SOCRAT Platform Design: A Web Architecture for Interactive Visual Analytics Applications

Alexandr A. Kalinin
University of Michigan
akalinin@umich.edu

Selvam Palanimalai
University of Michigan
selvam.palanimalai@gmail.com

Ivo D. Dinov
University of Michigan
dinov@umich.edu

ABSTRACT

The modern web is a successful platform for large scale interactive web applications, including visualizations. However, there are no established design principles for building complex visual analytics (VA) web applications that could efficiently integrate visualizations with data management, computational transformation, hypothesis testing, and knowledge discovery. This imposes a time-consuming design and development process on many researchers and developers. To address these challenges, we consider the design requirements for the development of a module-based VA system architecture, adopting existing practices of large scale web application development. We present the preliminary design and implementation of an open-source platform for Statistics Online Computational Resource Analytical Toolbox (SOCRAT). This platform defines: (1) a specification for an architecture for building VA applications with multi-level modularity, and (2) methods for optimizing module interaction, re-usage, and extension. To demonstrate how this platform can be used to integrate a number of data management, interactive visualization, and analysis tools, we implement an example application for simple VA tasks including raw data input and representation, interactive visualization and analysis.

CCS CONCEPTS

•**Human-centered computing** → *Interactive systems and tools;* •**Information systems** → *Web applications;* •**Software and its engineering** → *Software architectures;*

KEYWORDS

Visual Analytics, System Design, Web Platform Architecture

ACM Reference format:
Alexandr A. Kalinin, Selvam Palanimalai, and Ivo D. Dinov. 2017. SOCRAT Platform Design: A Web Architecture for Interactive Visual Analytics Applications. In *Proceedings of HILDA'17, Chicago, IL, USA, May 14, 2017,* 6 pages.
DOI: http://dx.doi.org/10.1145/3077257.3077262

1 INTRODUCTION

Over the two last decades, the web has proven itself to be a successful platform for the development of information visualizations [26]. Web-based visualization solutions dramatically reduce deployment issues by running directly in the desktop or mobile web browser, yielding a high degree of accessibility, and by avoiding complex installation, version update, incompatibility, and other problems characteristic of standalone software [32]. A number of recent web visualization frameworks, such as the ubiquitously used D^3 [14], incorporate proven and efficient practices for improved compatibility, accessibility, and performance. Recent rapid development of technologies, frameworks, and best design practices for complex interactive front-end applications, together with increased graphical and interactive capabilities native to modern web browsers (HTML5/JavaScript), enable enhancement of visualizations by user-focused interactions. However, VA web applications development practices vary more for multiple reasons [12], such as greater diversity of domains, dataset properties, required analysis and visualization, and desired results. Even under specific assumptions about these properties, designing web visual analytics systems is more challenging than visualizations, since VA requires the integration of data management, analysis, and highly interactive visualization solutions into a complex web application [24].

These challenges are related to the design principles of large scale general purpose interactive applications, which are described as non-trivial applications with in-browser data manipulation and display [27]. However, there is a lack of adoption of these techniques, which not only notably limits the capabilities of existing web VA applications, but also imposes greater costs and longer development times on their developers. Currently, developers spend the majority of their time re-implementing VA system architectures rather than focusing on interaction, innovation and creation of sophisticated web VA applications. This is even more important for complex analytics solutions, where re-implementation of data processing, modeling, and analysis methods are very expensive and error-prone [22]. Improved computational capabilities of modern web browsers [25] enable implementations of mathematical, statistical, and computing JavaScript libraries such as Math.js [4], jStat [3], Science.js [8], and ConvNetJS [2]. While some of these libraries provide out-of-the-box interface for visualizations, others are relatively easy to integrate into web visual analytics workflow.

These developments provide an exciting opportunity for creating more flexible, scalable, reusable, and sustainable web-based VA systems that could benefit from bringing together modern web browser capabilities and best practices for building complex interactive web applications. Development of systems that integrate in-browser solutions for data analysis and data management with a diverse set of web visualization tools would involve design decisions that balance their expressive, interactive and processing capabilities, efficiency, compatibility, and accessibility.

