Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.Doi Number

Software Fault-Proneness Analysis based on Composite Developer-Module Networks

Shou-Yu Lee[1], W. Eric Wong[1], Yihao Li[2], and William Cheng-Chung Chu[3]

[1]Department of Computer Science, University of Texas at Dallas, 800 West Campbell Road, Richardson, TX 75080, USA

[2]School of Information and Electrical Engineering, Ludong University, 186 Hongqi W Rd, Zhifu District, Yantai, Shandong 264025, China

[3]Tunghai University, No. 1727, Sec.4, Taiwan Boulevard, Xitun District, Taichung 40704, Taiwan

Corresponding author: W. Eric Wong (e-mail: ewong@utdallas.edu).

This work was funded by the USA National Science Foundation (Grant No. 1757828 and 1822137).

ABSTRACT Existing software fault-proneness analysis and prediction models can be categorized into software metrics and visualized approaches. However, the studies of the software metrics solely rely on the quantified data, while the latter fails to reflect the human aspect, which is proven to be a main cause of many failures in various domains. In this paper, we proposed a new analysis model with an improved software network called Composite Developer-Module Network. The network is composed of the linkage of both developers to software modules and software modules to modules to reflect the characteristics and interaction between developers. After the networks of the research objects are built, several different sub-graphs in the networks are derived from analyzing the structures of the sub-graphs that are more fault-prone and further determine whether the software development is in a bad structure, thus predicting the fault-proneness. Our research shows that the different sub-structures are not only a factor in fault-proneness, but also that the complexity of the sub-structure can affect the production of bugs.

INDEX TERMS Software Networks, Developer-Module Networks, Fault-Proneness Prediction, Human Aspects, Software Metrics

I. INTRODUCTION

Program failure has always been a major concern when it comes to software development [131],[134]. Nevertheless, as the trend of software scale and complexity continuously increase, the cost of software failure exaggerates riotously. According to a study by the National Institute of Standards and Technology, the annual cost of software bugs in the U.S. reached $59.5 billion in 2002 [85]. To achieve more cost-effective management, some software revalidation and testing techniques, such as fault[1] localization, have become an indispensable part of the software development and evolution process [12]. Although the techniques of fault localization are becoming more and more complete, it is still expensive to operate with a higher degree of precision. Therefore, applying fault-proneness prediction beforehand can be a clever step to ensuring software quality [18], [30], [48], [51], [62], [68], [91], [94], [128].

Among existing software fault-proneness analysis and prediction models, many essential factors have been considered [59]. Recently, discussions have risen with the impact of human activities on software quality [9], [10], [16], [20], [22], [26], [44]-[47], [72], [79], [103], [106], [118], [126], [127], [130]. With that in mind, a common approach is to include characteristics of a developer as a part of the fault-proneness prediction model by calculating them into metrics [60], [95], [137]. In contrast, some approaches visualize the aspects of development artifacts using social network analysis [36], [42], [90], [107], [139]. These studies either rely heavily on quantified data or use only simple relations between developer and software and result in ignorance to deeper and indirect dependencies in their models.

In this paper, we propose a new approach to build the fault-proneness analysis model that integrates human aspects by using complex network techniques. Recent studies [34], [141] have shown that certain structural patterns in the software network have high correlations with bugs or unexpected system performances. On top of that, our research aims to integrate the pattern of developers' activity into the software network, and by doing so, we will have a more comprehensive view on deeper and indirect developer-module dependency, and thus can further help to analyze the

[1] We use the term bug and fault interchangeably throughout the paper.

relationship between developers' behaviors and bugs they introduce.

The rest of the paper is structured as follows. Section II introduces the basic idea of software fault-proneness prediction using various approaches, how the developers' aspects can be included in the software fault-proneness prediction model, and the implementation of complex networks on software systems. The proposed Composite Developer-Module Networks and their attributes are presented in Section III. Our case studies and experiment results are given in Section IV. Section V discusses some threats to the validity of our study, and some related work is introduced in Section VI. Finally, the conclusions of our research and future work are listed in Section VII.

II. BACKGROUND

In this section, we will provide a brief introduction to software metrics and how they are utilized in building software fault-proneness prediction models. Additionally, we will introduce the thoughts of integrating human aspects into the prediction model and how the idea can be implemented by software network analysis.

A. SOFTWARE METRICS

Software metrics are useful and a widely applied measure to understand the insight of software project development. Numerous studies have been conducted and proved that many fault-proneness prediction approaches using software metrics can notably help the effectiveness of maintaining software quality and reliability [5], [25], [27], [38], [40], [49], [61], [63], [80], [89], [101], [102], [104], [109], [111], [116], [136].

There are two major categories of software metrics, namely, *product* metrics and *process* metrics [140]. Product metrics are used to measure the aspects of the software per se. The product metrics can be categorized into generic metrics and special metrics, and generic metrics consist of two categories: *static* metrics, which can be calculated by the attribute of source code, and *dynamic* metrics, which can only be collected during runtime. To name a few, static metrics include Line of Code [88], McCabe Complexity [71], and C.K. metrics [24], and dynamic metrics include dynamic coupling metrics [3], runtime cohesion metrics [76], and code coverage-based metrics [23], [77], [135]. Both static and dynamic metrics can be further divided into internal metrics, which examine the internal structure of a software module, and external metrics, which focus on the interactions between separate software modules [132], [133]. Special metrics such as [2] measure the attributes that are not directly related to the target software. Various commercial or free tools appear on the market that provide the feature of collecting these metrics, such as Scitool Understand [108].

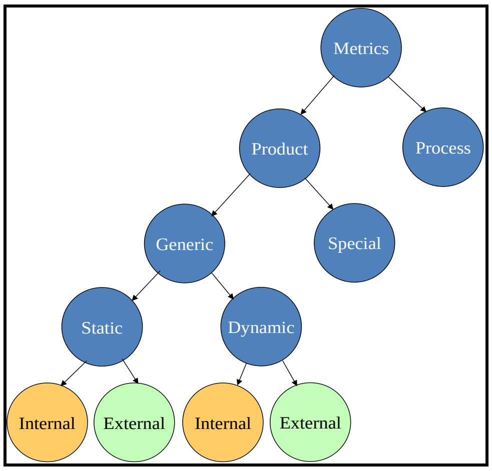


FIGURE 1. Taxonomy of software metrics

Process metrics are used to analyze the process of software development and evolution, and some examples are: code churn [81], [112], social network analysis-based metrics [9], [13]-[15], [20], [122], developer-based metrics [95], [127], [137], and organizational metrics [8], [79], [83]. **FIGURE 1** shows the hierarchy of the software metrics.

In this paper, although we do not use a single measure of metrics on fault-proneness prediction, we still employ the concept of static product metrics (module network), process metrics (developer network), and the idea of the external structure of software module to build our model.

B. METHODS TO BUILD FAULT-PRONENESS PREDICTION MODELS

Once we have software metrics as indicators that reflex the characteristics of software, a prediction model can be built to produce beneficial results that can be used to predict fault-proneness. To operate an effective fault-proneness prediction, loads of factors need to be considered during model construction [59].

Regression analysis is a straightforward method that measures the dependency of variables [17]. A regression model is given to estimate the value of some dependent variable through a regression equation that takes the value of several independent variables as inputs. Regarding software fault-proneness prediction, many forms of regression can be applied as prediction models. Take linear regression models as examples: the inputs can be a bunch of software metrics, and the output should be the tendency of fault-proneness [6], [38], [41], [64], [121]. Alternatively, logistic regression, another commonly used regression method, takes one of two different values on the dependent variable to divide the ssoftware modules into fault-prone or non-fault-prone classes [4], [29], [31], [58], [78], [114], [115], [143], [144].

Machine learning is a comparatively advanced approach used in software fault-proneness prediction that is based on various learning algorithms. With a given training set of input values (software metrics), the desired training output (index of fault-proneness) can be approached by iterative learning procedures, among which the fault-proneness prediction models can be built. Below are some samples of commonly used techniques and their brief introductions:

- Artificial neural networks (ANNs) [7], [52], [120], [142]: using a pre-defined network topology to process learning algorithm. Different designs of a network can considerably affect the outcome of training results.
- Decision tree-based modeling [54]-[56], [100], [112]: recursive partition into smaller subset. The technique can be easier to interpret on larger data set.
- Naïve Bayesian classifier [19], [75], [123] and the Bayesian Belief Network [1], [28], [93], [96]: both based on Bayes' theorem. the Naïve Bayesian classifier assumes that the value of attributes is independent. If dependencies exist between attributes, a BBN can be used instead.
- Support vector machine [32], [39], [53], [138]: maps the training tuples into a higher dimension and searches for the linear optimal separating hyperplane to separate the new tuples into different classes.
- Discriminant analysis [35], [50], [92], [119]: a discriminant function is built from the training dataset to assign data points into either the desired (fault-prone) or the non-desired (non-fault-prone) group.

C. DEVELOPER ASPECTS

Since the software is a pure cognitive product of human developers, the flaws in it are significantly caused by erroneous behaviors of humans involved [44]-[47], [66], [126], [127]. Therefore, several studies have tried to clarify the accurate association of human aspects and software bugs. One of the approaches separates the consistent characteristics of a human individual and evaluates their behavior, which can be used to predict his/her future performance [11], [43], [60], [70], [95]. Another approach tries to classify specific working environments and activities during the development process that could affect the performance of developers [9], [33], [83], [113]. Since the later approach focuses more on human activities and interaction than characteristics, it can be useful when there is no sufficient historical data on individual developers and is closer to our approach.

D. SOFTWARE NETWORK

A modern software system is composed of various elements, such as functions and variables in software modules, and commit records in development. These elements are all dependent on each other in some way. Therefore, we can construct an abstract model in a complex network if we regard the elements as nodes and relationships as edges. Hence, the network is called a software network.

Studies have shown that the theory of complex networks can be accommodating in the software reliability domain because of the small-world effects and scale-free properties [82], [124]. Also, the properties of high cohesion and low coupling in the software development process can modularize [117]. As an application in fault-proneness prediction, many techniques apply the topology characteristics and properties such as complexity and evolution of rules into quantitative indicators. Among them, network analysis-based metrics are derived from the network features such as closeness or betweenness centrality [75], [98]. Furthermore, the dependency for software modules can also work as a good gauge of fault-proneness in the system [87], [122], [145].

Although a complex network is a practical technique and relatively intuitive in describing the multifaceted relationship, the entire structure can grow enormously when the target model is from a large number of real-world objects and thus hold researchers back from investigating the network altogether. To tackle the issue, the researchers must identify the specific type of structures that can carry some important local properties in the network. Network Motif [110] serves as one of the properties that are recurrent and have some statistically significant patterns and is broadly implement in many domains such as biochemistry, neurobiology, and engineering, etc. Through these practices, various searching algorithms for certain types of network motifs and their application have been applied on software domains as well [67], [69], [141].

E. NETWORK WITH HUMAN ASPECTS

Although the software network can be helpful when it comes to fault-proneness prediction, the influence of human aspects is often overlooked (as states in Section II-C). To further improve the accuracy of prediction results, several approaches try to comprise the human aspects of the network prediction model. One simple tactic is to express developer contributions with a Contribution Network (also called a Developer-Module Network in some studies) [10], [16], [22], [97]. In the contribution network, a contribution edge always refers to a commit on a module made by a developer. The weights of edges are defined as the number of commits for a developer to the specific module. **FIGURE 2** depicts an example of the contribution network.

