Dynamic network analysis of software systems

Anjan Pakhira and Peter Andras
School of Computing Science
Newcastle University
Newcastle upon Tyne, UK
{anjan.pakhira,peter.andras}@ncl.ac.uk

***Abstract*— It is difficult to analyse large-scale integrated software systems with the purpose of improving their dependability and functionality through maintenance and evolution. Such systems contain many interactions between their components and can be represented as complex interaction networks similar to complex biological and socio-technical systems. Here we combine dynamic analysis and network analysis methods with the aim to determine and validate components of high functional importance in software systems. We use as a test case the JHotDraw 6.01b software and predict the method calls with high functional importance using network analysis methods. We validate the predictions by disabling the methods predicted to have high functional importance and evaluating the behaviour of the software following this. Our results show that network analysis methods are relatively good in predicting method calls of high functional importance. Such analysis can predict vulnerabilities or critical components of software systems and can be used to predict patching or updating needs of software systems.**

Keywords-dynamic analysis; network analysis;complex system; vulnerability determination; patching prediction

I. INTRODUCTION

Large-scale software systems became ubiquitous parts of everyday life in the last decades – just consider the collection of software that integrates work done on mobile phone PDAs, office desktop at work, and laptop used at home and at conferences. While software components became more reliable in parallel with the rapid expansion of software systems due to better development support and improved software development and management practices, the large scale of current software systems presents new challenges for the dependability evaluation and maintenance of these systems [1]. Such software systems evolve by integrating new components with old ones and as a whole they are practically tested mostly by their users as the usage of the software expands. This makes formal analysis aimed to assess software dependability [2]or to support software maintenance [3], [4] very difficult since usage patterns evolve as the software system is used by an increasing number of users and the blueprints of integrated components may not be available or compatible in the context of possible analysis or test scenarios [5].

Static analysis [6] of software provides a way to assess dependences between various software components based on the analysis of the source code of the software. For example, in case of object-oriented software, static analysis provides information about the interconnectedness of classes through their methods. Static analysis can be useful in evaluating the dependability and maintenance requirements of complex software systems; however it requires access to the source code of all components of the system. Dynamic analysis [7] offers a somewhat different view of the software system, capturing only interactions between components that actually happen during the running of the software. In a sense static analysis provides a static summary of all possible interactions, while dynamic analysis offers a dynamic slice of this static description that is characteristic of the software system in some usage context. Thus that dynamic analysis may be able to capture in a more valid manner what is truly important in terms of software component interactions within the software system, given a range of typical usage scenarios. In both cases of static and dynamic analysis however there are serious difficulties related to data complexity [8], visualisation [9] and practical interpretation [10].

As we noted above, both static and dynamic analysis generates a description of the software system in terms of interactions between components (e.g. method calls between classes or objects). This description of the software system can be considered as graph or network with nodes being classes (or objects) and edges (arcs) being the method calls between these. Thus, network analysis [11], [13] may lend us some help in extraction of meaningful information of highly complex networks of software component interactions [12]. However, most such analysis to date primarily to show that software as a network has certain network properties (e.g. being a small-world [14] or a scale-free network [15]), but do not really give guidance about how to use network analysis to make software better in terms of improved dependability or support for software maintenance.

We show here that the combination of dynamic analysis with network analysis methods can detect valid vulnerabilities of complex software. We also show that some network analysis methods are better than others in predicting the importance of network edges in the context of dynamic analysis of software. For the purpose of demonstration of our work we chose to analyse the JHotDraw 6.01b [16] software. We expect that combined application of dynamic analysis and network analysis of software systems may lead to better support of software maintenance and evolution in terms of determining likely vulnerabilities and also possibly informing about available options to mitigate such vulnerabilities (e.g. by indicating what to patch and how to patch in a large software

This work has been supported by an EPSRC DTA studentship grant

system to avoid malicious attack or context dependent dysfunction).

The rest of the paper is structured as follows. First we review briefly the related works in static and dynamic analysis of software and in relevant aspects of network analysis. Then we discuss software systems as networks and the relationship of this with static and dynamic analysis. Next we present our data and results. The paper is closed by the conclusions and future work section.

II. Related Works

A. Static Analysis

Static analysis is any form of analysis that does not require the system being analysed to be operated [6]. Static analysis of software typically involves analysis of source code or binary, using a static analysis tool. Usually these tools are language specific, and contain a non trivial model [7], [17] against which source code or binary code under investigation is analysed.

The two core concepts that are often used in defining static analysis metrics are: (1) coupling, which is a measure of strength of interconnection [18]; (2) cohesion, which is a measure of intermodular functional relatedness [18]. The key static analysis metrics defined by [19] are the following: (1) coupling between objects – CBO; (2) response for class – RFC; (3) lack of cohesion in methods – LCoM; (4) depth in inheritance tree – DIT; (5) number of children – NOC; (6) weighted method complexity – WMC. These metrics quantify coupling, cohesion, inheritance relationships and complexity of a class in object oriented systems [20].

