLayers 2's. From the other candidates, OpenLayers 3 provides a decent coverage, while virtual globes generally lack functions, or up to date extensions related to drawing.

Features designated to create rich representation models (e.g. styling, thematic maps, scale bar) are grouped in the representation group (Table 7). In this final collection, the results indicate some of the differences between web mapping libraries, and virtual globes, and how strongly the candidates affiliate to these concepts. Styling is equally important for both of them, however, as most of the functions related to cartographic elements are strongly related to digital cartography, and web mapping, the greater coverage of web mapping libraries is understandable. On the other hand, NASA Web World Wind implements an adequate number of cartographic elements, thus it represents the otherwise missing step from the continuous transition between web mapping libraries, and virtual globes.

3.3 Metrical results

From the candidates' static software metrics (Table 8) the difficulty of learning a library cannot be directly identified, however they still grant some valuable insight into their nature. Web mapping libraries aren't necessarily smaller or more lightweight, than virtual globes. Cesium is expected to have a large $\mathrm{ALC}_{\mathrm{JS}}$ value due to its size and capabilities, however NASA Web World Wind's smaller value compared to the OpenLayers libraries can be surprising. It can be explained with the number of features implemented into the OpenLayers libraries, and the young age of NASA Web World Wind. Leaflet is the smallest in both size, and LLOC, however its capabilities are also limited without extensions, and they can gradually increase the final application's size. Finally, OpenLayers 3 is not only more capable than its predecessor, but it is also smaller in size. This however, is not caused by a smaller code base, but a better compression method.

Table 8 Static software metrics of the candidate libraries.

Library	Size (KB)	LLOC	CC/F[a]	EF	ALC_{JS}
Cesium	11 420	292 500	2.08	911	51.27
Leaflet	162	3 639	2.00	200	18.47
NASA Web World Wind	1 452	13 037	2.50	187	30.64
OpenLayers 2	872	23 702	2.82	207	36.46
OpenLayers 3	499	21 451	2.36	223	33.90

[a]Cyclomatic complexity per function.

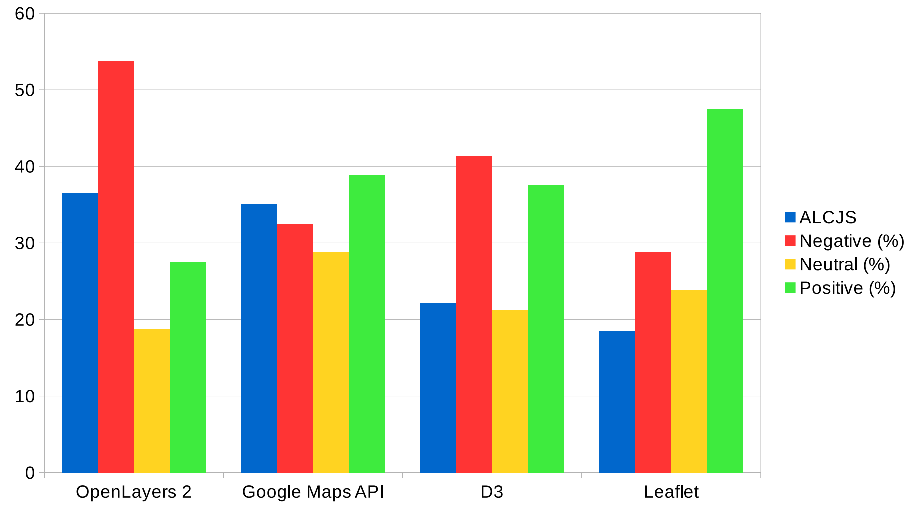

Fig. 2 ALC_{JS} compared to ratio of moods experienced by developers during a diary study (Roth et al., 2014).

In order to validate ALC_{JS}, a function with minimal complexity (Appendix C) was also measured along with jQuery, a library well-known for its user friendliness and calm learning curve (Lindley, 2009; Król and Szomorowa, 2015). The addNums function received a score of 0.30, while jQuery received 16.55, fulfilling the expectations about the metric. Furthermore, the metric nicely aligns with the ratio of positive and negative moods experienced by developers in Roth et al.'s diary study (Figure 2 and Table 9). The only exception is Google Maps API, which indicates, this metric can only be used for a rough approximation for the time and effort needed to learn a library. However, there are more, less quantifiable factors (e.g. documentation, support, personal qualities) which contribute to the real experience.

