Security Aware Visual Analysis of Software Systems

David Kavaler, Sugeerth Murugesan, Chetna Vig

Motivation

- Analyzing large software systems is expensive
  - 60%-90% maintenance

Motivation

- Analyzing large software systems is expensive
  - 60%-90% maintenance
- What makes it hard to understand?
  - Complexity
  - Size
  - Dynamic Evolution

Motivation

- Analyzing large software systems is expensive
  - 60%-90% maintenance
- What makes it hard to understand?
  - Complexity
  - Size
  - Dynamic Evolution
- Bugs make it even harder

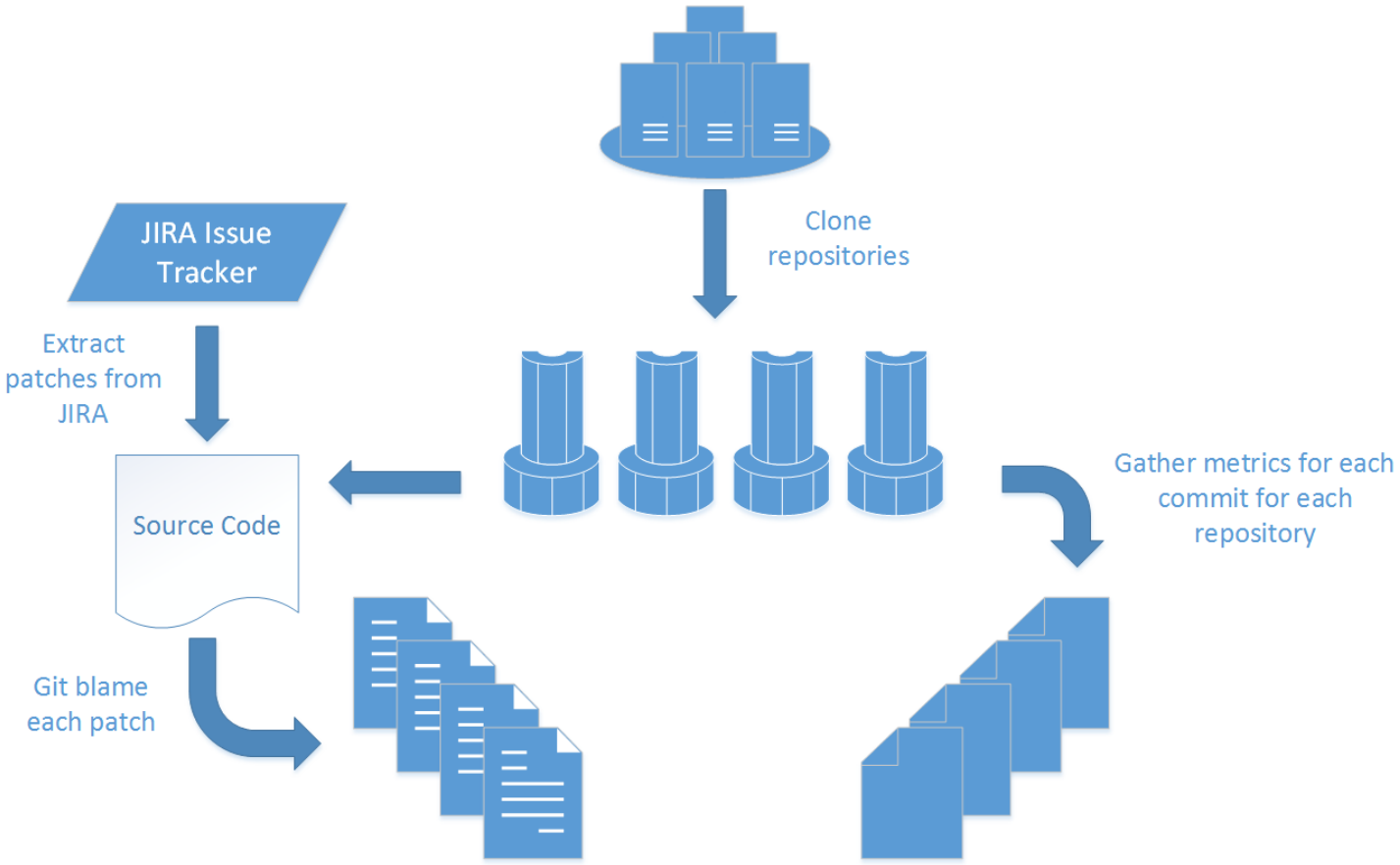
Motivation

- Analyzing large software systems is expensive
  - 60%-90% maintenance
- What makes it hard to understand?
  - Complexity
  - Size
  - Dynamic Evolution
- Bugs make it even harder