HILDA'17, Chicago, IL, USA

978-1-4503-5029-7/17/05…$15.00
DOI: http://dx.doi.org/10.1145/3077257.3077262

In this paper, we first consider design requirements for the development of module-based VA systems from developer's perspective. We discuss how existing practices of large scale web application development, together with modern visualization tools can be adopted to build integrative VA applications. We contribute the preliminary design and implementation of a platform for Statistics Online Computational Resource Analytical Toolbox (SOCRAT). This extensible, integrative platform defines: (1) a specification for an architecture for building VA applications with multi-level modularity, and (2) methods for optimizing modular interaction, re-usage, and extension. Through a simple use case, we demonstrate how this platform can be used to integrate a number of data management, interactive visualization and analysis tools into an exemplar web application with multi-source data input, raw data display, interactive charting and clustering analysis. Lastly, we briefly discuss ongoing work and prospective developments.

2 BACKGROUND AND RELATED WORK

2.1 Modular web architectures for interactive visualization

Clairvoyant work by Bender *et al.* [11] was one of the first to suggest framework architectures that implement web visualization pipeline as containing 3 separate steps: data preparation, representation and rendering. They defined a number of requirements for such frameworks, including an ease of extensibility, a standardized way to interact with system in real time, and a modular structure with dynamically loaded components. Central piece of such framework is a Kernel module, that is responsible for internal communication and initialization of all other components, including user interface (UI), renderer, data manager, and computational modules. This design, based on module independence, allowed flexible configurations with more or less operations executed on client vs. server.

The Statistics Online Computational Resource (SOCR) suite of tools [16] implemented an educational web-based framework that included interactive visualization components based on JFreeChart, a Java charting library [19]. SOCR Charts [16] and Motion Charts [10] allow interactive representation of data summary statistics, raw data, and data mapping. Over 20 various provided charts include 3D, spatial, and cartographic visualizations.

Prefuse [19] is the one of the first toolkits designed with modularity at the level of visualization designs. Based on experience of building prefuse and reviewing other visualization solutions, Heer and Agrawala in [18] noted the difficulty in identifying common design patterns within existing visualization tools, and, consequently, the high cost for users to learn and evaluate unfamiliar systems. Later, Protovis [13] was introduced as a toolkit based on declarative language for web visualization design that decoupled visualization specification from execution and improved expressiveness. Generalizing this approach, D^3 [14] provided a declarative framework for mapping data to visual elements in the document object model (DOM) and dynamically transforming them. However, D^3 existed as a single library until the recent version 4.0, when it was split into a collection of modular "microlibraries", flattening the hierarchical relationship between them, and allowing to create custom bundles with only a subset of modules compatible with current and prospective web module standards, such as ES6 modules [29].

A higher level modular approach for building visualization tools is represented by a recent mixed-initiative Voyager system [36, 37] that relies on a number of separately released Node.js packages in CommonJS format [29] used as building blocks, including D^3, declarative visualization grammars Vega [31] and Vega-Lite [30], visualization recommendation engine Compass [35], and custom data utility and UI libraries.

Finally, and likely closest to our work, is Plastic.js [20], a data display framework designed with modular architecture for extensibility, asynchronous data loading and display, default inner data format, and schema. It defines Query Modules, Data Modules, and Display Modules that are developed using software design patterns for large-scale JavaScript applications [27], including factory and facade patterns for module decoupling and global observer pattern to handle the asynchronous events for inter-module interactions [28]. As an integrative framework, it supports the use of multiple visualizations solutions, including D^3, and also provides dependency management and lazy loading.

Clearly, modular architectures are not uncommon among various web visualization solutions. However, modularity is introduced on very different levels of these frameworks and toolkits, from high-level components to single visualization specification to small implementation details, since such tools address a wide range of challenges in information visualization. These different levels of abstraction are not necessarily contradictory, and various approaches can be used in combination. Further, we argue that this kind of multi-level modularity is beneficial for building interactive visualization components that can be integrated into an extensible, flexible web-based VA system.

