Visualization Linter

Static and runtime check tools

Youssef Sherif
Wei Zheng

Demo

- Static Analysis tool
- Runtime Library

Background

- Why do we need a visualization linter?
  - To help the “average Joe” data visualization user adhere to visualization best practices
  - Educate the users about data vis in a user-friendly and intuitive manner
- Why two tools?
  - Each of the static analysis tool and run-time library have its own uses, pros, and cons.

Static Analysis Tools

- Not meant to check data-related issues
  - Why? Because data is dynamic
    - Example: Http request from a backend server
- Meant to check for logical problems

Runtime Tools

- Has access to data during runtime
- Cons
  - Warnings and Errors are displayed on runtime (not immediately)

Programming Language and Framework

- JavaScript
  - Web applications are on the rise
    - JavaScript is the default web language
- D3.js and Chart.js
  - Most popular
  - Open source

Implementation

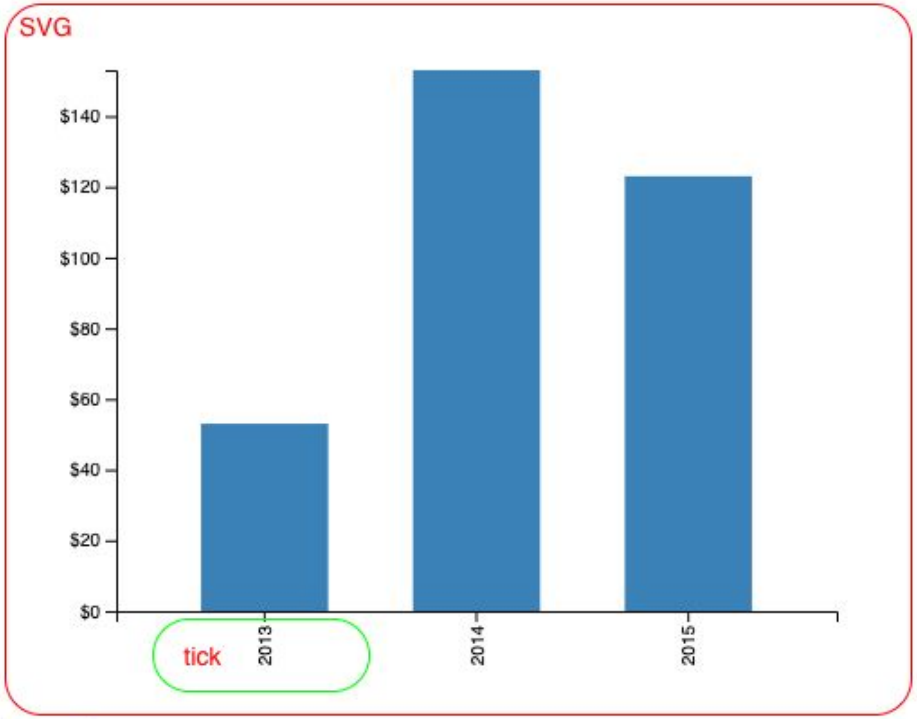
- Static Analysis Tool
  - ESlint plugin
    - ESlint is the most widely used JavaScript pluggable static linter

- Run-time library checker
  - A regular npm package

Resources for Rules for Best Practices

- Tamara's book "Visualization Analysis and Design"
  - Example: order of effectiveness
- The Visualization Guidelines Repository
- Yan Holtz's online guideline

Dealing with SVG



Check the rotation for the tick in green circle in the following way:

```
var svg = document.getElementsByTagName("svg");
var label = svg.querySelector(".tick > text");
var valueStr = label.getAttribute("transform");
```

In this way, we can check d3 at run time.

```
▸Uncaught TypeError: svg.querySelector is not a function   test3.js:50
    at checkTick (test3.js:50)
    at test3.js:57
```

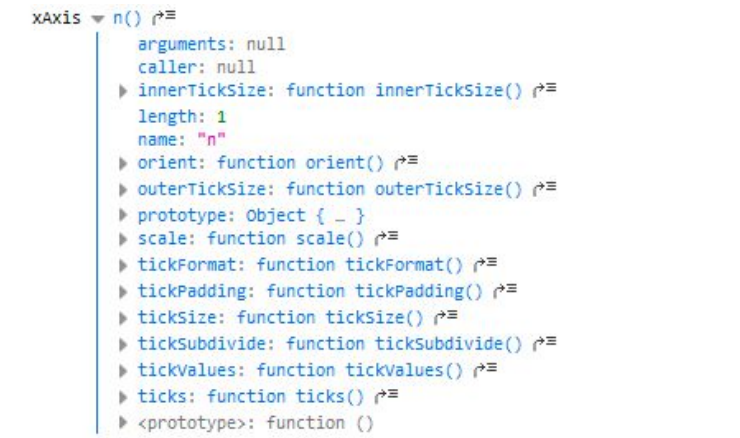
Dealing with SVG

- **Impossible for some rules, very hard for others**
- **Fragile**

Deal with other elements instead of SVG?

- Since dealing with SVG is hard, we thought about asking the user to manually provide in code features of the graph like
  - The type of graph
  - X-axis of the graph if applicable
  - Y-axis of the graph if applicable
- Unlike SVG's data structure, data structures of other graph's elements such as the axis can are easier to parse and comprehend

```
D3Checker.lint({
    type: 'barChart',
    xAxis: xAxis,
    yAxis: yAxis
})
```



Problems with this approach

- A hassle for the user to provide all of these data

A third approach

Fork D3.js and manipulate the code inside to add checkers

- This approach needs a lot of time of studying D3 internal code
- We abandoned this approach

Different Approaches Comparison

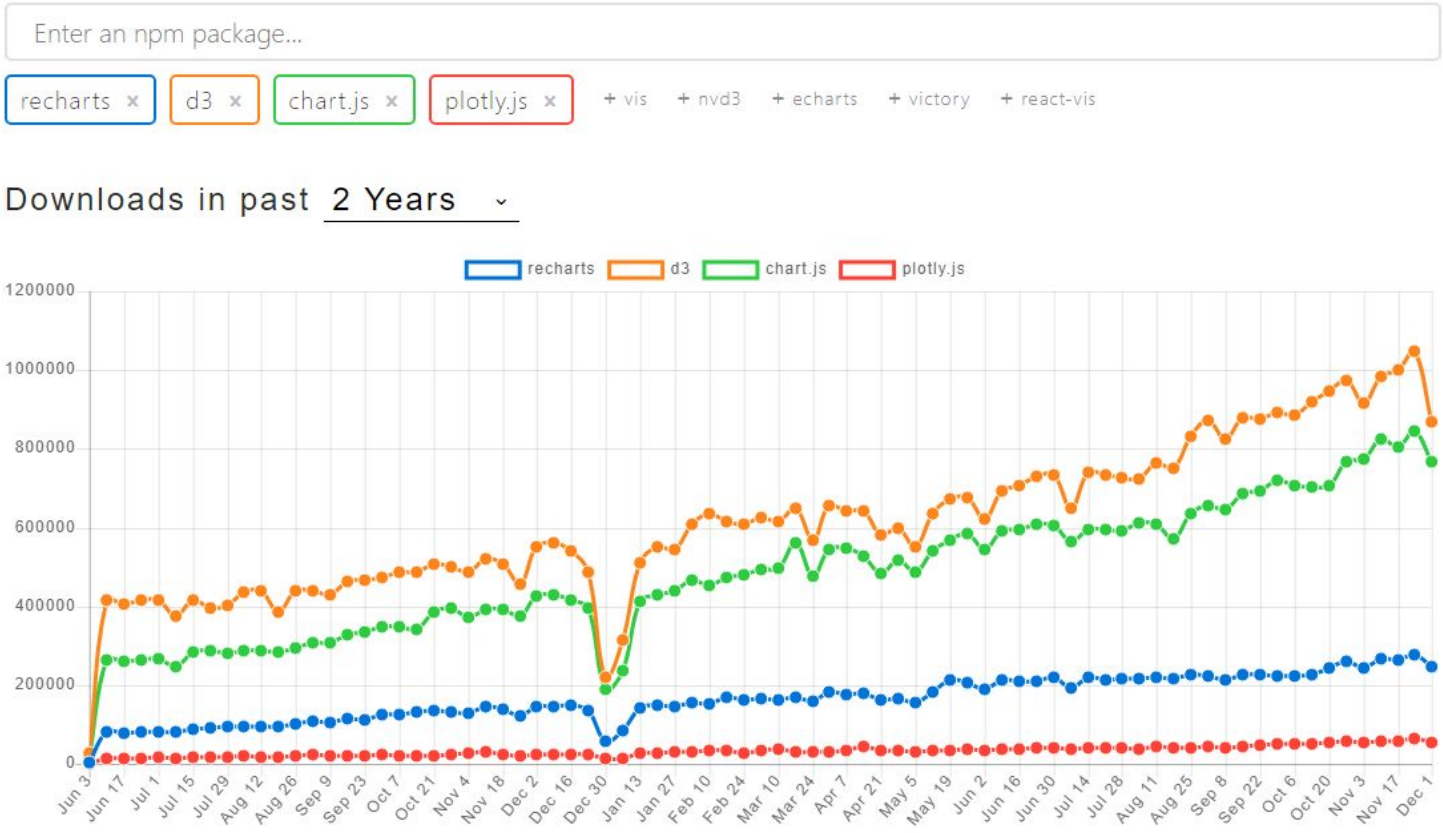
Features/ Approach	Parse SVG in D3	Fork D3 and modify internal code	Parse Smaller Elements in D3	Replace D3 with another library
Ease of detecting graph's properties	Possible	Unknown	Hard	Easy
Fragility	Medium	Unknown	Fragile	Not fragile
User's ease of use	Relatively easy	Very easy	Hard	Relatively easy
Time needed to invest in the project	A lot	Extreme	A lot	Reasonable
Probability pursing approach	Average	Low	Low	High

Final Approach: Use Chart.js instead of D3

- It appears that adding a runtime and static analysis checker might be
  - Fragile/inaccurate
    - Lots of false positive
- Reasons
  - It is hard to know the intent of what the user wants to visualize from the code or even the exposed data structure
    - This might make it hard/almost impossible to implement most of the rules
  - The variety and possibilities of visualizations that might be drawn via D3 is almost infinite
    - Lots of rules would be inconvenient for lots of cases
      - Example:

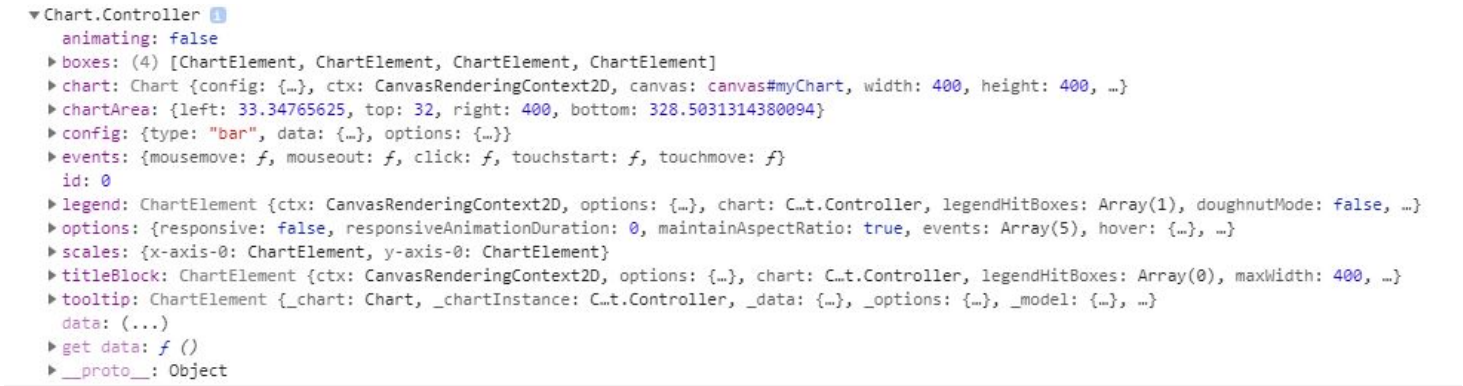
Why Chart.js?

- Easier data structure interface
- Most adopted after D3 (based on best of our knowledge)


Enter an npm package...
recharts ×
d3 ×
chart.js ×
plotly.js ×
+ vis
+ nvd3
+ echarts
+ victory
+ react-vis
Downloads in past 2 Years
recharts
d3
chart.js
plotly.js
1200000
1000000
800000
600000
400000
200000
0
Jun 3
Jun 17
Jul 1
Jul 15
Jul 29
Aug 12
Aug 26
Sep 9
Sep 23
Oct 7
Oct 21
Nov 4
Nov 18
Dec 2
Dec 16
Dec 30
Jan 13
Jan 27
Feb 10
Feb 24
Mar 10
Mar 24
Apr 7
Apr 21
May 5
May 19
Jun 2
Jun 16
Jun 30
Jul 14
Jul 28
Aug 11
Aug 25
Sep 8
Sep 22
Oct 6
Oct 20
Nov 3
Nov 17
Dec 1

ChartJS Simpler Data Structure

For example, we can directly know from the data structure that the chart is a “bar chart” the title is at the top of the chart



Comparison: applying rules of d3 and chart.js

Visualization Rules	Implementation Complexity in d3	Implementation Complexity in chart.js	Implemented
Don't plot more than 4 lines in one chart. If you need to display more, break them out into separate charts for better comparison.	Easy	Easy	√
Set the height of a line chart such that the data in the line chart takes up approximately two-thirds of the y-axis' maximum scale.	Hard	Easy	√
Use lines when connecting sequential data in time-series plots	Hard	Hard	×
Use solid lines only because dashed and dotted lines can be misleading.	Easy	Easy	√
Label the lines directly to enable readers identifying lines quickly using corresponding labels instead of referencing a legend.	Hard	Hard	×
Always include a Zero Baseline if possible.	Medium	Medium	√

Comparison: applying rules of d3 and chart.js

Visualization Rules	Implementation Complexity in d3	Implementation Complexity in chart.js	Implemented
Visual displays of information should present both cause and effect.	Impossible	Impossible	✗
Good visualizations should maximize data-ink ratio.	Hard	Hard	✗
Explaining the data helps viewers see the relevance in the information.	Impossible	Impossible	✗
Don't make users do "visual math".	Impossible	Impossible	✗
Change the layout for the graphical design to improve the readability of the design.	Hard	Hard	✗
Create an approximation of the overall structure but reduce the complexity so that it may be easier to comprehend.	Hard	Hard	✗

Comparison: applying rules of d3 and chart.js

Visualization Rules	Implementation Complexity in D3	Implementation Complexity in Chart.js	Implemented
Use horizontal labels. Avoid steep diagonal or vertical type, as it can be difficult to read.	Medium	Medium	√
Error bars considered harmful.	Hard	Hard	×
Use consistent colors. Use one color for bar charts. You may use an accent color to highlight a significant data point.	Medium	Easy	√
Don't use 3D effects either, especially in bar graphs. By making the bars look like cubes, the tops become obscured and it is difficult to discern where the top of the data really ends.	Hard	Hard	×
Order data appropriately. Order categories alphabetically, sequentially, or by value.	Hard	Medium	√
Space bars appropriately. Space between bars should be ½ bar width.	Medium	Already implemented	√ for d3

Comparison: applying rules of d3 and chart.js

Visualization Rules	Implementation Complexity in d3	Implementation Complexity in chart.js	Implemented
Don't use more than 6 colours together.	Medium	Easy	√
Don't use [RED and GREEN] or [ORANGE and GREEN] to make comparison on the same chart.	Medium	Easy	×
Rainbow color Map is considered harmful.	Medium	Easy	×
Use colours judiciously to indicate relationships and choose colour palettes that facilitate the message conveyed in the figure.	Medium	Easy	×
Double-check your colors for the color blind.	Medium	Easy	√

Comparison: applying rules of d3 and chart.js

Visualization Rules	Implementation Complexity in d3	Implementation Complexity in chart.js	Implemented
Do not use blow-apart effects.	Hard	Hard	×
Allow for direct interactions with objects that reveal new insights (e.g., sorting via drag).	Hard	Hard	×
Use interaction in visualization sparsely and cautiously.	Hard	Hard	×
Overview first, zoom and filter, details on demand.	Hard	Hard	×
Visualization should pass the Squint test. When you squint at your page, so that you cannot read any of the text, you should still 'get' something about the page.	Impossible	Impossible	×
All nouns should be encoded in black, verbs in red, adjectives in green, determiners in grey, particles in brown, conjunctions in blue, and Interjection in yellow.	Hard	Hard	×

Comparison: applying rules of d3 and chart.js

Visualization Rules	Implementation Complexity in d3	Implementation Complexity in chart.js	Implemented
Make sure all data adds up to 100%.	Medium	Medium	×
Visualize no more than 5 categories per pie chart.	Medium	Easy	√
Do not use multiple pie charts for comparison as slices sizes are very difficult to compare side-by-side.	Medium	Easy	×
Pie charts are bad and 3D pie charts are very bad.	Medium	Medium	×
Order slices correctly by first placing the largest section at 12 o'clock stretching clockwise, and then placing the remaining sections consistently either clockwise or counter-clockwise in descending order.	Hard	Easy	√

Comparison: applying rules of d3 and chart.js

Visualization Rules	Implementation Complexity in d3	Implementation Complexity in chart.js	Implemented
Do not cut y-axis in bar chart.	Medium	Easy	√
Do not cut y-axis in line chart	Medium	Easy	√

To be done next

- Fix the npm package for the chartjs-runtime-vis-linter
- Make warning messages more friendly and understandable
- If we would publish our work in a paper, we need to be more methodological on what rules to implement.
- Ability to turn on or turn off rules
- Categories (schools in DataVis), each with a batch of rules
- Implement more rules
- Add more resources for each rule
- Add tests for edge cases for the library

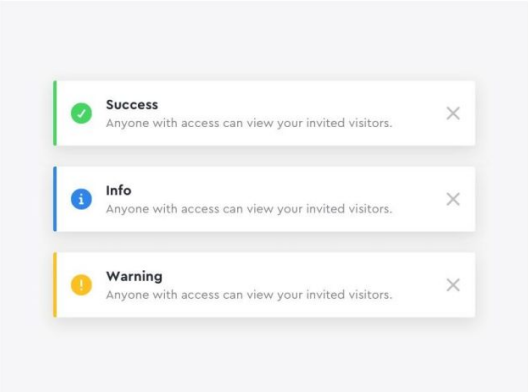
To be done next

- Implement auto-fix for static analysis rules
- For the runtime library, add the ability to do the check only during development for performance reasons

Future Advancements

Run-time library checker

- Unobtrusive toasts



Project Output

- Source code for the d3-eslint-plugin (static analysis)
- Source code for the chart.js runtime linter
- D3-eslint-plugin docs
- Chart.js runtime linter docs
- Chart.js runtime linter online demo
- D3-eslint-plugin npm package
- Chart.js runtime linter npm package