Is your code Noodle-free?

Predicting Defect-prone Areas Using Noodlr

Team:
Vishal Chaudhary
Bastin Headmon Gomez

CS 846: Empirical Software Evolution - Winter 2016

Overview

- ★ Motivation
- ★ Introduction of Noodlr
- ★ Implementation of Noodlr
- ★ Terminology and Algorithm
- ★ Evaluation
- ★ Threat to Validity
- ★ Future Work
- ★ Conclusion

Motivation

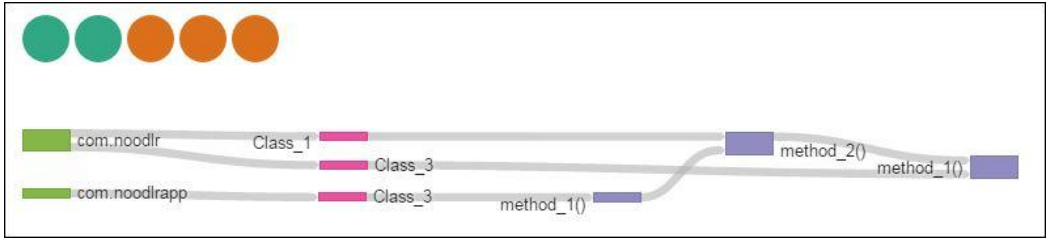
To improve the usage of dependency graphs:

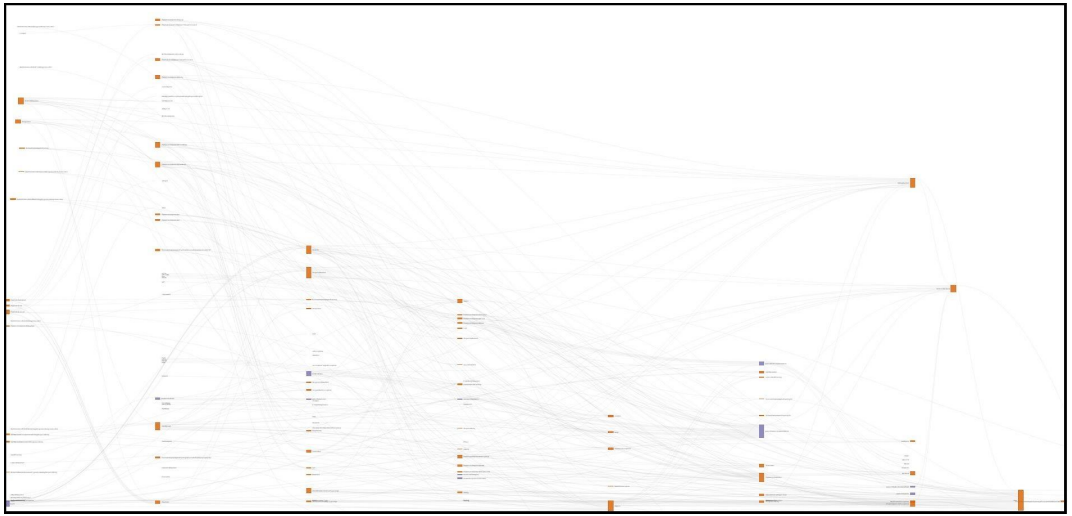
- ★ Detecting defect-prone areas by computing strongly connected components
- ★ Better allocation of resources in projects
- ★ Internal training purposes
- ★ Uses in various stages of Software development life cycle:
  - Development Stage
  - Maintenance Stage
  - Testing Stage

