Software structure evolution and relation to system defectiveness

Jean Petrić
Faculty of Engineering, University of Rijeka
Vukovarska 58
HR-51000 Rijeka, Croatia
jean.petric@riteh.hr

Tihana Galinac Grbac
Faculty of Engineering, University of Rijeka
Vukovarska 58
HR-51000 Rijeka, Croatia
tihana.galinac@riteh.hr

ABSTRACT

We still do not have clear figure about how software systems evolve and how we may control its evolution process. Software structure has been identified that may have the biggest impact, especially because it may be represented from numerous perspectives. Novelty introduced in this paper is the way how we define the structure of evolving complex software systems. The structure is represented with help of graph representations, and subgraph frequencies, the concept reused from the network analysis theory. The graph structure of a software system and its evolution over the system versions, as well as its relation to defectiveness, is empirically studied in terms of subgraph frequencies and motifs for more than 30 releases of three large open source software systems. We identified that the same set of subgraphs of software system is present across the system version, but different sets, although overlapping, are present in different software systems. Furthermore, we confirmed the continuous system evolution in terms of continuous structure change and we find some evidence for its relation to system defectiveness.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Measurement, Experimentation

Keywords

Software defect prediction, network structure, subgraphs and motifs, evolving complex software systems

1. INTRODUCTION

The aim of software defect prediction (SDP) is to predict defect–prone locations in software. Although there are numerous benefits of timely predicting defect–prone locations, one of the most important is guiding verification efforts and making them more efficient. It is not possible, within reasonable time and cost constraints, to verify the whole software system. Moreover, with continuous growth of the complexity of software systems, it becomes more and more important to early focus verification efforts on defect–prone locations in software.

EASE '14 May 13 - 14 2014, London, England, BC, United Kingdom
Copyright 2014 ACM 978-1-4503-2476-2/14/05$15.00.
http://dx.doi.org/10.1145/2601248.2601287

The SDP research area has developed numerous models, but with limited generalization. A major problem is the lack of adequate and non–biased datasets [1, 2]. However, several observations are frequently reported by different researchers. Firstly, the process data (e.g. defects), collected during software development process as a result of human activity, are found to be better predictors of defect–proneness than product metrics (e.g. Lines of code), [3]. Since human role in reporting the process data is crucial, the process data are more exposed to bias. Secondly, analyzing communication links within the software, it becomes clear that communication centric software modules are generally more defect–prone, [9].

The first finding motivated the research in direction of finding better product metrics that would be useful for SDP, while the second finding motivated investigation of communication structures. Complex system theory uses connected graph structures as representations of physical systems and tries to understand their behavior and evolution based on communicating paths between basic building structural elements. Recent findings in that area have really improved the existing knowledge base about some complex systems. Therefore, we were motivated to use graph structures as a representation of software systems, and to observe their communication substructures. Observing the software system evolution by using these concepts may possibly lead to new knowledge about software system behavior and evolution.

In this paper we employ concepts of network analysis to complex software systems analysis and evolution. More precisely, we use three node connected directed graph structures to represent the structure of a software system. Furthermore, we identify subgraphs that are present in analyzed software, the frequencies of their occurrence, how these frequencies change during software evolution, both in their absolute frequency or in terms of their frequencies measured with respect to their occurrence in random graphs. Finally, we study the evolution of software structure through its representation in subgraph frequencies and in relation to the system defectiveness. In this empirical study we analyze three Eclipse open source systems with 36 system release versions.

The paper is organized as follows. In Sect. 2 we describe what motivated this empirical study. In Sect. 3 we discuss the related work in analysis of software evolution and defectiveness using network analysis techniques. In Sect. 4 we introduce the concepts from network analysis theory that we apply in the empirical study. In Sect. 5 the empirical experiment is described in detail. The results are given in Sect. 6. Finally, in Sect.8 we conclude the paper.

2. MOTIVATION

Milo *et al.* [4] introduced in 2002 the concept of a simple building block, a directed connected subgraph structure, that can be used to characterize and classify complex networks, and thus, reflect the underlying process that generated each network class. For example, different biological and non-biological structures that surround us can be represented as a graph, so there is an open possibility to examine such structures and see if there are some sub-structures (sub-graphs, or building blocks) which are statistically more significant than others. Such statistically significant substructures are called motifs (see Sect. 4).

Since then, a number of studies has been undertaken to analyze network motifs [5, 6, 7]. Unfortunately, analysis of network motifs has some computational limitations, because number of sub-graphs increases exponential with respect to network size or the sub-graph size, and there is no known algorithm which can solve such problem in polynomial-time [6]. However, the analysis on 3 and 4 node subgraph have already succeeded to identify that motif dominance is a property of entire population, e.g. so-called feed forward loop is a motif dominant in gene regulation, neurons and electronic circuits, see Table 1.

Some interesting observations in network analysis using subgraphs and motifs have been made in [4]. Motif dominance is independent of network size, its concentration is the same in subnetworks of the analyzed network and in the complete network, but in the randomized network versions of the subgraph it decreases with size. The motifs tend to be more significant as network grows. Sets of network motifs are stable across the network class unless the network structure changes by random addition, removal or rearrangement of the edges by 20%. Another interesting observation from network theory is that k node subgraph frequencies, that in sum have probability one in the analyzed graph structure, are bounded. Moreover, there exists mathematically impossible combinations that bound some frequencies [10].

Changes of evolving systems, implemented in versions of the same software program, are normally driven by quality improvements. However, the newer versions may degenerate system defectiveness. These changes affect system structure over the system evolution. Therefore, an interesting general research question is *could we manage software quality by managing its structure*? Hence, it is interesting to analyze how the structure of software system and its defect proneness have changed over versions of evolution and if there is any correlation between them. It is not a novelty that changes in software structure are influencing defect proneness. If we can use the subgraph frequencies to characterize our software structure, then it would be interesting to analyze if subgraph frequencies are related to defect proneness. These could enable us to quantify this effect and use it as an early indicator of software improvement for lowering defect proneness. From the aforementioned observations on network subgraph frequencies and motifs behavior, we were motivated to investigate the following research questions in the context of software structure, its evolution and relation to system defectiveness.