Static analysis has been used in the context of software dependability evaluation for example to search for coding errors [21] and for supporting testing and verification [22]. Static analysis methods are also used to support software maintenance and software understanding. For example, [23] used static analysis to improve traceability, while [24] use combination of static methods with text analysis methods to improve maintenance and reuse of software components.

Static analysis is well understood and researched, even if the definitions of its core concepts and metrics may vary to some extent. Consequently it is well supported by various software engineering tools (e.g. Moose [25], JRipples [23] and can be integrated relatively easily into software engineering methodologies. However, an important deficiency of static analysis methods is that the information that they provide about the software system is a static summary of what the system might do and of how the system might behave. This means that there is no information provided about the likelihood and importance of various variants of action and behaviour.

B. Dynamic Analysis

Dynamic analysis looks at the software system at runtime [6]. This involves execution of the system, and often the executing of a certain planned usage scenario. For example, in case of runtime memory studies, commonly performed in the high performance computing (HPC) domain, executions are normally carried out on unloaded processes, so as to isolate results from interference from other software, sharing the process [26].

Dynamic analysis is realised by tracking and logging interactions between software components [9], for example by logging entries and exits of methods in the context of object oriented software. There are various methods and tools that are used for dynamic analysis (e.g. method entry and exit aspects in AspectJ, monitoring agents in Eclipse). In conceptual terms, dynamic analysis often follows the path set by static analysis by focusing on various dynamic coupling and cohesion metrics of the analysed software [27].

Dynamic analysis has been used to support software comprehension [28] and maintenance through visualisation of dynamic interactions between classes [9], [27] and through the analysis and visualisation of dynamic object flow [29]. Dynamic analysis has also been used to support the certification of software components in the context of dependable software engineering [30].

Dynamic analysis offers a partial view of the software system, since it includes information about events that happen during running of the software over a period of time. The execution of the software may follow pre-set scenarios or usual usage patterns, but in all cases it is likely that the data included in the dynamic analysis will leave out some possible interactions between software components that were not executed during the considered run-time data collection period. In this sense dynamic analysis provides a picture of the system that represents a slice of the static picture of the system that may be generated using static analysis. Usually the data resulting from dynamic analysis is also very complex (just as in the case of static analysis) and its interpretation and handling presents significant challenges.

C. Network Analysis

Network analysis has its roots in the random graph theory of Erdos and Renyi and its applications expanded rapidly in the last two decades [11]. It considers representations of systems as networks or graphs of nodes and edges, where nodes usually represent system component and edges their interactions. A few major types of networks with characteristic properties can be recognised by analysing the composition of the network, implying the validity of the characteristic properties for the analysed systems. Commonly used such major types of networks are: exponential random network, scale-free network and small-world network [31].

In general, complex systems can be represented as complex networks, which allow quantification of some aspects of the complexity of the system [32] and intuitive visualisation of the system [15] that can support reasoning about the system. A commonly accepted assumption of network analysis is that the functional integrity of the represented system and the structural integrity of the representing network (compared to the network representing the fully functional system) are closely interrelated. According to this assumption it is possible to analyse the structural features of network components in the context of the network representing the fully functional system and to derive the functional importance of system components

represented by these network components on the basis of this network structure analysis [13].

Network analysis methods include methods that establish the type of the networks, methods that measure the structural integrity of the network, and methods that analyse structural properties of network components with the aim of determining such components that are the key contributors to the structural integrity of the network [13]. The type of the network can be established for example by analysing the node connectedness distribution of the network. Structural integrity can be measured for example by calculating the average shortest path length or average clustering coefficient of the network. There are many methods developed in recent years that aim to measure the importance of network components [33]. These methods include the calculation of connectedness of nodes and implied connectedness of edges, the calculation of the betweenness value of nodes and edges, the determination of the frequent non-trivial network motifs, and many others [34].

Network analysis applied to various systems (e.g. the Internet [35], protein interaction systems of unicellular organisms [36]) led to impressive results predicting high vulnerability components or estimating the level of robustness of the system. These results were achieved by relying on the central assumption of these methods about the close relationship between functional and structural integrity of the represented system and the network representation of it. However, in general it is very difficult to establish the validity of claims that more sophisticated network analysis methods are able to determine indeed system components with high functional importance. This is because experimental validation of such predictions in case of complex biological or socio-technical systems is extremely difficult due to imprecision, complicatedness and high cost of experimental procedures or the practical impossibility of doing such experiments (e.g. because of ethical reasons, lack of sufficiently sensitive measurements, etc.).