On the other hand, some studies [9], [21], [57], [73], [74]

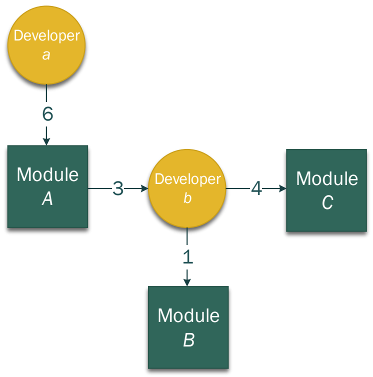


FIGURE 2. A Contribution Network with two developers and three software modules

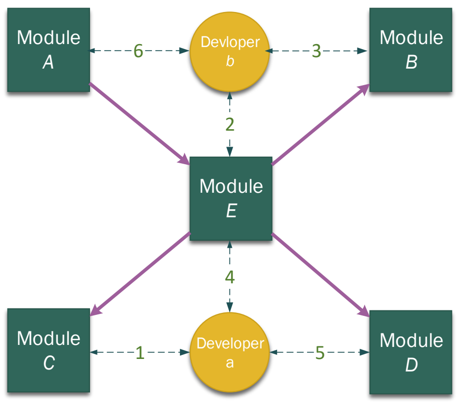


FIGURE 3. A Socio-Technical Network with two developers and five software modules

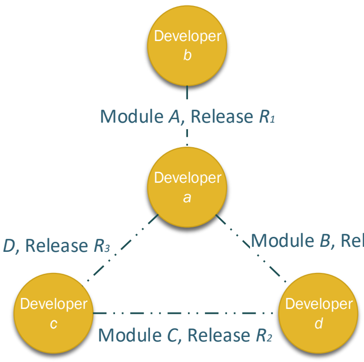


FIGURE 4. A Developer Collaboration Network with four developers

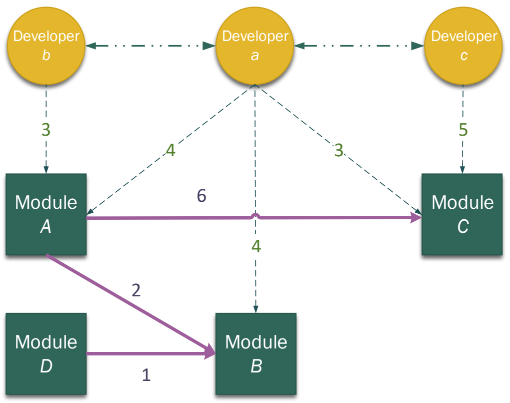


FIGURE 5. A Tri-Relation Network with three developers and four modules

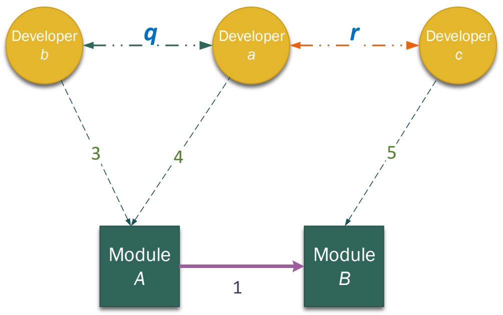


FIGURE 6. Direct and indirect relationship of developers

suggest that the module dependency should be used along with the contribution history of developers on fault-proneness prediction. Thus, the network is called a Socio-Technical Network, which is a hybrid of developer contribution network and a network of module dependency. There are two types of edges: the contribution edges are comparable to those in the contribution network but are now bidirectional; the module edges always have the weight of one. **FIGURE 3** depicts an example of a Socio-Technical Network.

Then again, another unique idea is to describe developers' collaborations by assigning edges to those that have worked on common modules. The network is called Developer Collaboration Network and involves developers [66], [129] solely. In the prediction model, the metrics can be generated by applying *Social Network Analysis* since the edges in the network are flexing some social relations. **FIGURE 4** depicts an example of a Developer Collaboration network.

A Tri-Relation Network (TRN) is the ultimate form of network that involves the human aspect [65]. The network combines three different kinds of relation, i.e., the developer contribution, module dependency, and developer collaboration. With those in hand, a more comprehensive view of activities in software development can be described. Moreover, *Social Network Analysis* can be applied to build a sounder and complete fault proneness prediction model. **FIGURE 5** depicts an example of the Tri-Relation Network. Despite the fact that all these approaches established the foundation of the human-aspect fault-proneness prediction model, they fail to illustrate developers' deeper and indirect dependency. Take the network in **FIGURE 6** as an example. The developer relationship ***q*** is evident because both developer *a* and *b* have contributions to Module *A*. However, since there is a function call from Module *A* on Module *B*, the dependency ***r*** between developer *a* and developer *c*, who has some contributions on Module *B*, should be considered but often ignored in the previous studies. Our approach aims to cover such indirect dependencies on developers and hence portray a more comprehensive picture of developer relationships and activities.

III. COMPOSITE DEVELOPER-MODULE NETWORKS

In this section, the insights of the Composite Developer-Module Networks elements used in our research will be explained. The motivation of the Composite Developer-Module Networks is to include deeper and indirect dependencies between developer and software modules. Thus, one can better analyze the more comprehensive

TABLE 1
NOTATIONS USED IN OUR STUDIES

$N(R)$	A CDM network for release R
$V(R)$	All vertices of $N(R)$
$E(R)$	All edges of $N(R)$
$V_f(R)$	All vertices in $N(R)$ representing functions
$V_d(R)$	All vertices in $N(R)$ representing developers
$E_f(R)$	All edges in $N(R)$ representing function calls
$E_c(R)$	All edges in $N(R)$ representing developer contributions
$N_{sub}(R)$	A sub-structure derived from $N(R)$
$V_{sub}(R)$	All vertices in sub-structure $N_{sub}(R)$
$E_{sub}(R)$	All edges in sub-structure $N_{sub}(R)$
v_F	A vertex representing a function F
u_D	A vertex representing a developer D
$e_{(F,G)}$	An edge representing a function call from F to G
$g_{(D,F)}$	An edge representing a developer D has contributions on F
$name(X)$	Name label of object X

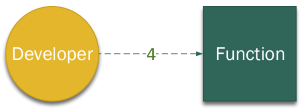


FIGURE 7. An example of an edge in $E_c(R)$

structural patterns of developers' behaviors and their correlation to the vulnerability of software bugs.

The notations we used in the following section are listed in **TABLE 1**.

A. DEFINITION OF COMPOSITE DEVELOPER-MODULE NETWORK

A Composite Developer-Module Network of a specific version of release R is defined as below:

$$N(R) = (V(R), E(R)) \tag{1}$$

$V(R)$ is a set of all vertices, including all developers before release version R, denoted as $V_d(R)$, and all functions in software modules of release version R, denoted as $V_f(R)$. This implies that

$$V_d(R) \cup V_f(R) = V(R) \tag{2}$$

$E(R)$ represents all edges in the network, including $E_c(R)$: links from developers to the functions he/she contributes to in a software module before release version R (**FIGURE 7**), and $E_f(R)$: represents links between every two functions in the software in release version R (**FIGURE 8**). This implies that

$$E_c(R) \cup E_f(R) = E(R) \tag{3}$$

Corollary 1. A Composite Developer-Module Network is a connected graph. If there exists a vertex $v \in V$, then (1) if $v \in V_d(R)$, then v denotes the developer that should have at least one contribution in software module; (2) if $v \in V_f(R)$, then v denotes the function which has either some contributing contributor or link with other function. Thus, all the existing v in V are connected.

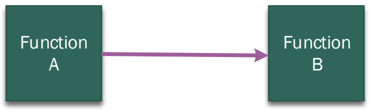


FIGURE 8. An example of an edge in $E_f(R)$

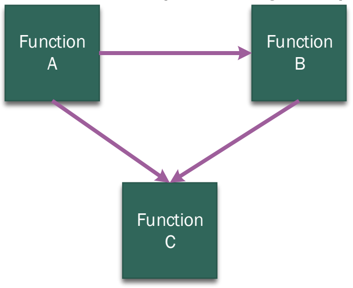


FIGURE 9. An example of Module Network

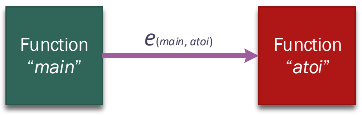


FIGURE 10. An example of $e_{(main,atoi)}$, where "*atoi*" contains a bug

A Module Network $N_m(R) \subset N(R)$ is a subset of Composite Developer-Module Network $N(R)$ in release version R. $N_m(R)$ only includes vertices of software module functions $V_f(R)$, and links between those functions $E_f(R)$. **FIGURE 9** shows an example of a Module Network.

$$N_m(R) = (V_f(R), E_f(R)) \tag{4}$$

For a vertex $v_F \in V_f(R)$ that denotes a function F which exists in the software modules, it has attributes that are defined as:

$$v_F = \{name(F), path(M), B\} \tag{5}$$

Where *name*(F) denotes the name label of the function F, *path*(M) denotes the path of software module where the function is located, and $B \in \{true, false\}$ indicates whether the function contents bug. The function contents bug will be denoted in red. For example, for a function "***atoi***" in module "***stdlib.h***" which contents bug, then the vertex of the function v_{atoi} is noted as:

$$v_{atoi} = \{\text{atoi}, \text{src/stdlib.h}, true\} \tag{6}$$

A link $e_{(F,G)} \in E_f = V_f(R) \times V_f(R)$ is defined as a directed edge from F to G when a function F has a function call on function G. The attributes of the $e_{(F,G)}$ are defined as:

$$e_{(F,G)} = \{name(F), name(G)\} \tag{7}$$

Where *name*(F) denotes the name label of function F, while *name*(G) denotes the name label of function G. Because the edge means "***F* has a function call on *G*,**" the direction should be ***F*→*G***. **FIGURE 10** shows the example of $e_{(main,atoi)}$.

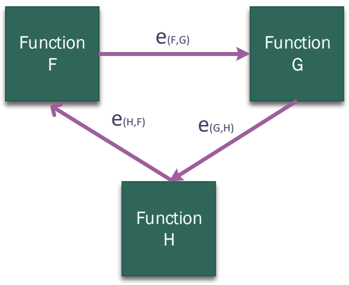


FIGURE 11. An example of an indirect two-way call

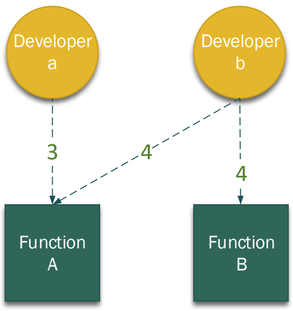


FIGURE 12. An example of Developer Network

Corollary 2. The link between two vertices in a Module Network is weightless. In case that multiple function calls between two specific functions may exist, i.e., if a function F calls function G more than once in the source code, we simply define a link without weight to eliminate confusion.

Corollary 3. The link between two vertices in a Module Network could only be a single unidirectional edge. Since a direct two-way function call is illegal in the sematic of software programs, only a single unidirectional edge is available in our network. However, an indirect two-way call could exist in the network. e.g. $E_f' = \{ e_{(F,G)}, e_{(G,H)}, e_{(H,F)} \}$ (**Error! Reference source not found.**).

A developer Network $N_d(R) \subset N(R)$ is a subset of Composite Developer-Module Network $N(R)$ in release version R. $N_d(R)$ which includes the relationship of developers' contribution to specific functions in software modules before release version R. In other words, a developer has a relationship to the function because he/she has a record of editing the code of the function in the current release or past release. The set of the edges that represent the contribution relation is denoted as $E_c(R)$, and vertex set $V(R)$ includes committer $V_d(R)$ and function $V_f(R)$ as stated in 4.1. Thus, the developer network is defined as:

$$N_d(R) = (V(R), E_c(R)) \quad (8)$$

FIGURE 12 shows an example of a Developer Network.

For a vertex $u_D \in V_d(R)$ that denotes a developer D who has contributions on some functions in $V_f(R)$, it has an attribute that is defined as:

$$u_D = \{name(D)\} \quad (9)$$

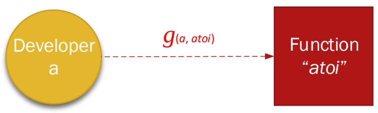


FIGURE 13. An example of $g_{(a,atoi)}$

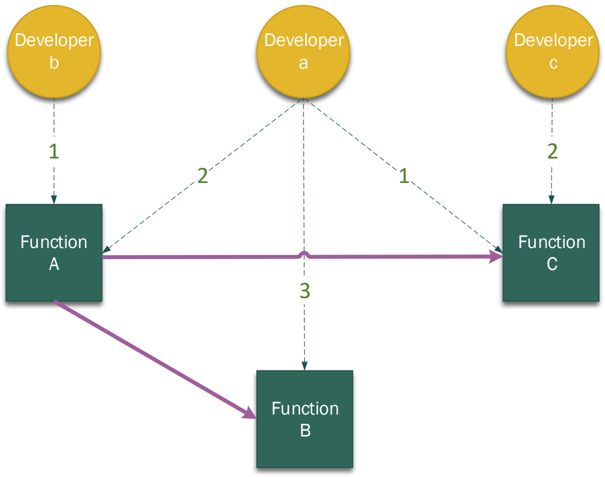


FIGURE 14. An example of Composite Developer-Module Network

Where $name(D)$ is the name label of the developer.

A link $g_{(D,F)} \in E_c(R) = V_d(R) \times V_f(R)$ is defined as a directed edge that represents the relationship when developer D made a commit on function F. The edge should connect a developer vertex u_D and a function vertex v_F. Thus, the attributes of $g_{(D,F)}$ is defined as:

$$g_{(D,F)} = \{name(D), name(F), T, I\} \quad (10)$$

Where $name(D)$ denotes the name label of the developer D, and $name(F)$ denotes the name label of function F. T denotes the commit time of the contribution (last time is selected if D has multiple commits), and $I \in \{true, false\}$ represents whether this commit introduced a bug. Because the edge is defined as "***D* has a contribution on *F*,**" the direction should be ***D*→*F***. Below is an example for such edge that a developer "***a***" made a commit on function "***atoi***":

$$g_{(a,atoi)} = \{\text{a, atoi, 2002/12/13 08:32:10, true}\} \quad (11)$$

FIGURE 13 shows an example of $g_{(a,atoi)}$.

Corollary 4 The edge in a Developer Network can only be a link between developer and function. The definition of edge in a Developer Network represents the dependency of developers and functions in software modules, so the edge can only exist between $V_d(R)$ and $V_f(R)$.

Corollary 5 The link in a Developer Network could only be a single unidirectional edge from developer to function. Function to committer link $V_f(R) \to V_d(R)$ is non-sense and is undefined in the network.

With the combination of Developer Network and Module Network, we could build a comprehensive view of a network that includes both relationships of (1) function-function dependency and (2) developer-function dependency. **FIGURE 14** shows an example of a Composite Developer-Module Network.