The usability of this static metric is limited, as it only involves a small set of easily obtainable static metrics. A better object-oriented metric for learning difficulty could also involve the proportion of private and public constructors, methods, constructor arguments, and coupling factors. However, obtaining those values is more cumbersome, and the methodology of calculating such a metric is yet to be developed. Furthermore, in theory, ALC_{JS} applies to every object-oriented language, however assessing its accuracy in other languages than JavaScript should be done prior to nominating it as a general metric.

Table 9 Order of the libraries by $\mathrm{ALC}_{\mathrm{JS}}$ (lowest first), the ratio of positive moods (highest first), and the ratio of negative moods (lowest first) experienced in the diary study (Roth et al., 2014).

Factor	First	Second	Third	Fourth
$\mathrm{ALC}_{\mathrm{JS}}$	Leaflet	D3	Google Maps API	OpenLayers 2
Positive moods	Leaflet	Google Maps API	D3	OpenLayers 2
Negative moods	Leaflet	Google Maps API	D3	OpenLayers 2

Table 10 Other properties of the candidate libraries.

Property	Cesium	Leaflet	NASA WWW[a]	OL 2[b]	OL 3[b]
Documentation[c]	Good	Good	Decent	Very good	Very good
Community[c]	Decent	Very good	Poor	Good	Good
Contributors	89 (29)	236 (8)	12 (5)	98 (16)	154 (27)
Open issues	409 (25%)	225 (7%)	40 (56%)	383 (64%)	414 (21%)
RF[d] (days)	28	68	N/A	32	24

[a]Web World Wind.
[b]OpenLayers.
[c]Possible values: poor, decent, good, very good.
[d]Release frequency.

The non-code metrics (Table 10) show a greater variance amongst the candidates. As of usability metrics, all of the libraries have good API documentations, and just enough tutorials or examples to set the developer on the right track. Furthermore, Cesium, Leaflet, and the OpenLayers libraries give enough help to create advanced applications, while NASA Web World Wind leaves much space for fiddling in such situations. The community support of virtual globes are relatively small, while there are much more questions related to web mapping libraries. NASA Web World Wind does not have a dedicated label, nor related questions on the examined forums, however it has a dedicated place on The NASA World Wind Forum with a small amount of questions. The usability metrics were evaluated on an ordinal scale, as the absolute numbers should be interpreted with caution. For example, Leaflet has the most answered questions (4 370, about 80% of active questions), OpenLayers 3 has the most tutorials (23), and OpenLayers 2 has the most examples (210). To put those numbers in contrast, D3 has 16 697 answered questions (about 78% of active questions) on the aforementioned forums, offers 81 tutorials, and 978 examples.

The stability metrics of the candidates show, all of them are maintained by an adequate number of developers. More popular libraries (e.g. Leaflet) have gradually more contributors, however most of them did only make a small amount of contributions. On the contrary, the number of major contributors (in parentheses) seems to be increasing with the volume and complexity of the library. From the number of open issues in contrast to their ratio to the total number of issues (in parentheses) it can be determined, every library has a decent support level. Most of the libraries have or had (OpenLayers 2) monthly releases, while Leaflet produces a new release at an average of 2 months. The release frequency of NASA Web World Wind could not been determined, as it only has a single release.

4 Conclusions

From the great amount of geographically enabled JavaScript libraries, there are currently five projects, which should be paid special attention to. They are Cesium, Leaflet, NASA Web World Wind, and the OpenLayers libraries. They can be categorized into two groups based on their specific design concepts: web mapping libraries, and virtual globes. Virtual globes have the advantage of native real 3D and 2.5D visualization, and hardware accelerated 2D rendering, while web mapping libraries are generally better at supporting GIS features.

Although the overall GIS feature coverage of Cesium is not outstanding, this library should not be underestimated. It is the biggest of the candidates by orders of magnitude. As a consequence, it has the steepest learning curve, but it pays off the invested time, if its capabilities are sufficient for the given project. Its main disadvantage is its lack of known projections. It can use the two most widely used projections in web mapping though: the Web Mercator, and the Plate Carrée.