III. Software Systems as Complex Networks

Here we consider software systems developed in an object oriented language environment (e.g. Java, C++). Such software systems are developed by defining classes that form class hierarchies by considering ancestor – descendent relationships (i.e. the descendent class is a modification of the ancestor class by specification, re-specification, or possibly addition of methods and variables). Class definitions imply instantiations of other classes as objects. The software system delivers its intended service by instantiating one or a few initial objects, which trigger the instantiation of many other objects. The interactions between objects are defined in the class specifications in form of method calls. Events are changes in the environment of the software system triggered either by the software itself or by other sources (e.g. the user, another computer, etc.). Objects may be notified of such events, which may imply the calling of their appropriate methods that handle the presence of these events. The dispatching of events is done by core or system objects that monitor the presence of such events (e.g. they monitor input from the mouse or keyboard). The distribution of event notifications in form of method calls and method calls in general constitute messages, which often have also parameters (i.e. the input variables of the called methods). This brief summary captures key concepts and elements of object oriented software. Of course, various realisations of object oriented development environments may have additions and variations in terms of actual implementation and usage of these concepts.

Static analysis of object oriented systems usually aims to analyse the software code by evaluating the features of classes, and their methods, variables and interactions in order to determine values of various coupling and cohesion metrics that characterise the software system or its parts [7], [37]. The raw data of this analysis can be seen as the network of classes that act as nodes and are linked by edges that represent methods that are called from other classes. Static analysis also considers additional information that can be seen as labels of nodes and edges such as variable passed as parameters, variables that get modified, and so on. In case of large scale software systems there are hundreds or thousands of classes and method calls and the static analysis network representation of the system is comparable to network representations of other complex systems such as unicellular biological organisms (seen as metabolic or protein interaction systems [38]) or socio-technical systems (seen for example as email network of individuals working in an organisation [39]).

Dynamic analysis considers only instantiated objects and the classes to which these belong together with the actual method calls that are effectuated during the run-time of the software systems [6], [27]. As we pointed out earlier this gives a dynamic slice of the static analysis picture of the software system. The advantage of considering this dynamic slice is that this indicates the actual likelihoods of instantiating classes as objects and of calling given methods of these classes in the context of some usage scenario (e.g. typical everyday usage). Similarly to the static case, the object / classes and the method calls linking them can be seen as a network of objects / classes linked by edges (arcs) representing the called methods. The dynamic analysis network represents the actual realisation of the software system, while the static analysis network represents an equal weight mixture of all possible realisations of the software system.

Considering software systems as complex systems represented as complex networks (i.e. network of classes / objects and methods resulting from static or dynamic analysis) means that we can apply network analysis methods to evaluate robustness of these software systems and also to search for vulnerabilities of these systems. Better understanding of system robustness and vulnerabilities is likely to help the analysis and improvement of system dependability and also the evolution through improvements of the system (e.g. development through patching). Naturally, it is an important question the extent to which network analysis methods provide valid measurements of robustness and vulnerability in case of software systems.

The static network of classes and method calls have been analysed for various software systems and usually these results show that the software system in question (e.g. the Linux kernel [32], [40] is a network that is close to scale-free networks in terms of network type metrics (e.g. comparison of connectedness distributions). However, most analyses so far

did not go much further than this in terms of actual analysis, but rather went on to discussion of potential advantages and benefits that could be derived from this fact or from further network analysis of the software system [41]. Notably [32] used network analysis to shed light on evolution features of the Linux kernel by finding nodes in the network representations of variants of the Linux kernel that showed unusual network evolution patterns.

Here we present the network analysis of the JHotDraw 6.01b [16] software and show that some network analysis methods can indeed detect valid vulnerabilities in software systems. We also show that some of these methods are much less effective than expected in terms of finding valid vulnerabilities in the analysed software system.

IV. Determination of Vulnerabilities in Software Systems

A. Data Collection

We chose the JHotDraw 6.01b [16] software as our test bed software. This is software resulted originally from a design experiment [9] and consequently it is considered a well designed software. The code has over 66K lines of code and includes 344 classes with a few thousand methods that can be called. Since this software has been subjected by others to dynamic and static analysis [9] it is a good starting point for the combined use of dynamic analysis and network analysis of software systems (being analysed by others offers possibilities of comparisons that can support the validation of the results).

To collect the dynamic analysis data we need to trace and log the interactions between objects / classes and the methods of which call instantiate these interactions. Dynamic analysis is practiced by many groups but there are relatively little details available about the actual techniques that are used to gather dynamic analysis data (see for example [10], [9], [6]). The main technical options that we considered were as follows: (1) the use of the TPTP Probekit agent in Eclipse[42]; (2) using the Java NetBeans profiler [43]; (3) aspect oriented implementation of crosscutting concerns for the detection of entry and exit of methods using AspectJ [44], [45]. We used as our dynamic analysis data generation method the TPTP Probekit agent including tracking the entry and exit of methods and the analysis of the stack trace. At the time points of entry and exit checking the Probekit agent writes into a log file tracking the execution of the program, and following the entry phase the agent also investigates the stack trace in order to determine the current class, the caller class, and the current class method that has been called by caller class.

The data that we analysed included around 900,000 entries for each run. We generated this reproducing each time the same operation sequence as the one used in [9] – i.e. we generated three drawing panels, placing on each after being generated five drawing objects. The entries that we analysed include the names of the caller class, the called class and the called method of the called class.

