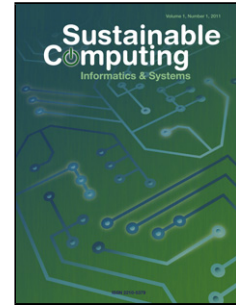


Accepted Manuscript

Title: Software power modeling method at architecture level based on complex networks

Author: Deguang Li Bing Guo Yan Shen Junke Li Yanhui Huang



PII: S2210-5379(16)30004-X
DOI: <http://dx.doi.org/doi:10.1016/j.suscom.2016.08.002>
Reference: SUSCOM 153

To appear in:

Received date: 19-1-2016
Revised date: 30-6-2016
Accepted date: 30-8-2016

Please cite this article as: Deguang Li, Bing Guo, Yan Shen, Junke Li, Yanhui Huang, Software power modeling method at architecture level based on complex networks, Sustainable Computing: Informatics and Systems <http://dx.doi.org/10.1016/j.suscom.2016.08.002>

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

Software power modeling method at architecture level based on complex networks

Deguang Li¹, Bing Guo¹, Yan Shen², Junke Li¹, Yanhui Huang¹

¹College of Computer Science, Sichuan University, Chengdu, 610065

²School of Control Engineering, Chengdu University of Information Technology, Chengdu, 610225

(Email:lideguang.00@163.com , guobing@scu.edu.cn)

Highlights

- We propose our power model at architecture level based on complex networks, which can estimate power consumption of the software with small error and meet requirements of software power modeling at high level.
- We test our power model on real computer platforms and the results validate the rationality of our assumption that there is a nonlinear relation between the network characteristics of software and its power consumption and our method of measuring the network characteristics of software is effective.
- BP neural network as a approximation tool is effectively reflecting the nonlinear relation between the network characteristics of software and its power consumption
- Our experiment proves that software power consumption could be analyzed from high level, which has important meaning for low-power software design.

Abstract

The architecture of software systems can be naturally represented in the form of complex networks, especially for object-oriented software. Software as a kind of artificial complex networks, where entities of the software are nodes and interactions between entities are edges. These interactions are data-flows, instruction-flows and control-flows of the software, and these flows driving hardware circuit is the internal cause of power consumption of software. In this paper, we model software systems as complex networks at architecture level, assuming that the relation between the network characteristics of software and its power consumption is nonlinear. Based on this assumption, we propose a software power modeling method at architecture level. The model first measures network characteristics of software and then fit the nonlinear relation between the network characteristics of software and its power consumption by BP neural network. Experimental results show that our model could accurately estimate power consumption of the software, and the error is less than 11.2% compared to the measured value, which indicates our assumption is reasonable and our model is effective.

Keywords: software power modeling; architecture; complex networks; BP neural network; network characteristics of software

1. Introduction

In recent years, power dissipation and energy saving have acquired comprehensive concerns. According to a report of Gartner entitled "Green IT: a new wave of industry impact" in 2007, carbon emissions of ICT (Information and Communications Technology) equipment one year accounts for about 2% of global carbon emissions, they pointed out that when we Google once, the energy consumed is equal to that consumed by boiling half pot of water [1]. Meanwhile, the power consumed by ICT equipment accounts for 10% of all power consumption of the United Kingdom in 2008, and 8% for the United States. With internet of things, cloud computing and new information technology applications developing, global ICT industry is still growing rapidly. According to a forecast by METI in 2010[2], the power consumed by ICT equipment is forecasted to reach about 20% of all power consumed worldwide by 2025.

