CLEMSON UNIVERSITY

Exploring Software Supply Chains from a Technical Debt Perspective

J. Yates Monteith

John D. McGregor

Strategic Software Engineering Research Group

Problem: Quality in the Software Supply Chain

- Due diligence requires that deliveries from suppliers be checked for acceptable quality.
- Software products are often subjected to acceptance test but these are superficial.
- One approach is to establish the reputation of the vendor.
- Another is to sample at high value targets.

Technical debt

- Many sources besides code
- We used SONAR in a standard configuration
- Need measures for non-code artifacts

Betweenness Centrality (BC)

- Ratio between the number of shortest paths a node lies in to the total number of shortest paths.
  - The node on the most shortest paths has the highest betweenness centrality.
- Graph theorists use this to identify nodes that are important to graph connectivity and information flow.

$$BC(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Experiments

- Sampled three versions of the Java Development Tools (JDT): 3.4, 3.5 and 3.6.
  - Maintenance builds.
- **Experiment 1:** Correlation test between technical debt and betweenness centrality.
- **Experiment 2:** Longitudinal Hotspot Evaluation.

BC-TD Correlation

- Measured betweenness centrality of each file in JDT 3.4, 3.5 and 3.6 using Cytoscape.
- Measured technical debt using Sonar Technical Debt plugin.
- Performed Pearson Correlation Coefficient test.

Version	Correlation Coefficient	One-tailed P Value
3.4	0.42765676	< 0.0001
3.5	0.42565911	< 0.0001
3.6	0.43607052	< 0.0001

Analysis

- Results were moderate, but significant correlation.
- There exists a positive relationship between technical debt and betweenness centrality.
- As one grows, the other grows, though not at the same rate.