2.2 Modular web visual analytics with SOCR

In our earlier work [16], the SOCR suite of tools implemented a web-based collection of tools for interactive modeling and visual data analysis. The SOCR Modeler provided interactive visual model fitting [16]. The SOCR Analyses component provided hypothesis testing, statistical modeling, and computation of power and sample size [15]. Together with the SOCR Charts, these components were implemented using an MVC pattern, allowing to decouple interactive visual representation from modeling techniques, such that the latter could be easily extended and used as external computational library. Although mainly designed for educational purposes, the suite of SOCR tools enabled visual analytics workflow for interrogation of the dataset interactively by visualizations and data analysis. Examples include California ozone pollution case study using SOCR Charts and Analysis [17] and visual analysis of big medicare, labor, census and econometric data with SOCR Data Dashboard [21].

However, architecturally original SOCR applets lack interoperability. Their modular structure provides extensibility and sustainability, but there is no integrative platform that provides resource sharing and runtime interaction. It is important, for example, for optimization of many repetitive user actions in data management, including input, storage, preprocessing, transforming, and querying. Moreover, most of SOCR applets eventually became practically unavailable to end users after new versions of browsers disabled Java by default as a response to numerous vulnerability reports [34]. Our approach in this study is based on our experience with

developing and maintaining SOCR tools, with a long-term goal of re-developing them as modules of a common integrative platform that would allow their integration into a unified system to address these limitations. From these experiences, we also conclude that implementation is as important as design, so we focus on both aspects in this work.

3 VISUAL ANALYTICS SYSTEM DESIGN

As discussed in the previous section, modular web configurations of visualizations, data analytics, and interactive techniques are quite different and it's difficult to combine them in any standardized way, since important considerations cover component design (Prefuse vs. D^3), system structure (Voyager vs. Plastic.js), and implementation (e.g. Java applets vs. ES6 modules). Component design itself entails various approaches to structuring visualizations, interactions, and analytics. Here, we consider some of the requirements, and possible solutions, for an integrative web VA architecture that would combine these approaches in a beneficial and diverse way. The following list of requirements was inspired by an analysis of [11], and is updated accordingly to include advancements in the field and those related to the studies mentioned above.

3.1 Requirements

Integrative web VA systems should should satisfy following requirements:

- Architecture should be *flexible* for easy addition, modification, and replacement of the modules. Use case: the developer should be able to easily change default data storage from in-browser to server-hosted database.
- System has to provide the means for module *interoperability*, such that they have a flexible way to interact with each other without introducing hard dependencies. Use case: the developer should be able to easily request data in an analysis module from a database by communicating with Database Module and without querying data directly.
- The system architecture has to allow easy *extension* of system components, for example, by adding support for various third-party visualization or computational libraries. Use case: the developer should be able to easily include and use a third-party dependency inside of VA application, for instance, vega-lite visualization grammar [30].
- Architecture has to be *robust* in runtime, meaning that a failure of one module should not affect the state of others. Use case: the developer should be sure that the application in production will not crash in case if one of the third-party modules fails.
- Components should be *reusable* across various activities. Use case: the developer should be able to easily use the specific calculations from various components that rely on them without re-implementation. Similarly, visualization specifications should be reusable across modules.
- System architecture should emphasize *expressiveness* of applications built on it, by providing access to fine-grained control over visualization and analytics methods. Use case: the developer should be able to directly control visual components on the level of DOM if there is need for it.
- Finally, *accessibility* includes simple protocols for creating new modules from scratch as well as from existing code together with detailed documentation. Use case: the developer, unfamiliar with the details of platform architecture, should be able to quickly prototype a VA application, from data input to visualization and interaction.