Corollary 6 A Composite Developer-Module Network is the union of Developer Network and Module Network. The definition of Developer Network is $N_d(R) = (V(R), E_c(R))$, and that of Module Network is $N_m(R) = (V_f(R), E_f(R))$. Since $V_f(R) \in V(R)$ and $E = E_c(R) \cup E_f(R)$, so $N_d(R) \cup N_m(R) = (V(R), E(R)) = N(R)$.

B. SUB-STRUCTURE OF COMPOSITE DEVELOPER-MODULE NETWORK

A sub-structure of the Composite Developer-Module Network $N_{sub}(R)$ is a sub-structure that can be sliced into an independent network that contents specific vertices $V_{sub}(R) \subset V(R)$ and edges that connect all the vertices $E_{sub}(R) \subset E(R)$. A category of patterns will be identified whether it has a relationship with software bugs. **FIGURE 15** shows an example of such sub-structure retrieval.

In our research, we analyze some specific sub-structure of the network and identify the most fault-prone patterns by calculating the relationship between the sub-structures with the bug-introduced log that derived from several real-world projects.

There are many possible sub-structures that can be pruned from a Composite Developer-Module Network. To compose an objective that is useful in our study, the sub-structure should obey the rules:

1) It consists of at least one developer vertex $u'_D \in V_{d\text{-}sub}(R)$, and a function vertex $v'_F \in V_{f\text{-}sub}(R)$.
2) Every developer vertex in $V_{d\text{-}sub}(R)$ should be connected to at least one function vertex $V_{f\text{-}sub}(R)$ with an edge in $E(R)$, and vice versa.
3) All edges in $E(R)$ consist of vertices in $V_{sub}(R)$ is in $E_{sub}(R)$, i.e., $E_{sub}(R) \subset E(R)$ and $E_{sub}(R)=\{e' \mid e' = V_{sub}(R) \times V_{sub}(R) \text{ and } e' \in E(R)\}$.
4) The sub-structure must be a connected graph.

With respect to the above rules, we can break down two simple forms of sub-structure consists of three vertices below:

- **The three-point *F-D-F form* (FIGURE 16)**: the form consists of one developer and two functions. It represents a developer who has multiple contributions on two different functions in software modules. In this case, there are two variants of the form: the link of the two functions can be absent according to the dependency between them.
- **The three-point *D-F-D form* (FIGURE 17)**: the form consists of two developers and one function. It represents two separated developers who have contributions on the same function in the software modules. In this case, the form has no variant because it can only have two contribution links that are strictly pointing from the two developers to the function.

We can further extend the form of sub-structures consisting of various developers and functions by applying the same rule. To name a few, **FIGURE 18** and **FIGURE 19** shows two different four-point sub-structures, while **FIGURE 20** and **FIGURE 21** show two different five-point sub-structures.

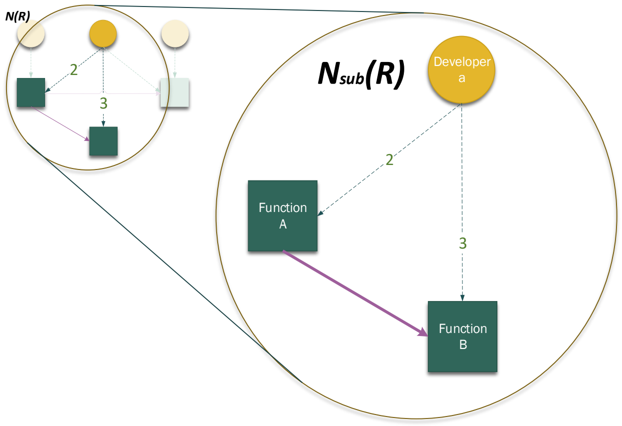


FIGURE 15. An example of sub-structure derived from a completed network

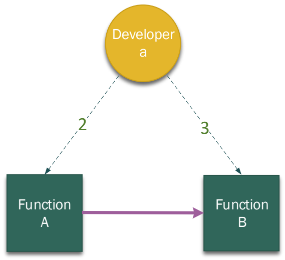


FIGURE 16. An example of *F-D-F Form*

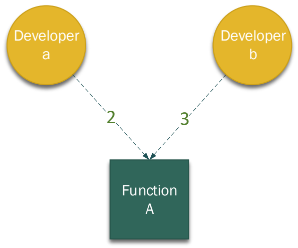


FIGURE 17. An example of *D-F-D Form*

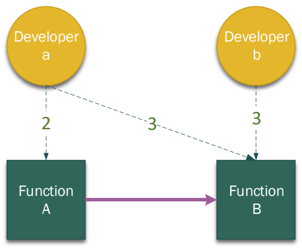


FIGURE 18. A four-point sub-structure

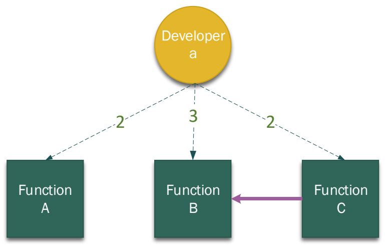


FIGURE 19. A four-point sub-structure

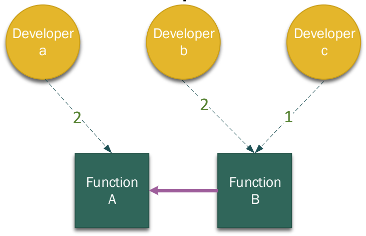


FIGURE 20. A five-point sub-structure

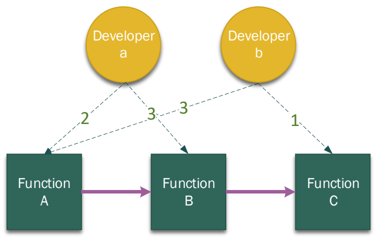


FIGURE 21. A five-point sub-structure

TABLE 2
DESCRIPTIVE INFORMATION FOR THE FOUR SOFTWARE PROJECTS IN THE STUDY

Project	Releases	LOC	Function	Developers	Faults
gedit	314	2012	1387.93	105	58.96
Nagios Core	93	1750	1069.51	22	4.82
NGINX	455	1975	1205.02	48	6.17
redis	173	2350	1177.69	144	57.31

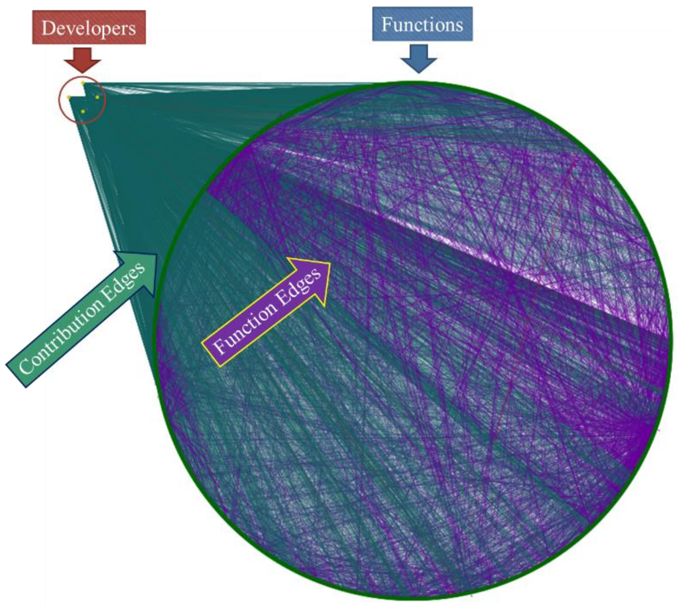


FIGURE 22. Sample graph of CDM network *NGINX: release-1.1.8*

IV. CASE STUDIES

A. EXPERIMENT METHODOLOGY

In this section, we shall discuss the research methodology in more detail. The purpose of our study is to answer the question:

- R (*is different Sub-Structures in the CDMN a factor of fault-proneness?*): Does the structure of Developers/Modules in a sub-graph in a software network effect introducing a bug in a software project?

The answer to the question will determine whether our proposed Composite Developer-Module Network is applicable in building a software fault-proneness analysis model and thus can be used in a prediction model for software fault-proneness in the future.

We use four open-source projects based on C/C++ language, namely: *gedit* [37], *Nagios Core* [84], *NGINX* [86], and *redis* [105], in our study. *gedit* is the default text editor in the GNOME desktop environment with a graphical interface. *Nagios Core* is a monitoring system for networks and infrastructure that has alert features. *NGINX* is a web server that is an asynchronous event-driven approach to handle requests. *redis* is an in-memory database system implementing distributed key-value storage. **TABLE 2** summarizes the descriptive information for the four software projects in our case studies. In the columns, from left to right, given the project name, there is the number of releases, with the maximum number of Line of Code in all functions, the average number of functions for each release, the maximum number of developers among all release, and the average number of faults among all releases. We use the terms *module* and *function* interchangeably in the following part as all the relationships of developers and modules are collected at the function level in the C/C++ language.

FIGURE 22 shows a sample graph of CDM network built by *NGINX* release 1.1.8: the yellow dots on the top left are the vertices of developers, the green ring is many vertices of functions. The green edges are contribution links, and the purple edges are function call links, respectively.

We derived the network into several categories of sub-structures in the study. The number of each sub-structure and the total faulty commits within the structure for all the releases of the target software project are collected. In addition, bugs that exist in each category of sub-structure (i.e., Total Bugs per Structure) are calculated. Bug numbers corresponding to the number of developers, functions, and points (developers and functions) are also calculated to evaluate the impact of these variables.

To describe each Sub-Structure in a text format, we use the terminology of:

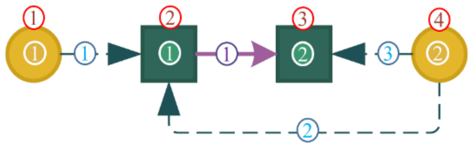


FIGURE 23. An example of Sub-Structure Category: *4-D2(3)-F2(1)*

$$[\textit{total points}]\text{-}D\{\textit{developers}\}\{(\textit{developer links})\}]\text{-}F\{\textit{functions}\}\{(\textit{function links})\} \quad (12)$$

[total points] means the number of total points, both developers and functions. {developers} and {functions} are the number of developers and functions, respectively. If there is only one exist, then the number is omitted for simplicity. {(developer links)} means the developer-function links exist in the structure, and {(function links)} means the function links in the structure. Both numbers are also omitted if only one developer or function is in the structure.

FIGURE 23 shows an example of the category: *4-D2(3)-F2(1)*. Thus, there are four points in the Sub-Structure, with two developers (D2) and two functions (F2). Also, there are three Developer-Function links and one Function link, thus denoted as D2(3) and F2(1).

All the categories of Sub-Structured we used in the study and will be used in the future are listed in the **Error! Reference source not found.** We chose 13 among the categories that have a sufficient amount of data to be collected in this study.

B. RESULTS

This section will present the collected data and analysis from the four projects in our study. 13 Sub-Structures are built by the points, either individual Developers or Function modules, with the links between the points. The 13 Sub-Structures are *01-2-D-F, 02-3-D-F2(0), 03-3-D-F2(1), 04-3-D2-F, 05-4-D-F3(0), 06-4-D-F3(1), 07-4-D-F3(2), 08-4-D-F3(3), 09-4-D2(3)-F2(0), 10-4-D2(3)-F2(1), 11-4-D2(4)-F2(0), 12-4-D2(4)-F2(1), 13-4-D3-F*. However, some Sub-Structures are presented due to the insufficient bug report (*08-4-D-F3(3)* in all four projects), or the Sub-Structure simply not existed (*redis*' *13-4-D3-F*). For the integrity of our analysis, we leave the label on the tables, though they will not be considered in the data analysis.

In the following data sets, six metrics are calculated to emphasize different perspectives on each structure:

- Total Bugs per Structure: the total bug introduction commits per structure from the whole software project all the releases.
- Average Bugs/Release: the average bug introduction commits per the structure of each release of the software project.
- Variance Bugs/Release: the variance of the bug introduction commits per the structure of each release of the software project.
- Average Bugs/Developer: Average Bugs/Release divides by the number of the developers of the structure $V_d(R)$.
- Average Bugs/Function: Average Bugs/Release divides by the number of functions of the structure $V_f(R)$.
- Average Bugs/Point: Average Bugs/Release divides by the number of the developer and the number of the functions of the structure $V(R)$.

Corresponding column graphs for the metrics are also provided for each data set.

TABLE 3 shows the data of each Sub-Structure we found in the *gedit* project from 314 releases of versions. Generally, the trend of fault-prone is growing as the complexity of the Sub-Structure increases, except for the most highly complex one that may be due to the insufficient amount of data. *12-4-D2(4)-F2(1)* is the most fault-prone Sub-Structure in this case. The trend can also be seen in **FIGURE** 24.

TABLE 4 shows the data of *Nagios Core*. Likewise, the more complex, the more fault-prone, especially the *4-D-F3* group, with more than one bug in each structure. **FIGURE 25** also shows a clearer view of the growing faults number.