We also used the Java NetBeans profiler [43] approach to collect comparable data to validate the results generated by the TPTP Probekit agent [42]. In addition we also used the Moose toolkit [25] to generate static analysis data about the JHotDraw 6.01b software in order to check the validity of our results. For the purpose of visualisation we used the Pajek graph visualisation software [46].

B. Network Analysis

First we processed the data to generate a network representation of it. We found 195 classes that were active during our sequence of operation. There were 817 methods of these classes that were called during the runs of the software. We ignored the direction of the calls and considered all method calls as undirected edges (and not as directed arcs). Note that two nodes representing classes may be connected by many edges representing different method calls between the two classes. It should be also noted that a method of a class may be called by more than one objects belonging to different classes, in such cases the method will be used to label all these interactions between classes, i.e. the same method label may appear attached to different edges. A network representation of the dynamic analysis data derived from the software is shown in Figure 1A.

We analysed the connectedness distribution of the network nodes, i.e. the connectedness of a node is the number of edges connecting a node to other nodes. Considering all edges for all nodes representing classes the best fitting distribution of the connectedness values is log-linear (see Figure 1B) with the probability density function of the connectedness values:

$$p(x = a) = \frac{-4.9\ln(a) + 45.167}{a} \quad (1)$$

While this is not a probability density function corresponding to a power law distribution (e.g. $p(x = a) = \alpha \cdot \frac{1}{a^{\gamma}}$, a typical connectedness distribution of a scale-free network has a γ value in the range of 2 – 4) this distribution also provides a much longer tail than an exponential distribution (e.g. $p(x = a) = \frac{1}{\lambda} e^{-\lambda a}$) – i.e. having very highly connected nodes is relatively likely – which makes meaningful the application of network analysis methods for the determination of components that have high importance for the structural integrity of the network.

We used network analysis methods to determine important edges of the network that are likely to contribute significantly to the network's structural integrity. According to the core assumption of network analysis such edges are likely to represent interactions that contribute critically to the functional integrity of the software system. To find important edges we

used three network analysis methods to calculate such importance values of edges:

(1) we calculated the hub connection score (HCS) of edges as the product of the connectedness values of the nodes that are connected by the edge, i.e. if the edge e connects nodes n and m , with connectedness values $v(n), v(m)$ the hub connection score of the edge is:

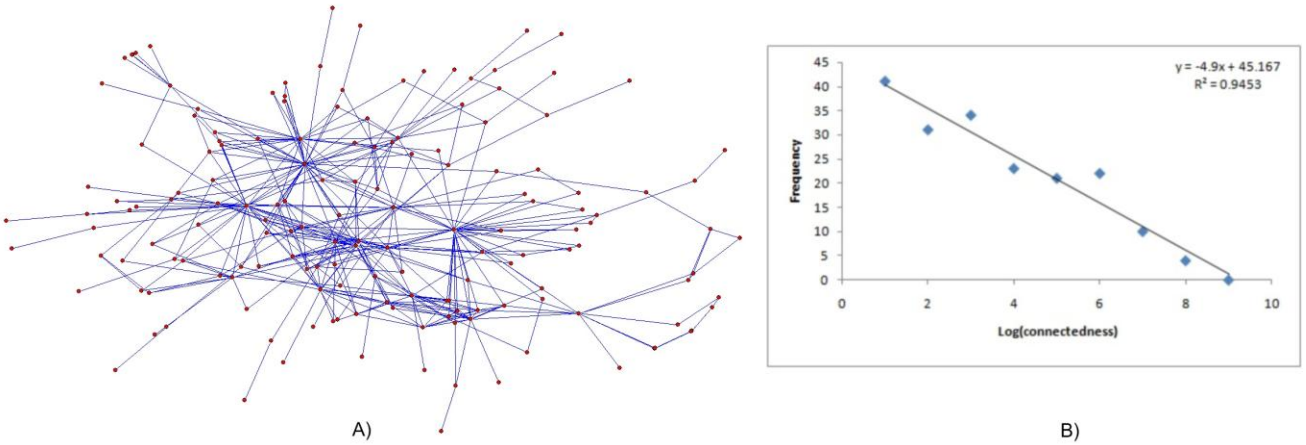

Figure 1. A) The network representation of the JHotDraw 6.01b; B) The distribution of the log(connectedness) values of the nodes of the network together with the best linear fit line and its equation and R^2 value in the upper right corner (R^2 value close to 1 indicates a good fit between the data points and the linear relationship estimation).