Computer system is a typical system controlled by software, which directly drives the underlying hardware. For example, different instruction execution, data access operation and other operations of the software drive the underlying hardware circuit, which indirectly result in power generation. Thus we can conclude that software is the consumer and manager of hardware, software itself does not cause energy consumption, energy consumption is "by-products" of software execution. Also we can say that software consumes energy actively and determines energy consumption of computer system. In order to have a better understanding and improve energy efficiency of computer system, much research has been done from different aspects. Such as upgrade manufacturing process, change circuit structure at circuit level[3-4]; compiler optimization, instruction transformation, instruction rearrangement, loop structure optimization and power analysis at instruction layer [8-11]; expression changes, optimizing data representation, program structure rearrangement, elimination of redundant computation, compressed data storage space, algorithm selection and estimation of energy consumption and execution time at source code level[10-15]; estimating the energy consumption of pervasive Java-based software and distributed Java-based at component level[20-21]and fine-grained power management using process-level profiling at process level[22-24];high-level software energy macro-modeling, system structure selection, transformation and simplification at architecture level[24-21]; and power management technique at system level, such as DPM, DVS, RTOS task scheduling, cooperative game theoretical technique which presents dynamic method for voltage-scaling based task scheduling for simultaneous optimization of performance, energy, and temperature under dynamically varying task and system conditions [22-29].

While software power consumption measuring and estimation is the basis of software power optimization, current software power consumption measuring and estimating method has different levels and granularity, including hardware-based level [5-7];instruction level [8-13];source code level [16-21];software component level [25-26] process level [27-29] and architecture level [30-32]. Hardware-based power measuring method always leverages multimeter to sample the current and voltage of the software and then calculates energy consumption of the software; Instruction-level power model first measures power consumption of each assembly instruction, and then calculates whole power consumption by summing all instructions of the program; while source code level power model first recodes execute paths and power consumption of the software, then calculates power consumption of each line of the code by linear regression; component level

power model which estimates energy consumption of software component by combining construction-time estimation and runtime estimation; architecture level modeling focuses on high level characteristics of software, finds the relation between the characteristics and its power consumption, which can quickly analyze and forecast energy consumption of the software. However, research on building power model at architecture level is few. Software as a kind of artificial complex system, especially for object-oriented software, its architecture can be naturally represented in the form of complex networks and many studies investigate different characteristics of the software from this perspective [27-32]. Inspired by this, we try to explore software power model at architecture level based on complex networks in this paper.

First we model software as complex networks, analyze different network characteristics of software and their influence on software power consumption. These network characteristics include number of nodes, number of directed edges, average path length, clustering coefficient and average degree of the software. Then we assume that the relation between these network characteristics of software and its power consumption is nonlinear, and we use BP neural network to fit the nonlinear relation. Finally we validate our model by experiments and the results show our model can achieve 11.2% error compared to measured value, which indicates our assumption is reasonable and our model is effective. Thus our contributions in this paper are:

- (1) A software power model at architecture level is proposed in this paper, which can estimate power consumption of the software with small error and meet requirements of high level software power modeling.
- (2) We test our power model on real computer platforms and the results validate the rationality of our assumption and our method of measuring network characteristics of software is effective.
- (3) Our experiment proves that software power consumption could be analyzed from high level, which has important meaning for low-power software design at architecture level.

2. Related work

Many software power models have been put forward from different levels, including hardware-based level [5-7], instruction level [8-13], source code level [16-21], software component level [25-26], process level [27-29] and architecture level [30-32]. Now we have a brief introduction to each model.

Hardware-based level: Hardware-based power measurements [5-7] mainly are divided into three types, which are power measurement with meters [5], measurement with special designed devices [6] and measurement by integrating sensors into hardware [7]. Direct power measurement with meters is a straight forward method to understand the power dissipation of devices and the full system, the differences of various measurement methods are which type of meters is used to do the measurement and at which place it is done. Though direct measurement with meters is simple, it does not supply methods to control the process of the measurement process, thus some special designed power measurement devices are presented to measure the power in these circumstances. The third type of approach is mainly used in high-performance servers, which are integrated with power sensors to monitor the power consumed, and then this information is supplied to the administrator for power management. Although we can get power information accurately by hardware, this method requires professional knowledge and it does not easy to operate for ordinary users.