The trend is more evident in the case of *NIGINX*, as shown in **TABLE 5** and **FIGURE 26**. The bug introductions of the *4-D-F3* group are almost ten times over the *3-D-F2* group. Likewise, in the data in **TABLE 6** and **FIGURE 27,** we can examine the trend of fault-prone leaning towards the more complex ones.

Also, an interesting observation is that the more functions one developer, or a group of developers, works on, the more chance a bug will exist. However, the number of developers, who work on the same set of functions, does not have much effect on the exists of bugs. This may be due to the concept of *personal workload***,** which can be considered as one of our following study topics.

C. DATA ANALYSIS

To answer our research question:

- R (*is different Sub-Structures in the CDMN a factor of fault-proneness?*)

We conducted an Analysis of variance (ANOVA) on the average bug introduction commits per structure to determine whether the dependency of the variant Sub-Structure and prove the result is not random, i.e., the difference in Sub-Structure can affect the fault-proneness. The null hypothesis H_0 and alternative hypothesis H_1 are:

- H_0: All the statistical results of the average bug introduction are identical.
- H_1: All the statistical results of the average bug introduction are not identical.

The result of the test show for all the four projects, the *F-stat*, the F value calculated by the data is far greater than *F-critical*, the F value by the degree of freedom with significant level α = 0.05. Therefore, we can safely say that the research data is valid, and Sub-Structures in the CDMN are a factor of fault-proneness.

TABLE 3
GEDIT (TOTAL RELEASED VERSIONS: 314)

Sub-Structures	Total Structure Counts	Total Bugs-Intro	Total Bugs per Structure	Average Bugs /Release	Variance Bugs /Release	Average Bugs /Developer	Average Bugs /Function	Average Bugs/Point
01-2-D-F	1000075	22481	0.02248	0.03045	0.00164	0.03045	0.03045	0.01522
02-3-D-F2(0)	339336356	13814193	0.04071	0.07798	0.01558	0.07798	0.03899	0.02599
03-3-D-F2(1)	1321142	70750	0.05355	0.07654	0.01208	0.07654	0.03827	0.02551
04-3-D2-F	1218585	68349	0.05609	0.06173	0.00433	0.03086	0.06173	0.02058
05-4-D-F3(0)	807430586	10998688	0.01362	0.00405	0.00122	0.00405	0.00135	0.00101
06-4-D-F3(1)	107403	1995	0.01857	0.02940	0.06424	0.02940	0.00980	0.00735
07-4-D-F3(2)	2282	78	0.03418	0.04684	0.17263	0.04684	0.01561	0.01171
08-4-D-F3(3)	N/A							
09-4-D2(3)-F2(0)	5855	24	0.00410	0.03045	0.00164	0.01522	0.01522	0.00761
10-4-D2(3)-F2(1)	3622	337	0.09304	0.07869	0.03793	0.03935	0.03935	0.01967
11-4-D2(4)-F2(0)	991	24	0.02422	0.02176	0.01190	0.01088	0.01088	0.00544
12-4-D2(4)-F2(1)	925	265	0.28649	0.16507	0.10800	0.08254	0.08254	0.04127
13-4-D3-F	4544	32	0.00704	0.00741	0.00359	0.00247	0.00741	0.00185

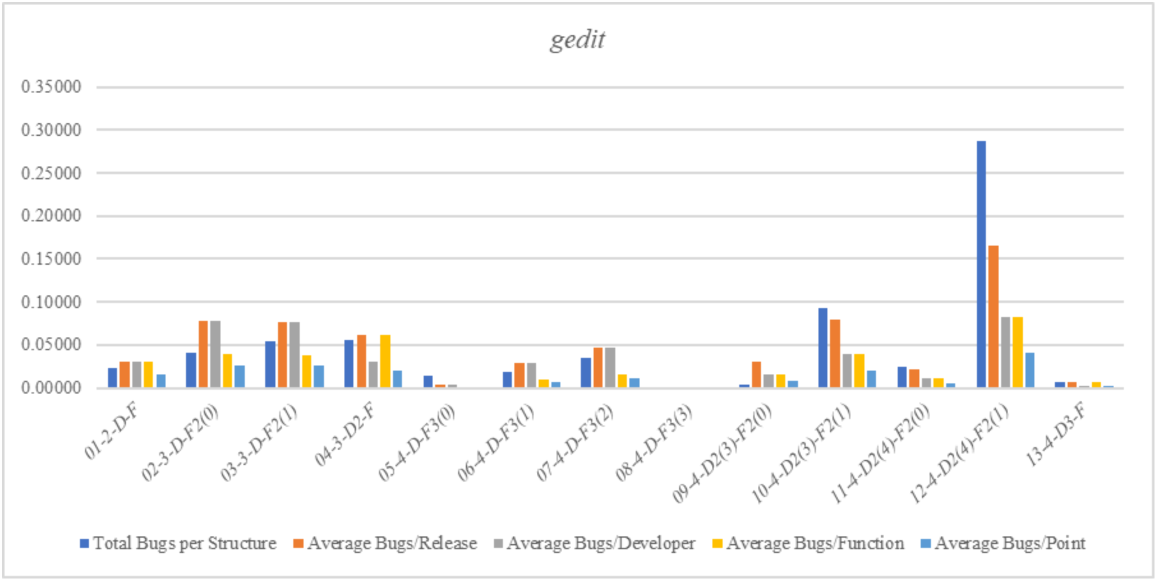


FIGURE 24. Clustered column graph of metrics from the data of *gedit*

TABLE 4
NAGIOS CORE (TOTAL RELEASED VERSIONS: 93)

Sub-Structures	Total Structure Counts	Total Bugs-Intro	Total Bugs per Structure	Average Bugs /Release	Variance Bugs /Release	Average Bugs /Developer	Average Bugs /Function	Average Bugs/Point
01-2-D-F	116960	38478	0.32898	0.48423	0.12293	0.48423	0.48423	0.24211
02-3-D-F2(0)	33281611	24908879	0.74843	1.04016	0.44509	1.04016	0.52008	0.34672
03-3-D-F2(1)	248471	241494	0.97192	1.27540	0.73397	1.27540	0.63770	0.42513
04-3-D2-F	71067	27475	0.38661	0.25661	0.09428	0.12830	0.25661	0.08554
05-4-D-F3(0)	41375464	43244247	1.04517	1.55821	2.66453	1.55821	0.51940	0.38955
06-4-D-F3(1)	62009	89948	1.45056	1.70745	8.67952	1.70745	0.56915	0.42686
07-4-D-F3(2)	1098	2158	1.96539	2.24870	22.97662	2.24870	0.74957	0.56217
08-4-D-F3(3)	N/A							
09-4-D2(3)-F2(0)	177	180	1.01695	0.50968	2.05436	0.25484	0.25484	0.12742
10-4-D2(3)-F2(1)	222	340	1.53153	0.65950	8.22459	0.32975	0.32975	0.16487
11-4-D2(4)-F2(0)	63	62	0.98413	0.51613	2.05680	0.25806	0.25806	0.12903
12-4-D2(4)-F2(1)	165	144	0.87273	0.66667	8.22464	0.33333	0.33333	0.16667
13-4-D3-F	44	37	0.84091	0.13978	1.55633	0.04659	0.13978	0.03495

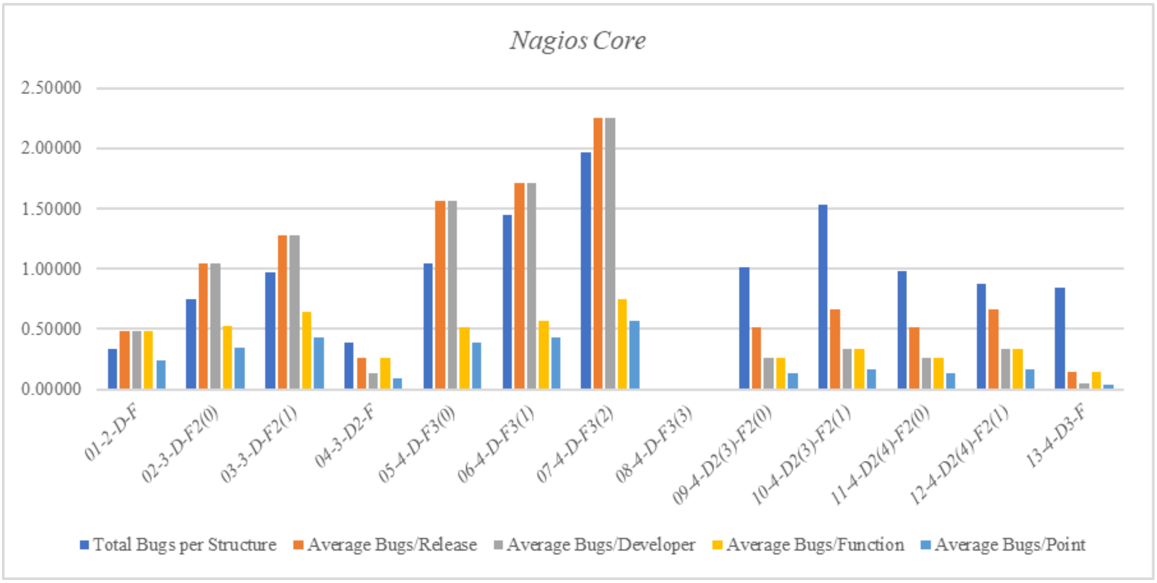


FIGURE 25. Clustered column graph of metrics from the data of *Nagios Core*

TABLE 5
NGINX (TOTAL RELEASED VERSIONS: 455)

Sub-Structures	Total Structure Counts	Total Bugs-Intro	Total Bugs per Structure	Average Bugs /Release	Variance Bugs /Release	Average Bugs /Developer	Average Bugs /Function	Average Bugs/Point
01-2-D-F	723073	86607	0.11978	0.15054	0.00433	0.15054	0.15054	0.07527
02-3-D-F2(0)	343784924	95743692	0.27850	0.30983	0.01528	0.30983	0.15491	0.10328
03-3-D-F2(1)	1324574	468261	0.35352	0.39289	0.02698	0.39289	0.19645	0.13096
04-3-D2-F	317807	33492	0.10538	0.03474	0.00451	0.01737	0.03474	0.01158
05-4-D-F3(0)	620024867	756121870	1.21950	0.80862	0.24929	0.80862	0.26954	0.20216
06-4-D-F3(1)	497389	1452341	2.91993	2.81783	2.19851	2.81783	0.93928	0.70446
07-4-D-F3(2)	2929	11088	3.78559	3.69991	3.93956	3.69991	1.23330	0.92498
08-4-D-F3(3)	N/A							
09-4-D2(3)-F2(0)	658	89	0.13526	0.05934	0.05594	0.02967	0.02967	0.01484
10-4-D2(3)-F2(1)	703	734	1.04410	0.39121	0.95675	0.19560	0.19560	0.09780
11-4-D2(4)-F2(0)	123	27	0.21951	0.05934	0.05594	0.02967	0.02967	0.01484
12-4-D2(4)-F2(1)	236	224	0.94915	0.38681	0.95577	0.19341	0.19341	0.09670
13-4-D3-F	242	26	0.10744	0.05714	0.05400	0.01905	0.05714	0.01429

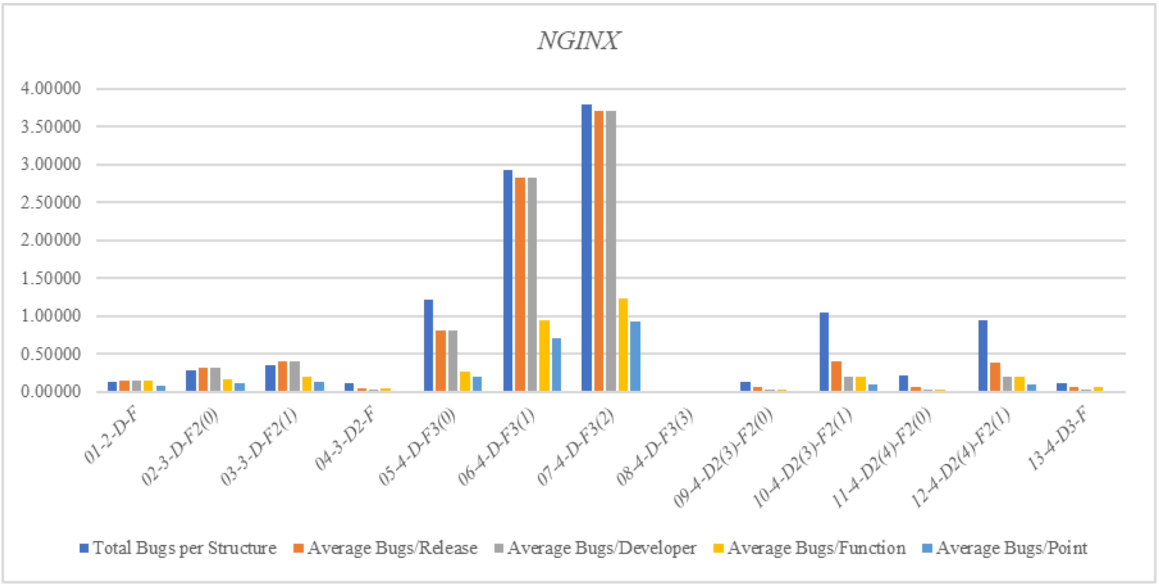


FIGURE 26. Clustered column graph of metrics from the data of *NGINX*

TABLE 6
REDIS (TOTAL RELEASED VERSIONS: 173)