$$HCS(e) = v(n) \cdot v(m) \quad (2)$$

(2) the frequency weighted connection score (WCS) of an edge e that connects the nodes n and m, with connectedness values $v(n), v(m)$, is calculated considering the call frequency of the method corresponding to the edge, $f(e)$, as:

$$WCS(e) = f(e) \cdot v(n) \cdot v(m) \quad (3)$$

(3) the betweenness score (BWS) of an edge is the number of shortest paths connecting nodes of the network that contain the edge; there may be more than one alternative shortest paths between two nodes; the length of an edge was set to be the inverse of the call frequency of the method represented by the edge:

$$BWS(e) = \left| \begin{cases} \{e_1,...,e_k\} \mid e \in \{e_1,...,e_k\}, \\ \sum_{i=1}^{k} \frac{1}{f(e_i)} \le \sum_{j=1}^{k'} \frac{1}{f(e'_j)}, \\ \forall \{e'_1,...,e'_{k'}\}: \\ |nodes(e_1) \cap nodes(e'_1)| \ge 1, \\ |nodes(e_k) \cap nodes(e'_{k'})| \ge 1, \\ |nodes(e_i) \cap nodes(e_{i+1})| = 1, \\ |nodes(e'_j) \cap nodes(e'_{j+1})| = 1, \\ i = 1,...,k,\ j = 1,...,k' \end{cases} \right| \quad (4)$$

where $nodes(e)$ determines the two nodes that are connected by the edge e.

All edges were ranked according to these edge importance metrics and we considered the highest ranked edges as the ones that are likely to have the highest contribution to the structural integrity of the network, according to the considered importance metric. Consequently, the prediction according to the assumptions of network analysis is that the corresponding methods are likely to have an important contribution to the functional integrity of the software system represented by the network. In addition, as an indirect validation of our ranking of methods, we also ranked the methods according to their call frequency determined through the use of NetBeans profiling. The top-5 ranked methods for all three network analysis based rankings and for the call frequency ranking are shown in Table 1.

C. Evaluation

Our aim is to assess to what extent are valid the predictions of network analysis methods about the critical importance of methods corresponding to highly ranked edges of the network representing the software system. To do this we disabled the highly ranked methods one-by-one and tried to execute the same sequence of operations that we executed to generate our dynamic analysis data. For each disabled method we tried to do the minimal damage to the software code in order to avoid trivial errors (e.g. if the method is expected to return a pointer to an object and does not return anything then it generates an error immediately).

We note that the choice of method disabling may be to some extent subjective. However we tried to do this in a principled manner by applying the same kind of disabling to similar kinds of methods and by keeping the way of disabling as simple as possible, while avoiding to cause trivial errors. For example, in case of methods for which the return type is 'void' (i.e. nothing is returned) we simply shortcut the method's entry and exit, without executing anything in between. Or, in case of methods that return an object we created a default object of the right type, which is returned by the method, while the actual execution of the method is skipped.

We also note that a method that is selected on the basis of an importance metrics ranking may also be called by other objects / classes as well, and not only by the class which is linked by it in the context of the network edge that led to the selection of the method as likely to be functionally important component of the software (i.e. if method M of class A is called by class B and this edge – B calls A.M – is ranked as highly

TABLE I. THE TOP-5 RANKED METHODS ACCORDING TO THE THREE NETWORK ANALYSIS BASED SCORES AND THE CALL FREQUENCIES. ALL METHOD NAMES START WITH 'ORG.JHOTDRAW.'.

	HCS scoring		**WCS scoring**
Rank	**Method**	**Rank**	**Method**
1	standard.StandardDrawingView.tool	1	util.PaletteButton.mousePressed
2	standard.StandardDrawingView.paintComponent	2	contrib.zoom.ZoomDrawingView$3.Constructor
3	contrib.AutoscrollHelper.Constructor	3	contrib.zoom.ZoomDrawingView$2.mouseMoved
4	framework.FigureAttributeConstant.getName	4	application.DrawApplication.toolDone
5	framework.FigureAttributeConstant.hashCode	5	contrib.AutoscrollHelper.Constructor
	BWS scoring		**Call frequency**
Rank	**Method**	**Rank**	**Method**
1	standard.StandardDrawingView.tool	1	framework.FigureAttributeConstant.hashCode
2	framework.FigureAttributeConstant.getName	2	figures.FigureAttributes.get
3	standard.StandardDrawingView.paintComponent	3	figures.AttributeFigure.getAttribute
4	framework.FigureAttributeConstant.hashCode	4	figures.AttributeFigure.getDefaultAttribute
5	framework.FigureAttributeConstant.setID	5	framework.FigureAttributeConstant.equals

important, it is possible that the same method M of class A is called by other classes C_1, ..., C_r). Of course, this means that disabling the method has an impact on the functionality of calls of the method from any other class as well. Thus, the disabling of the method implies the lost functionality of the edge that was selected, and also possibly of a number of other edges that represent the calling of the same method by other classes. In this sense the functional disabling of the method does not correspond strictly to the removal of an edge in the network representation of the system, but includes this and may also have additional side effects in terms of removal of the functionality of other edges as well.

Running the software after making the damage (i.e. disabling the normal functioning of a method) is expected to lead to a crash or some other significant error if the prediction about functional importance of the method is correct. Considering the top-50 highly ranked methods according to the four rankings that we generated, we found that in some cases the disabling of the method indeed caused crash or significant functional failure, while in other cases there was no notable effect of the disabling of the method. In particular we found that methods that were highly ranked according to the importance metrics HCS and WCS and according to the call frequency calculated using NetBeans profiling, were most likely to cause crash or significant functional defect to the software. On the other side in case of ranking of edges on the basis of betweeness scores did not produce very highly ranked nodes that would induce crash or functional defects after being disabled. Figure 2 shows a graphical summary of the effects of the disabling for the top-50 methods for all four method rankings. Figure 2 shows that ranking based on network metrics HCS and WCS and on call frequency find at least 6 methods out of the top-10 which are highly functionally important (the corresponding lines are above the 60% level), while this is not the case for the ranking based on the BWS metric.