Instruction level: Tiwari [8,9,10] first put forward the concept of software power consumption and proposed instruction-level power model, which was shown in formula (1), total power consumption E_p of the program p is composed of three parts:

$$E_p = \sum_i B_i \times N_i + \sum_{i,j} (O_{i,j} \times N_{i,j}) + \sum_k E_k \quad (1)$$

B_i is the base power cost of instruction i , $O_{i,j}$ is the power consumption caused by circuit state switch between instruction i and instruction j . E_k is the power consumption caused by other effects between instructions (such as pipeline stalls, Cache miss, etc...), all of them are determined by corresponding hardware circuit. N_i is execution times of the instruction i , $N_{i,j}$ is the number of occurrences in the program, all these two parameters are determined by program execution path, path information can be obtained by dynamic analysis of the program. The model helps in formulating instruction level power models which provides the fundamental information needed to evaluate power cost of the entire programs, and has been applied to two commercial microprocessors: Intel 486DX2 and Fujitsu SPARClite 934. Based on the instruction power model, Nikolaidis[6] and Leite[7] proposed fine-grained approach for power consumption analysis and prediction, also put forward new methods for low power applications.

Source code level: Brandolese [16] presented a fully automatic method which combines instruction-level simulation and static-time source characterization for estimating the execution time and power consumption of a C program, also Julien [18] proposed functional approach for estimating the power of an algorithm directly from the C code without compilation. Šimunić [17] employed a profiler, which exploits a cycle-accurate energy consumption simulator to relate the embedded system energy consumption and performance to the source code. Li [19] put forward a power model at source code level, which is implemented by combining hardware-based power measurements with program analysis [22] and statistical modeling. First they recorded code execution paths and their power consumption information by hardware profiling tool as software running, and then used linear regression to calculate the power consumption of each line by the statistical path information and energy consumption. What's more, they put forward software power model at method level and byte-code level [23] based on source code power model. Based on these power models, they found power consumption characteristics of mobile applications in [24], for example on average mobile applications spend 61% of their energy in idle states, network is the most energy consuming component, and only a few APIs (Application Program Interfaces) dominate non-idle energy consumption, which is helpful for improving energy efficiency of the applications. Hannah [21] evaluated power consumption of different algorithms and pointed out that input size and running time are key factors affecting power consumption of the algorithm.

Software component level: Seo [25-26] presented a framework which estimates the energy consumption of a pervasive Java-based software system at software component level. The estimation framework provides a novel approach that facilitates the estimation of a system's energy consumption during construction-time and during runtime, based on monitoring the changes in a small number of easily tracked system parameters. The framework allows the engineer to estimate the software system's energy consumption at construction time and refine it at runtime, which is well suited for a large class of today's distributed, embedded, and pervasive applications. In a large number of distributed application scenarios, the framework showed very good precision on the whole, giving results that were within 5% of the actually measured power losses incurred by executing the software.

Process level: Thanh [27] proposed a simple and efficient process-level power profiling tool pTop, which consists of a power profiling daemon running at the kernel space, continuously profiling resource utilization of each application process according to their ID and track all available system activity information, thus the model cloud provide real-time information about the power consumed by each process in terms of different resource components. Also in [28] they designed a group of APIs for power aware system modules to acquire real-time power information, then the modules make power-aware decisions based on this information. The module called EnergyGuard which can eliminate energy wasted by abnormal-behavior applications. Since pTop is software-based, requires no additional hardware, and it is easy to apply in various platforms. Thus in our paper, we use pTop to profile power consumption of the software for comparing.

Architecture level: To the best of our knowledge, Tan [30-31] made a first step toward considering energy saving could be done during the design of software architecture and proposed a systematic framework for software architectural transformations to reduce power consumption, his work demonstrates the impact of considering energy saving at architecture level, which is consistent with our idea that software energy could be analyzed at high level. What's more, in recent years, complex networks is now attracting an increasing research attention and a lot of studies investigate software from the perspective of complex networks [27-32]. Complex networks as an emerging theory which provides a powerful tool and a new perspective for the research of software systems. Inspired by this, we try to explore software power consumption at architecture level based on complex networks.

