Can We Avoid High Coupling?

Craig Taube-Schock, Robert J. Walker, and Ian H. Witten

ECOOP 2011

Introduction

- We have long heard the maxim of “high cohesion/low coupling" in software design.

- Unfortunately, while software can be poorly created with definitely excessive coupling, it is not immediately clear whether high coupling can be definitively eliminated in all circumstances.

Scale-Free Networks

- A scale-free network is one that has a power-law degree distribution, characterized by having a majority of nodes involved in few connections and a few nodes involved in many connections.

- This set of distributions are known as heavy-tailed

- The presence of any heavy-tailed distribution describing the degree counts in the connectivity network means that there must be nodes that are highly coupled, and even nodes that are very highly coupled, relative to the mean level for each system.

Contributions

1. A demonstration that overall connectivity follows a heavy-tailed distribution across the spectrum of granularity for a large number of open-source systems -- regardless of maturity, degree of active support, and level of use;

2. A demonstration that between-module connectivity ubiquitously follows a heavy-tailed distribution -- and thus highly-coupled nodes are ubiquitous;

3. An explanatory model as to why having some areas of high coupling is consistent with good software design practices.

Models leading to scale-free

- To offer an explanation as to how a power-law could develop, Barabasi and Albert considered the evolution of complex networks as they increased in size and noted that the preferential attachment model caused scale-free structure to emerge.

- In this model, newly added nodes preferentially attach to nodes that have been in the network the longest time,

Explaining scale-free structures

- To address this question, consider inlinks (inbound connections) and outlinks (outbound connections) for all source code entities.

- In general, outlinking is constrained to be reasonably small.
- Examples:
  - Class declarations have few direct superclasses and directly implement few interfaces.
  - Individual statements tend to be limited to the number of variable access or method invocations due to practical issues, such as style guidelines and the difficulty of reading statements that extend beyond a programmer's screen width.

- For these reasons, high connectivity is largely due to inlinks.
  - A source code entity that exhibits high connectivity is thus likely to do so because of its utilization in multiple contexts.

Explaining scale-free structures

- Consider a source code entity e such that deg(e) > dmax and for which the number of connections is largely due to inlinking.

- To replace e by two or more entities (ei) in an attempt to satisfy deg(ei) <= dmax would require that all the replacement nodes ei provide the same utility as e.

- This suggests the introduction of code clones, which is considered to be poor design. The ability to reuse source code entities suggests that an arbitrarily low dmax cannot be practically achieved.

Empirical Study

- The empirical data comes from the Qualitas Corpus, a collection of 100 open-source software systems written in the Java

- Source code entities include modules, classes, method declarations, blocks, statements, and variables; each is modelled as a node within a directed graph.

Qualitas Corpus

#	Name/Version	Nod	Cnx	Mod	Cls	Mth	Blk	Sta	Var
1	derby-10.1.1.0	318831	809952	135	1805	25067	56357	160555	74910
2	gt2-2.2-rc3	256838	651522	219	3453	26347	52556	106738	67523
3	weka-3.5.8	248704	682151	91	2019	19169	47561	124152	55710
4	jtopen-4.9	230394	593240	18	1940	20559	42206	112259	53410
5	tomcat-5.5.17	177249	433523	149	1777	17152	36247	80214	41708
6	compiere-250d	155379	388859	43	1260	18128	25458	73472	37016
92	jmoney-0.4.4	6310	17618	6	193	713	996	2989	1411
93	nekohtml-0.9.5	6606	17153	7	54	422	1453	2887	1781
94	jchempaint-2.0.12	5757	15844	8	125	419	1146	2696	1361
95	jasml-0.10	5482	15419	8	53	256	895	3011	1257
96	fitjava-1.1	3862	10296	5	96	462	786	1564	947
97	picocontainer-1.3	3771	9117	5	99	540	842	1155	1128

Table 1. Structural measures of the systems that were examined. "Nod" = nodes; "Cnx" = connections; "Mod" = modules; "Cls" = classes; "Mth" = methods; "Blk" = blocks; "Sta" = statements; "Var" = variables.

