Analysis of Software Design Principles under Complex Network Theory

Juan Pablo Royo Sales & Francesc Roy Campderrós

Universitat Politècnica de Catalunya

January 17, 2021

Contents

1 Introduction

One of the most well known Software Design principles in **Software Engineering** is High Cohesion (High Cohesion) and Low Coupling (Low Coupling), which is well described here [You79].

As this two principles states a *robust Software* should be design with Low Coupling between their modules and High Cohesion inside it.

In other words, a Software that fullfil this characteristics should be very connected in their minimum Functional Units (FU) (Functions inside same file, Methods inside a class, etc), and with few connections between their coarse grained FU (a.k.a Modules or Packages).

In this work, we are going to formulate some hypothesis which we believe it can been empirically proved and shown the relationship between these principles and how to measure with **Complex Network Theory (Complex Network Theory)**. At the same time, we are going to analysis different kinds of software of different sizes and build under different language paradigms to see if the tool set that Complex Network Theory provides are suitable for the general case.

2 Preliminaries

In this section we are going to describe how and why the different Language Paradigms are selected and what is the criterion for selection of different Software solutions to be evaluated.

On the other hand as well, we are going to formulate some hypothesis that are going to guide our work to see if our assumptions can empirically been proved using Complex Network Theory.

2.1 Context

We have selected the most important 2 main Language Paradigm to conduct the analysis: Functional Programming (Functional Programming) and Object Oriented Programming (Object Oriented Programming).

The reasons behind this decision are basically the following:

- **95%** of Software in the Industry are built with one of these 2 Paradigms according to the last results of this well-known survey [Inc20].

- Due to the intrinsic nature of each of those Paradigms we have some hypothesis that we are going to describe later that can lead to different conclusion and Metrics
- If we can deduce some Software Design properties analyzing these 2 Paradigms we can generalize for the rest because they are quite different in nature and covers almost the whole Industry.
- We also believe that Software Principles should apply indistinguishably the Paradigm.

On the other hand the selection of the programs to be analyzed are the following:

- Most of the software are Open Source or Free software that can be download publicly either from `https://github.com` or from `https://repo1.maven.org`.
- Software that are marked as **PRIVATE** are Big Projects from Privates Companies that don't want to reveal the Sources Code and Names for Commercial reasons.
- In the case of **PRIVATE** Functional Programming Solution, it belongs to a Company one of the authors of the current work is working right now.
- In the case of **PRIVATE** Object Oriented Programming Solution, it belongs to a Company one of the authors of the current work worked in the past.
- In both cases, taken anonymous data for conducting this analysis has been agreed with legal representatives of those Companies.

On the last hand we are going to use *Haskell* Programs for analyzing Functional Programming and *Java* Programs for Object Oriented Programming. We believe that right now both are the most representative ones in their Paradigm fields.

2.2 Hypothesis

In this work we are trying to prove the following **Hypothesis** that we consider can be proved using Complex Network Theory (Complex Network Theory).

Hypothesis 1. *Given any Software Program Solution, its Network Metrics should be between the* 1*st Quartile and* 3*rd Quartile, according to the aver-*

age of the Network Metrics that we have been identified in this work, to be considered as a well Software Designed Solution.

Hypothesis 2. *Any Object Oriented Programming Program have a better modularity in terms of Complex Network Theory Metric rather than Functional Programming Programs.*

Hypothesis 3. *The more Lines of Code (LoC) a Program have, the better Modularity it presents.*

Hypothesis 4. *If the Software follows the principle design of High Cohesion and Low Coupling, the Degree Distribution (Degree Distribution) of the Generated Graph should follow a power-law like.*

3 Results

In this section first we are going to describe the **Experiments** conducted and after that we have obtained after running the different experiments; this is what we call **Metrics** subsection.

3.1 Experiments

In this section we are going to described how the experiment have been set up in order to prepare the graphs for taking the desired metrics, that could allow us to explain and verify the proposed hypothesis.

3.1.1 Preliminaries

In order to determine if a Software fullfil the 2 Software Design Principles that we want to analyze, High Cohesion and Low Coupling, we need to extract the Call Dependency Graph (CDG) from the programs that we want to analyze.

