Structural dependencies between system fault distribution principles

Ana Vranković and Tihana Galinac Grbac
Faculty of Engineering
University of Rijeka
Vukovarska 58, 51000 Rijeka
Email: avrankovic13@gmail.com, tihana.galinac@riteh.hr

***Abstract*—In this paper, we use a network analysis approach to represent and analyse software structure evolution and its relation to defectiveness of subgraphs present in the software structure. We want to define structural dependencies between various empirical principles already confirmed about fault distributions in software systems. Main benefit is such approach is an tool for software evolution analysis independently of development context, programming language, underlaying platform. Our aim is to contribute to the theory of software evolution.**

THE RESEARCH TOPIC PRESENTATION

The fault distributions across system modules have been investigated by many authors. The first systematic study was established by [1] and was motivated by numerous earlier studies, in particular [2] and further replicated by [3], [4]. All these studies identified the uneven distribution of software faults over the system modules and empirically observed existence of the Pareto principle. Studies on the analytic model for fault distributions over the system modules have resulted with less consistent results in terms of best fit particular analytical distribution to empirical data of faults over the system modules [5], [6]. As step further in aim to understand fault behaviour we were motivated to understand structural dependencies on fault behavior.

One fraction within the network structure analysis is identification of significant network substructures aiming to uncover structural design principles in complex networks. Network motifs, proposed by [7], are patterns of interconnections (they observe just three node subgraphs that are presented in Fig. 1) in complex networks with an occurrence that is statistically higher than in random networks. It is identified that motifs could be useful for characterizing universal classes of different complex network structures in different scientific fields such as medicine, sociology, electrical engineering. In our previous work [8] we applied the same analysis approach to software structure. In Fig. 2 is presented an code example and its relation to graph structure. With this approach we were analysing motif evolution within software structure of Eclipse projects and confirmed previous findings that motif is characteristic of analysed artifact, like software and not of the development context. However, in the same analysis we identified that subgraph occurrence have significant relation to software maturity and that motivated us to further analyse subgraph behaviour and its relation to its defectiveness.

We analysed two Java projects (Eclipse JDT and PDE) that contained 25 versions in total. Structural analysis was applied based on subgraph types that are representing primitive 3 node communication patterns within the graph structure. We have shown that software programs have similar behaviors in terms of average subgraph type defectiveness and distributions of average subgraph frequencies coming from the same population. We found that subgraph defectiveness is a good predictor of number of defects in system version (see Fig. 3 and 4). New insights obtained with this approach may be useful in better fault prevention by providing software architects with new architecture design guidelines or by providing software quality personnel with a set of risky subgraphs so they may better plan verification and thus be more effective in defect detection activities. In our further work we aim to investigate how different subgraphs influence other metrics of the software and software evolution in other development contexts. Also, analysing software written in other languages could be interesting.

ACKNOWLEDGMENTS

This work has been supported in part by Croatian Science Foundation's funding of the project UIP-2014-09-7945 and by the University of Rijeka Research Grant 13.09.2.2.16.

REFERENCES

[1] N. E. Fenton, N. Ohlsson, "Quantitative Analysis of Faults and Failures in a Complex Software System," *IEEE Trans. Softw. Eng.*, vol. 26, no. 8 , pp. 797–814, Aug. 2000.

[2] E. Adams. Optimizing Preventive Service of Software Products. *IBM Research J.*, vol.28, no.1, pages 2–14, 1984.

[3] C. Andersson and P. Runeson, "A Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems," *IEEE Trans. Softw. Eng.*, vol. 33, no. 5, pp. 273–286, May 2007.

[4] T. Galinac Grbac, P. Runeson and D. Huljenić, "A Second Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems," *IEEE Trans. Softw. Eng.*, vol 39, no. 4, pp. 462–476, Apr. 2013.

[5] G. Concas, M. Marchesi, A. Murgia, R. Tonelli, and I. Turnu, "On the Distribution of Bugs in the Eclipse System," *IEEE Trans. Softw. Eng.*, vol. 37, no. 6, pp. 872–877, Nov./Dec. 2011.

[6] T. Galinac Grbac, D. Huljenić, "On the probability distribution of faults in complex software systems," *Inf. and Soft. Techn.*, vol. 58, pp. 250–258, Apr. 2015.

[7] R. Milo, S. Shen-Orr, S. Itzkovitz, et al. *Network motifs: simple building blocks of complex networks*. Science 2002; 298:824–27.

[8] J. Petrić and T. Galinac Grbac. Software structure evolution and relation to system defectiveness. In *Proc. of the 18th Int. Conf. on Evaluation and Assessment in Software Engineering EASE2014*, London, UK, 2014.