3. Our model

First we get the system class diagram of the software by reversing its source code and then convert it into a complex network, where nodes of the network represent classes of the software, and edges represent interactions between the classes. Thus the abstract software network can be defined by a tuple, namely G :

$$G = (V_s, E_s), V_s = \{s_i \mid i = 1, 2, \dots, N\}, E_s = \{e_j \mid j = 1, 2, \dots, M\} \quad (2)$$

Here V_s is the set of classes and E_s is the set of interactions between classes of the software, N is total number of nodes and M is total number of edges. Since we know that all the edges in the network are data-flows and control-flows of the software, and these flows driving hardware circuit is the internal cause of power consumption of software. While for the software network G , each class is modeled as a node, however, we know that the number of fields or methods are different among different classes, correspondingly, power consumption contributed by these different classes will differ, which is also the same for different edges (since the interactions between different classes are different). So in order to accurately reflect the influence of different nodes and different edges on energy consumption, we need to assign different weights to different edges and nodes in order to estimate power consumption of the whole software more accurately. So we present our new weighted software network WG based on the original software network, defined by a four tuple and its formula is (3).

$$WG = (V_s, E_s, H, K) \quad (3)$$

V_s is the set of classes of the software and E_s is the set of interactions between classes, H

represents the weights of each node, and K represents the weights of each edge. The specific weight calculation procedure is given in the next section.

For a given weighted software network, the number of nodes and directed edges describe the size of the software, the weight of each node describes possible execution paths at run time and the weight of each edge describes total interaction paths between two nodes. Average path length reflects communication link costs of message transferring in the network, clustering coefficient reflects the cohesion degree and the transmission property of the networks, and average degree reflects weights of the node for communications cost in the software network, these three parameters reflect communication cost of message passing and data exchange during software running, and all these five characteristics describe the essential characteristics of the software at architecture level. Previous studies [19,21-23,30,33] mainly use linear regression to fit the relation between characteristics of software and its power consumption, using linear regression method has a high speed in power analyzing, while there are many shortcomings. In theory, the assumption that the relation between characteristics of software and its power consumption is linear, which lacks real support; in practice, accuracy of the method is poor.

Based on above analysis, we consider that the relation between software characteristics and its power consumption is nonlinear (linear relation can be considered as a special nonlinear relation), by analyzing the relation between network characteristics of software and its power consumption, we propose our software power model as follows:

$$E_s = P \times T_s = f(m_s) \times T_s = f(V, E, L, C, K) \times T_s \quad (4)$$

In formula (4), E_s is the total power consumption of the software in T_s period, P is the average power consumption of the software, m_s is network characteristics of software and f is the nonlinear relation between network characteristics of software and its power consumption. V is the number of nodes, E is the number of directed edges, L is the average path length, C is the clustering coefficient and K is the average degree of the network. Now we present detail steps of our power model.

- (1) Assuming that there is a nonlinear relation between network characteristics of software and its power consumption.
- (2) Extract the network characteristics of software related with software power consumption at architecture level, which are V , E , L , C , and K of the software network.
- (3) Get E_s and T_s by running multi sets of benchmarks on the experimental platform and then calculate average power consumption P according to formula (5).
- (4) Training BP neural networks to fit the nonlinear correlation function f , input of the function f is the five network characteristics and output is average power consumption of the above benchmarks.
- (5) Input software network characteristics to the trained neural network, and then get its power consumption of the software.

$$P = E_s / T_s \quad (5)$$

In the following part, we present our detail analysis of the five characteristics affecting power consumption of software and describe how to calculate them, and describe the calculate method of the weight of each node and each edge.

4. Extract network characteristics of the software

4.1. Number of nodes

Software is converted into binary instructions that can be recognized by hardware after compiled, and computer performs various operations according to the instructions. In the execute process, microprocessor completes the operations of instruction fetch, decode, operand fetch, execution and so on. Among them, instruction fetch and operand fetch involve memory read/write access and data transfer, decoding involves internal arithmetic operation and logic operation of the microprocessor, which complete different operations according to the type of the instructions in execution stage, such as memory read and write, I/O read and write, etc. These operations will drive hardware circuit, resulting in energy consumption of the hardware.