3.2 General architecture

The proposed system is represented by a loosely coupled architecture [27] with functionality broken down into independent modules with, ideally, no inter-module dependencies. Modules are single-purpose parts of a system with limited permissions. They are interchangeable in the sense that the system is capable of supporting, adding, removing or replacing modules without the rest of the modules in the system failing, which facilitates flexibility and robustness. In this decoupled setting, modules do not directly communicate with each other. Instead, they provide a means to communicate with the Core module. The Core module is a central control piece responsible for the initialization and internal communications of other modules, and for satisfying the interoperability requirement, similar to the Kernel module in [11]. If necessary, it performs module runtime validation and monitoring. Extensibility is implemented in the form of plug-in support, which enables wrapping third-party components as modules, providing them with a standard for other modules API.

3.3 Module structure

The starting point for designing modular architecture is actually to break it down into independent functional pieces and then define their responsibilities. For web systems, it's common to design modules following MVC-like patterns (MV*), separating the data from the display, and organizing their interaction with the medium component. For VA application architecture, it's natural to initially separate visualization from data analytics and data storage, although, different views, on the contrary, should be made possible to combine. Thus, interactivity is more difficult to decouple into separate modules. Two simple options to approach this limitation are either to: (1) provide basic interactivity specifications within general intermediate display layer, that would be accessible by different modules, or (2) define more specific interactions individually in the scope of each module. The second option allows for more flexibility in terms of implementation, including easier third-party component integration.

In suggested architecture, we do not impose strict module classification. Instead, we suggest that there are a few loosely defined types of modules: for example, modules that perform specific actions on data, such as calculations without implementing the UI (background modules) or modules that use data to populate interactive visualization specifications (visual modules). Modules with new visualization capabilities should be able to implement high-level predefined specifications as well as low-level control definitions, depending on the task. To combine such variety of approaches, the module's components should be modular as well. Intra-module component structure should allow for simple ways to circulate information within a module. For inter-module communication, module inner components can be exposed by opening

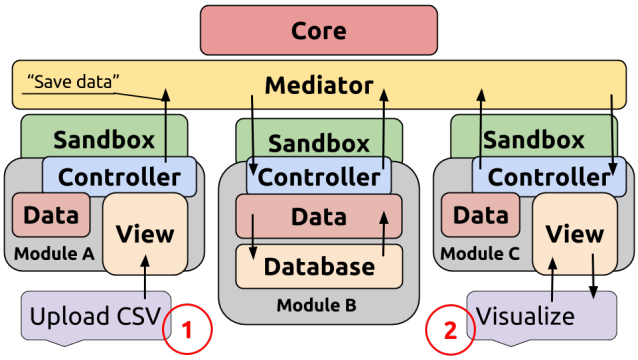

Figure 1: General modular platform architecture. Human-computer and inter-modular interactions (via Sandbox-Mediator pattern) are shown by arrows. From left to right: (1) user uploads CSV file using module A, which broadcasts "Save data" message; Core module redirects the message to module B that saves the data into database; (2) then user requests visualization of data in module C, which requests data from module B, receives the data, and displays it in the view component.

access to them for other modules via Core, for example, providing calculations-as-a-service or visualization-as-a-service. In practice, however, it is reasonable to expect that a typical module of a web VA system will be a hybrid of these types.

3.4 Core module for inter-module communications

Upon module initialization by the Core module, all modules are provided with Sandbox, an instance representing Facade software design pattern [28], that hides inner Core structure from the module behind high-level messaging interface. This interface, in turn, is represented by a Mediator pattern [28] that prevents modules from directly referring to each other and instead acts as an intermediary. Core can use Mediator to start, stop, and restart individual modules selectively in the runtime, without breaking the application. It is also responsible for answering module's request. For example, if a visual module requested specific data transformation that was outsourced to another module, which is currently not available, Core will use Mediator to provide visual module with negative result, such that it can display an appropriate error message or placeholder and/or try again later.