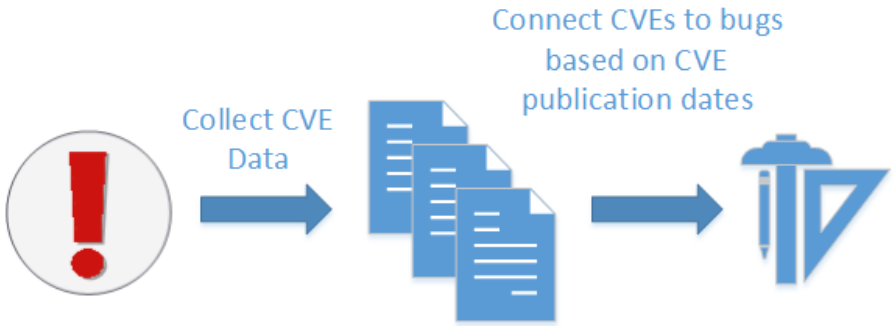
Visual analysis to the rescue:

- We aim to provide a visual analysis tool that can be used to identify fault-prone code using established metrics that can change value over time.

Data Gathering



Data Gathering



Selected Metrics

- Count metrics
  - Total count LOC
  - Count of executable LOC
  - Comment LOC
  - Churn (added and deleted LOC)
- Object-oriented feature metrics
  - Total number of functions
  - Total number of local public methods
  - Total number of local private methods
- Complexity
  - Ratio of comment lines to code lines
  - Sum of cyclomatic complexity
- Misc.
  - Authors and committers

Data Mining To Visualize Bug Probability

Motivation: To give probability of bugs in a project to the security analysts for ready reference.

Data Mining To Visualize Bug Probability

Objective: To predict bugs in a project by using Machine Learning tools

Data Mining To Visualize Bug Probability (contd.)

How?

Data Mining To Visualize Bug Probability (contd.)

By using various metrics related to a project such as:

- add_count
- delete_count
- countline_code
- countline_comment
- ratio_comment_to_code
- sum_cyclomatic etc.

Tools used

We used “Weka” as a data mining tool.

What is Weka?

Tools used (contd.)

Weka is a collection of machine learning algorithms for data mining tasks developed by The University of Waikato.

Tools used (contd.)

Methodology:

- Analysing all the metrics one by one
- Filtering out the most relevant metrics out of those
- Feature selection

Building a model for bug identification

We combined the data of two files “git_log_wmetrics” and “bug_data” to actually see them in a single file to train a classifier in weka.

Building a model for bug identification (contd.)

We ran different classifiers and found J-48 as the appropriate one so far for our model.

Results: Sum of Add and Delete Count

Test options

- Use training set
- Supplied test set — Set...
- Cross-validation — Folds 10
- Percentage split — % 66

More options...

(Nom) Buggy Commit

Start | Stop

Result list (right-click for options)

- 09:14:45 - trees.J48
- 09:15:10 - trees.J48
- 09:17:38 - trees.J48
- 09:17:50 - trees.J48
- 09:20:46 - trees.J48
- 09:20:54 - trees.J48
- 09:25:49 - trees.J48
- 09:30:40 - trees.J48
- 09:32:54 - trees.J48
- 09:37:23 - trees.J48
- 09:49:10 - trees.J48
- 12:22:13 - trees.J48
- 12:22:21 - trees.J48
- 12:22:31 - trees.J48
- 12:22:45 - trees.J48graft
- 12:22:54 - trees.J48
- 12:33:22 - trees.J48