For comparison we considered 100 randomly selected methods and we determined experimentally the methods that were functionally important in this random selection of methods. We found that the chance of finding a functionally important method in the JHotDraw 6.01b software is around 60%. This means that the network analysis based rankings that produce over 60% functionally important methods in their top range perform better than chance.

We also investigated pair wise combined disabling of a selection of methods that ranked high according to at least one of the importance metrics but did not cause significant functional defect after being disabled on their own. However, this analysis did not find any combined disabling that would have caused significant faulty behaviour in the software.

The above described results show that some network analysis methods (the application of the HCS and WCS metrics) can lead to the determination of valid vulnerabilities in software systems analysed as networks considering dynamic analysis data. The results also show that this is not the case with the ranking on the basis of betweenness scores. At the same time the simple call frequency ranking also provides relatively good predictions about functionally important methods (of course, we should note that call frequency may also be seen as a network analysis importance metric). This is a promising result that shows that applying network analysis methods can provide useful information for improving software dependability and for supporting software maintenance. However, our results also show that the applicability of network analysis methods has to be validated and more research is needed to find effective methods of network analysis that can detect valid vulnerabilities of software systems.

V. CONCLUSIONS AND FUTURE WORK

We analysed here the joint application of network analysis methods [13] and dynamic analysis of software systems [7] to detect vulnerabilities of the software in terms of methods with critical contribution to the functional integrity of the software. Applying these methods to a test case, the JHotDraw 6.01b [16] software, we have shown that some network analysis methods (hub connection score, weighted connection score, and call frequency ranking) can detect valid vulnerabilities in terms of functionally critical methods. We have also shown that some of these methods do not work as expected (betweenness score ranking).