RQ1 Which subgraphs/motifs with three nodes are present in software systems?

RQ2 Are the same subgraphs/motifs present in different systems and in different versions of the same evolving system?

RQ3 How software structure, in terms of subgraph frequencies and motifs, evolve over software versions?

RQ4 Is there a relation between subgraph frequencies/motifs and system defectiveness?

3. RELATED WORK

Evolving software systems have been analyzed from many perspectives. Special attention is given to open source systems because of available data.

There are several empirical studies trying to understand Lehman's statements about evolution of software systems and their interrelations. These statements originated from the analysis of large program growth dynamics [11]. Review of empirical studies analyzing open source evolution, given in [12], has identified several findings that we summarize as follows. The support has been provided for continuing change for systems which achieved maturity and succeeded to pass initial development stage. Continuing growth seems to well apply for OSS systems. Declining quality in respect to defect rates is generally confirmed, but in a wider context of the quality definition should be verified more. This statement can be taken from Lehman's law of software evolution. Increasing complexity has been analyzed by many empirical studies, but contradictory evidence is obtained. The authors argue that structural complexity has many dimensions and probably this is the main reason for such diversity in results. In this paper we contribute to the existing body of knowledge by observing the evolution of structural complexity in terms of subgraph frequencies. Subgraph frequencies is an approach to complex graph analysis that is widely investigated area in structural research of complex systems that are in focus of scientific interests in number of environments (medicine, biology, Web, traffic, telecommunications).

Different metrics are used in empirical studies to measure system growth or change and its relation to defects – e.g. the source code metrics such as software source code size [13], types, global variables, cumulative number of additions and deletions, file change ([14]), etc. It is impossible to list them all here so we mention just a few. Here we employ completely different metrics for measuring software structure change with the help of the coordinate system defined in terms of frequencies of subgraph occurrence in the source code. We did not find other similar study in the field of software evolution analysis. The software architecture is considered as one of the most important attributes of software evolvability.

The defect prediction in evolutionary developed systems, trying to reuse the same SDP model from the release to release has been analyzed by Zimmermann, Nagappan *et al.* [15, 16]. They find that SDP models might not work well if reused from release to release. In [2] it is confirmed that fewer serious failures occur in components implementing commonality and that these components have less changes over time, while components that implement variations are subject to significant change continuously. They bring into relation software structure and the role of the structural elements in the software structure to its defectiveness. Dependency graphs are used in [15] to identify central program units that are most likely to face defects. They compared network measures and complexity measures. Network measures are obtained using *Ucinet* tool distinguishing between ego measures that measure node connectivity with its neighbors, global measures that measure the role of the node within the complete network and measures for structural holes. Their empirical study was performed on the binaries of a *Windows Server 2003* and identified that SDP models built from network measures are better than models built from complexity metrics. Their study has been replicated in [17] for five more projects, and they concluded that network measures are not suitable for small scale projects. Network measures were also used in the longitudinal empirical study involving eleven large OSS programs [18]. In the study product and process artifacts were represented as graphs and were quantified with help of basic graph metrics such as average degree, clustering coefficient, node rank, graph diameter, assortativity, edit distance and modularity ratio. Correlating these measures they observed some interesting and new observations. Among others, they observed that software structures are similar across programs with lower and upper bound of variation. With help of these measures they were able to detect significant system change for which they have even found evidence in the history of system changes in terms of significantly higher amount of modification requests. In terms of evolution, the software have to change continuously, so that there could exist a more sensitive way for identification of structural change. Therefore, in this paper we analyze structure and structure change using subgraph frequencies.

Approaches similar to our approach are those aiming to identify design patterns and anti–patterns (patterns that should be avoided) by identifying micro–architectures in a software program and relate them to software defects. Numerous design pattern detection approaches exist, but in majority they are based on finding a pre–known pattern templates. Here we are focusing on finding connected subgraphs. In [19] the authors present the algorithm for finding three, four, and five node connected subgraphs, implemented in *SGFinder* tool, from class connections of the small to medium sized Object–oriented software systems. Their work focused on characterization of the specific subgraphs and their relation to defectiveness. Here, in this paper we use the connected subgraphs to represent the whole system and try to understand its evolution trends and defectiveness. Moreover, our study involved much larger systems with more releases.

4. NETWORK SUBGRAPHS AND MOTIFS

Network graph is a directed graph whose vertices are network nodes and edges represent directed connections between these nodes. Network subgraphs resemble the topological structure of a network. Motifs are subgraphs which are statistically over-represented structures in a network. This means that they appear statistically more often than usual. If there is a repeatedly occurring sub-structure in a network, which occurs much more often than other sub-structures, for further investigation, we must assure that this has not happened by chance. Network motifs are distinguished according to the number of nodes and number of edges connecting those nodes. All motifs are topologically unequal to each other. Figure 1 presents all types of topologically unequal three node connected and directed subgraphs. Frequency of these graphs is the subject of this paper.