Classifier output

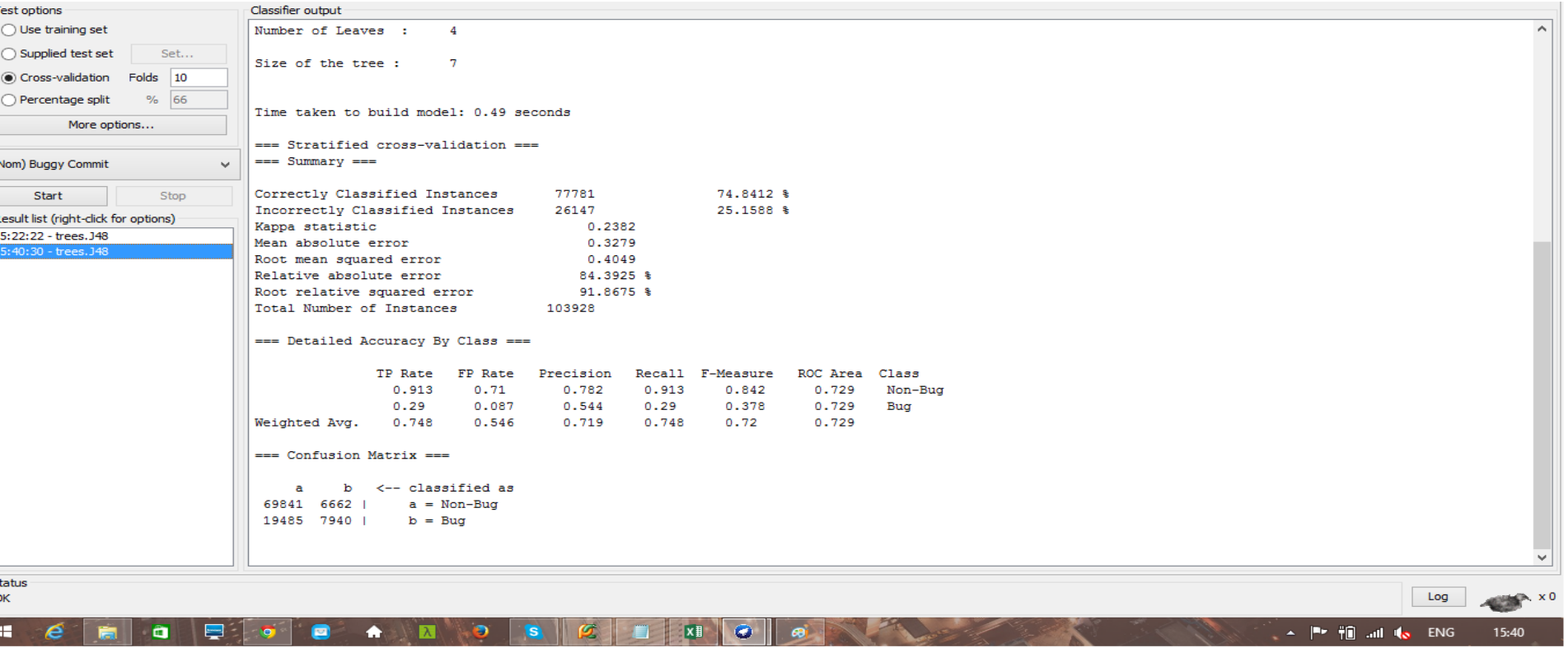
```
Number of Leaves  :     4

Size of the tree :      7


Time taken to build model: 0.75 seconds

=== Evaluation on test split ===
=== Summary ===

Correctly Classified Instances       26294               74.4114 %
Incorrectly Classified Instances      9042               25.5886 %
Kappa statistic                          0.2104
Mean absolute error                      0.3368
Root mean squared error                  0.4097
Relative absolute error                 86.6484 %
Root relative squared error             92.8753 %
Total Number of Instances            35336

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.92      0.745      0.775     0.92      0.841      0.715    Non-Bug
                 0.255     0.08       0.534     0.255     0.346      0.715    Bug
Weighted Avg.    0.744     0.569      0.711     0.744     0.71       0.715

=== Confusion Matrix ===

     a     b   <-- classified as
 23907  2082 |     a = Non-Bug
  6960  2387 |     b = Bug
```

Status
OK

Log x 0

ENG 12:34

Deviation of Add Count from Mean



Deviation of All features together

Test options

- Use training set
- Supplied test set — Set...
- Cross-validation — Folds 10
- Percentage split — % 66

More options...