4 SOCRAT CORE IMPLEMENTATION

We present a prototype of SOCRAT, a multilevel modular platform for building VA applications. Current implementation relies on AngularJS 1.5 [1], a widely used web framework with detailed documentation and a large, established community. Most importantly, it provides many useful features out-of-the box that fit well in the context of proposed architecture. Implementation of the Core module was also inspired by scaleApp framework that realizes similar basic architectural principles [7]. Modularity with declarative component definitions is present at the system level (third-party dependencies), platform level (SOCRAT modules), and module level (AngularJs components and routing). Below, we further discuss each level in greater detail.

4.1 Module level

From a high-level perspective, each SOCRAT module is implemented as an instance of a wrapper class for AngularJS module [1], which automatically pre-defines required structures. The AngularJS module declaratively specifies its structural parts, such as Services, Components or Directives, and Controllers, which fit organically into the suggested architecture of the platform. Conveniently, these sub-modular parts can be loaded in any order, lazily initialized and easy to test, and they support dependency injection. Familiar to many developers, the MV* structure of the framework fits very well into VA application requirements. Services, that exist in AngularJS application as singletons, are used to implement logic, transfer and process data, along with various types of calculations. At the same time, Components provide a means of constructing reusable visualization specifications, with a possibility for low-level control using Directives. Two-way data binding enables easy automation of many interactive tasks in a declarative fashion, such that user actions can be reflected in visualizations or data. Each sub-modular component is implemented following a Module design pattern [27] in CommonJS format, which allows to define them in declarative way and load them synchronously or asynchronously during the runtime. We defined a number of base wrapper classes, such that new sub-module components can be created using OOP-like native prototypal inheritance in JavaScript. With the same purpose, these classes provide a number of convenience methods, including, for example, intra-module communication via broadcasting or requesting data from Database module. This wrapping hides the service code developers would otherwise have to write, and instead allows working on higher levels of abstraction with interactions between users, visualizations, and data. However, this does not limit module-level flexibility, since these classes can be easily extended and their methods overloaded.

Besides providing standard AngularJS components of a module, SOCRAT allows custom inclusion of a third-party dependency by composition. This is needed when a large third-party component is a dependency for a single module and there is no need to wrap it as a separate SOCRAT module. For example, ngHandsontable is an AngularJS component that implements an Excel-like interactive data grid editor [5]. If the VA applicationfis design simply displays raw data on input, ngHandsontable does not have to be a SOCRAT module itself. In this case, we allow for tight coupling via direct dependency injection, but we gain extensibility and save the time that would have been spent on wrapping the ngHandsontable interface into the SOCRAT module.

4.2 Platform level

In order to satisfy the rest of design requirements, we impose additional structures onto the AngularJS modular application.

First, we automate module and component definitions, such that the module can be described in a fully declarative way. This abstracts the inner details of the chosen framework on the platform level, and makes the creation of new modules much easier for the

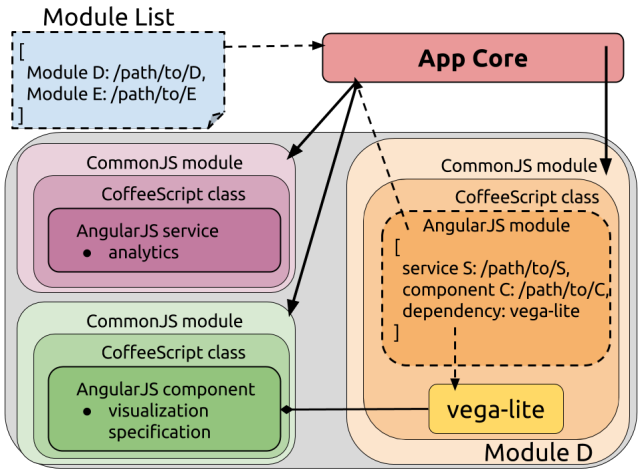

Figure 2: Module-level implementation. An example of a declarative list of modules. The Core module parses the list and initializes modules D and E by reference. Then, similarly, module D's declarative configuration is parsed and its components are initialized. Module D also includes vega-lite [30] by composition as third-party dependency available to module's visual component.

developer, who will not have to deal with AngularJS structures, so instead can focus on important functionality. It is also specifically convenient for VA application development, because it easily indicates which modules to include into a specific application, using them as building blocks. For example, if the prospective application has to be suited for working with time-series data, the developer just has to include references to specific CommonJS modules containing necessary visualizations and analytical methods. At the application runtime those modules will be automatically initialized and included into platform infrastructure.