Tangled vs Untangled Noodles

vs

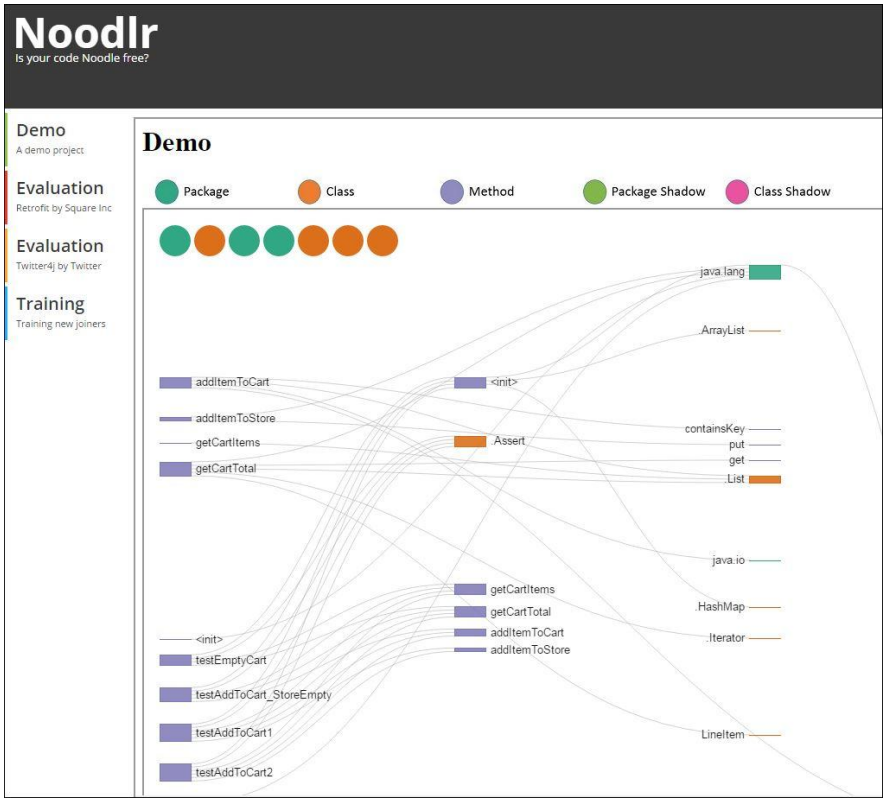
Tangled vs Untangled Graph





Noodlr:

- ★ A web based tool
- ★ Shows the call dependencies
- ★ Shows more defect-prone areas
- ★ Works with any java based project
- ★ Can be used in
  - Finding defect-prone areas & resource allocation
  - Training purposes
  - Various stages of SDLC (esp. Maintenance stage)



Implementation of Noodlr:

- ★ Backend
  - Implemented in Java
  - Input: Java based projects
  - Output: 2 Json files

- ★ Frontend
  - Web based UI with D3.js library for visualization
  - Input: 2 Json files
  - Output: Interactive dependency graph

Implementation(cont'd):