Nom) Buggy Commit

Start | Stop

Result list (right-click for options)

- 5:22:22 - trees.J48
- 5:40:30 - trees.J48
- 5:42:48 - trees.J48
- 5:44:55 - trees.J48
- 5:47:52 - trees.J48
- 5:49:16 - trees.J48
- 5:49:45 - trees.J48
- 7:22:00 - trees.J48

Classifier output

```
Size of the tree :      1761


Time taken to build model: 13.62 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       79670               76.6588 %
Incorrectly Classified Instances     24258               23.3412 %
Kappa statistic                          0.3384
Mean absolute error                      0.3018
Root mean squared error                  0.3973
Relative absolute error                 77.6783 %
Root relative squared error             90.1355 %
Total Number of Instances           103928

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.893     0.585      0.81      0.893     0.849      0.769    Non-Bug
                 0.415     0.107      0.581     0.415     0.484      0.769    Bug
Weighted Avg.    0.767     0.459      0.749     0.767     0.753      0.769

=== Confusion Matrix ===

     a     b   <-- classified as
 68300  8203 |     a = Non-Bug
 16055 11370 |     b = Bug
```

Status
OK

Log x 0

ENG 17:25

Visualization Design

Visualization goal:

- Allow security analysts to visualize large software systems to facilitate detection of where:
  - bugs are introduced (or)
  - bugs can be predicted

.

Visualization Design

Visualization goal:

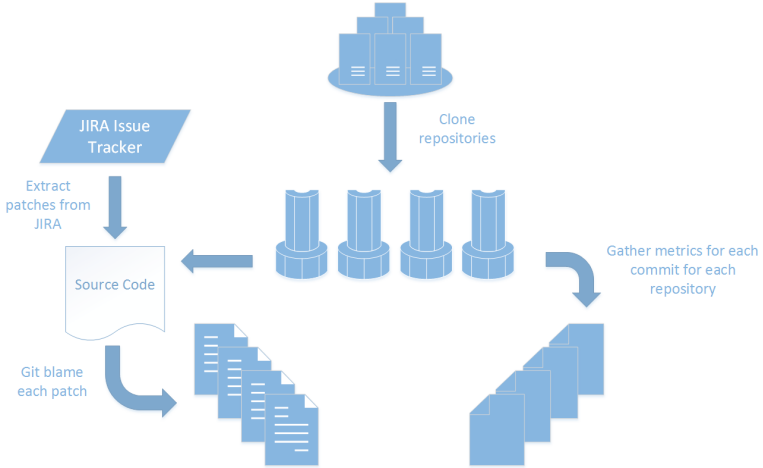
- Allow security analysts to visualize large software systems to facilitate detection of where:
  - bugs are introduced (or)
  - bugs can be predicted
- Large software comprehension for software maintenance