Sub-Structures	Total Structure Counts	Total Bugs-Intro	Total Bugs per Structure	Average Bugs /Release	Variance Bugs /Release	Average Bugs /Developer	Average Bugs /Function	Average Bugs/Point
01-2-D-F	298498	25591	0.08573	0.08417	0.00182	0.08417	0.08417	0.04209
02-3-D-F2(0)	129060969	28140056	0.21804	0.20614	0.01100	0.20614	0.10307	0.06871
03-3-D-F2(1)	621064	151399	0.24377	0.08417	0.00182	0.08417	0.04209	0.02806
04-3-D2-F	176291	24065	0.13651	0.15371	0.00375	0.07685	0.15371	0.05124
05-4-D-F3(0)	529847196	101014328	0.19065	0.67702	0.28596	0.67702	0.22567	0.16926
06-4-D-F3(1)	182665	24273	0.13288	0.14005	0.03247	0.14005	0.04668	0.03501
07-4-D-F3(2)	2895	156	0.05389	0.05740	0.10726	0.05740	0.01913	0.01435
08-4-D-F3(3)	N/A							
09-4-D2(3)-F2(0)	314	127	0.40446	0.26803	0.19987	0.13402	0.13402	0.06701
10-4-D2(3)-F2(1)	594	23	0.03872	0.02890	0.02823	0.01445	0.01445	0.00723
11-4-D2(4)-F2(0)	133	7	0.05263	0.04046	0.09719	0.02023	0.02023	0.01012
12-4-D2(4)-F2(1)	102	3	0.02941	0.01734	0.01714	0.00867	0.00867	0.00434
13-4-D3-F	N/A							

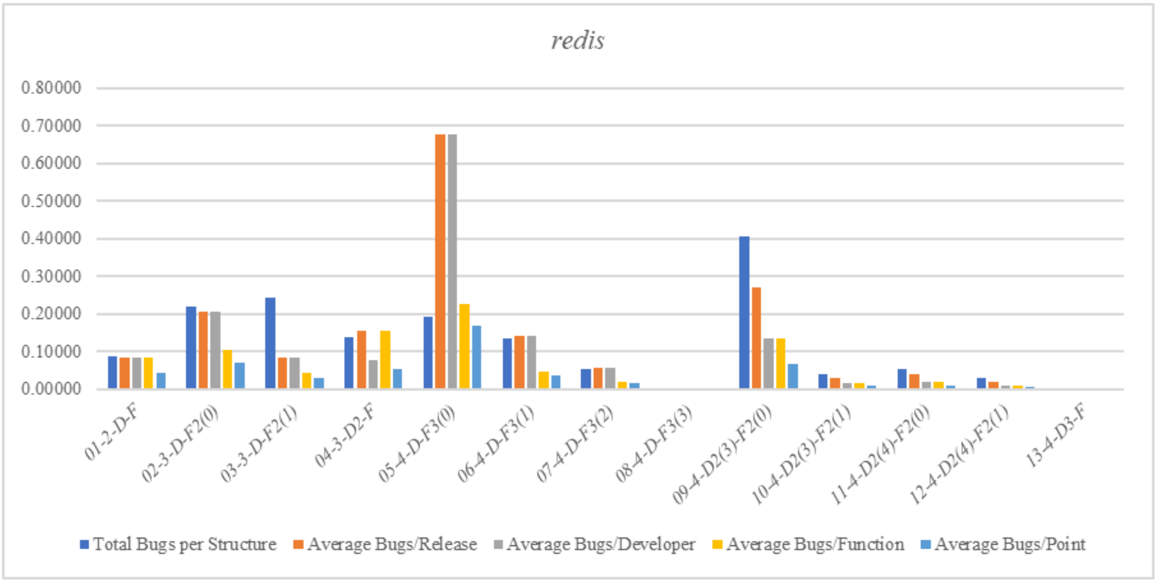


FIGURE 27. Clustered column graph of metrics from the data of redis

V. THREATS TO VALIDITY

In this section, we will discuss some threats that may affect the validity of the study. First, all the programs we analyzed are based on C language. Therefore, there is a chance that our results will not be as effective in projects developed in other programming languages. To alleviate the threat, we can expand our research on different projects with other programming languages involved. Second, only four open-source projects are considered in this research. However, as all total 1026 releases of all the software projects are included, we believe the generality of the study should be satisfied. Third, since the uniqueness of our methodology, very few comparisons with other related techniques could be made to evaluate our research. To improve the threat, we can extend our study to compare some of the general fault-proneness prediction models. Finally, since the issue tracking system we used is maintained by humans, the collected bug information may not satisfy completeness if the maintaining staffs are not responsible. In the study, we have tried to lighten the threat by inspecting each case manually. Nevertheless, if we could further improve our data collection technique and have a more comprehensive data set in the future, and we can conduct a more thorough study on the topic.

VI. RELATED WORK

Bird et al. [8], [10] mention that low ownership of certain software modules, i.e., the modules that have too many minor contributors, can result in a great possibility of software defects. The study suggests that developers should communicate and work carefully with experienced contributors on the objects regarding the desired modification. Ell [33] and Simpson [113] use a Failure Index

(F.I.) as an indicator of the possibility of some specific developer pairs making a mistake, which appears as a primary instance of the appliance to the developer activities in software fault-proneness prediction. Ohira et al. [90] apply social network analysis on developers across different projects, and the results identify that expertise on particular subjects can affect software quality in the development process.

Valverde et al. [124], [125] introduce the concept of network motif into the system software. They find that the frequent network motifs can be a consequence of network heterogeneity. On top of that, they propose a duplication-divergence model that can explain the motif that appears in software evolution. Qian et al. [99] discovered that the nature of software networks could be split into three clusters that match with renowned super-families of network types. However, since the meaning of network motifs in software networks is still unclear, more studies are needed to support the theory as well as the implementation of the software fault-proneness prediction model.

Nagappan et al. [83] propose eight metrics that aim to measure the complexity of software development from an organizational point of view, e.g., the ratio of engineers who left the organization that has modified the codes in some software module. These metrics have proven to be helpful in fault-proneness prediction. In [97], a Developer-Module Network, which only includes the contribution relationship of developers and software modules in their definition, is proposed to build a fault-proneness prediction model. In [145], they further introduce the model using Social Network Analysis on the dependency graph of developers. These studies show that the centrality regarding the developers' contribution has a high correlation to the failure of the software. On top of that, the technique is further extended by operating it on socio-technical networks [9].

Cataldo et al. [21], [22] propose a socio-technical congruence framework that portrays the coordination patterns of developers. The studies suggest that the resolution time of modification requests is drastically reduced for specific coordination patterns. Thus, the impact of working congruence on productivity in the process of software development can be significant. Their studies also indicate that logical dependency is related to product dependency and can have more influence on development quality compared to data dependencies.

VII. CONCLUSION AND FUTURE WORK

In this paper, we propose Composite Developer-Module Networks that feature both software module dependency and developer-module dependency. To build the network, we firstly express all functions of modules and developers involved in the software project into vertices, respectively. Secondly, two different types of dependencies are defined as edges in the network: the software module dependencies, which are decided by the function call between each module, and the developer-module dependencies, which are determined by functions in the software module the developer worked on. After the network is built, we evaluate the bug introduction with the sub-structure derived from the Composite Developer-Module Network and find the most fault-prone one that appears in the network that suggests the most fault-prone pattern during the software development process. We evaluate four open-sourced projects: *gedit*, *Nagios Core*, *NGINX*, and *redis* by constructing Composite Developer-Module Networks for every release version, respectively. Our analysis results have shown that the distinct Sub-Structures in the Composite Developer-Module Networks is a factor in fault-proneness, and the more complex structures can cause more faults in general.

For our future work, we plan to further evaluate our research by applying the method to more software p with a variety of characteristics. More evidence of erroneous patterns can be found through a more comprehensive data set, thus constructing an accurate and effective fault-proneness prediction model. Furthermore, we plan to apply machine learning techniques to our method to discover more potential vulnerable sub-structures automatically. Finally, as the goal of our works, we plan to integrate the technique into a practical software development environment and hope to benefit the massive software industry eventually.

APPENDIX

A table of Sub-Structures analyzed in the study is attached at the end of this paper.

ACKNOWLEDGMENT

This work was funded by the USA National Science Foundation (Grant No. 1757828 and 1822137). Special thanks to Professor Jun Ai and his team at Beihang University, China for providing the raw data of the four open-source projects used in the paper.

REFERENCES

[1] S. Amasaki, Y. Takagi, O. Mizuno, and T. Kikuno, "A Bayesian Belief Network for Assessing the Likelihood of Fault Content," in *Proceedings of the 14th International Symposium on Software Reliability Engineering*, Denver, CO, USA, November 2003, pp. 215-226.

[2] F. D. Anger, J. C. Munson, and R. V. Rodriguez, "Temporal Complexity and Software Faults," in *Proceedings of the 5th IEEE International Symposium on Software Reliability Engineering*, Monterey, CA, USA, November 1994, pp. 115-125

[3] E. Arisholm, "Dynamic Coupling Measures for Objected-Oriented Software," in *Proceedings of the Eighth IEEE International Symposium on Software Metrics*, Ottawa, Canada, June 2002, pp. 33-42.

[4] E. Arisholm and L. C. Briand, "Predicting Fault-Prone Components in A Java Legacy System," in *Proceedings of the Fifth ACM/IEEE International Symposium on Empirical Software Engineering*, Rio de Janeiro, Brazil, September 2006, pp. 8-17.

[5] V. R. Basili, L. C. Briand, and W. L. Melo, "A Validation of Object-oriented Design Metrics as Quality Indicators," *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 751-761, May 1996.

[6] V. R. Basili and D. H. Hutchens, "An Empirical Study of a Syntactic Complexity Family," *IEEE Transactions on Software Engineering*, vol. 9, no. 6, pp. 664-672, November 1983.

[7] M. Bezerra, A. Oliveira, and S. Meira, "A Constructive RBF Neural Network for Estimating the Probability of Defects in Software Modules," in *Proceedings of the 19th International Joint Conference on Neural Networks*, Orlando, FL, USA, August 2007, pp. 2869-2874.

[8] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy, "Does Distributed Development Affect Software Quality? An Empirical Case Study of Windows Vista," in *Proceedings of the 31st International Conference on Software Engineering*, Vancouver, Canada, May 2009, pp. 518-528.

[9] C. Bird, N. Nagappan, H. Gall, B. Murphy, and P. Devanbu, "Putting it all together: using socio-technical networks to predict failures," in *Proceedings of the 20th International Symposium on Software Reliability Engineering*, Toulouse, France, October 2009.

[10] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't Touch My Code!: Examining the Effects of Ownership on Software Quality," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, Szeged, Hungary, September 2011, pp. 4-14.

[11] R. M. Bell, T. J. Ostrand, and E. J. Weyuker, "The limited impact of individual developer data on software defect prediction," *Empirical Software Engineering*, vol. 18, no. 3, pp. 478-505, September 2013.

[12] K. H. Bennett and V. T. Rajlich, "Software maintenance and evolution: a roadmap," in *Proceedings of the Conference on the Future of Software Engineering*, Limerick, Ireland, June 2000, pp. 73-87.

[13] N. Bettenburg and A. Hassan, "Studying the Impact of Social Interactions on Software Quality," *Empirical Software Engineering*, vol. 18, no. 2, pp. 375-431, April 2013.

[14] M. Bezerra, A. Oliveira, P. Adeodato, and S. Meira, "Enhancing RBF-DDA Algorithm's Robustness: Neural Networks Applied to Prediction of Fault-Prone Software Modules," in *Proceedings of the 20th IFIP World Computer Congress*, Milano, Italy, September 2008, pp. 119-128.

[15] S. Biçer, A. B. Bener, and B. Çağlayan, "Defect Prediction Using Social Network Analysis on Issue Repositories," in *Proceedings of the Seventh International Conference on Software and Systems Process*, Honolulu, HI, USA, May 2011, pp. 63-71.

[16] K. Blincoe, G. Valetto, and D. Damian, "Do all task dependencies require coordination? the role of task properties in identifying critical coordination needs in software projects," in *Proceedings of the Ninth Joint Meeting on Foundations of Software Engineering*, August 18-26, 2014, Saint Petersburg, Russia.

[17] A. G. Bluman, *Elementary Statistics, A Step by Step Approach*, Sixth ed. New York, NY: McGraw-Hill, 2007.

[18] C. L. Briand, J. Wüst, J. W. Daly, and D. V. Porter, "Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems," *Journal of Systems and Software*, vol.51, no.3, pp. 245-273, May 2000.

[19] C. Catal, U. Sevim, and B. Diri, "Practical Development of An Eclipse-based Software Fault Prediction Tool using Naive Bayes Algorithm," *Expert Systems with Applications*, vol. 38, no. 3, pp. 2347-2353, March 2011.

