Recent Findings in Software Engineering

JIN LIN LHUN TAO

Institute of SE Research, Xianghiaj
Email: jllTao@ioser.Xian.com

Software Engineering embeds all software life, from the project phase to the development and to the maintainance. The research fiend is immensely vast and this report focuses on main streams which are subfields of Software Engineering, like refactoring, defect analysis, metrics, and, finally, Software Networks. In this report I acknowledge of some of the main results obtained in this field of research as distinguished per topics.

Keywords: Software Engineering

Received 02 October 2015; revised 02 October 2015

1. INTRODUCTION

The need for measuring software has become more and more impelling about two decades ago, with many documents devoted to software measures, both in the software industry and in the scientific literature. Without measurements software management can be very uneffective, since software products are extremely complex, and planning, estimations, and control become inaccurate. Software metrics were created in order to improve the measuring and controlling its essential parameters. Even if the meaning is (or was) used in a broad sense, software metrics generally refer to: product, process, resource, or project measurements. We restric our attention on the first meaning, dealing with product metrics alone, which may be distinguished in size metrics, complexity metrics, and quality metrics, generally related to each other. A milestone in the definition of a useful set of software metrics was the work of Kidamber and Kemerer [1], the first trial in addressing the problem of implementing a new suite of metrics for Object Oriented (OO) design.

The paradigm adopted considers modern OO software systems as complex software networks, according to its modern view derived from Barabasi-Albert works for the WWW network. This paradigm offers the opportunity of introducing software metrics which were out of the more traditional schemes of software engineering for measuring software, and of assessing software quality using new tools. The main reference points of this thesis for the estimation of software quality will be software defects, which will be considered as pivotal points, with all the other software metrics rounding around. Barabasi and Albert were the first to investigate the concept of complex network, giving rise to the modern complex network theory.

The concept of complex network has been extended to software systems [2, 3] and to OO software, were the networks nodes were identified with software components, and the networks edges with the interactions among them. As a consequence, power law distributions, scale free and small world properties and even fractals properties of complex networks [70] are necessarily present in software systems. Despite such amount of recent researches in this field, the applications to software engineering, and in particular to the improvement of software quality, is largely lacking, but some recent applications have recently appeared [4].

Such schematic view allows immediately to understand how defects can affect software in different proportions: a code defect introduced into a class linked to a very few classes will probably be less effective than a defect affecting a largely linked class. The implications of such a simple concept for the software industry can be very large and can range from software maintainability costs, to defect detection strategies, to resources allocation and so on.

2. DISCUSSION

An overview of the concept of complex software network, and a state of the art of related works is prvided in [5, 6, 7, 8]. In fact, typically, software systems are built out of many interacting modules and subsystems, at many levels (functions, classes, interfaces, libraries, source files, packages, etc.). This modular structure, where software entities interact reciprocally, suggests a graph based representation, where software entities can be represented as nodes, and all different relationships, like object interactions

The Computer Journal, Vol. 00 ??, No. 00 ??, ???
and routine calls, can be represented as connections between them. Papers [9, 10, 11, 12] present the metrics used for understanding the structure and properties of software networks. In particular they describe the Social Network Analysis metrics (SNA metrics) and the Fractal Dimension for complex networks. In [13, 14, 15, 16] the statistical distribution chosen for exploring the structure of complex networks and the statistics of empirical data measured in large software systems are discussed. It also presents the generative model associated to the more important statistical distributions, and describe how these generative models can be practically applied to large software systems

In [17, 18, 19, 20] are discussed in detail the Yule process and described applications for modeling the behavior of various software systems written in Java. It is also shown how the evolution of such systems through different releases can be fit very well by the model. In [21, 22, 23, 24] are described the statistical properties of bugs and a generative model for explaining the empirical distributions. In fact, the distribution of bugs in software systems has been shown to satisfy the Pareto principle, and typically shows a power-law tail when analyzed as a rank-frequency plot. It is further discussed the subject from a statistical perspective, using as case studies five versions of Eclipse, to show how log-normal, Double Pareto and Yule-Simon distributions may fit the bug distribution at least as well as the Weibull distribution. In [25, 26, 27, 28] are presented analysis of software networks by using the SNA metrics. It is investigated if the new proposed SNA metrics possess the same statistical properties found for bugs and have similar empirical distributions. Moreover, the possible correlations with Bugs and/or with other metrics and properties are studied. It is also investigated if the analytical distribution functions may be used to forecast future properties of the software systems. In [29, 30, 31, 32] the analysis of the software graphs of 96 systems of the Java Qualitas Corpus, obtained parsing the source code and identifying the dependencies among classes is presented. For two systems, Eclipse and Netbeans, it is computed also the number of bugs, identifying the bugs affecting each class, and finding that some SNA metrics are highly correlated with bugs, while others are strongly anticorrelated. In [33, 34, 35, 36] tools for characterizing software quality the Fractal Dimension of software networks are presented with an algorithm for computing the fractal dimension of a software network, and compare its performances with two other algorithms. In [37, 38] various releases of the publically available Eclipse software systems are analyzed, calculating the fractal dimension for twenty sub-projects, randomly chosen, for every release, as well as for each release as a whole. The results display an overall consistency among the sub-projects and among all the analyzed releases. A very good correlation between the fractal dimension and the number of bugs for Eclipse and for twenty sub-projects is found. Various works discuss statistical distributions of refactored classes and the analysis shows that, despite common believes, the coupling is not so much affected by multiple refactoring applications of move method and add method refactorings in particular.

3. CONCLUSIONS

We presented a selection of researches on software engineering which use modern approaches to analyze software systems using state-of-art theoretical techniques and tools.

In particular the analysis of software systems as complex networks allows to detect new metrics, like fractal dimension, clustering and others that can be related to software quality. Statistical distributions and mathematical models can be applied to the analysis of software defects and to model software systems growth.

Finally refactoring have also been analyzed with respect to the common belief that coupling is affected by multiple applications of move method and add method refactorings.
REFERENCES


[38] Monni, C. et Al. 2015 Predicting Software Defectiveness through Network Analysis. SATToSE, 36-47