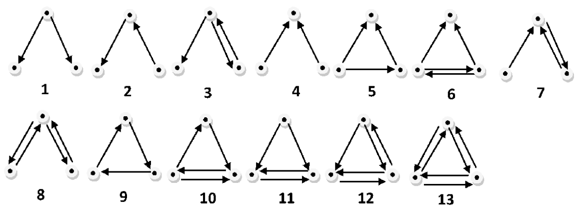


Figure 1: All types of three node subgraphs

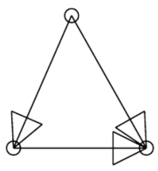
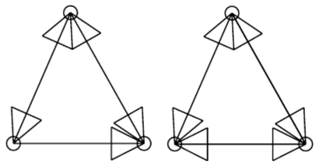
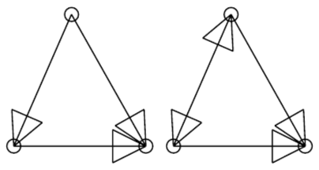
Over time, it was shown that there are network motifs which are more common in various networks than others [4]. Those network motifs are given in Table 1.

Because we are always doing analysis on one, so called real network, we must somehow compare this network with some others to perform statistical calculation and determine if some network subgraphs are motifs, i.e., statistically significant or not. For this purpose we use random networks. Random networks, in our analysis, are those networks which have same number of nodes and edges like real network, but connections between nodes are made randomly.

The statistical significance is an important property in the motif analysis, and can be calculated using the p-value or Z-score. According to those values, we assure that some motif is statistically significant or not. The p-value is the probability that frequency of appearance of the motif in a randomized network is equal or greater than the frequency in the real network. Thus, smaller p-value indicates lower possibility that appearance of the motif will be more often in the random networks than in the real network, resulting in a conclusion that the motif is significant. But, to calculate reliable p-value, at least one thousand random networks should be taken into consideration, [8].

Another important statistical value is a Z-score. To define Z-score, we calculate frequencies of a candidate motif in the real network (f_{real}), and its frequencies in random networks (f_{random}). Let $\overline{f}_{random}$ and σ^2_{random} be the mean and variance of the frequencies in random networks. The Z-score is

Table 1: Characteristic motifs in various fields of science

Characteristic network	Known as	Motif
Gene regulation Neurons Electronic circuits	Feed-forward loop	
World wide web	Feedback with two mutual dyads and fully connected triad, respectively from left to right	
Object-oriented software	Feed-forward loop and up-linked mutual dyad, respectively from left to right	

then defined as

$$Z = \frac{f_{real} - \overline{f}_{random}}{\sqrt{\sigma^2_{random}}}.$$

Important difference between the p-value and Z-score is that Z-score can be calculated even if there are less number of random networks [4]. One hundred random networks could be enough for getting reliable results. Finally, sub-graphs are occasionally declared as motifs if p-value is less or equal than 0.01 or Z-score is grater than 2, [6]. There are also other ways to determine statistical significance of the network motifs, which is beyond the scope of this paper.

Network motifs are considered in different fields of science, especially in biology, medicine, electrical engineering, etc. The purpose of network motif analysis is to find some characteristics which are common in some group of interest. If we take medicine for instance, it would be very interesting to find common patterns for some diseases. Also it would be interesting if medical researchers could be in position to understand evolution of such diseases and be ready to prevent it. On the other hand, revealing the network motifs can be helpful in understanding how people think while "surfing" on the Internet, providing for example better results in search engines. Understanding network motifs can also help in connecting people on social networks, because some common patterns which connect those people could be found.

Some of the tools that are seeking network motifs are Kavosh, mFinder, MODA, etc. Difference between them is in the algorithms used, which leads to faster or slower execution time [20]. Because of diversity in network motif detection tool algorithms, different results will be obtained on the same observed network. All these tools in the process of motif determination are calculating the subgraph frequencies, given in Figure 1, within analyzed network. Note that the major difference in these tools is in algorithms for generating random networks not in search for subgraph frequencies.

5. STUDY DESIGN

5.1 Hypotheses

The main research questions listed in the Sect. 2 are addressed with the help of the following hypotheses.

Hypotheses about subgraphs presence
H1 The same subgraphs are present across different versions of the same software program.
H2 The same subgraphs are present across different software system programs.

Hypotheses about structural evolution
H3 Subgraph frequencies do not change over system evolution. Distribution of subgraph frequencies is the same in all versions of the system.
H4 Subgraph frequencies tend to stabilize during system evolution.

Hypotheses about effects of structural evolution on defect proneness
H5 Subgraph frequencies are good predictors of number of defects in system version.

Hypotheses about motif stability in software structures
H6 The same motifs are present across all software systems and all system versions.

5.2 Data collection

We have chosen three open source projects written in Java to take needed information for this research, which includes three different Eclipse plugins: Business Intelligence and reporting tools (BIRT), Plug-in Development Environment (PDE), and Java development tools (JDT). Reasons for such selection are several:

- they are complex enough for network analysis,
- they evolve over a number of system releases,
- we have access to 9 versions of BIRT and 13 versions of PDE and 14 versions of JDT that gives us frankly good sample for our statistical analysis,
- we have access to open defect repository used in software development.

Number of other projects satisfy the above stated criteria but they are left for future work. Also, all three plugins are developed during long-time period. BIRT plugin is developed in period of 7 years in 9 different versions we have access, JDT plugin in period of 12 years through 14 different versions and PDE plugin in period of 11 years through 13 different versions. Process of collecting data for this research is divided in few steps:

1. collecting relevant source code files from Eclipse repositories for different plugins (Source code files)
2. creating network graphs from source code files (Network files)
3. determining subgraph frequencies and motifs with Kavosh and mFinder
4. getting number of defects from Eclipse Bugzilla pages
5. grouping and preparation of files for data analysis

The above process is repeated for each version of each plugin. Each step of the data collection procedure will be elaborated in more detail in the following paragraphs.

Source code files. Firstly, we find all tags for the above stated Eclipse plugin projects from the Web site of Eclipse.org Git repositories[1] and downloaded compressed project files for each release within the project separately. Secondly, we removed all test files from each project file by removing all folders that contain word *test* in its name. Please refer to Naming convention web page for Eclipse projects[2].