NASA Web World Wind could be considered as a lightweight alternative of Cesium, however there is one major difference, which should be taken into account for a given project. It supports eight (mostly polar) projections including the two supported by Cesium, and as it is a virtual globe, it has a full support for on the fly transformation. If a project needs an application, which can transform vector data correctly between global, and polar projections, this is the only suitable library. On the other hand, if a project starts out as a simple application, but expected to be more complicated in the future, the increased learning curve due to the lack of advanced documentation makes Cesium a better choice.

Leaflet is known for its lightweight nature, and its mild learning curve, which was confirmed by this study. Furthermore, it has a vast number of third party extensions; most of them are out of the scope of the study. It's low coverage value is mostly the consequence of giving partial scores for features supported by an extension. If every feature supported by a dedicated Leaflet extension received a score of 1, the library's coverage would have risen to 55%, slightly above OpenLayers 2's. This library should be the preferred choice for projects, where scalability is not among the requirements, but a capable Web GIS is needed. As a final point, its good documentation level and very good support makes it an excellent starting library for learning web mapping.

Among the OpenLayers libraries, OpenLayers 3 outgrew its predecessor in almost every aspect. It offers similar capabilities, but in a more advanced form. However, there are still a few points, where OpenLayers 2 is stronger. If supporting legacy browsers is a criterion for the application, it can be considered along Leaflet. It also has a good support for building queries, and filters, therefore it could act as a stronger foundation for an application with statistical capabilities.

From the perspective of building a massive Web GIS client based on the parameters provided by this study, there is no trivial best choice. OpenLayers 3 has the best GIS feature coverage, and a moderate learning curve. However, its limited WebGL support should be extended prior to using it as a massive Web GIS client. Creating 3D visualizations, and analyzing multidimensional spatial data has an increasing demand, and is a native feature in many desktop GIS software. OpenLayers 3, however, is not only incapable of creating 3D visualizations on its own, but it also has a limited support for even drawing one–, or two dimensional features on a two dimensional plane with hardware acceleration. If 3D support is a hard criterion, Cesium could be easier to extend, although it needs at least a custom projection handling system implemented. Leaflet could also be considered, however besides the lack of WebGL support, one could argue, the cohesion between modules in the other libraries are stronger than between Leaflet and its third party extensions.

Acknowledgements I would like to thank the numerous anonymous reviewers for their precious time dedicated to the evaluation of this paper, their valuable comments and suggestions, which helped me shape it in its current, enhanced form. I would also like to thank my colleague, András Hervai, for sharing his thoughts on software metrics, which can directly affect user experience.

A JavaScript routine for measuring the exposed functions of a library

```
function measure(obj, visited) {
  visited = visited || [];
  var count = 0;
  if (obj !== window) {
    visited.push(obj);
    for (var i in obj) {
      if (visited.indexOf(obj[i]) === -1) {
        visited.push(obj[i]);
        if (typeof obj[i] === 'object') {
          count += measure(obj[i], visited);
        } else if (typeof obj[i] === 'function') {
          count++;
        }
      }
    }
  }
  return count;
};
```

B Detailed support table of the candidate libraries

Category	Cesium	Leaflet	NASA WWW[a]	OL 2[b]	OL 3[b]
Rendering					
Hardware acceleration	1	0	1	0	0.5
Render geometry	1	1	1	1	1
Render raster	0	0	0	0	0
Render image	1	1	1	1	1
Blend layers	1	0	0	0	0.5
Formats – Vector					
ESRI Shapefile	0.5	0.5	1	0.5	0.5
KML	1	0.5	0.5	1	1
GeoJSON	1	1	1	1	1
WFS	0	0.5	0	1	1
Write transaction	0	0.5	0	1	1
Formats – Raster					
GeoTiff	0.5	0.5	0.5	0.5	0.5
Arc/Info ASCII GRID	0	0	0	0	0
WCS	0	0	0	0	0
Formats – Image					