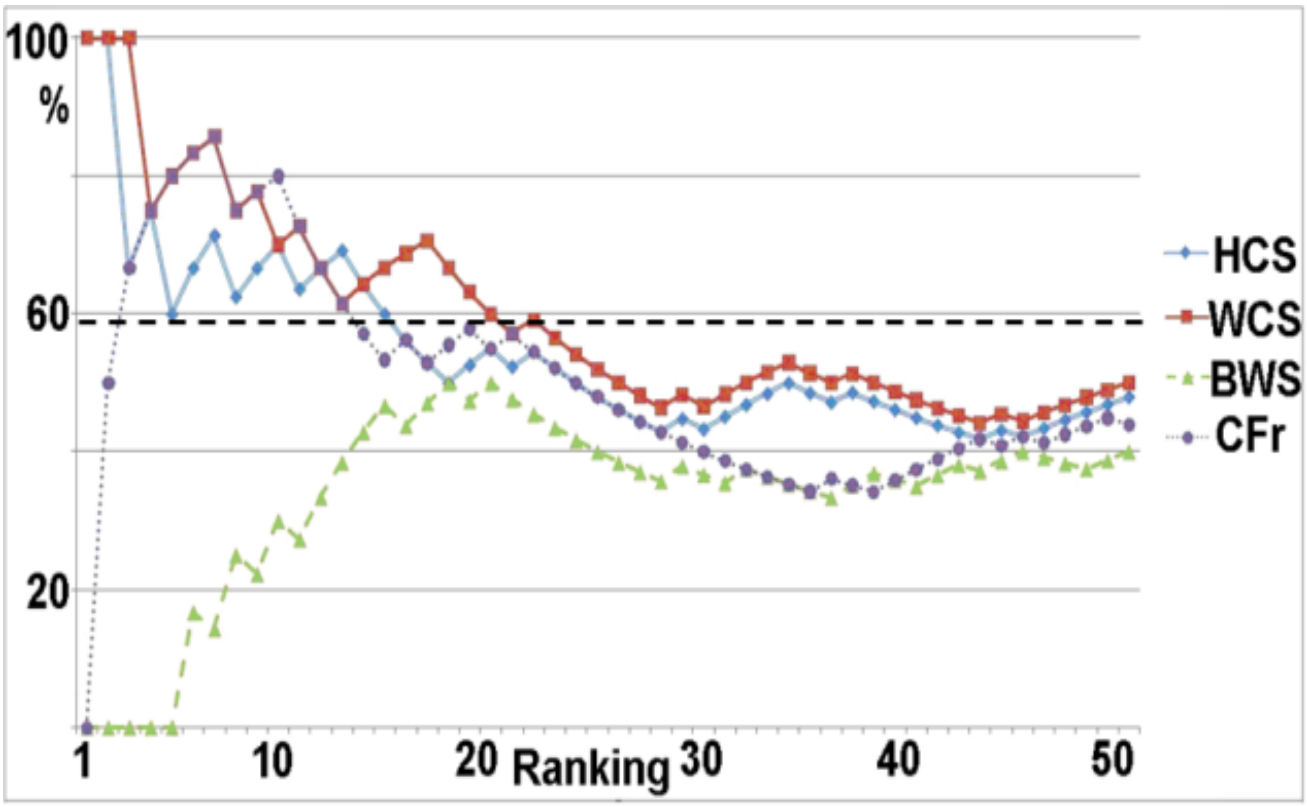

Figure 2. Graphical presentation of the effectiveness of edge ranking methods to determine methods which if disabled cause software crash or significant malfunction: the lines present for the results for the four ranking methods (based on network metrics HCS, WCS, BWS, and call frequency – CFr) by showing the percentage of the methods up to a given rank, which are validated as highly functionally important methods (i.e. 100% for a rank r means that all methods up to rank r are highly functionally important, 50% means that only half of them up to this rank are experimentally validated as highly functionally important). The dashed line shows the likelihood of a randomly chosen method to be functionally important.

We expect that by appropriate selection of network analysis methods and fine tuning of them we can build up a methodology of combination of dynamic analysis and network analysis that can deliver effective methods to support improvement of software maintenance and software analysis methods applied to software systems are needed, including validation tests of these methods.

We aim to extend this work to much larger software systems and also to software written in other language environments (e.g. C++). Ideally this kind of analysis should work in a programming language independent manner and without dependence on access to source code of the software. We aim to move in this direction and possibly achieve this in the longer term.

We also aim to use the results of the analysis to define and derive intervention policies and practical implementations (e.g. patches) that can improve software systems in terms of their dependability and can support the maintenance and evolution of them towards improved functionality. This may become possible by considering the larger network environment of identified software vulnerabilities and by elaboration of some form of (semi-)automated reasoning that may lead to such policies and implementations (e.g. patching need prediction).

REFERENCES

[1] Greg Goth, "Ultralarge Systems: Redefining Software Engineering?," IEEE Software, vol. 25, pp. 91-94, 2008.

[2] A. Avižienis, J.-C. Laprie, Brian Randell, Carl Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, pp. 11-33, Jan.-Mar. 2004, doi:10.1109/TDSC.2004.2

[3] Lawrence D. Alexander, "On the Challenges of Maintaining Large-Scale Software Systems at Lockheed Martin,", 22nd IEEE International Conference on Software Maintenance (ICSM'06), pp.2 ???, 2006.

[4] Kostas Kontogiannis, Panos Linos, Kenny Wong, "Comprehension and Maintenance of Large-Scale Multi-Language Software Applications," icsm, 22nd IEEE International Conference on Software Maintenance (ICSM'06), pp.497-500, 2006

[5] D. Jackson, "A direct path to dependable software," *Commun. ACM*, vol. 52, pp. 78-88, 2009.

[6] M. D. M. Couture , M. Desnoyers, P. M. Fournier, G. Matni, D. Toupin, "Monitoring and tracing of critical software systems State of the work and project definition", DRDC Valcartier TM 2008-144, *Technical Memorandum*, 2008.

[7] M.D. Ernst, "Invited Talk Static and dynamic analysis: synergy and duality," in Proceedings of the 5th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering, Washington DC, USA,pp.35-35, 2004.

[8] S. G. M. Cornelissen, "Evaluating Dynamic Analysis Techniques for Program Comprehension," PhD Thesis,TUDelft, 2009.

[9] B. Cornelissen, A. Zaidman , D. Holten *et al.*, "Execution trace analysis through massive sequence and circular bundle views," *J. Syst. Softw.*, vol. 81, pp. 2252-2268, 2008.

[10] S. Voigt, J. Bohnet, J. Dollner, "Object aware execution trace exploration,", 2009 IEEE International Conference on Software Maintenance, pp.201-210, 2009

[11] A. L. Barabasi, and R. Albert, “Emergence of scaling in random networks,” *Science*, vol. 286, pp. 509 - 512, 1999.

[12] S. Jenkins, and S. R. Kirk, “Software architecture graphs as complex networks: A novel partitioning scheme to measure stability and evolution,” *Inf. Sci.*, vol. 177, pp. 2587-2601, 2007.

[13] R. Albert, and A.-L. Barabási, “Statistical mechanics of complex networks,” *Reviews of Modern Physics*, vol. 74, pp. 47, 2002.

[14] D. J. Watts, and S. H. Strogatz, “Collective dynamics of 'small-world' networks,” *Nature*, vol. 393, pp. 440 - 442, 1998.

[15] Barabási, Albert-László, Bonabeau, Eric, "Scale-Free Networks". "Scientific American", vol.288, pp.50-59, May 2003.

[16] JHotDraw. http://www.jhotdraw.org/.

[17] R. Marinescu, "Measurement and Quality in Object-Oriented Design," icsm, pp.701-704, 21st IEEE International Conference on Software Maintenance (ICSM'05), 2005

[18] Y. Edward, and L. C. Larry, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*: Prentice-Hall, Inc., 1979.