Note that we analyzed just org.eclipse.ui[.*] components (workbench) for JDT and PDE plugins. Other components were too small for the motif and subgraph analysis.

Network files. Kavosh and mFinder tools expect network file as an input. Those files are provided through rFind tool. rFind tool seeks for relation between classes/modules. When two classes communicate, through method invocation, return type or parameters rFind detects it and marks down that relation. In this way graph is built, and network file is produced.

[1]http://git.eclipse.org/c/

[2]https://wiki.eclipse.org/Naming_Conventions

Subgraph frequencies and motifs. For subgraph frequencies and network motif determination we used mFinder [1] and Kavosh [2] tools, freely available from the web pages. In the analysis we use motif size 3, number of random networks 100 and Z-score acceptance ≥ 2. In motif detection process both tools calculate frequencies for all directed subgraphs of given motif size. There are 13 possible subgraphs with three nodes. Full enumeration of subgraphs along with mFinder algorithm is described in [4].

Defects. Number of defects per version for each plugin are collected from official *Eclipse Bugzilla* repository[3]. Only defects which satisfy special condition are taken into consideration. The condition which needs to be satisfied is that resolution of the defect must have status *fixed*. That ensures that fix for this bug is checked into the tree and tested. For other condition options which *Eclipse Bugzilla* offers, all values are taken into account, which means defects with any status, severity, priority, hardware and operating system on which the defect is present. This is just preliminary analysis and further analysis may refine this criteria.

Data analysis files. For the purpose of data analysis we created three tables, one per Eclipse plugin. Tables are stacked and presented in Table 2. Rows in the table contain information collected for one release. Therefore, in the first column we have release version identity, the next 11 columns contain information about subgraph frequencies per each subgraph and the last 13th column contains the total number of defects.

5.3 Data analysis techniques

For the purpose of analysis, besides observational conclusions, we also used several statistical techniques.

In analysis of hypothesis H3 we have to identify if there is a significant change in structure during software evolution. We perform the analysis with the help of subgraph frequencies, which are some kind of representation of network structure. Since, subgraph frequencies may be considered as a categorical variable, the Pearson chi-square statistic is used as analysis technique.

Hypothesis H5 is analyzed using Pearson and Spearman rank correlation between subgraph frequencies for each subgraph and the number of defects. Pearson analysis requires normally distributed data and Spearman rank correlations is more robust to data type. Although, in the case of categorical data the Spearman rank correlation may be more appropriate choice, because of completeness of the analysis we perform both calculations.

6. ANALYSIS

Table 2 provides dataset used in analysis. Analysis results are discussed in separate subsections that are structured in accordance to stated hypothesis.

6.1 Hypotheses about subgraphs presence

This group of hypotheses we analyze just by simple observation of data collection results in Table 2. We can identify that in all observed cases the set of subgraphs (subgraph ids

[3]https://bugs.eclipse.org/bugs/

Table 2: Dataset used for the analysis

	1 (6)	2 (12)	3 (14)	4 (36)	5 (38)	6 (46)	7 (74)	8 (78)	9 (98)	10 (102)	11 (108)	Defects
BIRT 2.0	77408	25578	635	3601110	5129	78	4348	84	1	1	46	2864
BIRT 2.1	37050	11206	290	873768	2282	35	1676	34	1	0	20	2279
BIRT 2.2	93256	29529	611	4934605	5861	81	4606	85	1	1	48	1905
BIRT 2.3	103713	31058	676	6077470	6295	83	4585	94	1	1	50	858
BIRT 2.5	52110	14239	348	1398471	3059	48	2221	47	1	0	22	665
BIRT 2.6	107850	30498	609	7007340	6533	82	4204	74	1	1	43	175
BIRT 3.7	125269	26946	487	9010159	6801	79	1694	28	1	0	31	46
BIRT 4.2	117325	25804	457	8212615	6503	73	1658	28	1	0	30	31
BIRT 4.3	113963	25411	450	7733108	6467	73	1650	28	1	0	29	26
JDT 1.0	23620	2417	38	178207	755	3	12	1	0	0	0	2584
JDT 1.1	56342	4577	47	646440	1574	5	21	3	0	0	0	1481
JDT 2.0	68460	4812	44	803718	1763	5	21	3	0	0	0	1058
JDT 3.0	80520	6380	50	1119079	2122	6	26	3	0	0	0	864
JDT 3.1	86917	7311	49	1248497	2372	7	26	3	0	0	0	606
JDT 3.2	97234	848	107	1388250	3029	20	74	6	0	0	0	318
JDT 3.3	107234	8781	100	1516278	3033	16	84	3	0	0	0	252
JDT 3.4	110531	8846	99	1558709	3149	16	84	3	0	0	0	194
JDT 3.5	109755	8930	98	1565121	3159	16	84	3	0	0	0	185
JDT 3.6	115995	9060	99	1588862	3250	16	85	3	0	0	0	80
JDT 3.7	6734	720	0	59406	228	0	0	0	0	0	0	66
JDT 3.8	116351	9052	99	1591644	3255	16	85	3	0	0	0	59
JDT 4.2	115995	9060	99	1588862	3250	16	85	3	0	0	0	17
JDT 4.3	76411	5987	48	1007275	2028	6	26	3	0	0	0	0
PDE 2.0	5253	104	0	132343	24	0	0	0	0	0	0	992
PDE 2.1	14539	515	0	195885	213	0	0	0	0	0	0	699
PDE 3.0	38611	2956	16	543494	1012	3	0	0	0	0	0	695
PDE 3.1	32408	2246	3	990694	703	2	0	0	0	0	0	580
PDE 3.2	3416	82	0	64528	25	0	0	0	0	0	0	547
PDE 3.3	31383	2049	17	361370	726	3	0	0	0	0	0	489
PDE 3.4	30159	2038	0	863484	655	0	0	0	0	0	0	486
PDE 3.5	4957	161	0	108232	43	0	0	0	0	0	0	410
PDE 3.6	33845	2268	12	1099887	733	3	0	0	0	0	0	301
PDE 3.7	34684	2292	12	1137434	751	3	0	0	0	0	0	144
PDE 3.8	35158	2322	36	1182209	761	11	0	0	0	0	0	84
PDE 4.2	35387	2324	12	1179663	755	3	0	0	0	0	0	49
PDE 4.3	35387	2324	12	1179663	755	3	0	0	0	0	0	42