It can be seen that the execution of binary instructions is the essence of software power consumption. Tiwari [8,9,10] pointed that each binary instruction has a fixed power consumption value in a specified platform. The more instructions executed, the greater power consumed. For a specific weighted software network, the node set and the edge set determine the number of binary instructions of the software. As for the node set, we know that possible execution paths inside each node is different, thus in order to reflect actual impact of different node on software power consumption, we assign the number of possible execution paths inside the node as its weight. As we know that each node in the network is a specific class, and the number of possible execution paths inside a class is the number of all possible execution paths for all the methods in this class, and the number of possible execution paths for each method is the complexity cyclomatic of the method, so the complexity cyclomatic of all the methods in the class is Weighted Methods per Class (WMC), thus the weight of each node is WMC of the class, which we can get by metrics plugin 1.3.6 [50] in the reversing process, so the weight of all the nodes $H=\{WMC_i/i=1,2,\dots,N\}$, N is the number of nodes of the software network. In this paper, the node set is the specific entity classes of the software, and the abstract classes and interfaces are not included since all of them are just containers of specific classes, which does not drive hardware circuit, thus the calculation of number of nodes V is presented in formula (6), V_i is node i and its initial value is 1, WMC_i is the WMC value of class i and V_S is the node set.






$$V = \sum_{i=1}^N V_i * WMC_i \quad (6)$$

4.2. Number of direct edges

Every edge in the network represents interaction between different nodes of the software, general type of the interactions are message passing, data exchange, and program calls. Message passing is usually unidirectional, and data exchange is bidirectional, the essence of program call is also data exchange, since data and program have the same expression. Message passing and data exchange will drive state switch of the corresponding hardware, thus result in power consumption indirectly.

Table 1 Classification of the directed edge

Relation between classes	edge direction	Edge collection
--------------------------	----------------	-----------------

Generalization		$\langle V_i, V_j \rangle$
Dependency		$\langle V_i, V_j \rangle$
Association		$\langle V_i, V_j \rangle \quad \langle V_j, V_i \rangle$
Aggregation		$\langle V_j, V_i \rangle$
Composition		$\langle V_j, V_i \rangle$

In a weighted software network, when one of the following five kinds of relation (shown in Table 1) exists between two nodes, we say that V_1 and V_2 are related, and this relation will be included in the set of directed edges. For generalization relation, we only calculate the relation between two classes. For dependency and association relation, V_1 mainly depends on V_2 , V_2 is usually a local variable, method parameter, or a call to static method. For aggregation relation, usually V_1 contains V_2 , but V_2 is not an integral part of V_1 . For composition relation, if V_1 contains V_2 , which means V_1 contains the global object of V_2 and V_2 is created at the moment when V_1 is created.

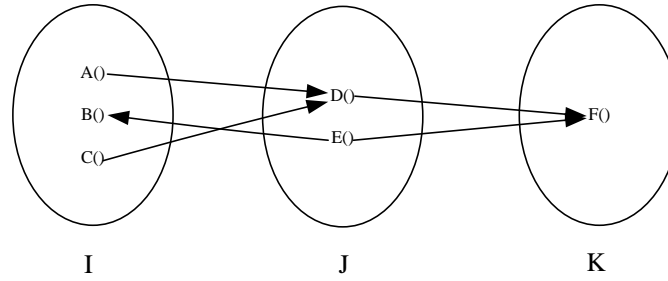


Figure 1 Dependency among different nodes

In the five above directed edges, a directed edge does not accurately describe the interaction between two nodes. For example, as shown in figure 1, node I contains three methods (method A , B and C), node J has two methods (method D and E) and node K contains method F , and the method J and C of node I depend on the method D of node J , while in software network (2), the edges between node I and node J are only e_{ij} and e_{ji} , while these two edges cannot accurately describe the interactions between the nodes, thus in this paper we also assign different weights to the edges between different nodes in weighted network (3), its weight is defined as: $W_{ij} = \sum M_{ij}$, M_{ij} is the method in node I depends on node J , so W_{ij} is the total number of methods in node I dependent on node J , as shown in figure $W_{ij}=2$ and $W_{ji}=1$, so the weight of all the edges $K = \{W_{ij} / i=1,2,\dots,N, j=1,2,\dots,N, i \neq j\}$, N is the number of nodes of the software network. Thus the calculation of number of direct edges E is presented in formula (7), E_S is the edge set and K is the weight of all the edges.