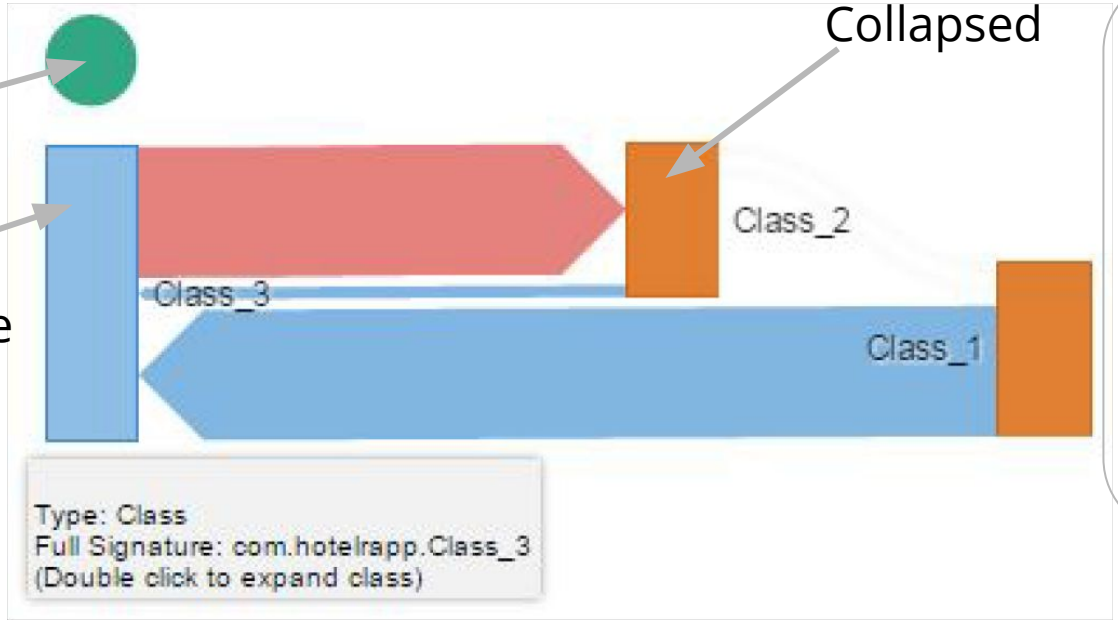
1st Json file format:

```
{
"type"      :   "Class",
"id"        :   "c1",
"parent"    :   "p1",
"name"      :   "Class_1",
"full_name" :   "p1.Class_1"
}
```

2nd Json file format:

```
{
"source"    :   "m1",
"target"    :   "m2",
"value"     :   "calls / depends on",
}
```

Implementation(cont'd):



Measures:

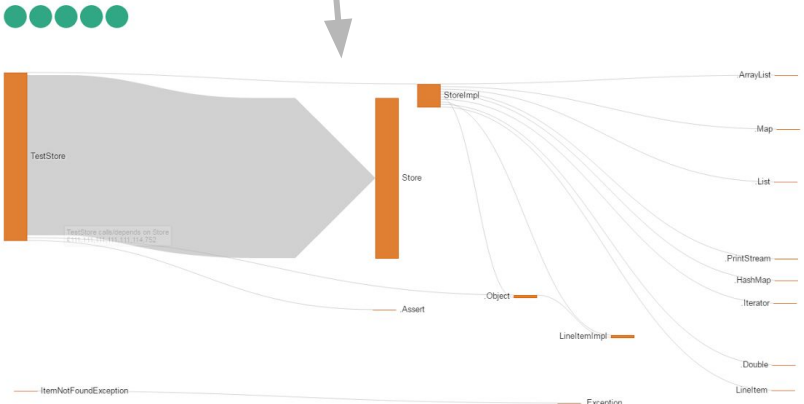
- Shape
- Color
- Height
- Arrows
- Hovered Text

Coloring Index:

Package Class Method Package Shadow Class Shadow

Implementation:

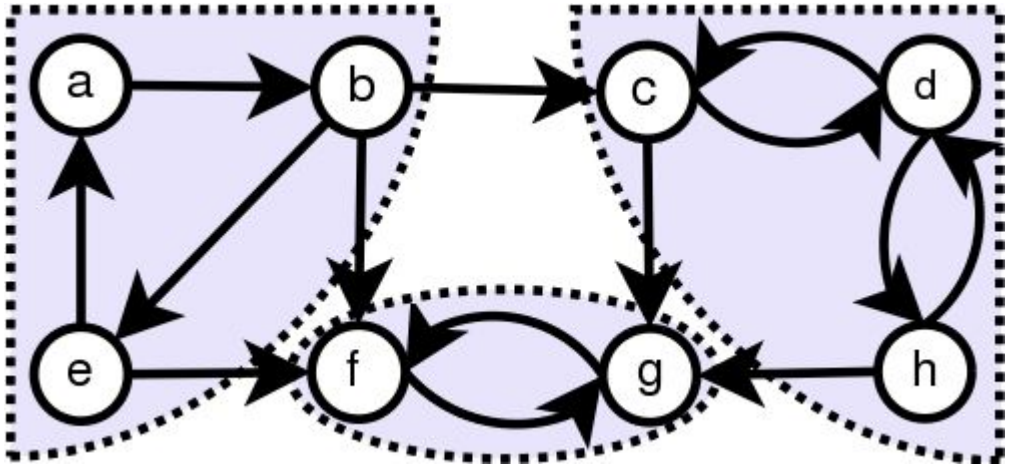
Package view (Fully collapsed)