are presented between parentheses in header of Table 2) that is present in initial version of software system is also present in all further software versions. But, not all software systems have all subgraphs represented in their structure and there are differences in subgraphs present in structures of different software systems. However, for the analyzed systems the subgraphs with 1(id6), 2(id2), 3(id14), 4(id36) and 5(id38) are present in all software systems and all systems versions. These conclusions are obtained by analyzing the following hypotheses.

H1 *The same subgraphs are present across different versions of the same software program.* In Table 2 we can observe that in all three analyzed Eclipse projects the same set of subgraphs is present along the project evolution, just the subgraph frequencies are changed as the system evolved. Therefore, solely based on this observation we can say that we have evidence in favor to this hypothesis for observed projects. This indicates that the system structure does not evolve radically, by involving new subgraphs. Perhaps this conclusion is very much dependent on level of system change.

H2 *The same subgraphs are present across different software system programs.* Projects in Table 2 are ordered according to number of connections between classes (short CBC). CBC simply counts how many relation, per one Eclipse plugin, between classes exists (e.g. method in class A invokes method in class B which is equal to 1 CBC). Eclipse BIRT, the first given in the table has the biggest CBC number and equals 22126, then followed by JDT.UI which has CBC 13731 and PDE.UI with CBC 7012. CBC values are calculated as average value from all versions of each plugin. Observing the subgraph frequencies we can notice that different subgraphs are represented in each Eclipse project structure. Software system with biggest CBC number has also the most subgraphs represented in its structure. Probably, as the CBC grows the more complicated subgraph structures occur in the software structure. However, we can not claim this statement is generally valid.

6.2 Hypotheses about structural evolution

Some studies have been using a set of subgraph frequencies to form a coordinate system where different system structures could be presented [10]. We reuse this concept and assume that system structure could be represented by subgraph frequencies of three node subgraphs present in some software structure. Therefore, we study if analysis of software structure through subgraph frequencies could help us in better understanding of system evolution.

H3 *Subgraph frequencies do not change over system evolution. Distribution of subgraph frequencies is the same in all versions of system.* Calculating χ^2 test statistic by using contingency table with all versions of the software system, we identified that the distribution of system frequencies is associated with system version. The value of χ^2 statistic, degrees of freedom (*df*) along with p-values at $\alpha = 0.05$ significance level are given in Table 3. Therefore, we may say that the hypothesis H3 is rejected and that subgraph frequencies change over system evolution.

Table 3: Results of chi-square test

Project	Last # of versions	χ^2 statistic	df	p–value
BIRT	2	92,76	9	<0.00001
	3	362,87	18	<0.00001
	4	6465,31	30	<0.00001
	5	74216,22	40	<0.00001
	6	76914,12	50	<0.00001
	7	83865,1	60	<0.00001
	8	135468,14	70	<0.00001
	all versions	145015,28	80	<0.00001
JDT	2	0,13	7	<0.00001
	3	0,17	14	<0.00001
	4	68,79	24	<0.00001
	5	162,29	28	<0.00001
	6	194,83	35	<0.00001
	7	6474,05	42	<0.00001
	8	6625,09	49	<0.00001
	9	6669,14	56	<0.00001
	10	8536,73	63	<0.00001
	11	10569,84	70	<0.00001
	12	20196,64	77	<0.00001
	13	20260,47	84	<0.00001
	all versions	21834,71	91	<0.00001
PDE	2	17,81	5	<0.00001
	3	28,37	10	<0.00001
	4	45,24	15	<0.00001
	5	67,28	20	<0.00001
	6	285,19	25	<0.00001
	7	804,75	30	<0.00001
	8	26033,57	35	<0.00001
	9	51857	40	<0.00001
	10	57952,27	45	<0.00001
	11	58193,03	50	<0.00001
	12	58364,38	55	<0.00001
	all versions	58704,49	60	<0.00001

H4 *Subgraph frequencies tend to stabilize during system evolution.* Visual analysis of the Table 2 indicates that the frequencies of particular subgraphs changes slightly comparing to some earlier versions. Therefore, we start our stability analysis from these last versions and comparing with the previous. Calculating χ^2 test statistics by using contingency table with increasing the number of the last versions of the software system in the analysis we identified that we CAN NOT claim the system frequencies are not associated with system version. The value of χ^2 statistic, degrees of freedom (*df*) along with p-values at $\alpha = 0.05$ significance level are given in Table 3. Therefore, we may say that the hypothesis H4 is rejected and that subgraph frequencies are in continuous change with system versions over the system evolution.

It is possible that different conclusions would be obtained if the analysis is performed with respect to release time instead of release sequence number. Another possibility is that the distribution of subgraph frequencies is stabilized when only most frequent subgraphs are considered. These issues are left for future work.

6.3 Hypotheses about effects of structural evolution on defect proneness

In previous hypotheses some interesting findings about software structure evolution have motivated us to extend the investigation of this study to identify if there are relationship between software structure and defectiveness of system during its evolution. As we observed that system is structurally changed significantly, we were motivated to verify if some of dimensions of structural change are more related to system defectiveness than others. Some studies have identified that SDP models could not be just simply reused from projects in the same domain or with the same process [16]. One possible explanation may be because the defectiveness of software system is significantly correlated to software structure that is also significantly changed during the system evolution.