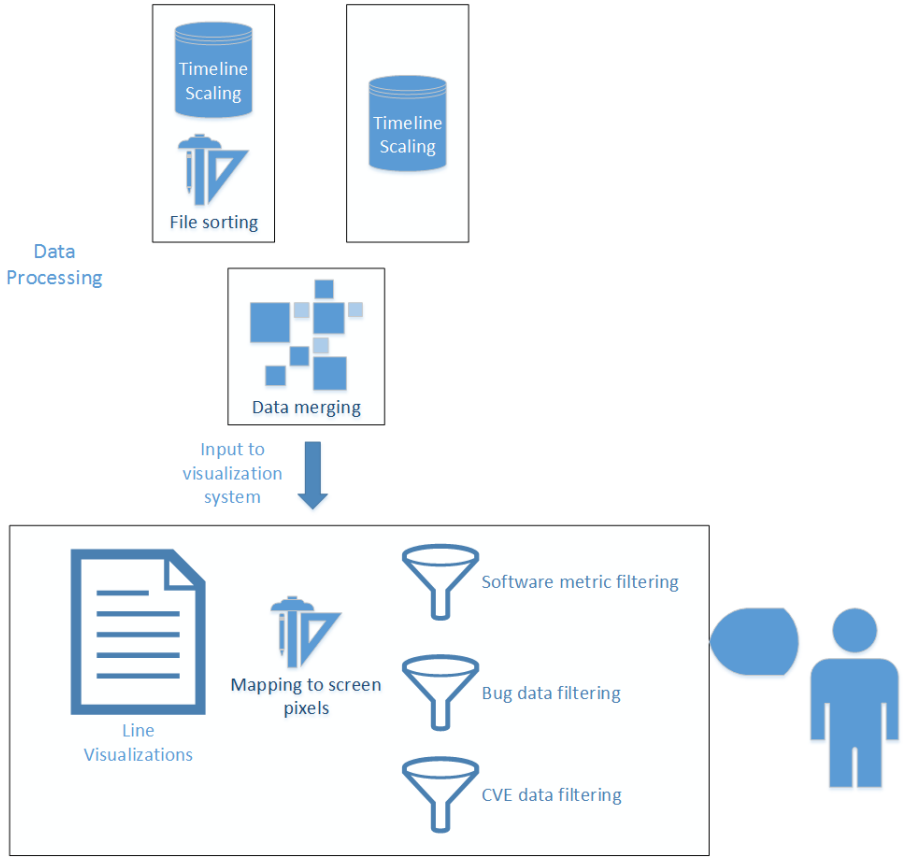
Visualization Design

Visualization goal:

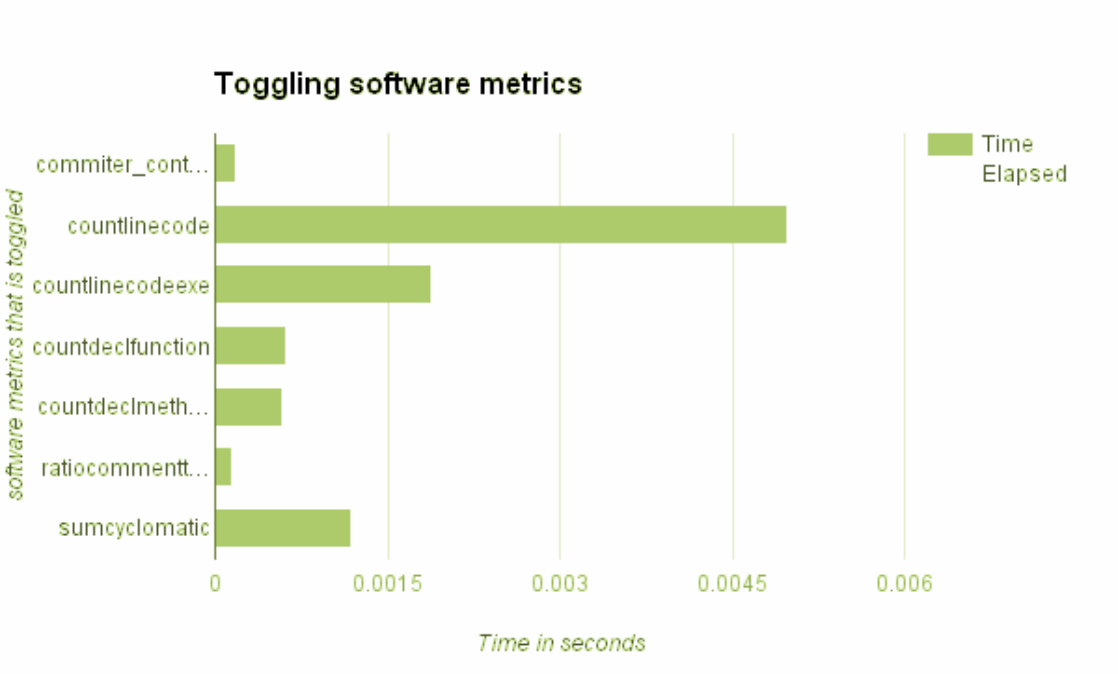
- Allow security analysts to visualize large software systems to facilitate detection of where:
  - bugs are introduced (or)
  - bugs can be predicted
- Large software comprehension for software maintenance
- Quality control tool to test and analyze software engineering hypothesis