Class view (Partially expanded)

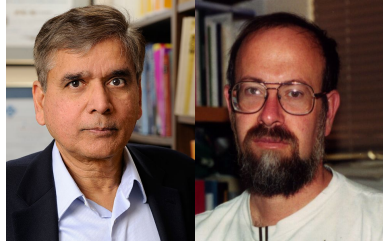
Method view (Fully expanded)

Strongly connected components (SCC)



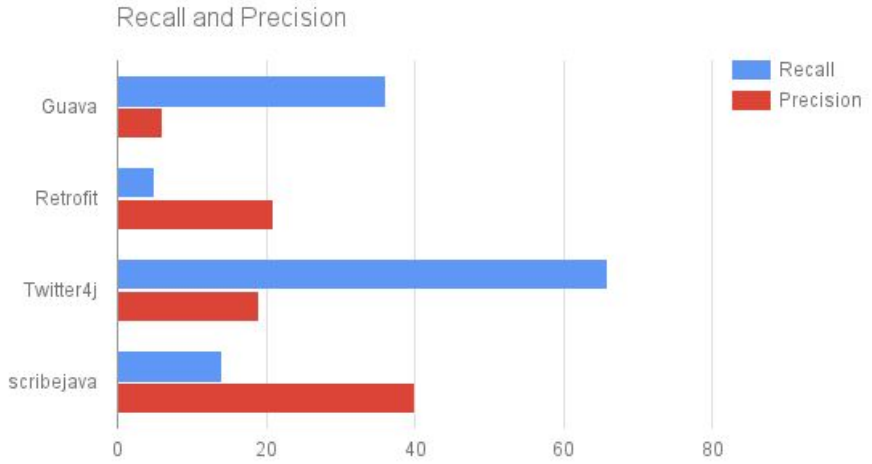
Application: In social network to find communities and their interests

Kosaraju-Sharir algorithm to find SCC



- ★ Two pass depth-first search on a directed graph (**G**)
- ★ First pass on the graph $\mathbf{G^{rev}}$ (direction of edges reversed) to find the ordering of nodes
- ★ Second pass on the original graph **G** based on the order obtained from the first pass