H5 *Subgraph frequencies are good predictors of number of defects in system version.* Pearson and Spearman correlation coefficients are given in Table 4. Pearson and Spearman rank correlation coefficients are calculated for number of defects in relation to different variables measured in evolution of analyzed software systems. Variables that were calculated for each version of software system are as follows: number of nodes, number of edges, and frequencies for each of subgraphs given in Table 1. Since subgraphs 12 and 13 do not appear in any of the software projects, we omit them from the table. The analyzed software systems were BIRT, JDT and PDE. The same correlation coefficients are calculated for all projects together, that is referred in the table as ALL in the Projects column. All correlation coefficients that are significant (p-value is lower than 0.5) are in bold font.

According to the results presented in Table 4, we may observe that number of nodes and edges is strongly correlated with number of defects, and this correlation is negative. This means that with the rise of number of nodes and edges in the software graph, the number of defects is falling. Another interesting observation is that the frequency of the most frequent subgraph (4, id36), in all of the observed software graphs, is very much correlated with number of defects. This correlation is even higher (-0.34, -0.8, -0.75). It is also interesting to note the correlation between number of defects and subgraph 6 (id46). This subgraph is very rare compared to subgraph 4 (id36), but is also relatively strongly correlated with number of defects (-0.53, -0.61). Note that all correlations are calculated for number of defects reported on the whole system. In majority of studies correlations are calculated among various variables and number of defects measured at the module or class level. All identified correlations are negative! That may be an indication that as system grows it is getting more stabilized in terms of number of defects. This is expected result since the evolution of software systems is driven with quality improvements.

6.4 Hypotheses about motif stability

H6 *The same motifs are present across all software systems and all system versions.* In all three different plugins we have found that motif 5 (id38) is present, occasionally with high Z-score value. Also, we have found that in BIRT and JDT plugins motif 6 (id46) is present for some or all versions of software. Table 5 shows the summary for all projects.

Table 5: Results from mFinder and Kavosh tools

Project	Version	Motif	Motif	$\overline{Z}$	$\overline{Z}$
	count	no and id	count	mFinder	Kavosh
BIRT	9	5, id38	9	13.95	9.88
		6, id46	9	7.14	2.38
JDT	14	5, id38	14	15.66	11.08
		6, id46	7	4.96	-
PDE	13	5, id38	10	7.94	7.81
		6, id46	1	-	2.69

In Table 5 results from all three plugins are grouped and shown together. Version count shows how many versions of particular plugin are processed, in the column motif its ID is presented (according to [4]), motif count presents the number of versions where specific motif is detected, $\overline{Z}$ is the average value of Z obtained from mFinder or Kavosh. Figure 2 presents the structure of motifs in Eclipse plugins.

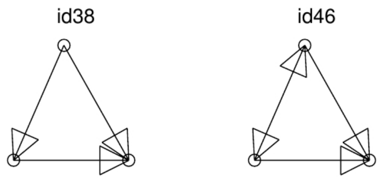


Figure 2: Dominant motifs from Eclipse plugins

Motif 5 (id38) is also known as feed-forward loop. This motif is frequent in many different fields of science. For example, in biology, it is shown that feed-forward loop appears 10 standard deviations more often than in randomized network (in bacterium E. coli). In this research, Z-value indicates that feed-forward loop is also very frequent in observed plugins, so we can tell that this has not happened by accident. Motif 6 (id46) appears less frequently than motif 5 (id38), but we can not ignore more than 4 standard deviations greater appearance than in random network. Thus, these two motifs could tell us something more about how communication between objects affects some software attributes. So we took two approaches to analyze them. The first approach is to see how Z-value is changed over new versions of software, and the second approach looks if there is any relation between number of reported defects in every single version of plugin and Z-score. Results of both approaches are shown in the subfigures of the Figure 3.

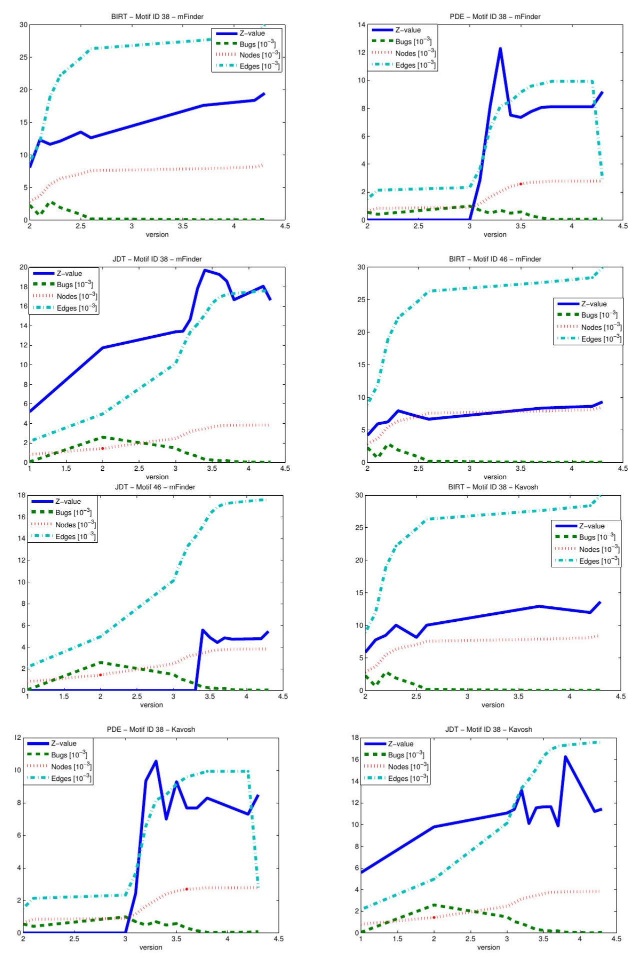


Figure 3: Relations between motifs obtained from mFinder and Kavosh tool and average number of defects through versions