Method

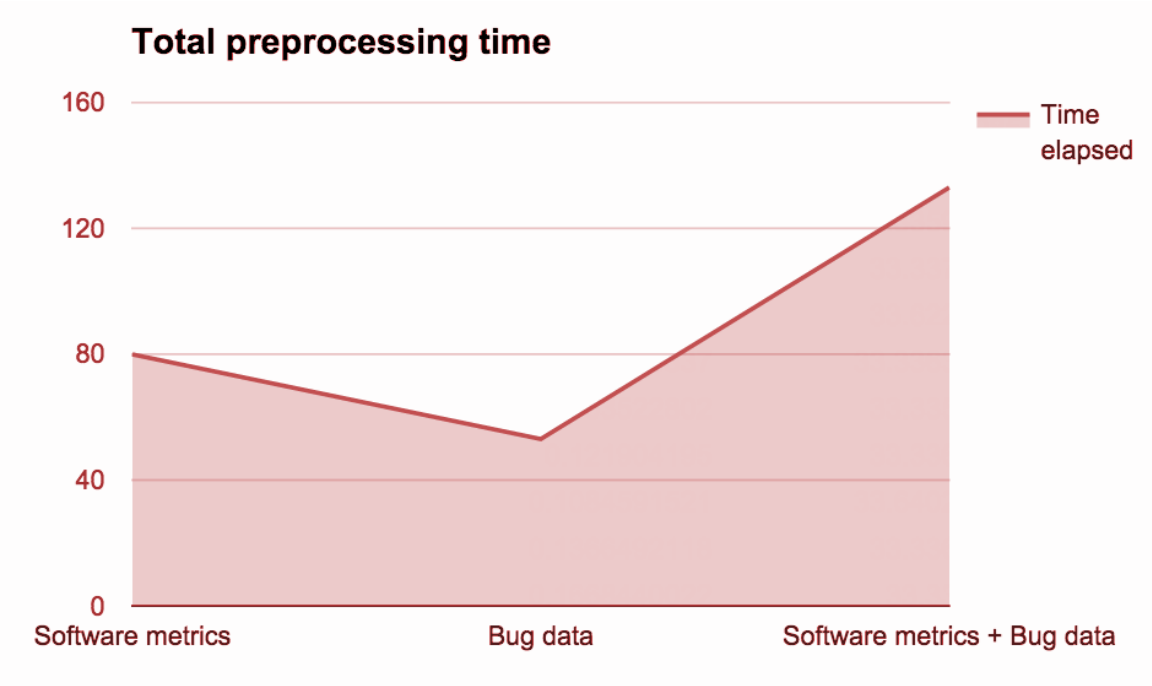

JIRA Issue Tracker
Clone repositories
Extract patches from JIRA
Source Code
Gather metrics for each commit for each repository
Git blame each patch

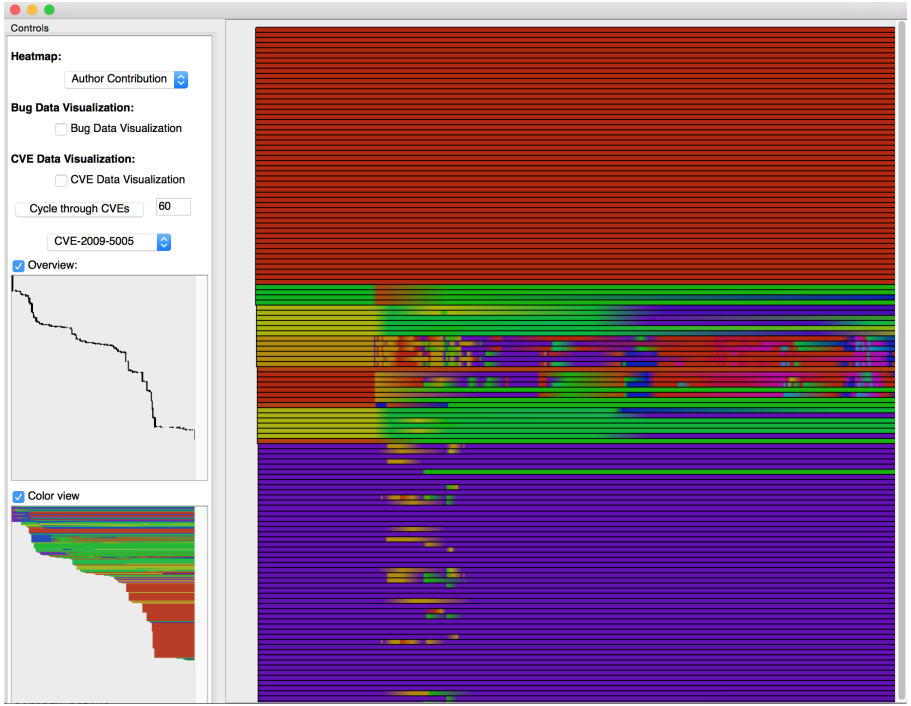

Timeline Scaling
File sorting
Timeline Scaling
Data Processing
Data merging
Input to visualization system
Visualization System
Line Visualizations
Mapping to screen pixels
Software metric filtering
Bug data filtering
CVE data filtering