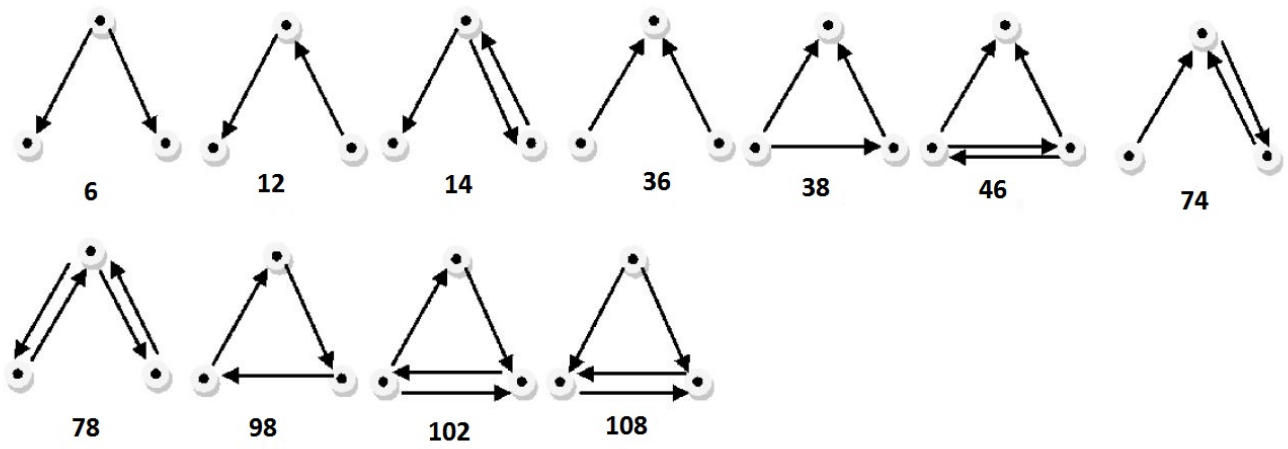

Fig. 1: All types of three node subgraphs

```
class Point {
     int row, column;
}
class Chessmen {
     Point pos;
     int iGetValue();
}
class Move {
     Point start_pos;
     Point end_pos;
     bool bIsAllowedMove();
}
class Pawn : public Chessmen {
     Move *moves;
     int iGetValue();
}
```

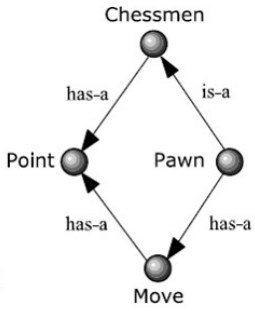

Fig. 2: Code to graph example

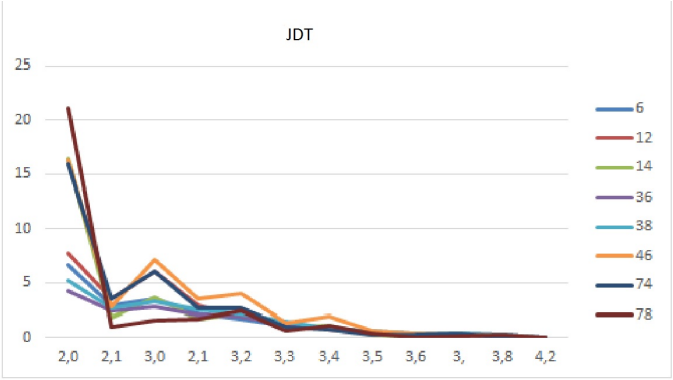

Fig. 3: Frequency and defects ratio over versions in JDT project

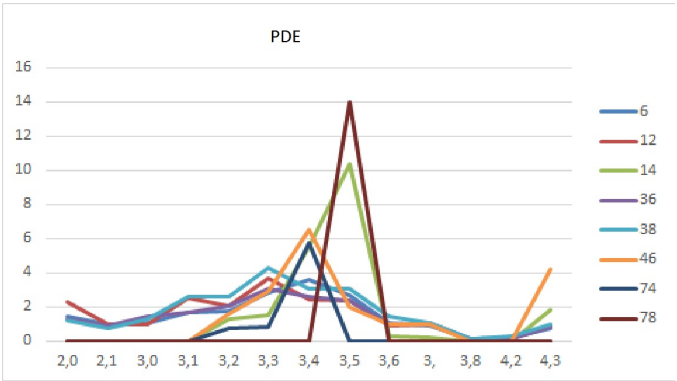

Fig. 4: Frequency and defects ratio over versions in PDE project