[20] M. Cataldo and J. D. Herbsleb, "Coordination breakdowns and their impact on development productivity and software failures," *IEEE Transactions on Software Engineering*, vol. 39, no. 3, pp.343-360, March 2013.

[21] M. Cataldo, J. D. Herbsleb, and K. M. Carley, "Socio-Technical Congruence: A Framework for Assessing the Impact of Technical and Work Dependencies on Software Development Productivity," in *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, Kaiserslautern, Germany, October 2008, pp. 2-11.

[22] M. Cataldo, A. Mockus, J. A. Roberts, and J. D. Herbsleb, "Software Dependencies, Work Dependencies, and Their Impact Failures," *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 864-878, November 2009.

[23] M.-H. Chen, M. R. Lyu, and W. E. Wong. "An Empirical Study of The Correlation Between Code Coverage and Reliability Estimation," In *Proceedings of the 3rd International Software Metrics Symposium*, March 1996, pp. 133-141, IEEE.

[24] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476-493, June 1994.

[25] S. D. Conte, H. E. Dunsmore, and Y. E. Shen, *Software Engineering Metrics and Models*, 1st ed. San Francisco, California: Benjamin-Cummings Publishing Co., Inc., 1986.

[26] C. R. B. De Souza and D. F. Redmiles, "The awareness network, to whom should I display my actions? And, whose actions should I monitor?" *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 325-340, May-June 2011.

[27] T. DeMarco, *Controlling Software Projects: Management, Measurement & Estimation*, 1st ed. Englewood Cliffs, New Jersey: Prentice Hall, June 1986.

[28] K. Dejaeger, T. Verbraken, and B. Baesens, "Toward Comprehensible Software Fault Prediction Models Using Bayesian Network Classifiers," *IEEE Transactions on Software Engineering*, vol. 39, no. 2, pp. 237-257, February 2013.

[29] G. Denaro, S. Morasca, and M. Pezzè, "Deriving Models of Software Fault-Proneness," in *Proceedings of the 14th ACM International Conference on Software Engineering and Knowledge Engineering*, Ischia, Italy, July 2002, pp. 361-368.

[30] G. Denaro and M. Pezzè, "An Empirical Evaluation of Fault-Proneness Models," in *Proceedings of the 24rd IEEE International Conference on Software Engineering*, Limerick, Ireland, May 2002, pp. 241-251.

[31] K. El Emam, W. Melo, and J. C. Machado, "The Prediction of Faulty Classes Using Object-Oriented Design Metrics," *Journal of Systems and Software*, vol. 56, no. 1, pp. 63-75, February 2001.

[32] K. O. Elish and M. O. Elish, "Predicting Defect-Prone Software Modules Using Support Vector Machines," *Journal of Systems and Software*, vol. 81, no. 5, pp. 649-660, 2008.

[33] J. Ell, "Identifying failure inducing developer pairs within developer networks," In *Proceedings of the 35th International Conference on Software Engineering*, San Francisco, CA, USA, May 2013, pp. 1471-1473.

[34] M. Fan, J. Liu, X. Luo, K. Chen, T. Chen, Z. Tian, X. Zhang, Q. Zheng, and T. Liu, "Frequent Subgraph Based Familial Classification of Android Malware," In *Proceedings of the IEEE 27th International Symposium on Software Reliability Engineering*, Ottawa, Canada, October 2016, pp. 24-35.

[35] M. Gegick and L. Williams, "Toward the Use of Automated Static Analysis Alerts for Early Identification of Vulnerability- and Attack-Prone Components," in *Proceedings of the Second International Conference on Internet Monitoring and Protection*, San Jose, CA, USA, July 2007.

[36] R. A. Ghosh, "Clustering and Dependencies in Free/Open Source Software Development: Methodology and Tools," *First Monday*, vol. 8, no. 4, April 2003.

[37] GNOME gedit, https://wiki.gnome.org/Apps/Gedit

[38] B. Goel and Y. Singh, "Empirical Investigation of Metrics for Fault Prediction on Object-Oriented Software," *Computer and Information Science*, pp. 255-265, Springer Berlin Heidelberg, 2008.

[39] I. Gondra, "Applying Machine Learning to Software Fault-Proneness Prediction," *Journal of Systems and Software*, vol. 81, vol. 2, pp. 186-195, February 2008.

[40] A. Halim, "Predict Fault-Prone Classes using the Complexity of UML Class Diagram," in *Proceedings of the 1st IEEE International Conference on Computer, Control, Informatics and Its Applications*, Jakarta, Indonesia, November 2013, pp. 289-294.

[41] A. E. Hassan, "Predicting Faults Using the Complexity of Code Changes," in *Proceedings of the 31st IEEE International Conference on Software Engineering*, Vancouver, Canada, May 2009, pp. 78-88.

[42] J. Howison, K. Inoue, and K. Crowston, "Social Dynamics of Free and Open Source Team Communication," in *Proceedings of the IFIP Second International Conference on Open Source Systems*, Lake Como, Italy, June 2006 pp. 319- 330.

[43] F. Huang, "Software Fault Defense based on Human Errors," Ph.D., School of Reliability and Systems Engineering, Beihang University, Beijing, 2013.

[44] F. Huang, B. Liu, and B. Huang, "A Taxonomy System to Identify Human Error Causes for Software Defects," in *Proceedings of the 18th International Conference on Reliability and Quality in Design*, Boston, MA, USA, August 2012, pp. 44-49.

[45] F. Huang, B. Liu, Y. Song, and S. Keyal, "The links between human error diversity and software diversity: Implications for fault diversity seeking," *Science of Computer Programming*, vol. 89, Part C, pp. 350-373, September 2014.

[46] F. Huang, B. Liu, S. Wang, and Q. Li, "The impact of software process consistency on residual defects," *Journal of Software: Evolution and Process*, vol. 27, No. 9, pp. 625-646, September 2015.

[47] F. Huang, B. Liu, and Y. Wang, "Review of Software Psychology," *Computer Science*, vol. 40, pp. 1-7, March 2013.

[48] Y. Jiang, B. Cuki, T. Menzies, and N. Bartlow, "Comparing Design and Code Metrics for Software Quality Prediction," in *Proceedings of the Fourth International Workshop on Predictor Models in Software Engineering*, Leipzig, Germany, May 2008, pp. 11-18.

[49] T. Kamiya, S. Kusumoto, and K. Inoue, "Prediction of Fault-Proneness at Early Phase in Object-Oriented Development," in *Proceedings of the Second IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, Saint-Malo, France, May 1999, pp. 253-258.

[50] T. M. Khoshgoftaar and E. B. Allen, "Classification of Fault-Prone Software Modules: Prior Probabilities, Costs, and Model Evaluation," *Empirical Software Engineering*, vol. 3, no. 3, pp. 275-298, 1998.

[51] T. M. Khoshgoftaar, E. B. Allen, K. S. Kalaichelvan, and N. Goel, "Early quality prediction: A case study in telecommunications," *IEEE Software*, vol. 13, no. 1, pp. 65-71, January 1996.

[52] T. M. Khoshgoftaar and R. M. Szabo, "Using Neural Networks to Predict Software Faults During Testing," *IEEE Transactions on Reliability*, vol. 45, no. 3, pp. 456-462, September 1996.

[53] S. Kim, E. J. Whitehead Jr., and Y. Zhang, "Classifying Software Changes: Clean or Buggy?" *IEEE Transactions on Software Engineering*, vol. 34, no. 2, pp. 181-196, March 2008.

[54] P. Knab, M. Pinzger, and A. Bernstein, "Predicting Defect Densities in Source Code Files with Decision Tree Learners," in *Proceedings of the Fourth International Workshop on Mining Software Repositories*, Shanghai, China, May 2006, pp. 119-125.

[55] A. G. Koru and J. Tian, "An Empirical Comparison and Characterization of High Defect and High Complexity Modules," *Journal of System and Software*, vol. 67, no. 3, pp. 153-163, September 2003.

[56] A. G. Koru and J. Tian, "Comparing High-Change Modules and Modules with the Highest Measurement Values in Two Large-Scale Open-Source Products," *IEEE Transactions on Software Engineering*, vol. 31, no. 8, pp. 625-642, August 2005.

[57] I. Kwan, A. Schroter, and D. Damian, "Does Socio-Technical Congruence Have an Effect on Software Build Success? A Study of Coordination in a Software Project," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 307-324, May 2011.

[58] L. Layman, G. Kudrjavets, and N. Nagappan, "Iterative Identification of Fault-Prone Binaries Using In-Process Metrics," in *Proceedings of the Second ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, Kaiserslautern, Germany, October 2008, pp. 206-212.

[59] S.Y. Lee, D. Li, and Y. Li, "An Investigation of Essential Topics on Software Fault-Proneness Prediction," in *Proceedings of the 2016 IEEE International Symposium on System and Software Reliability*, Shanghai, China, October 2016, pp. 37-46.

[60] S.Y. Lee and Y. Li, "DRS: A Developer Risk Metric for Better Predicting Software Fault-Proneness," in *Proceedings of the Second International Conference on Trustworthy Systems and Their Applications*, Hualien, Taiwan, July 2015, pp. 120-127.

[61] B. Lennselius, C. Wohlin, and C. Vrana, "Software Metrics: Fault Content Estimation and Software Process Control," *Microprocessors and Microsystems*, vol. 7, no. 35, September 1987, pp. 365-375.

[62] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485-496, August 2008.

[63] H. Li and W. K. Cheung, "An Empirical Study of Software Metrics," *IEEE Transactions on Software Engineering*, vol. 13, no. 6, pp. 697-708, June 1987.

[64] W. Li and S. Henry, "Object-Oriented Metrics that Predict Maintainability," *Journal of Systems and Software*, vol. 23, no. 2, pp. 111-122, November 1993.

[65] Y. Li, W. E. Wong, S.Y. Lee, and F. Wotawa, "Using Tri-Relation Networks for Effective Software Fault-Proneness Prediction," *IEEE Access*, vol. 7, pp. 63066-63080, May 2019.

[66] Y. Li, D. Li, F. Huang, S.Y. Lee, and J. Ai, C. Yang, and D. Li, "An Exploratory Analysis on Software Developers' Bug-Introducing Tendency over Time," in *Proceedings of the 2016 Annual Conference on Software Analysis, Testing and Evolution*, Kunming, China, November 2016, pp.12-17.

[67] Z. Lin, Q. Gunqun, and Z. Li, "Network motif and triad significance profile research on software system," *Systems Engineering - Theory & Practice*, vol. 30, no. 2, pp. 361-368, February 2010.

[68] Y. Ma and C. Bojan, "Adequate and precise evaluation of quality models in software engineering studies," in *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, Minneapolis, MN, USA, May 2007.

[69] Y. Ma, K. He, and J. Liu, "Network Motifs in Object-Oriented Software Systems," *Dynamics of Continuous, Discrete and Impulsive Systems (Series B: Applications & Algorithms)*, vol. 14, no. S6, pp. 166-172, 2007.

[70] S. Matsumoto, Y. Kamei, A. Monden, K. Matsumoto, and M. Nakamura, "An analysis of developer metrics for fault prediction," in *Proceedings of the Sixth International Conference on Predictive Models in Software Engineering*, Timisoara, Romania, September 2010.

[71] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308-320, March 1976.

[72] A. Meneely, M. Corcoran, and L. Williams, "Improving developer activity metrics with issue tracking annotations," In *Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics*, Cape Town, South Africa, May 2010.

[73] A. Meneely and L. Williams, "Socio-Technical Developer Networks: Should We Trust Our Measurements?" in *Proceedings of the 33rd International Conference on Software Engineering*, Honolulu, HI, USA, May 2011, pp. 281-290.

[74] A. Meneely, L. Williams, W. Snipes, and J. Osborne, "Predicting Failures with Developer Networks and Social Network Analysis," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Atlanta, GA, USA, November 2008, pp. 13-23.

[75] T. Menzies, J. Greenwald, and A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictors," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 2-13, January 2007.

[76] T. Menzies, J. Di Stefano, M. Chapman, and K. McGill, "Metrics that Matter," in *Proceedings of the 27th Annual NASA Goddard Software Engineering Workshop*, Greenbelt, MD, USA, December 2002, pp. 51-57.

[77] A. Mitchell and J. F. Power, "An Approach to Quantifying the Runtime Behaviour of Java GUI Applications," in *Proceedings of the 2004 Winter International Symposium on Information and Communication Technologies*, Cancun, Mexico, January 2004, pp.1-6.

[78] P. Mittal, S. Singh, and K. S. Kahlon, "Identification of Error Prone Classes for Fault Prediction Using Object Oriented Metrics," *Advances in Computing and Communications*, pp. 58-68. Springer Berlin Heidelberg, 2011.

[79] A. Mockus, "Organizational volatility and its effects on software defects," in *Proceedings of the eighth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, San Diego, CA, USA, November 2010.

[80] R. Moser, W. Pedrycz, and G. Succi, "A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction," in *Proceedings of the 30th International Conference on Software Engineering*, Leipzig, Germany, May 2008, pp. 181-190.