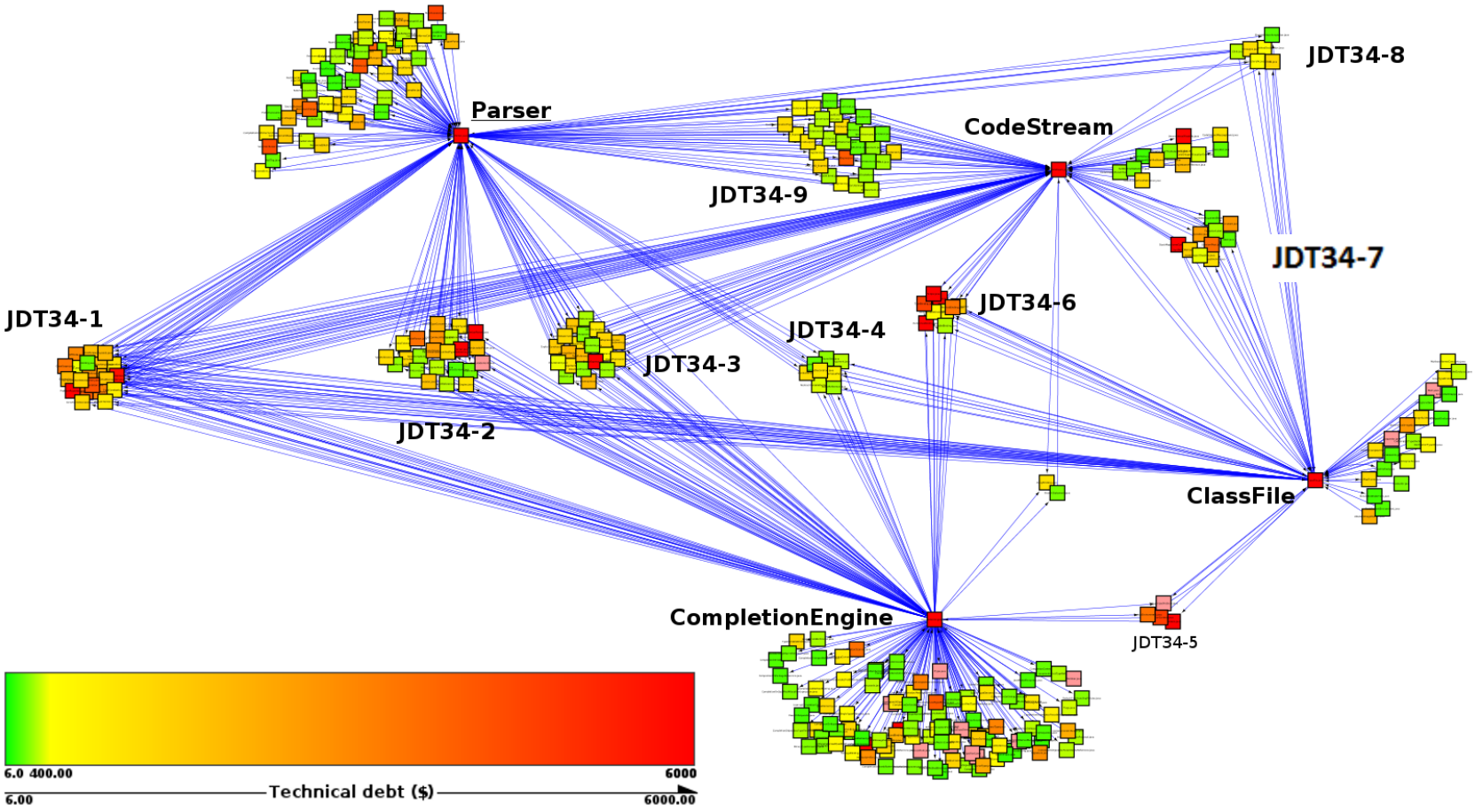
Longitudinal Evaluation

- Utilized same three maintenance versions of JDT: 3.4, 3.5 and 3.6.
- Generated dependency graphs for code structures using Cytoscape.
- Measured betweenness centrality.
  - i.e. Nodes that depended or were dependent on the four principle files.