Graph-based source code representation

- The basis of our analysis is a directed graph representation of source code, where
  - Nodes represent source code entities (packages, classes, methods, blocks, statements, and variables),
  - And links (directed arcs) represent connections between entities (hierarchical containment, method invocation, superclass, implementation, type, variable usage, and method overriding).

Meta-model

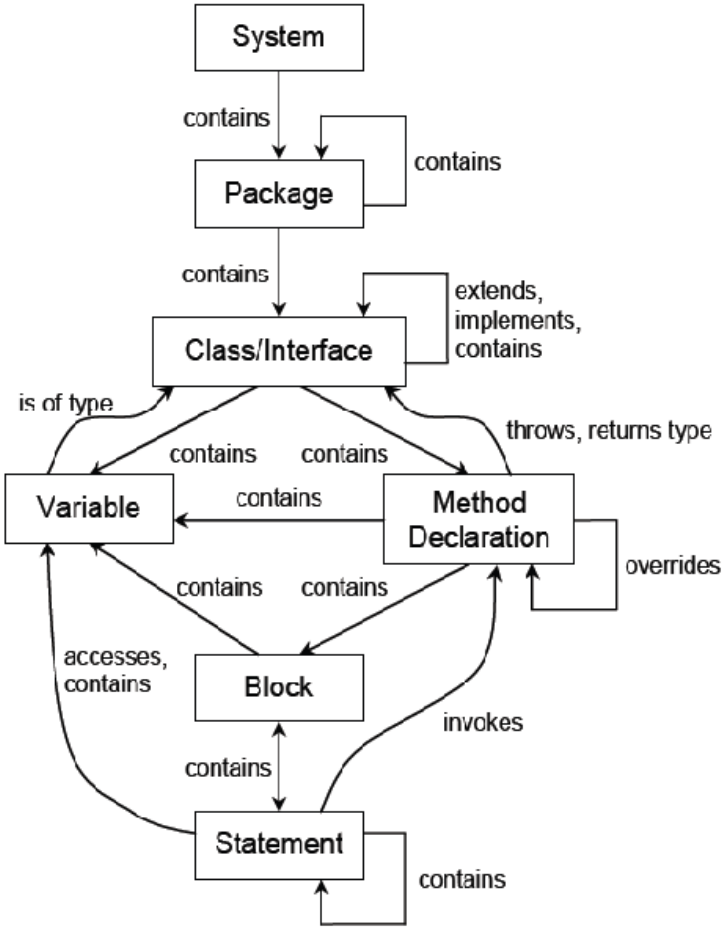


Fig. 2. Metamodel applied in our analysis.

Example

```
package p1;
public class X {
    private Y var1;
    private int var2;
    public int m1() {
        int temp;
        temp = var1.getValue() + getVar2();
        return temp;
    }
    private int getVar2() {
        return var2;
    }
}

package p2;
public class Y {
    public int getValue() { ... }
}
```

Fig. 3. Sample Java source code listing.