[81] J. C. Munson and S. Elbaum, "Code Churn: A Measure for Estimating the Impact of Code Change," in *Proceedings of the 13th*

International Conference on Software Maintenance, Bethesda, MD, USA, November 1998, pp. 24-31.

[82] C. Myers, "Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs", *Physical Review E*, vol. 68, no. 4, October 2003.

[83] N. Nagappan, B. Murphy, and V. Basili, "The influence of organizational structure on software quality: an empirical case study," In *Proceedings of the 30th International Conference on Software Engineering*, Leipzig, Germany, May 2008.

[84] Nagios Core, https://www.nagios.org/projects/nagios-core/

[85] National Institute of Standards and Technology, "The Economic Impacts of Inadequate Infrastructure for Software Testing," *National Institute of Standards and Technology (NIST) Planning Report 02-3*, May 2002.

[86] NGINX, https://nginx.org/

[87] T. Nguyen, B. Adams, and A. Hassan, "Studying the Impact of Dependency Network Measures on Software Quality," in *Proceedings of the 26th IEEE International Conference on Software Maintenance*, Timisoara, Romania, September 2010, pp. 1-10.

[88] V. Nguyen, S. Deeds-Rubin, T. Tan, and B. Boehm, "A SLOC counting standard," in *Proceedings of the 22nd International Forum on COCOMO and Systems/Software Cost Modeling*, Los Angeles, CA, USA, October 2007.

[89] A. Nugroho, M. Chaudron, and E. Arisholm, "Assessing UML Design Metrics for Predicting Fault-Prone Classes in A Java System," in *Proceedings of the Seventh IEEE Working Conference on Mining Software Repositories*, Cape Town, South Africa, May 2010, pp. 21-30.

[90] M. Ohira, N. Ohsugi, T. Ohoka, and K. Matsumoto, "Accelerating Cross-project Knowledge Collaboration Using Collaborative Filtering and Social Networks," in *Proceedings of the Second International Workshop on Mining Software Repositories*, Saint Louis, MO, USA, May 2005, pp. 1-5.

[91] N. Ohlsson, and H. Alberg, "Predicting Fault-Prone Software Modules in Telephone Switches," *IEEE Transactions on Software Engineering*, vol. 22, no.12, pp. 886-894, December 1996.

[92] N. Ohlsson, M. Zhao, and M. Helander, "Application of Multivariate Analysis for Software Fault Prediction," *Software Quality Journal*, vol. 7, no.1, pp. 51-66, 1998.

[93] A. Okutan and O. T. Yildiz, "Software Defect Prediction using Bayesian Networks," *Empirical Software Engineering*, vol.19, no. 1, pp. 154-181, February 2014.

[94] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Predicting the Location and Number of Faults in Large Software Systems," *IEEE Transactions on Software Engineering*, vol. 31, no. 4, pp. 340-355, April 2005.

[95] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Programmer-based fault prediction," in *Proceedings of the Sixth International Conference on Predictive Models in Software Engineering*, Timisoara, Romania, September 2010.

[96] G. J. Pai and J. B. Dugan, "Empirical Analysis of Software Fault Content and Fault Proneness Using Bayesian Methods," *IEEE Transactions on Software Engineering*, vol. 33, no. 10, pp. 675-686, October 2007.

[97] M. Pinzger, N. Nagappan, and B. Murphy, "Can developer module networks predict failures?" in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Atlanta, GA, USA, November 2008, pp. 2-12.

[98] R. Premraj and K. Herzig, "Network Versus Code Metrics to Predict Defects: A Replication Study," in *Proceedings of the Fifth International Symposium on Empirical Software Engineering and Measurement*, Banff, Canada, September 2011.

[99] G. Qian, Z. Lin and Z. Li, "Applying Complex Network Method to Software Clustering," in *Proceedings of the 2008 International Conference on Computer Science and Software Engineering*, Wuhan, China, December 2008.

[100] J. R. Quinlan, "Induction of Decision Tress," *Machine Learning*, vol. 1, no. 1, pp. 81-106, March 1986.

[101] D. Radjenovic, M. Hericko, R. Torkar, and A. Zivkovic, "Software Fault Prediction Metrics: A Systematic Literature Review," *Information and Software Technology*, vol. 55, no. 8, pp. 1397-1418, August 2013.

[102] F. Rahman and P. Devanbu, "How, and Why, Process Metrics Are Better," in *Proceedings of the 35th International Conference on Software Engineering*, San Francisco, CA, May 2013, pp. 432-441.

[103] F. Rahman and D. Premkumar, "Ownership, experience and defects: a fine-grained study of authorship," in *Proceedings of the 33rd International Conference on Software Engineering*, Honolulu, HI, May 2011.

[104] S. S. Rathore and A. Gupta, "Investigating Object-Oriented Design Metrics to Predict Fault-Proneness of Software Modules," in *Proceedings of the CSI Sixth International Conference on Software Engineering*, Madhya Pradesh, India, September 2012, pp. 1-10.

[105] Redis, https://redis.io/

[106] R. Robbes and D. Röthlisberger, "Using developer interaction data to compare expertise metrics," in *Proceedings of the Tenth International Workshop on Mining Software Repositories*, San Francisco, CA, USA, May 2013.

[107] A. Sarma, L. Maccherone, P. Wagstrom, and J. Herbsleb, "Tesseract: Interactive Visual Exploration of Socio-Technical Relationships in Software Development," in *Proceedings of the 31st IEEE International Conference on Software Engineering*, Vancouver, Canada, May 2009, pp. 22-33.

[108] Scitools Understand, http://scitools.com/features/

[109] R. Shatnawi and W. Li, "The Effectiveness of Software Metrics in Identifying Error-Prone Classes in Post-Release Software Evolution Process," *Journal of Systems and Software*, vol. 81, no. 11, pp. 1868-1882, November 2008.

[110] S. Shen-Orr, R. Milo, S. Mangan and U. Alon, "Network motifs in the transcriptional regulation network of Escherichia coli," *Nature Genetics*, vol. 31, no. 1, pp. 64-68, May 2002.

[111] M. Shepperd and D. Ince, *Derivation and Validation of Software Metrics*. Oxford, United Kingdom: Clarendon Press, 1993.

[112] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, "Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities," *IEEE Transactions on Software Engineering*, vol. 37, no. 6, pp. 772-787, December 2011.

[113] B. Simpson, "Changeset based Developer Communication to Detect Software Failures," in *Proceedings of the 35th IEEE International Conference on Software Engineering*, San Francisco, CA, USA, May 2013, pp. 1468-1470.

[114] S. Singh and K. S. Kahlon, "Effectiveness of Encapsulation and Object-Oriented Metrics to Refactor Code and Identify Error Prone Classes Using Bad Smells," *ACM SIGSOFT Software Engineering Notes*, vol. 36, no. 5, pp. 1-10, September 2011.

[115] S. Singh and K. S. Kahlon, "Effectiveness of Refactoring Metrics Model to Identify Smelly and Error Prone Classes in Open Source Software," *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 2, pp. 1-11, March 2012.

[116] Y. Singh, A. Kaur, and R. Malhotra, "Empirical Validation of Object-Oriented Metrics for Predicting Fault Proneness Models," *Software Quality Journal*, vol. 18, no. 1, pp. 3-35, March 2010.

[117] J. Sliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?", *SIGSOFT Software Engineering Notes*, vol. 30, no. 4, p. 1, July 2005.

[118] D. E. Strode and S. L. Huff, "A taxonomy of dependencies in agile software development," in *Proceedings of the 23rd Australasian Conference on Information Projects*, Geelong, Australia, December 2012.

[119] R. Takahashi, "Software Quality Classification Model based on McCabe's Complexity Measure," *Journal of Systems and Software*, vol. 38, no. 1, pp. 61-69, July 1997.

[120] M. M. T. Thwin and T. Quah, "Application of Neural Networks for Software Quality Prediction Using Object-Oriented Metrics," *Journal of Systems and Software*, vol. 76, no. 2, pp. 147-156, May 2005.

[121] P. Tomaszewski, L. Lundberg, and H. Grahn, "Improving Fault Detection in Modified Code: A Study from the Telecommunication Industry," *Journal of Computer Science and Technology*, vol. 22, no. 3, pp. 397-409, May 2007.

[122] A. Tosun, B. Turhan, and A. Bener, "Validation of Network Measures as Indicators of Defective Modules in Software Systems," in *Proceedings of the Fifth International Conference on Predictor*

Models in Software Engineering, Vancouver, Canada, May 2009, pp. 5:1-5:9.

[123] B. Twala, "Predicting Software Faults in Large Space Systems Using Machine Learning Techniques," *Defence Science Journal*, vol. 61, no. 4, pp. 306-316, July 2011.

[124] S. Valverde, R. Cancho, and R. Solé, "Scale-free networks from optimal design," *Europhysics Letters (EPL)*, vol. 60, no. 4, pp. 512-517, November 2002.

[125] S. Valverde and R. Solé, "Network motifs in computational graphs: A case study in software architecture," *Physical Review E*, vol. 72, no. 2, April 2005.

[126] G. M. Weinberg, *The Psychology of Computer Programming*: VNR Nostrand Reinhold Company, 1971.

[127] E. J. Weyuker, T. J. Ostrand, and R. M. Bell, "Do too many cooks spoil the broth? using the number of developers to enhance defect prediction models," *Empirical Software Engineering*, vol. 13, no. 5, pp. 539-559, 2008.

[128] E. J. Weyuker, T. J. Ostrand, and R. M. Bell, "Comparing the Effectiveness of Several Modeling Methods for Fault Prediction," *Empirical Software Engineering*, vol. 15, no. 3, pp. 277-295, 2010.

[129] T. Wolf, A. Schroter, D. Damian, and T. Nguyen, "Predicting Build Failures Using Social Network Analysis on Developer Communication," in *Proceedings of the 31st International Conference on Software Engineering*, Vancouver, Canada, May 2009, pp. 1-11.

[130] S. Wong and Y. Cai, "Generalizing evolutionary coupling with stochastic dependencies," in *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering*, Lawrence, KS, USA, November 2011.

[131] W. E. Wong, V. Debroy, A. Surampudi, H. Kim, and M. F. Siok. "Recent Catastrophic Accidents: Investigating How Software Was Responsible," In *Proceedings of the Fourth International Conference on Secure Software Integration and Reliability Improvement*, June 2010, pp. 14-22, IEEE.

[132] W. E. Wong, J. R. Horgan, M. Syring, W. Zage, and D. M. Zage. "Applying Design Metrics to A Large-Scale Software System," In *Proceedings of the Ninth International Symposium on Software Reliability Engineering* (Cat. No. 98TB100257) , November 1998, pp. 273-282, IEEE.

[133] W. E. Wong, J. R. Horgan, M. Syring, W. M. Zage, and D. M. Zage, "Applying Design Metrics to Predict Fault-Proneness: A Case Study on Large-Scale Software System," *Software: Practice and Experience*, vol. 30, no .14, pp. 1587-1608, November 2000.

[134] W. E. Wong, X. Li, P. A. Laplante, "Be More Familiar with Our Enemies and Pave the Way Forward: A Review of the Roles Bugs Played in Software Failures," *Journal of Systems and Software*, vol. 133, pp. 68-94, November 2017.

[135] W. E. Wong, Y. Qi, and K. Cooper, "Source Code-Based Software Risk Assessing," in *Proceedings of the 20th ACM Symposium on Applied Computing*, Santa Fe, NM, USA, March 2005, pp. 1485-1490.

[136] F. Wu, "Empirical Validation of Object-Oriented Metrics on NASA for Fault Prediction," *Advances in Information Technology and Education*, pp. 168-175. Springer Berlin Heidelberg, 2011.

[137] Y. Wu, Y. Yang, Y. Zhao, H. Liu, Y. Zhou, and B. Xu, "The Influence of Developer Quality on Software Fault-Proneness Prediction," in *Proceedings of the Eighth International Conference on Software Security and Reliability*, San Francisco, CA, USA, June-July 2014, pp. 11-19.

[138] F. Xing, P. Guo, and M. R. Lyu, "A Novel Method for Early Software Quality Prediction Based on Support Vector Machine," in *Proceedings of the 16th International Symposium on Software Reliability Engineering*, Chicago, IL, November 2005, pp. 213-222.

[139] J. Xu, Y. Gao, S. Christley, and G. Madey, "A Topological Analysis of the Open Source Software Development Community," in *Proceedings of the 38th IEEE Annual Hawaii International Conference on System Sciences*, Big Island, HI, USA, January 2005, pp. 198-208.

[140] P. Yu, H. Muller, and T. Systa, "Predicting Fault-Proneness Using O.O. Metrics: An Industrial Case Study," in *Proceedings of the Sixth European Conference on Software Maintenance and Reengineering*, Budapest, Hungary, March 2002, pp. 99-107.

[141] S. Zhang, J. Ai, and X. Li, "Correlation Between the Distribution of Software Bugs and Network Motifs," In *Proceedings of the 2016 IEEE International Conference on Software Quality, Reliability and Security*, Vienna, Austria, August 2016, pp. 202-213.