JPEG	1	1	1	1	1
PNG	1	1	1	1	1
WMS	1	1	1	1	1
Formats –Image – Tile service					
WMTS	1	0.5	1	1	1
TMS	1	1	0	1	0.5
Slippy map	1	1	1	1	1
Google Maps	0	0.5	0	1	0.5
ArcGIS REST API	1	0.5	0	1	1
Bing Maps	1	0.5	1	1	1
Database – Connection					
PostGIS	0	0	0	0	0
SpatiaLite	0	0	0	0	0
MySQL	0	0	0	0	0
Database – Functionality					
Using DBMS	0	0	0	0	0
Query/filter	0	0.5	0	1	0
Query language	0	0	0	0	0
Data – Pre-process					
On the fly transformation	1	0	1	0	0.5
Read attribute data	1	1	0	1	1
Z, and M coordinates	1	0	0.5	0	1
Geometry types	0.5	0.5	0.5	0.5	0.5
Spatial indexing	0	0	0	0	1
Geometry validation	0	0	0	0	0
Geometry simplification	0.5	0.5	0	0	1
Attribute table	0	0	0	0	0
Data – Conversion					
Interpolate	0	0	0	0	0
Raster to vector	0	0	0	0	0
Vector to raster	0	0	0	0	0
Data – Manipulation					
Update attribute data	1	1	0	1	1
Update geometry	0	1	0	1	1
Field calculator	0	0	0	0	0
Add/remove layer	1	1	1	1	1
Change layer order	1	1	0.5	1	1
Typed layers	0	0	0	1	0
Data – Analysis					
Basic geoprocessing	0.5	0.5	0.5	0.5	0.5
Topological analysis	0.5	0.5	0.5	0.5	0.5
Modify image	0	0	0	0	0.5
Modify raster	0	0	0	0	0
Raster algebra	0	0	0	0	0
Classification	0	0	0	0	0
Convolution[c]	0	0	0	0	0
Write WPS request	0	0.5	0	1	0.5
Projection					
Transform vector	1	1	1	1	1
Warp raster	1	0	1	0	1
Well-known projections[d]	0.5	0.5	0.5	0.5	0.5
Custom projections	0	0.5	0.5	1	1
Interaction					
Draw features	0	0.5	0	1	1
Modify features	0	0.5	0	1	1
Snap points	0	0.5	0	1	1
Modify view[e]	1	0.5	1	0.5	1
Select features	1	0.5	0.5	1	1
Query	0	0.5	0.5	0.5	0.5

Measure	0	0.5	0	1	0
Change time	1	0.5	0	0.5	0
Mouse coordinates	0	0.5	1	1	1
Representation – Styling					
Style vector	1	1	1	1	1
Style raster	0	0	0	0	0.5
Thematic maps[f]	1	1	1	1	1
Representation – Carto. e.[g]					
Scale bar	0	1	0	1	1
North arrow	0	0	1	0	0
Legend	0	0	0	0	0
Graticule	0	0.5	0	1	1
Text box	0	0	0	0	0
Overview map	0	0.5	0	1	0.5

[a]Web World Wind.
[b]OpenLayers.
[c]Also known as moving window (GRASS) and focal statistics (ArcGIS).
[d]Projections in the EPSG database.
[e]Ability to pan, zoom, and rotate the map.
[f]Ability to create choropleth and proportional symbol maps from vector data.
[g]Cartographic elements.

C Example JavaScript function

```
var addNums = function(a, b) {
    return a + b;
};
```

References

Adobe (2016). Adobe® roadmap for the Flash® runtimes. Technical report.

Agrawal, S. and Gupta, R. D. (2014). Development and Comparison of Open Source Based Web GIS Frameworks on WAMP and Apache Tomcat Web Servers. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL(4):1–5.

Albrecht, J. (1998). Universal analytical GIS operations – a task-oriented systematization of data structure-independent GIS functionality. *Geographic information research: Transatlantic perspectives*, pages 577–591.

Amato, M. and Ring, K. (2015). Getting Serious with JavaScript. In Cozzi, P., editor, *WebGL Insights*, chapter 4, pages 49–70. CRC Press.

Bostock, M., Ogievetsky, V., and Heer, J. (2011). D^3: Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309.

Brackin, R. and Gonçalves, P. (2014). OGC OWS Context Conceptual Model Version 1.0.0. Technical Report 12-080r2, Open Geospatial Consortium.

Doyle, A. (2000). OpenGIS Web Map Server Interface Implementation Specification Revision 1.0.0. Technical Report 00-028, Open Geospatial Consortium.

Eriksson, O. and Rydkvist, E. (2015). An in-depth analysis of dynamically rendered vector-based maps with WebGL using Mapbox GL JS. Master's thesis, Linköpings universitet, Linköping, Sweden.

Esri (2015). ArcGIS 10.3.1 for Server Functionality Matrix. Technical report.

Farkas, G. (2015). Comparison of Web Mapping Libraries for Building WebGIS Clients. Master's thesis, University of Pécs, Pécs, Hungary.