In CDG, **nodes** are Functional Units: *Functions* in the case of Functional Programming and *Methods* in the case of Object Oriented Programming. An **edge** is when from inside a FU another FU is called or invoked.

Therefore, we need to build this Call Dependency Graph in order to establish how the different Modules are interconnected in order to measure Low Coupling and High Cohesion.

3.1.2 Building Graphs from Programs

In order to achieve the desired Call Dependency Graph we are going to use some specific tooling for each Paradigm. The following has been used on each case:

- **Functional Programming**: *function-call-graph* `https://github.com/dustinnorwood/function-call-graph` is a Program that given a *Haskell* Source Code it outputs a *DOT* file with the Call Dependency Graph
- **Object Oriented Programming**: *java-callgraph* `https://github.com/gousiosg/java-callgraph` is a Program that given a *Java* Compiled Jar it outputs a in *stdout* the Call Dependency Graph. In order to use only *DOT* files we have converted this output into *DOT* using a script that is under `code/script_java.sh`

All the resulting graphs are under their respective folders as you can see here A.

3.1.3 R Scripts

3.2 Metrics

4 Discussion and Analysis

5 Conclusions

References

[Inc20] Stack Exchange Inc. 2020 development survey. `https://insights.stackoverflow.com/survey/2020`, 2020.

[You79] E. Yourdon. *Structured Design Fundamentals of a Discipline of-Computer Program and Systems Design.* Prentice-Hall Inc, Reno, 1979.

A Organization

- **code**: Under this folder you are going to find *R Scripts* code for conducting this analysis.
- **fp_graphs**: *DOT* files that contains the Dependency Graph Representation of each Functional Programming Program

- **oop_graphs**: *DOT* files that contains the Dependency Graph Representation of each Object Oriented Programming Program
- **report**: This report in Latex and PDF format.

B Programs Details - Line of Codes

B.1 FP Programs

Table 1: FP Analyzed Programs

Program	LoC
aeson	6948
amazonka	715531
async	743
attoparsec	4718
beam	20151
cabal	102525
co-log	1436
conduit	12963
containers	19556
criterion	2421
cryptol	30740
cryptonite	18763
dhall	29058
free	4472
fused-effects	4145
ghcid	1664
haskoin	12066
hedgehog	8277
helm	2071
hlint	6306
lens	16691
liquid	133740
megaparsec	8144
mios	6178
mtl	932
optparse	3220
pandoc	69179
pipes	1969
postgresql	6596

Table 1: FP Analyzed Programs

Program	LoC
protolude	1901
quickcheck	5077
reflex	10062
relude	2913
servant	15725
snap	5310
stm	1550
summoner	4025
text	9783
vector	12166
yesod	19971
PRIVATE PROGRAM	26975

B.2 OOP Analytzed Programs

Table 2: OOP Analyzed Programs

Program	LoC
akka-actor_2.10-2.3.9	35702
commons-cli-1.4	830
commons-codec-1.10	2231
commons-csv-1.8	607
commons-email-1.4	760
disruptor-3.4.2	1131
ftpserver-core-1.0.6	4052
grpc-core-1.34.1	9142
guava-28.1-jre	37216
hbase-client-2.4.0	33209
hsqldb-2.4.1	30578
jackson-databind-2.12.0	22516
javax.servlet-api-4.0.1	901
jedis-3.4.1	12640
jersey-core-1.19.4	5656
jetty-7.0.0.pre5	6727
joda-time-2.10.6	8891
jsch-0.1.54	4833
jsoup-1.13.1	5955
junit-4.13.1	5803

Table 2: OOP Analyzed Programs

Program	LoC
mail-1.4.7	8817
mariadb-java-client-2.7.1	9229
mongo-java-driver-3.12.7	35676
mx4j-3.0.2	6902
org.eclipse.jgit	42417
pdfbox-2.0.22	26413
poi-4.1.2	44691
postgresql-42.2.18	15199
resteasy-jaxrs-3.14.0.Final	15267
runtime-3.10.0-v20140318-2214	921
slf4j-api-1.7.30	763
spring-security-core-5.4.2	4726
spring-web-5.3.2	21393
tomcat-embed-core-10.0.0	47185
zookeeper-3.6.2	19207
PRIVATE PROGRAM	2143