$$E = \sum_{i,j=1,i \neq j}^N e_{ij} * W_{ij} \quad (7)$$

4.3. Average path length

Average path length describes the degree of separation of nodes in the network, and if average path length is short, which indicates the interaction between nodes is close. Recent studies show that although the number of nodes in many real networks is huge, average path length of the network is very short, which shows small world characteristics of the network. From the perspective of system dynamics, average path reflects the strength of the entire software coupling, the coupling strength directly affects the difficulty of software maintenance and change. For

software network, small average path length means strong coupling strength between nodes, thus the transmission cost (such as defect and change of the nodes) is small, and vice versa.

For software network from the power perspective, average path length reflects the average communication cost of message passing of the overall system. When the link is long, more paths are passed through when transferring message, which will incur more state transitions of hardware, thus more energy is consumed. So the average path length reflects the average power consumption of message passing of the whole system. Average path length is calculated by finding the shortest path between all pairs of nodes, adding them up, and then dividing by the total number of pairs. Defining the short path between two nodes V_i and V_j is d_{ij} , if nodes V_i and V_j are not connected, $d_{ij} = 0$. Thus the formula of average path length L of direct network is (8), N is the number of nodes in the network, and it has the same definition as shown in equation (2).

$$L = \frac{1}{N(N-1)} \sum_{v_i \neq v_j} d_{ij} \quad (8)$$

4.4. Clustering coefficient

In general, clustering coefficient and average path length are often mentioned together, since both are two important properties of the "small world" effect; these two parameters reflect degree of cohesion and transitivity of the network. The greater aggregation coefficient, the better transmission of the network, and the communicate path is relatively small when any two nodes of the network for communication, thus the power consumption is relatively less. For software systems, clustering coefficient reflects cohesion of entities in the software system and system organization layering trend.

Clustering coefficient is a measure of the "all-my-friends-know-each-other" property. More precisely, the clustering coefficient of a node is the ratio of existing links connecting a node's neighbors to each other to the maximum possible number of such links. The clustering coefficient for the entire network is the average of the clustering coefficients of all the nodes. A high clustering coefficient for a network is another indication of a small world. Suppose that node i has k_i edges connecting to other nodes, so the clustering coefficient of node i is (9). Where E_i is the existing links connecting to other nodes of node i , and $k_i(k_i-1)$ is the maximum possible number of such links of node i . And the clustering coefficient of the whole network is the average value of C_i of all nodes and the formula is (10), N is the number of nodes of the network.

$$C_i = \frac{E_i}{k_i(k_i-1)} \quad (9)$$

$$C = \frac{1}{N} \sum_i C_i \quad (10)$$

4.5. Average degree

In terms of software system, degree distribution characterizes the connectivity of each node in the network and reflects the reuse degree and complexity of the node. Therefore, the distribution can be used to reflect heterogeneity of the system structure and qualitative analysis uncertainties of the software network. The degree k_i of node i is defined as the number of other

nodes connected to the node, the degree of a node in a directed network is divided into out-degree and in-degree, out-degree of one node is the number of edges from the node to other nodes, in-degree of one node is the number of edges other nodes point to the node. Intuitively, the greater the degree of a node means the more important of the node in the network, the average degree of all nodes in the network is the average degree of the network, it can be defined as $\langle k \rangle$ and shown in formula (11), E is the collection of direct edges of the network and N is the number of nodes of the network.

$$\langle k \rangle = \frac{E}{N} \quad (11)$$

One node with great in-degree reflects more nodes communicating with this node, this is same to the node with great out-degree which reflects it depending on other more nodes, and both reflect the weight of one node for communication in the network. One node with great degree indicate more probability of communication link passing through this node in the network, then the frequency of the node driving hardware circuit is high, and thus generates more power consumption.