Evaluation



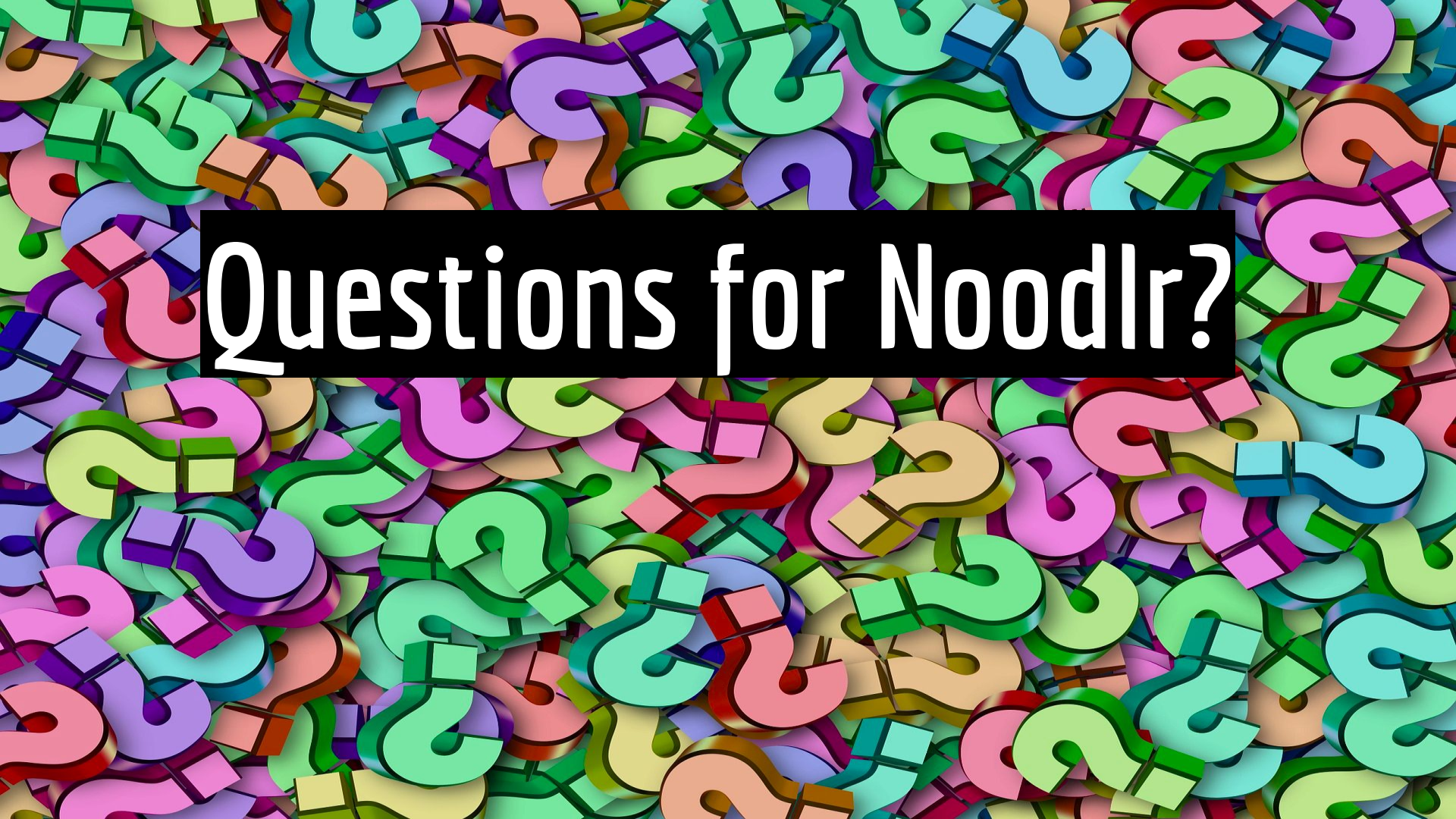
- ★ Four projects from Github
  - ○ Google Guava
  - ○ Square Retrofit
  - ○ Twitter4j
  - ○ Scribejava
- ★ Projects selected based on the programming language and no. of pull requests
- ★ Ran our tool to generate files using Kosaraju's algorithm to find SCCs
- ★ Retrieved files from the repo's pull requests which are related to issues
- ★ Compared them and computed precision and recall

Future work

- ★ To change to a different style of graph to support large projects
- ★ Search functionality
- ★ Support for projects with other programming languages
- ★ Integration as plugin to major IDE’s like Eclipse, Intellij or Netbeans

Conclusion

- ★ Knowing the structure of the project important for different reasons
  - Resource allocation
  - Internal training purpose
  - Tracking defect-prone areas
  - Track dependencies during development phase
  - Check impact of any change during maintenance phase
  - Helps in creating better test plans after knowing the different dependencies
- ★ Noodlr tool to help different functional levels
  - Managers, developers, testers etc.
- ★ Noodlr is web-based tool with a Java backend
  - D3.js used for graph visualization
  - Java used for input jar file processing and running required algorithms
- ★ Good scope for future work for Noodlr


Questions for Noodlr?