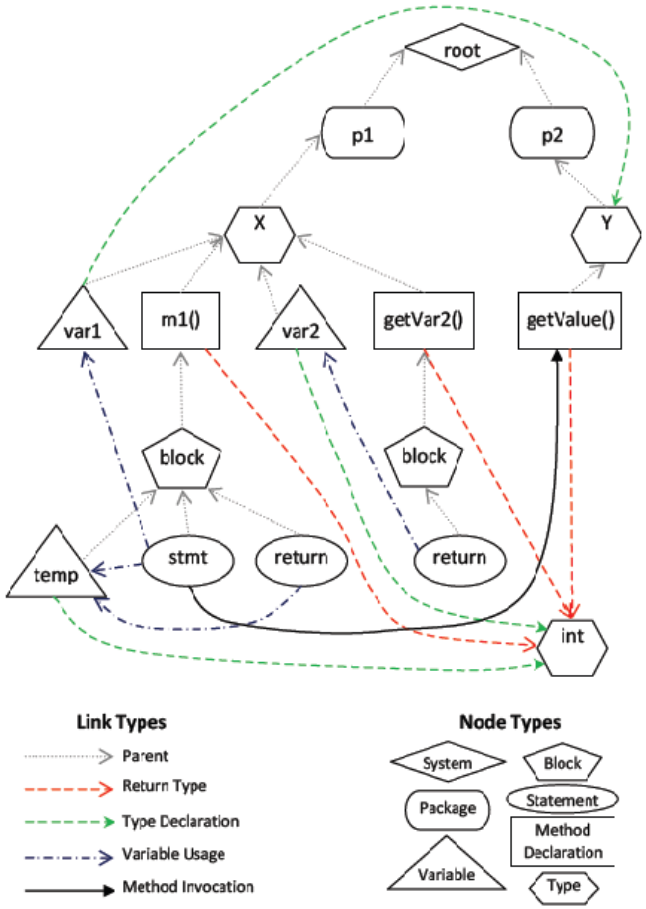


Fig. 4. Directed graph of the sample source code.

Within-module vs between-module links

- We consider class to be the defining aspect of modular boundaries;
- For this reason, hierarchical links are used to identify structural boundaries that are relevant to the analysis but are not included as part of the computation of degree distributions.

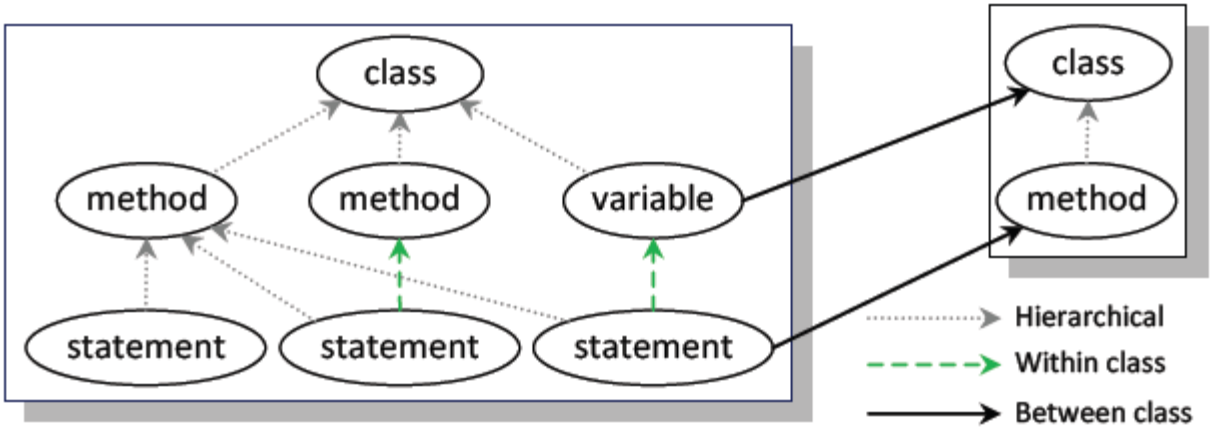


Fig. 5. Identifying within-module and between-module dependency.

Analysis - Overall connectivity

- All the plots exhibit characteristics of heavy-tailed distributions.