5. Nonlinear fitting of BP neural network

In order to describe the nonlinear relation between network characteristics of the software and its power consumption accurately, it is necessary to choose a reasonable and effective numerical approximation method. BP neural network [48] as a numerical approximation method can approximate any nonlinear relation and has a high degree of fitting, therefore we use BP neural network to fit the relation in this paper.

5.1. Nonlinear function fitting procedure of BP neural network

Steps of using BP neural network to realize nonlinear fitting are as follows:

- (1) Get network characteristics of the sample programs. First we select 112 java open-source programs from github and sourceforge as our dataset, the programs are general programs for ordinary users, which mainly are games, media player, office and general use programs. Among them, we randomly choose one hundred programs as the training set, twelve programs as the test set. Then for each program, we write scripts to get network characteristics by reversing its source code and calculate different characteristic values according to the definition in section 4. Next, we normalize these characteristics values and use these values as the inputs of BP neural network.
- (2) Get average power consumption of the training set. For each program, we take the same steps to get its average power consumption, detail experiment configuration and steps are described in section 6. Then normalize power values of the programs and use them as the outputs of BP neural network.
- (3) Design the structure of BP neural network. The important factors include the number of hidden layers, hidden layer nodes, hidden layer transfer function and output transfer function, detail design processes are presented in section 5.2.
- (4) Train BP neural network to get the optimal network. Input network characteristic and average power consumption values of the training set to train BP neural network, get best optimal weights and thresholds of the hidden layer under a predetermined mean square error.

- (5) Input characteristics of test programs to the trained BP neural network and get the forecast power consumption values, and then compare with actual values to verify our software power model.

5.2. Design the structure of BP neural network

Designing a reasonable structure of BP neural network is important to achieve best fitting effect, and the key parameters include the number of hidden-layers, the number of hidden nodes and transfer function of each layer. What's more, neural network learning rate, approximation error, convergence speed and memory usage are also important factors that need to be considered in the training process. Now we list key parameters of our BP neural network.

- (1) Determine the number of hidden-layers. The number of hidden layers determines the network error, but too many hidden layers will increase the complexity of the network, also increase the training time of the network and lead the over fitting tendency. In literature [48], which noticed that a single layer BP neural network can approximate any continuous function in a closed interval, and three layers BP network can complete any mapping of n dimension to m dimension, so we use one hidden layer in our paper.
- (2) Determine the number of hidden nodes. In BP neural network, the number of hidden layer nodes has a great influence on the performance of BP network, while there is still a lack of scientific and general methods to determine the number of nodes in the hidden layer in theory, and the number of hidden nodes (n_1) is always determined by the following empirical formula (10).

$$n_1 = \sqrt{n + m} + a \quad (12)$$

In the formula, n is the number of nodes in input layer, m is the number of nodes in output layer, and a is a constant number between 1~10. In our power model, there are five inputs (number of nodes, number of directed edges, average path length, clustering coefficient and average degree) in the input layer and one output (average power consumption) in the output layer, thus the scope of n_1 is 3~12. In our training process, we find that when the number of nodes in the hidden layer is 11, approximation error and convergence speed can achieve satisfactory results, thus 11 is assigned to n_1 in our work.

- (3) Determine transfer function of each layer. Different transfer functions have different accuracy and learning rate for the network, and there are various options for choosing the transfer function of hidden layer and output layer, such as tansig, hardlim, purelin, hardlims, logsig, etc. Through multiple sets of experiments, we find using the tansig as hidden layer transfer function and purelin as output layer transfer function can achieve satisfactory results. Therefore, we use the tansig and purelin as the transfer function for hidden layer and output layer in the trained BP neural network.