[19] S. R. Chidamber, and C. F. Kemerer, “A Metrics Suite for Object Oriented Design,” *IEEE Trans. Softw. Eng.*, vol. 20, pp. 476-493, 1994.

[20] J. W. Lionel, C. Briand, “Empirical Studies of Quality Models in Object-Oriented Systems,” *Advances in Computers*, vol. 56, pp. 98-167, 2002.

[21] A. Nathaniel, P. William, J. D. Morgenthaler *et al.*, “Evaluating static analysis defect warnings on production software,” in Proceedings of the 7th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering, San Diego, California, USA, pp. ??? 2007.

[22] B. Hailpern, and P. Santhanam, “Software debugging, testing, and verification,” *IBM Syst. J.*, vol. 41, pp. 4-12, 2002.

[23] M. Collin, P. Denys, and R. Meghan, “Combining textual and structural analysis of software artifacts for traceability link recovery,” in Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering, pp. 41-48, 2009.

[24] Emily Hill, Lori Pollock, K. Vijay-Shanker, "Automatically capturing source code context of NL-queries for software maintenance and reuse," 2009 IEEE 31st International Conference on Software Engineering, pp.232-242, 2009.

[25] Moose. http://www.moosetechnology.org/tools.

[26] Kurt B. Ferreira, Patrick Bridges, Ron Brightwell, "Characterizing application sensitivity to OS interference using kernel-level noise injection," Proceedings of the 2008 ACM/IEEE conference on Supercomputing, pp.1-12, 2008.

[27] Erik Arisholm, Lionel C. Briand, Audun F, "Dynamic Coupling Measurement for Object-Oriented Software," IEEE Transactions on Software Engineering, vol. 30, pp. 491-506, 2004.

[28] Bas Cornelissen, Andy Zaidman, Arie van Deursen, Leon Moonen, Rainer Koschke, "A Systematic Survey of Program Comprehension through Dynamic Analysis," IEEE Transactions on Software Engineering, vol. 35, pp. 684-702, 2009..

[29] Lienhard, A., Ducasse, S., and Gırba, T. 2009. Taking an object-centric view on dynamic information with object flow analysis. Comput. Lang. Syst.Struct.,vol.35, pp.63-79, 2009.

[30] J. Voas, J. Payne, "Dependability certification of software components", Journal of Systems and Software, vol.52, pp.165-172, 2000.

[31] Mark Newman, Albert-László Barabási, & Duncan J. Watts, “The Structure and Dynamics of Networks”, Princeton University Press, 2006.

[32] Lei Wang, Zheng Wang, Chen Yang, Li Zhang, Qiang Ye, "Linux kernels as complex networks: A novel method to study evolution,", IEEE International Conference on Software Maintenance, pp.41-50, 2009

[33] M. E. Newman, “Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality,” *Phys Rev E Stat Nonlin Soft Matter Phys*, vol. 64, pp. 016132, 2001.

[34] Stefan Bornholdt, Heinz Georg Schuster,*"Handbook of Graphs and Networks: From the Genome to the Internet"*, John Wiley & Sons, Inc., 2002.

[35] R. Cohen, K. Erez, D. ben-Avraham *et al.*, “Resilience of the Internet to Random Breakdowns,” *Physical Review Letters*, vol. 85, pp. 4626, 2000.

[36] C. Huthmacher, C. Gille, and H. G. Holzhutter, “A computational analysis of protein interactions in metabolic networks reveals novel enzyme pairs potentially involved in metabolic channeling,” *J Theor Biol*, vol. 252, pp. 456 - 464, 2008.

[37] R. Marinescu, “Measurement and Quality in Object-Oriented Design,” PhD. thesis, Faculty of Automatics and Computer Science of the ”Politehnica” University of Timisoara 2002.

[38] H. Jeong, B. Tombor, R. Albert *et al.*, “The large-scale organization of metabolic networks,” *Nature*, vol. 407, pp. 651 - 654, 2000.

[39] R. Guimerà, L. Danon, A. Díaz-Guilera *et al.*, “Self-similar community structure in a network of human interactions,” *Physical Review E*, vol. 68, pp. 065103, 2003.

[40] Andras, P, Idowu, O, and Periorelis, P (2006). Fault tolerance and network integrity measures: the case of computer-based systems. In Proceedings of AISB Convention 2006, pp.90-97.

[41] W. Pan, B. Li, Y. Ma *et al.*, “Class structure refactoring of object-oriented softwares using community detection in dependency networks,” *Frontiers of Computer Science in China*, vol. 3, pp. 396-404, 2009.

[42] EclipseTPTP. http://www.eclipse.org/tptp/.

[43] NetBeans. "NetBeans," http://netbeans.org/.

[44] AspectJ. http://eclipse.org/aspectj/.

[45] L. Ramnivas, *AspectJ in Action: Enterprise AOP with Spring Applications*: Manning Publications Co., 2009.

[46] Pajek. http://vlado.fmf.uni-lj.si/pub/networks/pajek/.