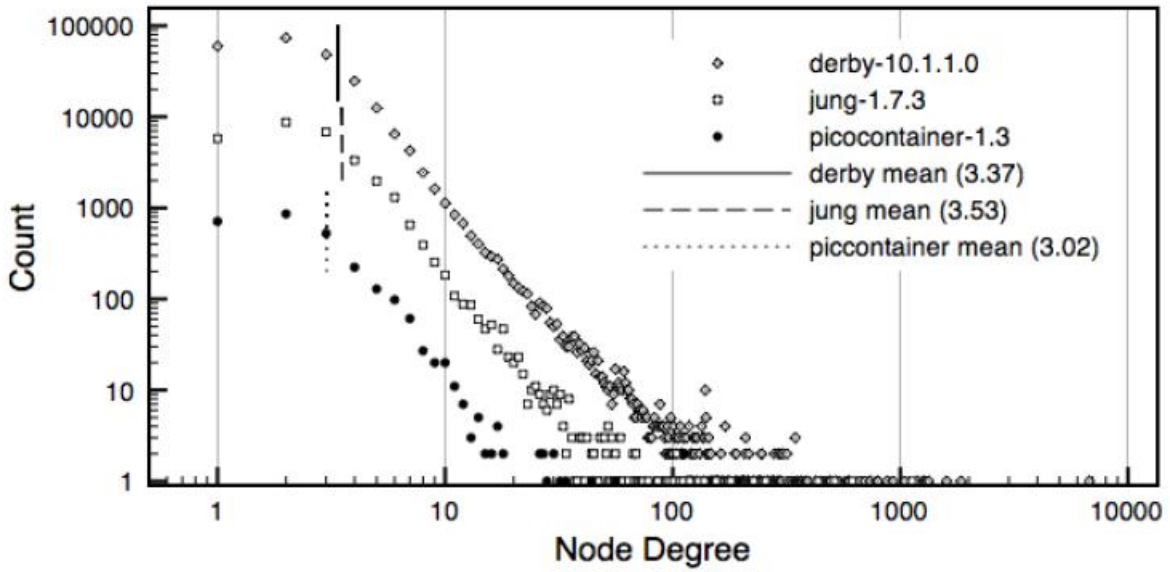


Fig. 6. Distribution of overall connectivity for the example systems.

Analysis - Between-module connectivity

- Figure 8 demonstrates that the between-module connectivity distributions are similar in shape to those computed to show overall connectivity (Figure 6).