Second, we do not allow a SOCRAT module to be injected as a dependency into another module, except for inclusion of dependencies by composition as described above. Instead, we use the proposed Sandbox-Mediator pattern combination to implement inter-module communications via Core module. In fact, modules are not started until Core parses the module list and automatically creates their components. When a module is ready to start, Core first performs a validation to ensure that the module's interface is compatible with the platform. Then, Core gives the module access to an instance of Sandbox, and subscribes to messages from a module through Mediator. What is left for the developer is to link the various module messages into pairs or chains, such that Core can invoke an interface method of one module in response to request from another, see Fig. 1. Exposing the interface of a module into messaging systems makes it reusable by other modules.

Similarly to Plastic.js [20], SOCRAT implements a simple, standard data format that is used for messaging information exchange between modules, without limiting it to a specific scheme. On data input, SOCRAT will try to flatten or map data onto a 2D grid, and will store it as a JSON object if that fails. In the future, we plan to include better standardized inner data representation.

4.3 User interface

Similar to the UI designs of Tableau (formerly Polaris) [33], Voyager and PoleStar [36, 37], SOCRAT implements a top bar menu, sidebar and main central area, see Fig. 3.

The main menu is used to provide access to SOCRAT modules that implement UI. It is indicated in declarative fashion in every module definition, along with its components. On the applicationfis start, SOCRAT Core recursively parses the module list and automatically adds links to all UI modules into main menu, while registering their URL in the routing scheme. This takes the menu-building burden from the developer, and allows to organize the main menu in many configurations, including nested dropdown sub-menus.

Sidebar is a common UI pattern that is typically used for auxiliary control placement. Both the main area and hideable sidebar are automatically initialized for every SOCRAT module, along with the methods for their interaction. Sidebar typically implements requests for current datasets and displays some of their properties, while central area can be used as a view for particular visualization specifications. This is the only UI restriction that SOCRAT imposes onto modules with UI; the developer can choose any visualization library in combination with any analytical methods that will be further used to build the application.

4.4 System level

We use npm [6] to include system dependencies defined in a declarative way. Runtime dependencies may include various libraries, frameworks, and custom Angular-enabled components that can be directly injected into modules. Webpack [9] is used to build the application and to separate resulting code into chunks that can be loaded dynamically, which represent the highest modularity level in the proposed architecture.

5 APPLICATION EXAMPLE USE CASE

In this short example we combined: a) ngHandsontable [5], a dynamic Excel-like data grid editor for raw data input and display, b) datavore [23], a small and fast in-browser database for data storage and querying, c) a low-level charting D^3 library [14] for visualization, and d) jStat [3] JavaScript statistical library. While ngHandsontable, D^3, and jStat are injected directly into the Data Input module, Charting module, and Clustering module correspondingly, the datavore is wrapped into a separate Database SOCRAT module. This application provides an interface to use one of the predefined SOCR Datasets, or to upload and parse custom CSV/TSV file, build few various predefined charts, and perform visualized clustering of dataset using k-Means algorithm, see Fig. 3.

6 DISCUSSION AND ONGOING WORK

In this paper, we discuss the design considerations for the development of complex web-based VA applications and present a design of a web platform that realizes multi-level modular architecture, adopting best design practices for building large scale interactive front-end applications to VA-specific requirements. We then demonstrate a preliminary implementation of SOCRAT, an example application for simple VA tasks that uses proposed architecture to combine raw data input and representation with interactive visualization and analysis.