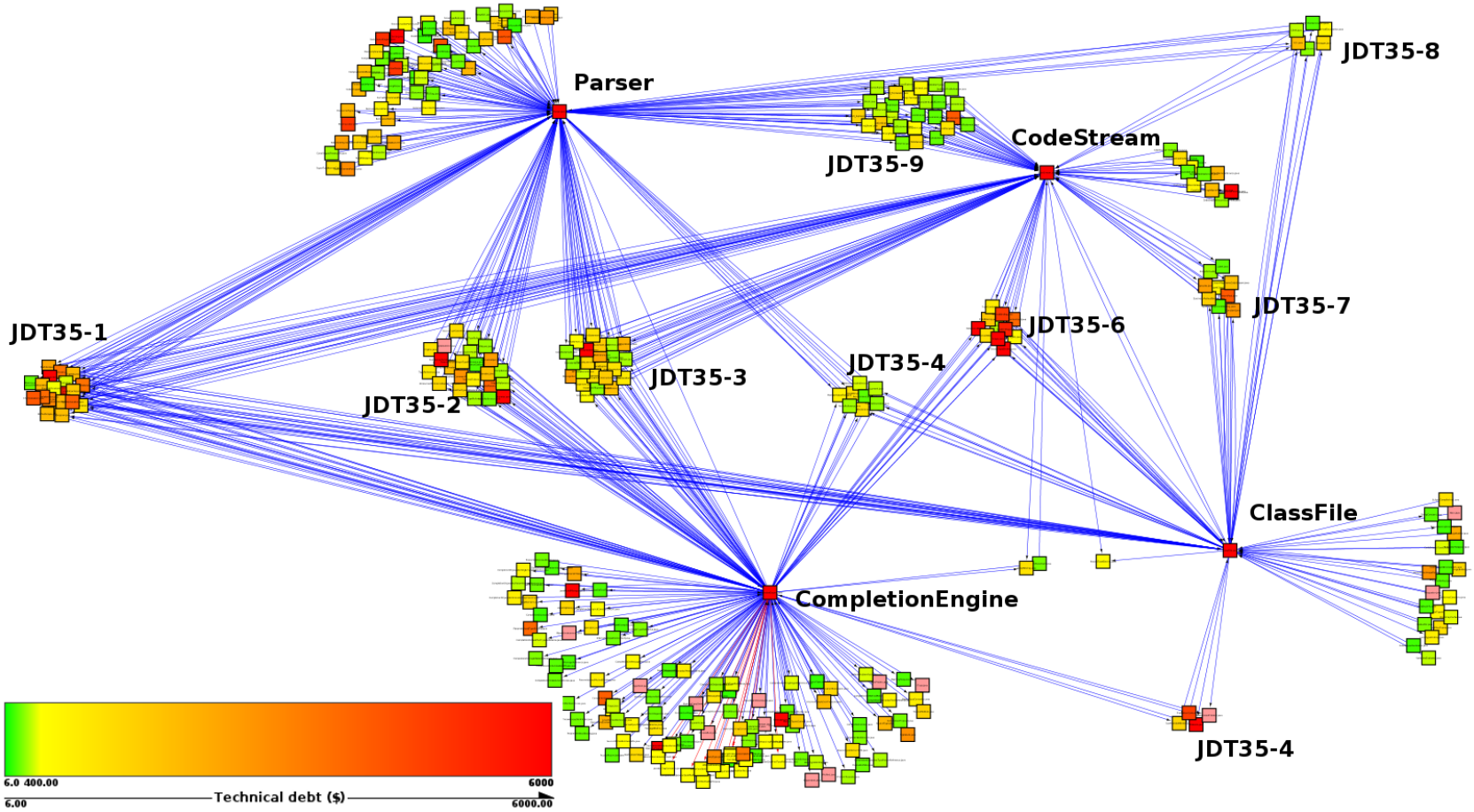
Longitudinal Evaluation

- Isolated four principle files via high technical debt: *ClassFile*, *Parser*, *CodeStream* and *CompletionEngine*.
- Reduced graph to four principle nodes and first neighbor nodes
- Performed the Force-Directed Spring-Embedded layout with weighting on betweenness centrality.
  - Nodes act as repulsive elements (think electrons).
  - Edge length determine by betweenness centrality.