Fenton, N. E. and Neil, M. (1999). Software metrics: successes, failures and new directions. *Journal of Systems and Software*, 47(2–3):149 – 157.

Fowler, M., Beck, K., Brant, J., Opdyke, W., and Roberts, D. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional.

Haklay, M., Singleton, A., and Parker, C. (2008). Web Mapping 2.0: The Neogeography of the GeoWeb. *Geography Compass*, 2(6):2011–2039.

Hamilton, E. L. (2014). Client-side Versus Server-side Geoprocessing. Master's thesis, University of Wisconsin – Madison, Madison, WI, USA.

Król, K. and Szomorowa, L. (2015). The possibilities of using chosen jQuery JavaScript components in creating interactive maps. *Geomatics, Landmanagement and Landscape*, 2:45–54.

Lindley, C. (2009). *jQuery Cookbook: Solutions & Examples for jQuery Developers*. O'Reilly Media, Inc.

Maguire, D. J. (1991). An overview and definition of GIS. *Geographical information systems: Principles and applications*, 1:9–20.

Maguire, D. J. (2008). ArcGIS: General Purpose GIS Software System. In Shekhar, S. and Xiong, H., editors, *Encyclopedia of GIS*, pages 25–31. Springer, NY, USA.

McCabe, T. J. (1976). A Complexity Measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320.

Meaden, G. J. and Chi, T. D. (1996). Geographical information systems Applications to marine fisheries. Technical report, Food and Agriculture Organization of the United Nations, Rome.

Nguyen, V., Deeds-Rubin, S., Tan, T., and Boehm, B. (2007). A SLOC Counting Standard. Technical Report USC-CSSE-2007-737, Center for Systems and Software Engineering.

Orlik, A. and Orlikova, L. (2014). Current Trends in Formats and Coordinate Transformations of Geospatial Data – Based on MyGeoData Converter. *Central European Journal of Geosciences*, 6(3):354–362.

Percivall, G. (2010). Progress in OGC Web Services Interoperability Development. In Di, L. and Ramapriyan, H. K., editors, *Standard-Based Data and Information Systems for Earth Observation*, pages 37–61. Springer, Berlin Heidelberg.

Poorazizi, M. E. and Hunter, A. J. (2015). Evaluation of Web Processing Service Frameworks. *OSGEO Journal*, 14:29–42.

Ramsey, P. (2007). The State of Open Source GIS. Technical report, Refractions Research Inc., Suite 300 – 1207 Douglas Street Victoria, BC, V8W-2E7.

Revesz, P. (2008). Constraint Databases, Spatial. In Shekhar, S. and Xiong, H., editors, *Encyclopedia of GIS*, pages 157–160. Springer, NY, USA.

Roth, R., Donohue, R., Sack, C., Wallace, T., and Buckingham, T. (2014). A Process for Keeping Pace with Evolving Web Mapping Technologies. *Cartographic Perspectives*, 0(78):25–52.

Shalloway, A. and Trott, J. R. (2002). *Design Patterns Explained A New Perspective on Object-Oriented Design*. Addison-Wesley Professional.

Steiniger, S. and Bocher, E. (2009). An Overview on Current Free and Open Source Desktop GIS Developments. *International Journal of Geographical Information Science*, 23:1345–1370.

Steiniger, S. and Hunter, A. J. (2013). The 2012 free and open source GIS software map – A guide to facilitate research, development, and adoption. *Computers, Environment and Urban Systems*, 39:136–150.

Thrall, S. E. and Thrall, G. I. (1999). Desktop GIS software. In Longley, P. A., Goodchild, M. F., Maguire, D. J., and Rhind, D. W., editors, *Geographical Information Systems Abridged*, pages 331–345. John Wiley & Sons, Inc.

Wang, Z. Y. and Wu, W. M. (2014). Technique of Javascript Code Obfuscation Based on Control Flow Tansformations. *Applied Mechanics and Materials*, 519–520:391–394.

Wendlik, V., Karut, I., and Behr, F. (2011). Tiling Concepts and Tile Indexing in Internet Mapping APIs. In *Applied Geoinformatics for Society and Environment 2011 - Geoinformation for a better world*, pages 116–121.

Westervelt, J. (2004). GRASS Roots. In *Proceedings of the FOSS/GRASS Users Conference*, pages 1–10.