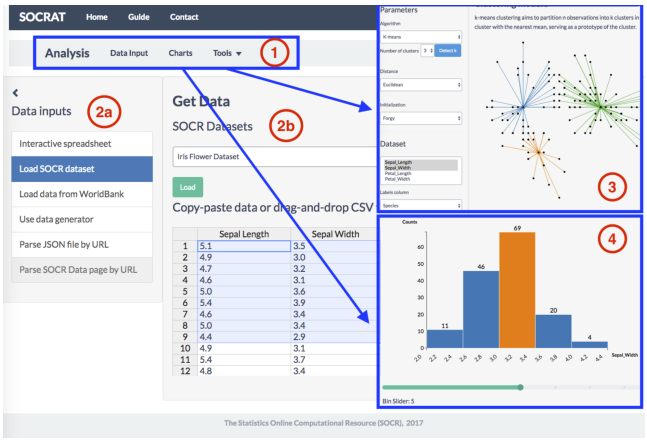

Figure 3: Screenshot of SOCRAT interface: (1) main menu automatically generated from module list, (2) data input module interface with various data sources in sidebar (2a) and raw data display in central panel (2b), (3) interactive clustering module with the results of k-means, (4) interactive histogram with variable number of bins.

As a next step, we plan to extend SOCRAT to support more of various existing and custom data management, visualization, and analytical components. We also will update design consideration from analyst point of view, with the focus on user experience. We hope that this work will engage the VA community into a discussion of best architecture design practices for creating more effective and feature-rich VA solutions.

ACKNOWLEDGMENTS

The work is partially supported by the National Science Foundation under Grants No.: 1023115, 1022560, 1022636, 0089377, 9652870, 0442992, 0442630, 0333672, 0716055, the National Institutes of Health under Grants No.: P20 NR015331, P50 NS091856, P30 DK089503, U54 EB020406, and the Elsie Andresen Fiske Research Fund.