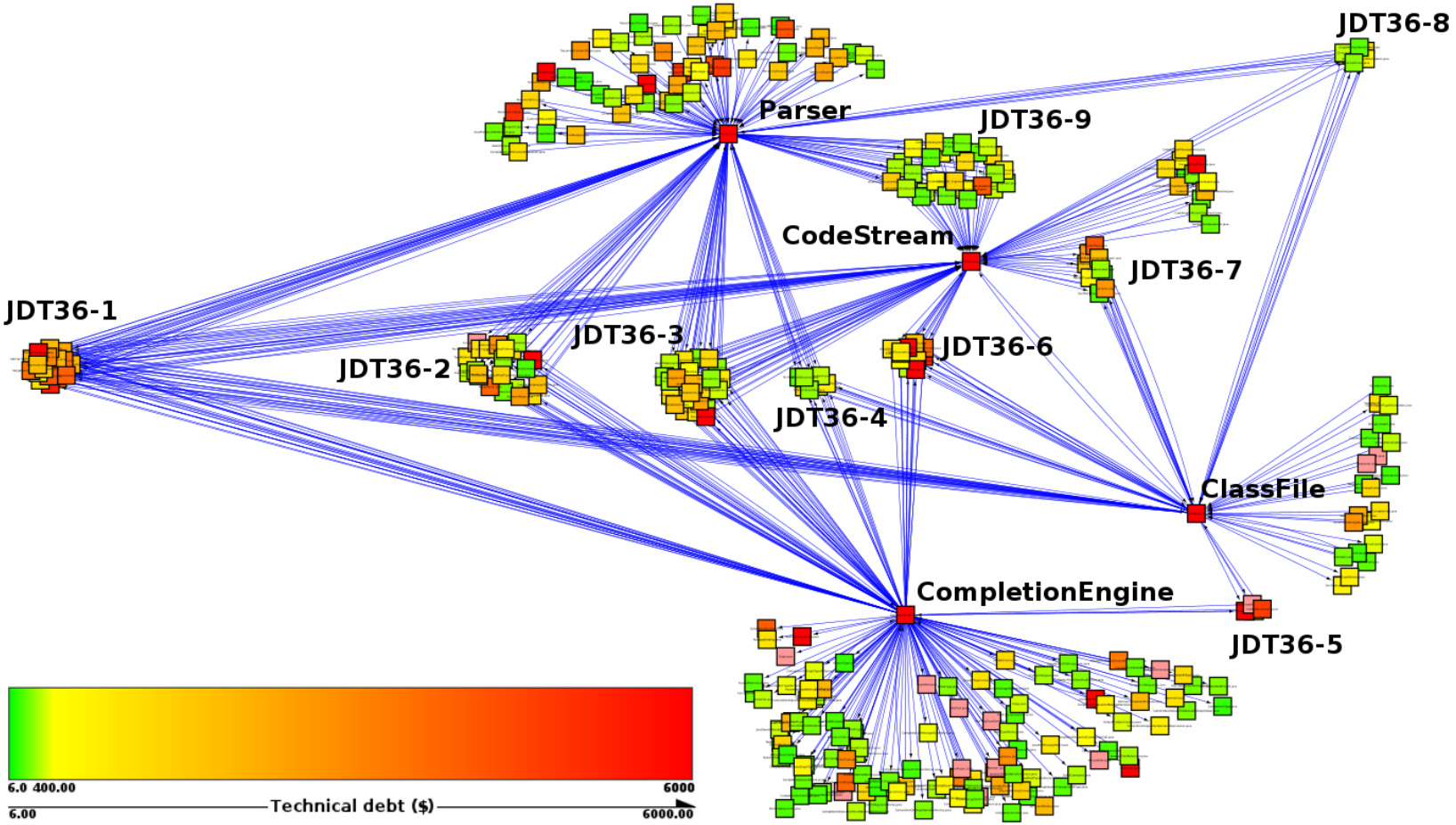
Eclipse – JDT 3.4



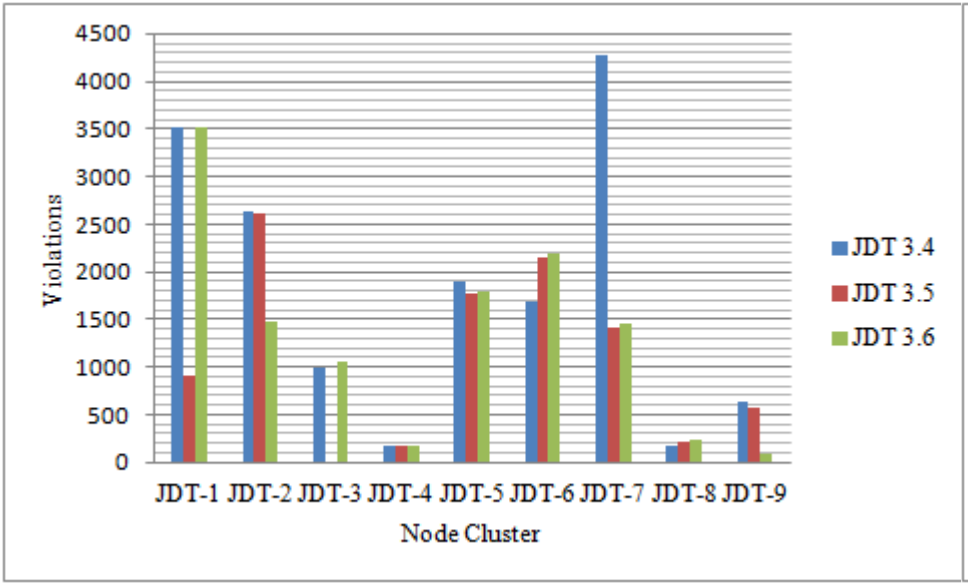
Eclipse – JDT 3.5

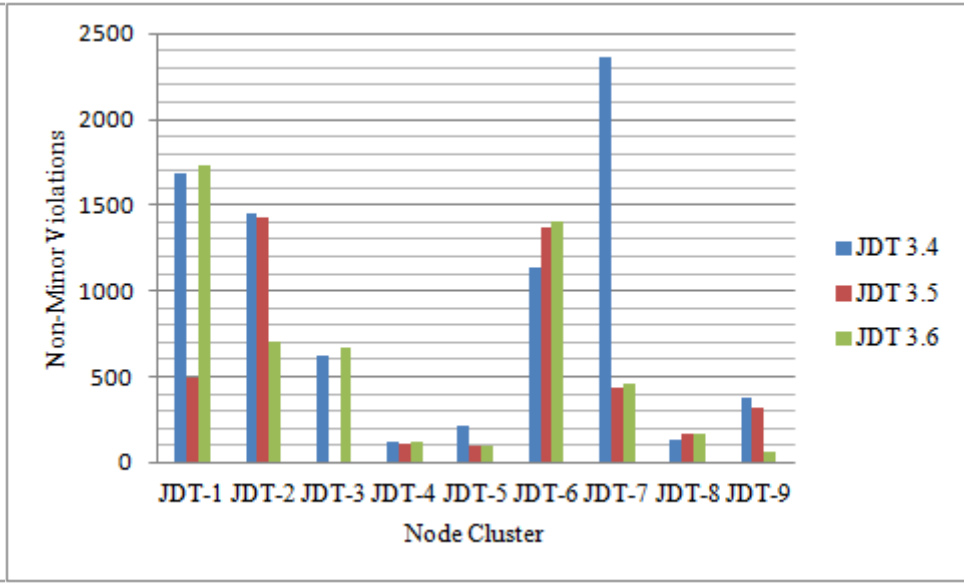


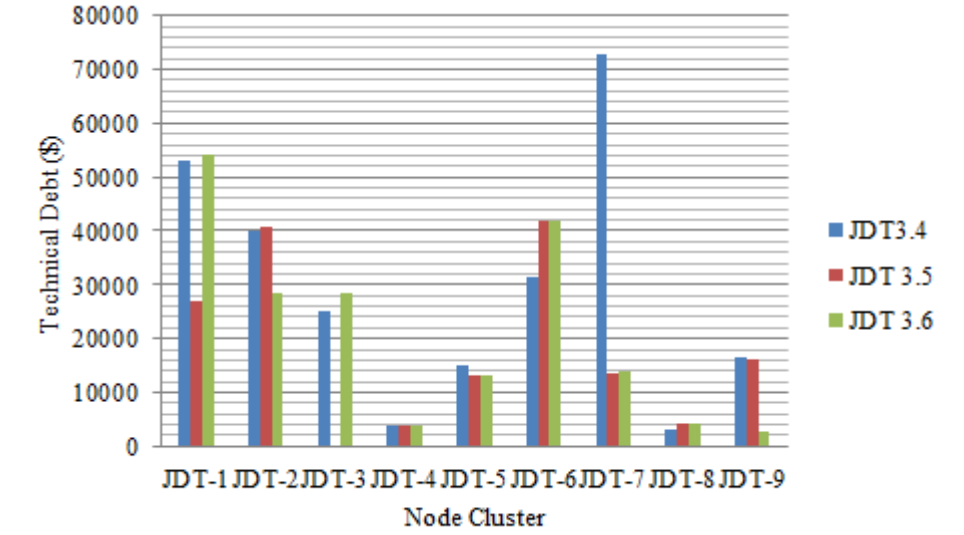
Eclipse – JDT 3.6

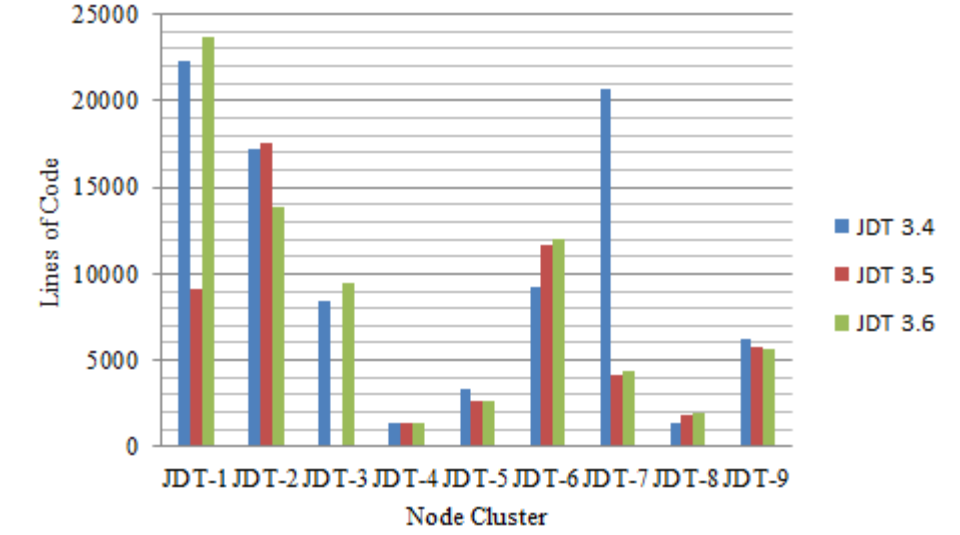


Coding standard violations









Analysis

- Examination of node clusters showed cluster 7 was an outlier: excessive technical debt, minor violations, non-minor violations and code size.
  - However, not the largest cluster in terms of lines of code.
- Analysis across versions showed significant refactoring of code, resulting in significantly reduced lines of code, violations and technical debt.
- Our technique consistently identified places where the professional staff spent time modifying design and code.

Conclusion

- Betweenness centrality has a positive relationship with technical debt.
- Using whichever of the two is easiest to compute we can identify regions of code that need renovation.
- We can also identify the vendors in an ecosystem that are best to use from a technical debt perspective.