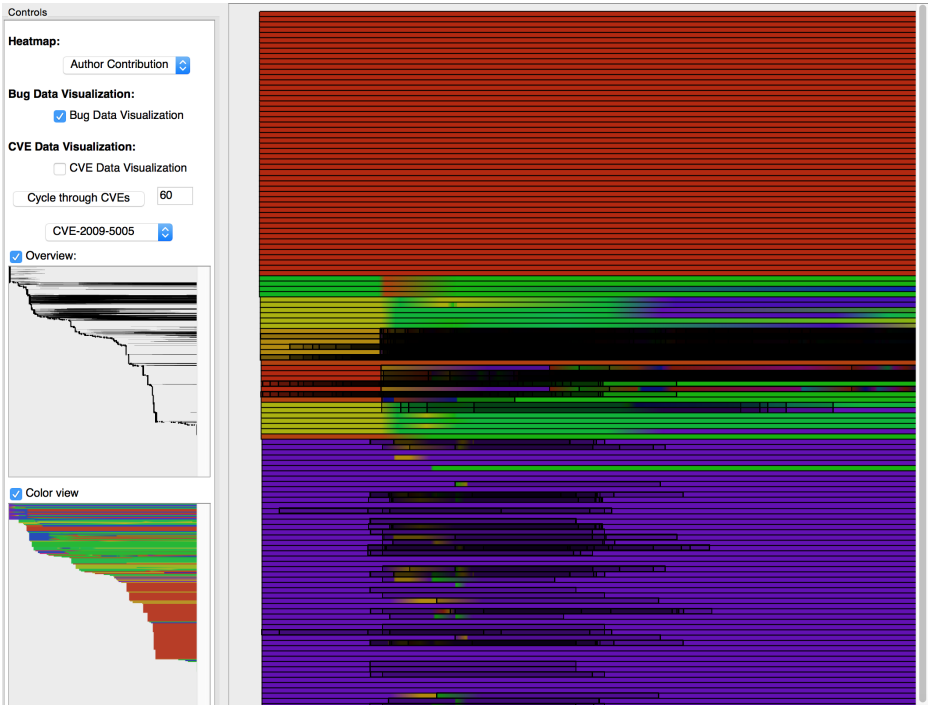
Performance Analysis



Performance Analysis




Controls
Heatmap:
Author Contribution
Bug Data Visualization:
Bug Data Visualization
CVE Data Visualization:
CVE Data Visualization
Cycle through CVEs
60
CVE-2009-5005
Overview:
Color view


Controls
Heatmap:
Author Contribution
Bug Data Visualization:
Bug Data Visualization
CVE Data Visualization:
CVE Data Visualization
Cycle through CVEs
60
CVE-2009-5005
Overview:
Color view

Controls

Heatmap:

Author Contribution

Bug Data Visualization:

Bug Data Visualization

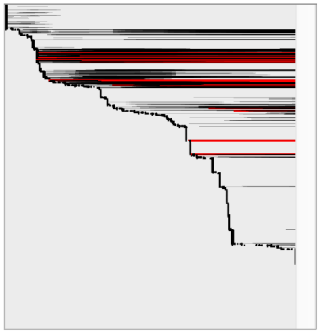
CVE Data Visualization:

CVE Data Visualization

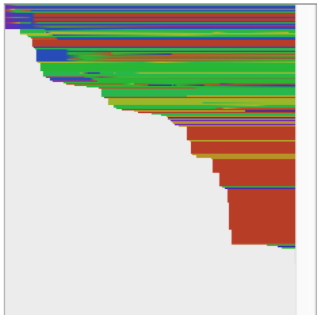
Cycle through CVEs 60

CVE-2009-5005

Overview:



Color view



Conclusion and Future Work

We present varying visualization and machine learning techniques that visualize the entire software source base along with bug and CVE data

Future Work:

- Visualizing bugs and metrics in the form of cities
- Adding a navigation box to the overview widget
- Incorporate prediction results into visualization tool
- Performance with overview widgets enabled

Demo Link: https://www.youtube.com/watch?v=eUnrEkfbQjw&feature=youtu.be