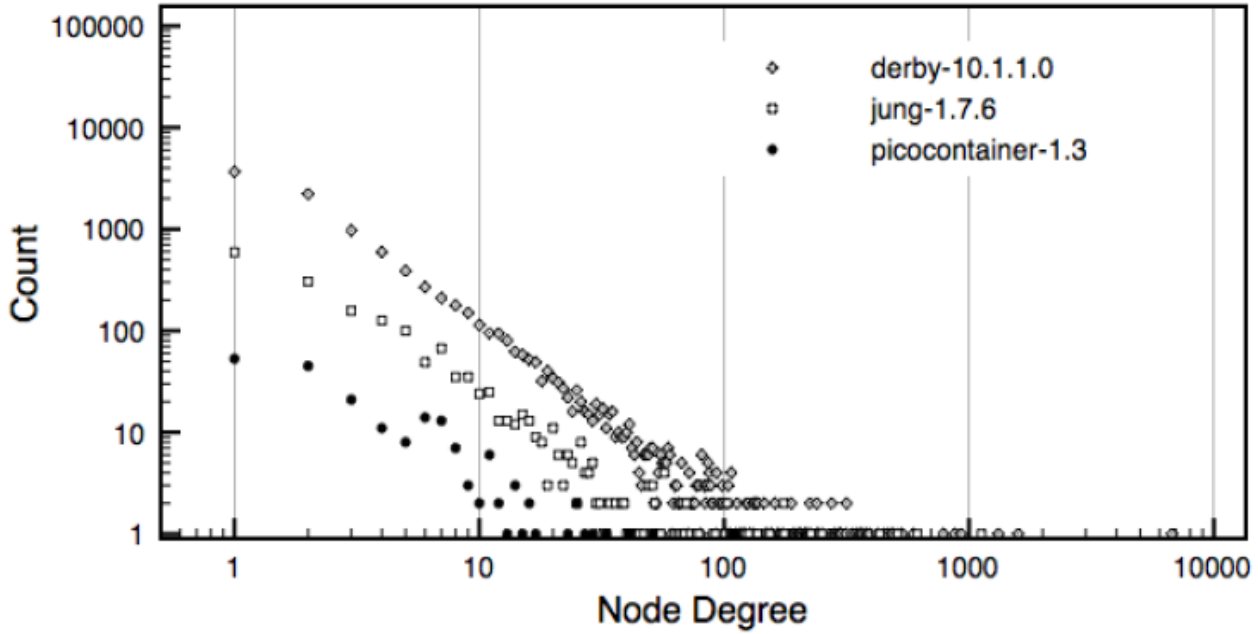


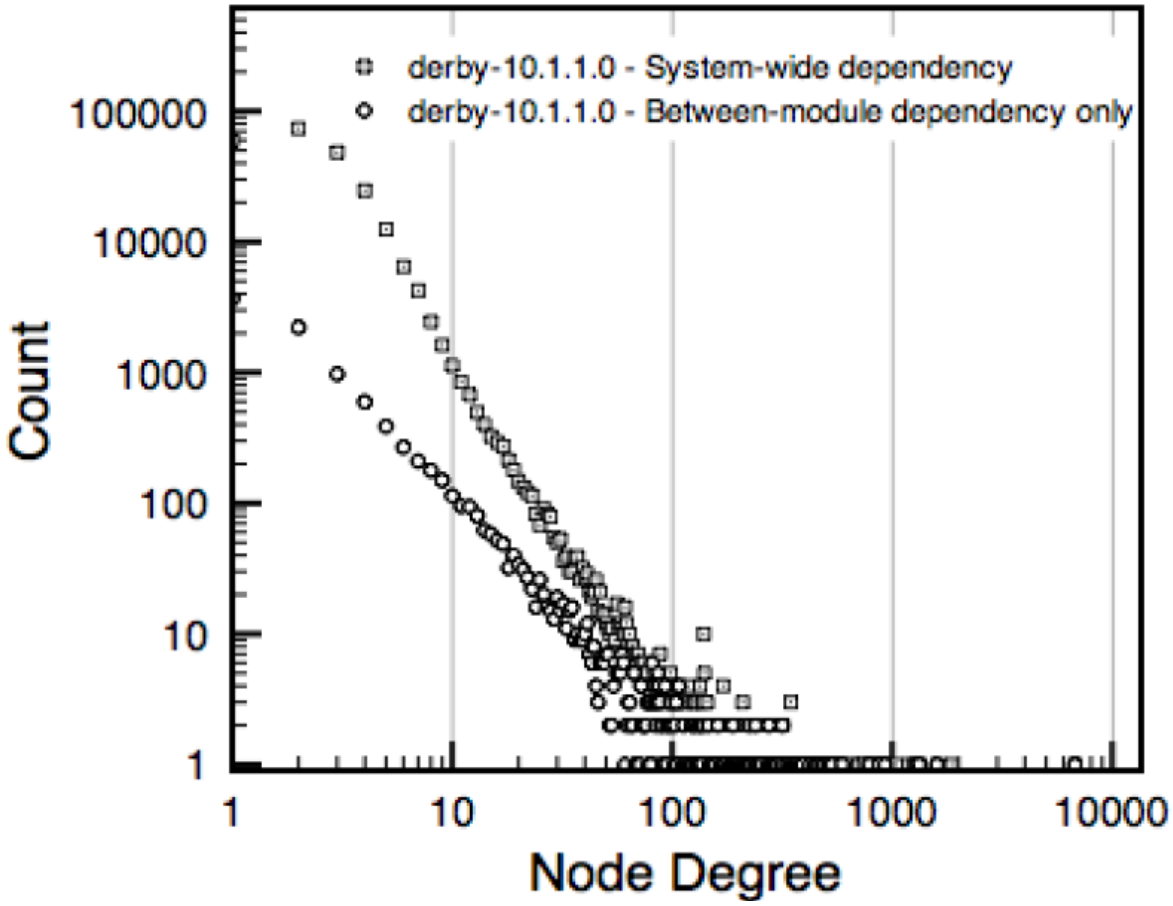
Fig. 8. Degree distributions for the sample systems (between-module only).

Analysis - Between-module connectivity

- We observe heavy tails in both distributions, which overlap when we plot them on the same graph.

- This demonstrates that the nodes that appear in the heavy tail of the overall connectivity distributions are the same nodes that appear in the heavy tail of the between-module connectivity distributions, and are responsible for the presence of high-coupling.

Overall vs between-module connectivity



Conclusions

- We conclude that high coupling is impracticable to eliminate entirely from software design.
- The maxim of “high cohesion/low coupling" is interpreted by some to mean that all occurrences of high coupling necessarily represent poor design.
- In contrast, our findings suggest that some high coupling is necessary for good design.