REFERENCES

[1] AngularJS. Retrieved March 2, 2017 from https://angularjs.org/
[2] ConvNetJS. Retrieved March 2, 2017 from http://convnetjs.com/
[3] jStat. Retrieved March 2, 2017 from https://jstat.github.io/
[4] Math.js. Retrieved March 2, 2017 from http://mathjs.org/
[5] ngHandsontable. Retrieved March 2, 2017 from https://handsontable.github.io/ngHandsontable/
[6] NPM. Retrieved March 3, 2017 from https://www.npmjs.com/
[7] scaleApp. Retrieved March 2, 2017 from http://scaleapp.org/
[8] Science.js. Retrieved March 2, 2017 from https://github.com/jasondavies/science.js/
[9] Webpack. Retrieved March 3, 2017 from https://webpack.github.io/
[10] Jameel Al-Aziz, Nicolas Christou, and Ivo D Dinov. 2010. SOCR motion charts: An efficient, open-source, interactive and dynamic applet for visualizing longitudinal multivariate data. *Journal of statistics education: an international journal on the teaching and learning of statistics* 18, 3 (2010).
[11] Michael Bender, Ralf Klein, Andreas Disch, and Achim Ebert. 2000. A functional framework for web-based information visualization systems. *IEEE Transactions on Visualization and Computer Graphics* 6, 1 (2000), 8–23.
[12] Paul Booth, Wendy Hall, Nicholas Gibbins, and Spyros Galanis. 2014. Visualising data in web observatories: a proposal for visual analytics development & evaluation. In *Proceedings of the 23rd International Conference on World Wide Web*. ACM, 1055–1060.
[13] Michael Bostock and Jeffrey Heer. 2009. Protovis: A graphical toolkit for visualization. *IEEE transactions on visualization and computer graphics* 15, 6 (2009).
[14] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics* 17, 12 (2011), 2301–2309.
[15] Annie Chu, Jenny Cui, and Ivo D Dinov. 2009. SOCR analyses–An instructional java web-based statistical analysis toolkit. *Journal of online learning and teaching/MERLOT* 5, 1 (2009), 1.
[16] Ivo D Dinov. 2006. Socr: Statistics online computational resource. *Journal of Statistical Software* 16, 11 (2006).
[17] Ivo D Dinov and Nicolas Christou. 2011. Web-based tools for modelling and analysis of multivariate data: California ozone pollution activity. *International journal of mathematical education in science and technology* 42, 6 (2011), 789–805.
[18] Jeffrey Heer and Maneesh Agrawala. 2006. Software design patterns for information visualization. *IEEE transactions on visualization and computer graphics* 12, 5 (2006).
[19] Jeffrey Heer, Stuart K Card, and James A Landay. 2005. Prefuse: a toolkit for interactive information visualization. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 421–430.
[20] Simon Heimler. 2014. Development of a Modular JavaScript Data Display Framework. In *Applied Research Conference 2014: 5th July 2014, Ingolstadt*. Shaker Verlag GmbH, Aachen, Germany.
[21] Syed S Husain, Alexandr Kalinin, Anh Truong, and Ivo D Dinov. 2015. SOCR Data dashboard: an integrated big data archive mashing medicare, labor, census and econometric information. *Journal of big data* 2, 1 (2015), 13.
[22] Darrel C Ince, Leslie Hatton, and John Graham-Cumming. 2012. The case for open computer programs. *Nature* 482, 7386 (2012), 485–488.
[23] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 3363–3372.
[24] Daniel Keim, Gennady Andrienko, Jean-Daniel Fekete, Carsten Görg, Jörn Kohlhammer, and Guy Melançon. 2008. Visual analytics: Definition, process, and challenges. In *Information visualization*. Springer, 154–175.
[25] Faiz Khan, Vincent Foley-Bourgon, Sujay Kathrotia, Erick Lavoie, and Laurie Hendren. 2014. Using javascript and webcl for numerical computations: A comparative study of native and web technologies. In *ACM SIGPLAN Notices*, Vol. 50. ACM, 91–102.
[26] Shixia Liu, Weiwei Cui, Yingcai Wu, and Mengchen Liu. 2014. A survey on information visualization: recent advances and challenges. *The Visual Computer* 30, 12 (2014), 1373–1393.
[27] Addy Osmani. 2011. Patterns For Large-Scale JavaScript Application Architecture. Retrieved March 2, 2017 from https://addyosmani.com/largescalejavascript/
[28] Addy Osmani. 2012. *Learning JavaScript Design Patterns: A JavaScript and jQuery Developer's Guide*. O'Reilly Media, Inc.
[29] Axel Rauschmayer. 2015. Exploring ES6: Upgrade to the next version of JavaScript. Retrieved March 2, 2017 from https://leanpub.com/exploring-es6/
[30] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2017. Vega-lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (2017), 341–350.
[31] Arvind Satyanarayan, Ryan Russell, Jane Hoffswell, and Jeffrey Heer. 2016. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE transactions on visualization and computer graphics* 22, 1 (2016), 659–668.
[32] Chad A Steed, Katherine J Evans, John F Harney, Brian C Jewell, Galen Shipman, Brian E Smith, Peter E Thornton, and Dean N Williams. 2014. Web-based visual analytics for extreme scale climate science. In *Big Data (Big Data), 2014 IEEE International Conference on*. IEEE, 383–392.
[33] Chris Stolte, Diane Tang, and Pat Hanrahan. 2002. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics* 8, 1 (2002), 52–65.
[34] US-CERT. 2013. Oracle Java Contains Multiple Vulnerabilities. Alert (TA13-064A). Retrieved March 2, 2017 from https://www.us-cert.gov/ncas/alerts/TA13-064A
[35] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2016. Towards a general-purpose query language for visualization recommendation. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*. ACM, 4.
[36] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2016. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE transactions on visualization and computer graphics* 22, 1 (2016), 649–658.
[37] Kanit Wongsuphasawat, Zening Qu, Dominik Moritz, Riley Chang, Felix Ouk, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2017. Voyager 2: Augmenting Visual Analysis with Partial View Specifications. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM.