Refactoring Functional-style JavaScript Code

Implications on Performance and Readability

TECHNISCHE UNIVERSITÄT DARMSTADT

Program Testing and Analysis (WiSe17/18)

By:
Rohit Gowda (2404248)
Subhadeep Manna(2793470)

Guidance : Marija Selakovic
Supervising Prof: Michael Pradel

Motivation

Functional Style Vs **Iterative Style**

Functional Style (ForEach, Map, Reduce, Filter, Concat etc)

Pros : Code Readability and Maintainability

Cons: Poor Performance

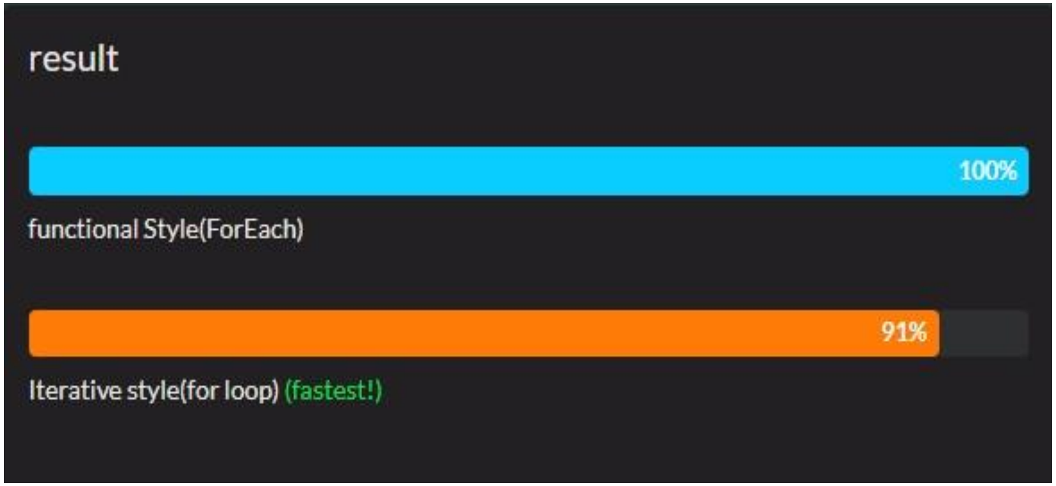


Fig 1:Simple Performance measurement at [http://jsperf.com (Functional](http://jsperf.com) vs Itertive Style)

When and *How* performance difference gets Significant ?

Our Goal

- Refactor Functional style Javascript into Iterative style. (*ForEach, Filter, Map and Reduce*)

- Analyse the Performance, Complexity, Readability and Maintainability of refactored code.

- Using selected Software Metrics(*Halstead*) and Tools.

Approach (Step 1: Static Analysis)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Generate an (Abstract Syntax Tree) AST from static code using JavaScript parsing Tools such as *Esprima*.
- Traverse the tree and look for Patterns which matches using a traversal library such as *Estraverse.*

```
var k = planetSize.map(function(x)
{
        return x+10;
});
```

esprima →

```
"init": {
        "type": "CallExpression",
          "callee": {
            "type": "MemberExpression",
            "computed": false,
            "object": {
              "type": "Identifier",
              "name": "planetSize"
            },
            "property": {
              "type": "Identifier",
              "name": "map"
            }
          },
```

Fig 2: AST Parsing using Esprima

Approach (Step 2: AST Manipulation)

- Collect the patterns based on the scope of each element.
- Manipulate the AST using built in functions like estraverse.replace().
- The manipulation changes the nodes to iterative pattern.

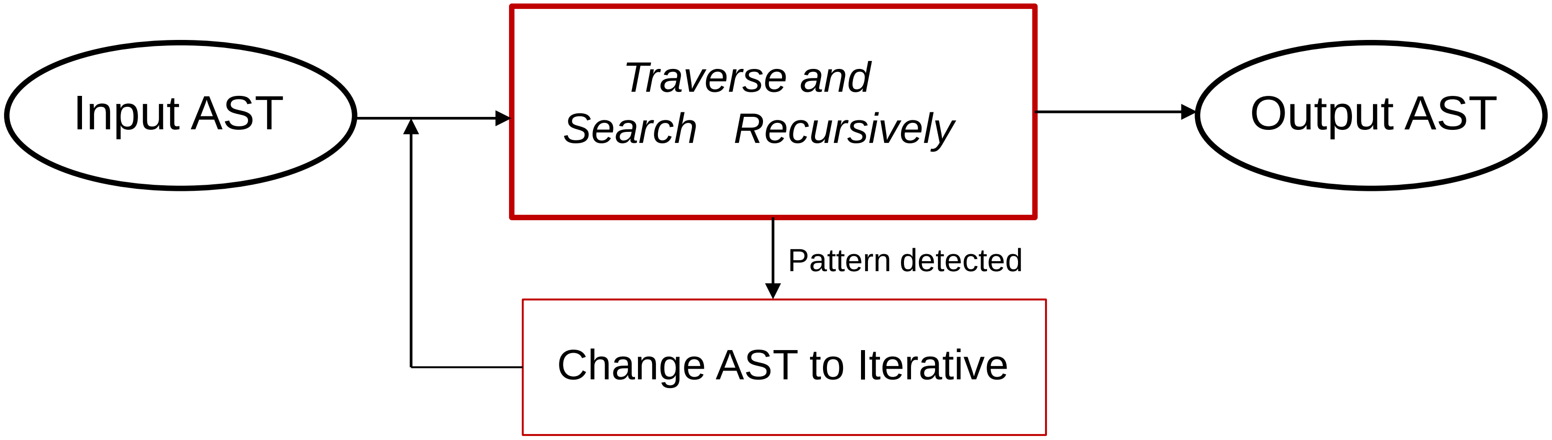


Fig 3: AST Manipulation Process

Approach (Step 3: Code Generation)

- Generate Code based on the manipulated AST using tools such as *escodegen*.
- Incase the AST manipulation is incorrect error is thrown.
- Important to take care of error Handling.

Example:

RefactoredCode = escodegen.generate(Changed AST)

Short-Summary of the Refactoring Tool 1

TECHNISCHE UNIVERSITÄT DARMSTADT

Part 1: Pattern Detection.

- *Traverse the tree recursively and return the patterns matching (forEach, Map, filter and Reduce)*

Code Snippet

```
if(node.callee)
        {
         if(node.callee.property)
           {
             typeOfOp=node.callee.property.name;
           }
         }

         if(typeOfOp==<operatorName>) {return obj;}
```

Short-Summary of the Refactoring Tool 2

TECHNISCHE UNIVERSITÄT DARMSTADT

Part 2: Data Store.

Store the scoping data using the extracted values from AST.

Format JSON:

```
{"parentSignature":parentSign,"bodySection":bodySection,
"arrayOperatedOn":arrayOperatedOn,
"argVariableName":argVariableName}
```

Part 3: AST changes and code generator

1. Using **predefined template** for each element type.
2. **Traverse** the tree for same pattern.
3. On match get the value from datastore and **substitute** it in AST iterative template.
4. Push the new Iterative section in appropriate position determined by scoping.

Tests and Results Verification

- Tested the refactoring tool against npm packages (*minimist, rimraf, opitimist, semver and loadash*). Extras: nanomist.

Methodology:

- Check the correctness of refactoring(Regression Testing)
- Using the testSuite provided with each package (also debugging and manual inspection).

```
test/big-numbers.js ................................ 11/11
test/clean.js ...................................... 12/12
test/gtr.js ...................................... 141/141
test/index.js .................................. 1626/1626
test/ltr.js ...................................... 149/149
test/major-minor-patch.js .......................... 27/27
test/prerelease.js ................................... 9/9
total .......................................... 1975/1975

  1975 passing (4s)

  ok
-----------|----------|----------|----------|----------|----------------|
File       |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
-----------|----------|----------|----------|----------|----------------|
All files  |    97.67 |    95.39 |    96.15 |     98.2 |                |
 semver.js |    97.67 |    95.39 |    96.15 |     98.2 |... 487,488,546 |
-----------|----------|----------|----------|----------|----------------|
```

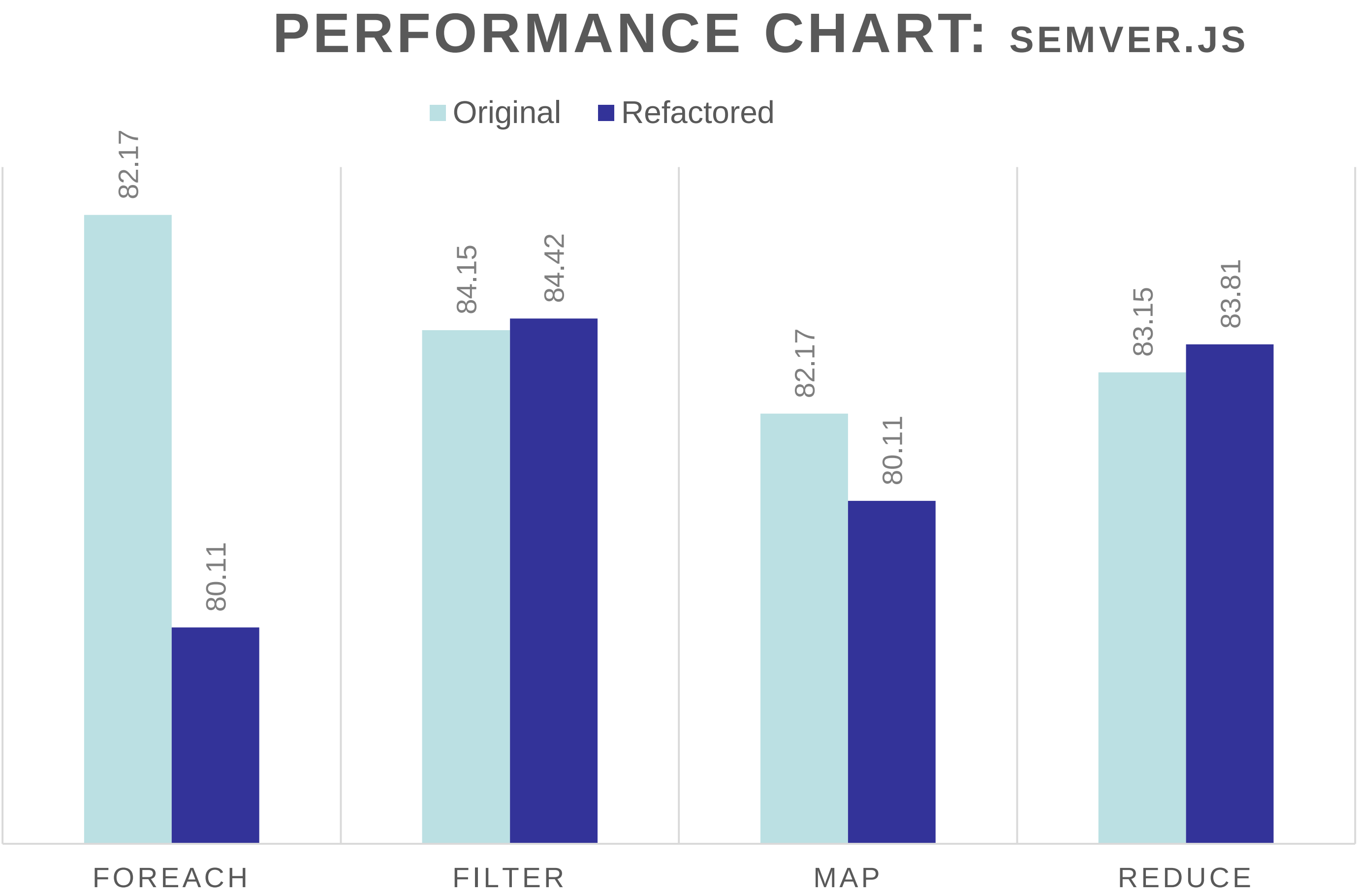
Fig 4: ForEach refactoring tests for semver.js.

Performance Analysis

- Verifying individual test suites against the Refactored code and Original code for each of the packages.

- Calculating the avg. time taken to run (say10) iterations of testSuite over each of the packages.(*Synchronously*)

Performance Analysis



Time in seconds

Code Metrics

Cyclomatic complexity:

- Number of distinct(independent) path taken in the code.
- *Lower is better*.
- Estimated using *escomplex* library.

Refactored semver.js vs Original semver.js

***Usually Refactored code has a higher cyclomatic complexity*.**

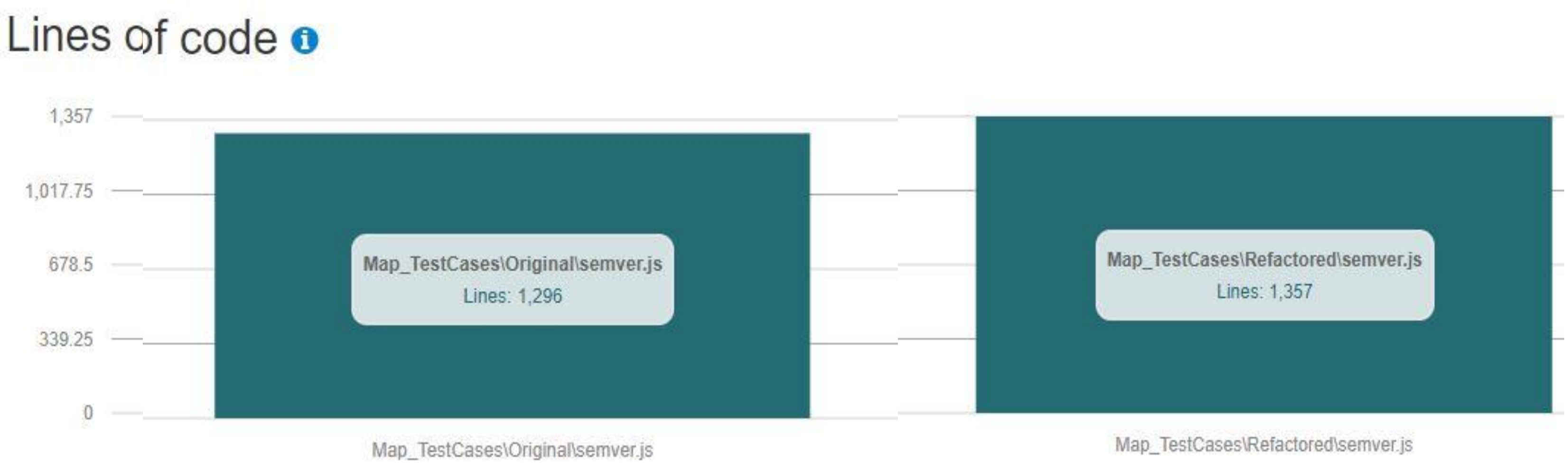
Fig 5: Map code metrics for semver.js.

Readability and Maintainability and LOC

Maintainability

- Calculated from lines of code(LOC) and cyclomatic complexity.
- Higher the value better is the maintainability.

Estimations using Escomplex and Plato



Conclusions

- Performance gain from refactoring depends on number of functional styles converted to iterative style.
- Refactoring to iterative style usually causes lower maintainability index and higher cyclomatic complexity.
- Performance gain is also dependent on the size of data (array, collection etc).

References

- Marija Selakovic and Michael Pradel. Performance issues and optimizations in javascript: An empirical study. *In Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 61–72, New York, NY, USA, 2016. ACM.

- https://github.com/jquery/esprima
- https://github.com/estools/estraverse
- https://github.com/estools/escodegen
- https://github.com/escomplex/escomplex
- https://github.com/es-analysis/plato
- https://www.npmjs.com/package/performance-benchmarking
- https://github.com/substack/minimist
- https://github.com/substack/node-optimist
- https://github.com/npm/node-semver
- https://github.com/isaacs/rimraf
- https://github.com/lodash/lodash
- http://jsoverson.github.io/plato/examples/jquery/

Q & A

THANK YOU!