From Figure 3 it is evident that different tools give different results. The reason for this is different algorithm which is used in those tools. While mFinder gives us motif 6 (id46) with average Z-score 7.14 and 4.96 in BIRT and JDT plugins respectively, Kavosh gives us much smaller Z-value for BIRT – 2.38 and for JDT plugin does not find motif 6 (id46) to be over-represented in real network at all. Furthermore, from Figure 3 we can point on two more things. The first is that in BIRT plugin Z-value of both motifs increases with every new version. So, developers of this plugin in every new version of plugin use more and more communication between classes which matches motif 5 (id38) and motif 6 (id46). Similar is also with JDT and PDE plugins, but with JDT there is some slight decrease after version 3.5, and with PDE there is a jump between versions 3.3 and 3.4. In general, the newer version of plugin, the more stable motif is obtained – or more class connections with motif types 5 (id38) and 6 (id46) are made. The second thing is that there is no relationship between number of defects per version and the Z-value. But, because this paper employs the total number of defects per version, we can not make conclusion that there is no relation between those two parameters. The possibility of a logical connection between defects and motifs is not excluded at

Table 4: Correlation coefficients

Proj.	Measure	Nodes	Edges	1	2	3	4	5	6	7	8	9	10	11
All	P coef.	-0,26	**-0,33**	-0,21	0,15	0,28	-0,11	0,02	0,18	**0,39**	**0,41**	**0,29**	**0,40**	**0,33**
	p–val.	0,13	**0,05**	0,21	0,37	0,10	0,52	0,93	0,29	**0,02**	**0,01**	**0,09**	**0,01**	**0,05**
All	S coef.	**-0,49**	**-0,47**	**-0,38**	-0,08	-0,01	**-0,37**	-0,22	-0,09	0,03	0,11	0,10	0,31	0,12
	p–val.	**0,00**	**0,00**	**0,02**	0,63	0,94	**0,03**	0,20	0,60	0,88	0,50	0,58	0,07	0,49
BIRT	P coef.	**-0,98**	**-0,98**	**-0,66**	-0,28	0,10	**-0,70**	-0,54	-0,26	0,42	0,47	NaN	0,41	0,23
	p–val.	**0**	**0**	**0,05**	0,46	0,79	**0,03**	0,13	0,5	0,26	0,21	NaN	0,28	0,55
BIRT	S coef.	**-1**	**-1**	**-0,8**	-0,08	0,30	**-0,80**	**-0,72**	0,03	**0,65**	**0,7**	NaN	0,52	0,27
	p–val.	**0**	**0**	**0,01**	0,84	0,44	**0,01**	**0,04**	0,96	**0,07**	**0,05**	1	0,19	0,49
JDT	P coef.	**-0,88**	**-0,88**	**-0,58**	-0,43	-0,44	**-0,62**	**-0,57**	**-0,53**	**-0,56**	-0,27	NaN	NaN	NaN
	p–val.	**0**	**0**	**0,03**	0,12	0,12	**0,02**	**0,03**	**0,05**	**0,04**	0,36	NaN	NaN	NaN
JDT	S coef.	**-0,97**	**-0,97**	-0,51	-0,48	-0,31	-0,52	-0,52	-0,37	**-0,55**	-0,03	NaN	NaN	NaN
	p–val.	**0**	**0**	0,06	0,08	0,29	0,06	0,06	0,19	**0,04**	0,91	NaN	NaN	NaN
PDE	P coef.	**-0,91**	**-0,64**	**-0,56**	-0,51	**-0,56**	**-0,76**	-0,49	**-0,57**	NaN	NaN	NaN	NaN	NaN
	p–val.	**0**	**0,01**	**0,05**	0,07	**0,05**	**0,00**	0,09	**0,04**	NaN	NaN	NaN	NaN	NaN
PDE	S coef.	**-1**	**-0,75**	-0,50	**-0,55**	-0,45	**-0,75**	**-0,56**	**-0,61**	NaN	NaN	NaN	NaN	NaN
	p–val.	**0**	**0**	0,08	**0,05**	0,13	**0,00**	**0,04**	**0,03**	NaN	NaN	NaN	NaN	NaN

some different level, but this question is left for future work.

7. THREATS TO VALIDITY

Object oriented systems are systems with high code reuse. Therefore investigation of system subgraph frequencies may be an indicator of system structure but may suffer from generalizations to some closed industrial code with proprietary languages.

In our study we analyzed the evolution of the Eclipse system plugins, BIRT, PDE and JDT. This may represent the threat to generalizing of the results in sense that our findings are limited to the Eclipse plugins. Only complex parts of systems are taken into account, because other parts are not big enough for motif/subgraph analysis (like is already mentioned in section 5.2.1). Removal of small parts could represent bias.

The evolution is analyzed with respect to release time instead of release sequence number. The version sequence number may not be aligned with time and that may represent a threat. That is the reason why we decided to extend this analysis in future to observe system evolution over the release time dimension.

The motif detection algorithm could influence the results obtained in the motif analysis. Therefore, in order to eliminate this kind of bias we used the mFinder and Kavosh tools. Although tools give different results these are not completely different and even more the similarities are repeated in all observed versions. So we avoid to make definite conclusions based on the analysis results with these differences.

