Analysis of Software Design Principles under Complex Network Theory

Juan Pablo Royo Sales & Francesc Roy Campderrós

Universitat Politècnica de Catalunya

January 14, 2021

Contents

1 Introduction **1**

2 Preliminaries **4**
2.1 Context 4
2.2 Hypothesis 5

3 Results **5**
3.1 Experiments 5
3.2 Metrics 5

4 Discussion and Analysis **5**

5 Conclusions **5**

References **5**

A Organization **5**

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 (Functions inside same file, Methods inside a class, etc), and with few connections between their coarse grained functional units (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

2.1 Context

Program	LoC	Repository
aeson	6948	https://github.com/haskell/aeson.git
amazonka	715531	https://github.com/brendanhay/amazonka
async	743	https://github.com/simonmar/async.git
attoparsec	4718	https://github.com/haskell/attoparsec.git
beam	20151	https://github.com/haskell-beam/beam.git
cabal	102525	https://github.com/haskell/cabal.git
co-log	1436	https://github.com/kowainik/co-log
conduit	12963	https://github.com/snoyberg/conduit
containers	19556	https://github.com/haskell/containers.git
criterion	2421	https://github.com/haskell/criterion.git
cryptol	30740	https://github.com/GaloisInc/cryptol
cryptonite	18763	https://github.com/haskell-crypto/cryptonite.git
dhall	29058	https://github.com/dhall-lang/dhall-haskell.git
free	4472	https://github.com/ekmett/free.git
fused-effects	4145	https://github.com/fused-effects/fused-effects.git
ghcid	1664	https://github.com/ndmitchell/ghcid.git
haskoin	12066	https://github.com/haskoin/haskoin-core.git
hedgehog	8277	https://github.com/hedgehogqa/haskell-hedgehog.git
helm	2071	https://github.com/z0w0/helm.git
hlint	6306	https://github.com/ndmitchell/hlint.git
lens	16691	https://github.com/ekmett/lens.git
liquid	133740	https://github.com/ucsd-progsys/liquidhaskell.git
megaparsec	8144	https://github.com/mrkkrp/megaparsec.git
mios	6178	https://github.com/shnarazk/mios.git
mtl	932	https://github.com/haskell/mtl.git
optparse	3220	https://github.com/pcapriotti/optparse-applicative.git
pandoc	69179	https://github.com/jgm/pandoc.git
pipes	1969	https://github.com/Gabriel439/Haskell-Pipes-Library.git
postgresql	6596	https://github.com/haskellari/postgresql-simple.git
protolude	1901	https://github.com/protolude/protolude
quickcheck	5077	https://github.com/nick8325/quickcheck.git
reflex	10062	https://github.com/reflex-frp/reflex.git
relude	2913	https://github.com/kowainik/relude
servant	15725	https://github.com/haskell-servant/servant.git
snap	5310	https://github.com/snapframework/snap.git
stm	1550	https://github.com/haskell/stm.git
summoner	4025	https://github.com/kowainik/summoner
text	9783	https://github.com/haskell/text.git
vector	12166	https://github.com/haskell/vector.git
yesod	19971	https://github.com/yesodweb/yesod.git
small	449	PRIVATE
PRIVATE PROGRAM	26975	PRIVATE

Table 1: FP Analyzed Programs

2.2 Hypothesis

- Que si escribes un programa, las métricas de su grafo deben estar más o menos parecidas .95 percentil al avarage de las que hemos calculado.
- Que los OOP programs estan mejor modularizados que los FP programs
- Que la modularización de un programa mejora cuantas más lineas de código tiene

3 Results

3.1 Experiments

3.2 Metrics

4 Discussion and Analysis

5 Conclusions

References

[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 *C++* code for simulating and generating the different networks using the different strategies: Growth + Preferential Attachment (G Pref Attachment), Growth + Random Attachment (G Random Attachment) and No Growth + Preferential Attachment (NG Pref Attachment), as well as the *R* scripts for generating plots and doing graph analysis.
- **code/data**: Data Generated for each strategy
- **report**: This report in Latex and PDF format.