[142] J. Zhou, P. S. Sandhu, and S. Rani, "A Neural Network Based Approach for Modeling of Severity of Defects in Function Based Software Systems," in *Proceedings of 1st International Conference on Electronics and Information Engineering*, Kyoto, Japan, August 2010, pp. 568-575.

[143] Y. Zhou and H. Leung, "Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults," *IEEE Transactions on Software Engineering*, vol. 32, no. 10, pp. 771-789, October 2006.

[144] Y. Zhou, B. Xu, and H. Leung, "On the Ability of Complexity Metrics to Predict Fault-Prone Classes in Object-Oriented Systems," *Journal of Systems and Software*, vol. 83, no. 4, pp. 660-674, April 2010.

[145] T. Zimmermann and N. Nagappan, "Predicting Defects Using Social Network Analysis on Dependency Graphs," in *Proceedings of the 30th IEEE International Conference on Software Engineering*, Leipzig, Germany, May 2008, pp. 531-5.

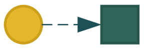
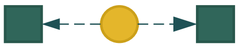
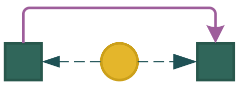
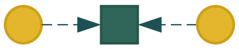
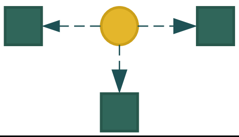
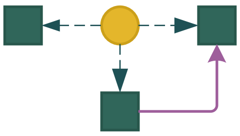
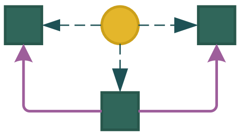
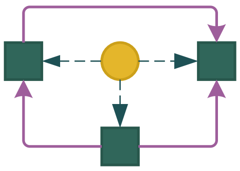
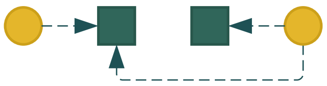
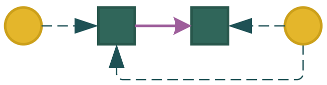
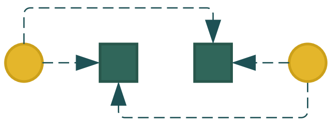
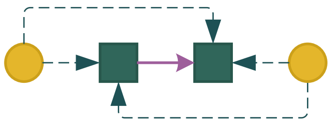
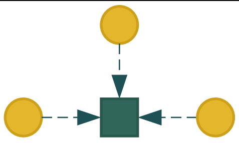
SHOU-YU LEE received the B.S. degree in computer science from National Tsing Hua University, Taiwan, and the M.S. degree in computer science from Tunghai University. He is currently pursuing a Ph.D. degree with The University of Texas at Dallas, Richardson, TX, USA, under the supervision of Prof. W. E. Wong. His current research interests include software fault localization, context-sensitive computing, and software risk analysis. (Email: sxl128630@utdallas.edu)

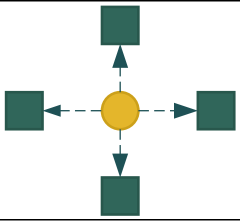
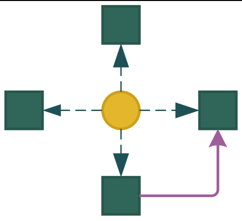
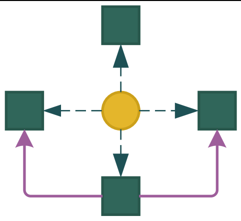
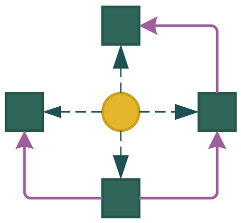
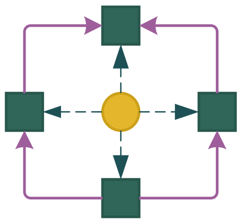
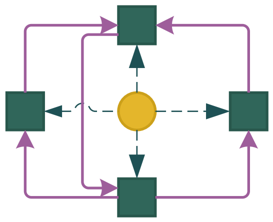
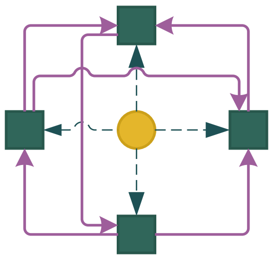
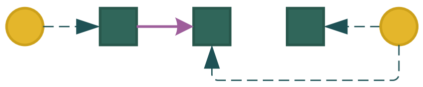
W. ERIC WONG received the M.S. and Ph.D. degrees in computer science from Purdue University. He is a full professor and the founding director of the Advanced Research Center for Software Testing and Quality Assurance in Computer Science, University of Texas at Dallas (UTD). He also has an appointment as a guest researcher with National Institute of Standards and Technology (NIST), an agency of the US Department of Commerce. Prior to joining UTD, he was with Telcordia Technologies (formerly Bellcore) as a senior research scientist and the project manager in charge of Dependable Telecom Software Development. In 2014, he was named the IEEE Reliability Society Engineer of the Year. His research focuses on helping practitioners improve the quality of software while reducing the cost of production. In particular, he is working on software testing, debugging, risk analysis/metrics, safety, and reliability. He has very strong experience developing real-life industry applications of his research results. Professor Wong is the Editor-in-Chief of IEEE TRANSACTIONS ON RELIABILITY. He is also the Founding Steering Committee Chair of the IEEE International Conference on Software Quality, Reliability, and Security (QRS). (Email: ewong@utdallas.edu)

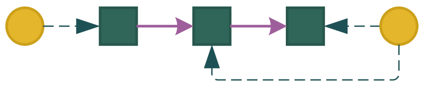
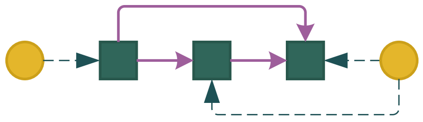
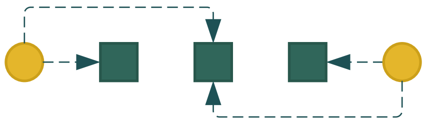
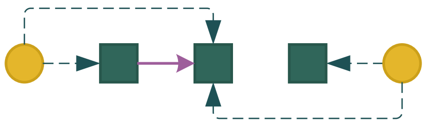
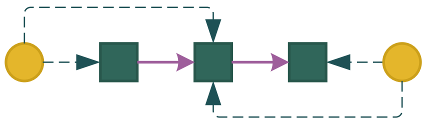
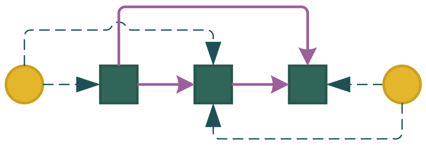
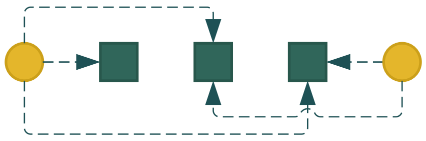
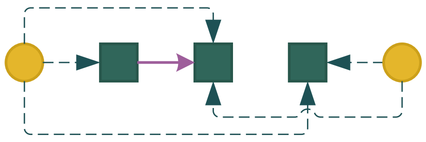
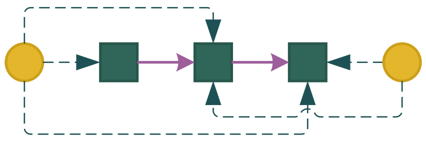
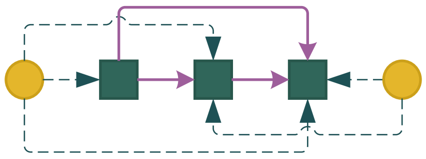
YIHAO LI received the B.S. degree in software engineering from the East China Institute of Technology, the M.S. degree in computer science from Southeastern Louisiana University, and the Ph.D. degree from the Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA. He held a postdoctoral position at the Graz University of Technology. He is now an Assistant Professor at School of Information and Electrical Engineering, Ludong University, China. His research interests include software risk analysis, software fault-proneness prediction, software fault localization, and program debugging. (Email: yihao.li@ldu.edu.cn)

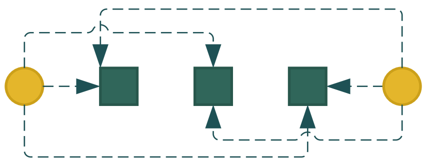
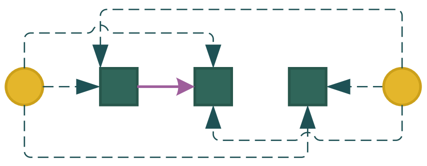
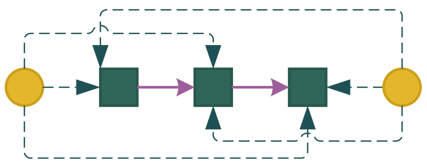
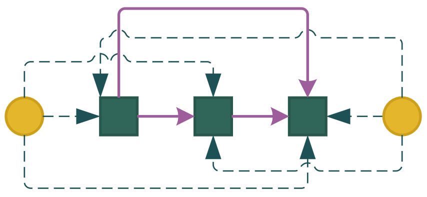
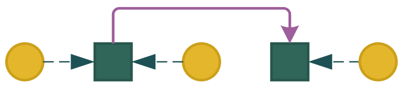
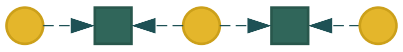
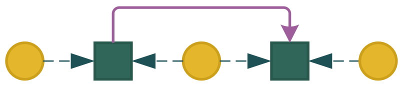
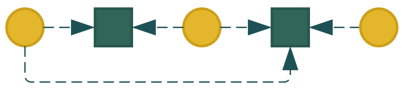
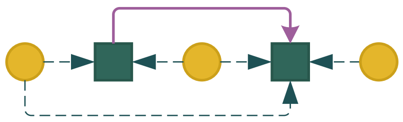
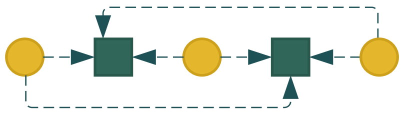
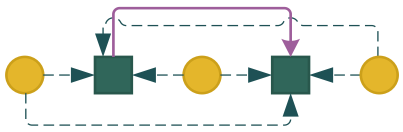
WILLIAM CHENG-CHUNG CHU received the M.S. and Ph.D. degrees in computer science from Northwestern University, Evanston, IL, USA, in 1987 and 1989, respectively. He is currently a Distinguished Professor with the Department of Computer Science, Tunghai University, Taichung, Taiwan, where he had served as the Director of Software Engineering and Technologies Center from 2004 to 2016 and as the Dean of Research and Development office from 2004 to 2007. He was a Research Scientist with the Software Technology Center, Lockheed Missiles and Space Company, Inc. In 1992, he was also a Visiting Scholar with Stanford University. Prof. Chu was the recipient of special contribution awards in both 1992 and 1993 and a PIP Award in 1993 at Lockheed Missiles and Space Company, Inc. He is an Associate Editor for the IEEE TRANSACTIONS ON RELIABILITY, the Journal of Software Maintenance and Evolution, the International Journal of Advancements in Computing Technology, and the Journal of Systems and Software. (Email: cchu@thu.edu.tw)

APPENDIX: SUB-STRUCTURES ANALYZED IN THE STUDY

Vertices	(Dev., Func.)	Contrib.	Calls	Name	Sample
2	(1, 1)	1	0	2-D-F	
3	(1, 2)	2	0	3-D-F2(0)	
		2	1	3-D-F2(1)	
	(2, 1)	2	0	3-D2-F	
4	(1, 3)	3	0	4-D-F3(0)	
			1	4-D-F3(1)	
			2	4-D-F3(2)	
			3	4-D-F3(3)	
	(2, 2)	3	0	4-D2(3)-F2(0)	
			1	4-D2(3)-F2(1)	
		4	0	4-D2(4)-F2(0)	
			1	4-D2(4)-F2(1)	
	(3, 1)	3	0	4-D3-F	

5	(1, 4)	4	0	5-D-F4(0)	
			1	5-D-F4(1)	
			2	5-D-F4(2)	
			3	5-D-F4(3)	
			4	5-D-F4(4)	
			5	5-D-F4(5)	
			6	5-D-F4(6)	
	(2, 3)	3	1	5-D2(3)-F3(1)	

			2	5-D2(3)-F3(2)	
			3	5-D2(3)-F3(3)	
		4	0	5-D2(4)-F3(0)	
			1	5-D2(4)-F3(1)	
			2	5-D2(4)-F3(2)	
			3	5-D2(4)-F3(3)	
		5	0	5-D2(5)-F3(0)	
			1	5-D2(5)-F3(1)	
			2	5-D2(5)-F3(2)	
			3	5-D2(5)-F3(3)	

		6	0	5-D2(6)-F3(0)	
			1	5-D2(6)-F3(1)	
			2	5-D2(6)-F3(2)	
			3	5-D2(6)-F3(3)	
	(3, 2)	3	1	5-D3(3)-F2	
		4	0	5-D3(4)-F2(0)	
			1	5-D3(4)-F2(1)	
		5	0	5-D3(5)-F2(0)	
			1	5-D3(5)-F2(1)	
		6	0	5-D3(6)-F2(0)	
			1	5-D3(6)-F2(1)	

	(4, 1)	4	0	5-D4-F