8. CONCLUSIONS AND FUTURE WORK

Network analysis approach to software evolution and defect prediction is a very promising area of research. Since it is independent of metrics from the development process and metrics involving real code complexity, considering only communication paths, it is very promising area for moving predictions in very early phases, e.g. architecture definition. The potential is in guiding the software evolution, not just as controlling and correcting factor but also in its design. In this paper we reuse the concepts of network analysis theory. The software system is represented as graph. The software structure is represented by the frequency of occurrence of all three node subgraphs. In this representation there may be some very frequent subgraphs and some statistically frequent subgraphs in comparison to its occurrence in the random graph called motifs.

We have several contributions in this paper. Firstly, we observe that same set of subgraphs are present in all versions of system evolution. This set is changed in different systems but with overlapping in subgraphs present. We proved that analyzed systems evolve continuously and the change in their structure is statistically significant. We could not confirm that the analyzed systems tend to stabilize. This is an important finding because it mean that with help of subgraph frequencies we can better differentiate the software systems and on quantitative basis. Also, continuous change of software structure can explain weak reusability of software defect prediction models across the system versions. We identified that defectiveness is correlated with some subgraphs. Motifs are shown to be consistent across system versions and across different software systems. Their significance it seams to grow with the system grow and system maturity.

For this research few important steps have been realized, which include the elaboration of ideas for this and future work, realization of rFind tool which will help in further data collection from other Java source codes and unify the way of how the data are collected. Future work would certainly go deeper in finding how other software attributes affect on subgraphs and motifs, especially how defects on class level have influence on structure, subgraph occurrence and motif significance. Moreover, we want to replicate the same analysis for different application domains, i.e. telecommunications, medical software etc. Also, in future, we will expand gathering of different complexity measures for different software systems. In this work we have look evolution of software systems only by its versions, so in future we will also include time-period of software releases, because then we will have better picture of how stability of motifs changes during time. There is also a plan to extend rFind tool to work for other programming or scripting languages such as C++, Python, etc. This will make it possible to collect data for different kind of software (CAD, financial, web applications) and perform some analysis for wider sample of software systems.

Acknowledgment

The work is supported by the University of Rijeka research grant 13.09.2.2.16.

9. REFERENCES

[1] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. *A systematic literature review on fault prediction performance in software engineering.* IEEE Trans. Softw. Eng., 38(6):1276, 1304, Nov. 2012.

[2] S. Krishnan, R. R. Lutz, and K. Goševa-Popstojanova. *Empirical evaluation of reliability improvement in an evolving software product line.* In Proceedings of the 8th Working Conference on Mining Software Repositories (MSR '11). ACM, New York, NY, USA, 103-112.

[3] T. Galinac Grbac, P. Runeson, and D. Huljenić. *A second replicated quantitative analysis of fault distributions in complex software systems.* IEEE Trans. Softw. Eng., 39(4):462–476, 2013.

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

[5] Y. Ma, K. He, J Liu. *Network Motifs in Object-Oriented Software Systems.* Dynamics of Continuous, Discrete and Impulsive Systems (Series B: Applications and Algorithms), 14(S6): 166-172, 2007; arXiv:0808.3292.

[6] E. Wong, B. Baur, S. Quader, C. Huang. *Biological network motif detection: principles and practice.* Briefings in Bioinformatics, 2011;

[7] L. Wen, D. Kirk, R. G. Dromey. *Software Systems as Complex Networks. Cognitive Informatics*, 6th IEEE International Conference, 2007;

[8] B. H. Junker, F. Schreiber. *Analysis of Biological Networks.* Wiley, 2008.

[9] A. von Mayrhauser, J. Wang, M.C. Ohlsson and C. Wohlin. *Deriving a Fault Architecture from Defect History.* Proceedings International Symposium on Software Reliability Engineering, 1999;

[10] J. Ugander, L. Backstrom, and J. Kleinberg. *Subgraph frequencies: mapping the empirical and extremal geography of large graph collections.* In Proc. of the 22nd Internat. Conference on World Wide Web (WWW '13), pages 1307-1318. International World Wide Web Conferences Steering Committee, Geneva, Switzerland, 2013.

[11] M. M. Lehman and L. A. Belady (Eds.). *Program Evolution: Processes of Software Change.* Academic Press Prof., Inc., San Diego, CA, 1985.

[12] J. Fernandez-Ramil, A. Lozano, M. Wermelinger, and A. Capiluppi. *Empirical studies of open source evolution.* In T. Mens and S. Demeyer (Eds.), *Software Evolution*, pages 263–288, Springer, Berlin–Heidelberg, 2008.

[13] I. Herraiz, J. M. González-Barahona, G. Robles, D. M. Germán. *On the prediction of the evolution of libre software projects*, ICSM 2007, 405-414.

[14] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy. *Predicting Fault Incidence Using Software Change History.* IEEE Trans. Softw. Eng. 26, 7 (July 2000), 653-661.

[15] T. Zimmermann and N. Nagappan. *Predicting defects using network analysis on dependency graphs.* ICSE 2008: 531-540.

[16] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy. *Cross-project defect prediction: a large scale experiment on data vs. domain vs. process.* In Proceedings of the ESEC/FSE '09. ACM, New York, NY, USA, 91-100.

[17] A. Tosun, B. Turhan, and A. Bener. *Validation of network measures as indicators of defective modules in software systems.* In Proceedings of the 5th International Conference on Predictor Models in Software Engineering (PROMISE '09). ACM, New York, NY, USA, Article 5 , 9 pages.

[18] P. Bhattacharya, M. Iliofotou, I. Neamtiu, M. Faloutsos. *Graph-based analysis and prediction for software evolution.* ICSE 2012: 419-429

[19] A. Belderrar, S. Kpodjedo, Y. Guéhéneuc, G. Antoniol and P. Galinier. *Sub-graph Mining: Identifying Micro-architectures in Evolving Object-Oriented Software.* CSMR 2011: 171-180

[20] M. Zuba. *A Comparative Study of Network Motif Detection Tools.* UConn Bio-Grid, REU Summer, 2009;