After the structure of BP neural network is determined, we start training our BP neural network. The scope of initial weight value is [-1,1], learning rate $\eta=0.7$ and momentum factor $\alpha=$, 0.01, minimum mean square error of the training is set to 0.0001, the maximum number of iterations is 10000. Then input normalized characteristic values and normalized power consumption values of the training set to the network. Figure 3(a) presents the training error of our BP network in the training process, from the figure we can see that minimum mean square error of the training reaches the required accuracy after 86 iterations. After training, input the test set to the

trained BP neural network and get the predicted power consumption of test set. Here we get the optimal weights (W_1, W_2) of the BP neural network and the threshold values of B_2 and B_1 , as shown in Figure 2.

$$W_1 = \begin{bmatrix} -0.36493 & -0.58707 & -0.1003 & 1.035724 & 1.855533 & -0.41245 \\ -1.20467 & -0.78416 & 1.065832 & -0.61765 & 0.764129 & 0.602211 \\ -0.12742 & -0.51417 & -0.40698 & -1.73954 & -2.02058 & 0.531697 \\ 0.802849 & 1.218968 & -0.15769 & 0.978275 & -0.51417 & -0.91123 \\ 0.114258 & 0.26563 & -0.88062 & -0.94464 & 1.173781 & -0.23418 \\ -0.23331 & -1.22303 & -1.42125 & 0.575745 & -0.02367 & 0.086923 \\ -1.0424 & -0.99477 & 0.406443 & -0.84588 & -0.4762 & -0.89998 \\ -1.20386 & -1.37115 & -0.60392 & -0.7068 & 0.397846 & 0.067678 \\ 1.284278 & 0.607484 & -0.5369 & -1.04817 & 0.114137 & 0.324494 \\ 1.187568 & 1.702296 & -1.4756 & -0.76927 & 0.102407 & -0.13686 \\ -1.08965 & 0.853758 & -1.24692 & 0.521057 & -0.96292 & -0.54297 \end{bmatrix}, W_2 = \begin{bmatrix} 0.645321 \\ -0.85821 \\ -0.42707 \\ 0.311938 \\ -1.03065 \\ 0.111164 \\ 0.05828 \\ -0.05678 \\ 0.512042 \\ 1.193173 \\ -0.07466 \end{bmatrix}, B_1 = \begin{bmatrix} 1.940801 \\ 1.801807 \\ -1.16769 \\ -0.94284 \\ -0.84151 \\ 0.151878 \\ -0.44778 \\ -0.85067 \\ 1.560673 \\ 1.750395 \\ -1.91844 \end{bmatrix}, B_2 = -0.76784$$

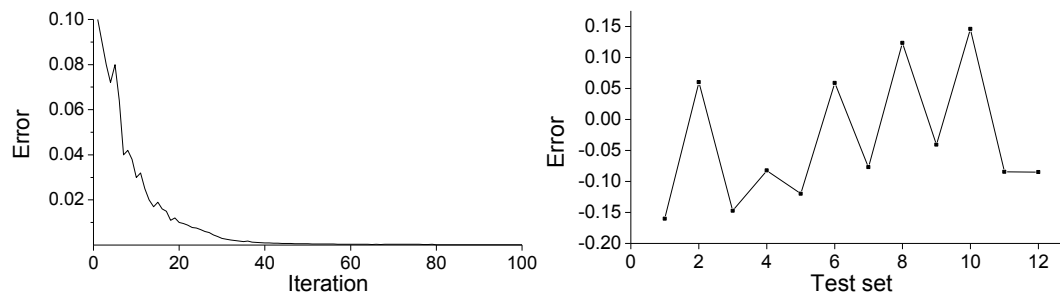
Figure 2 Weight and threshold value of BP neural network

6. Experiment validation and experiment analysis

In order to verify the accuracy of our software power model, we use test set to validate the model. The test programs contain open-source games, media player and general use software. Five characteristics of the test set are shown in Table 3, we evaluate our power model on Ubuntu 14.04.3 platform, which is a normal desktop computer and its configuration shown in Table 2, and using KA3005P DC power supply to provide stable and controllable voltage for the computer, and then we leverage HOIKI 3334 multi-function power measuring instrument to measure instantaneous power consumption and cumulative power consumption of the software.

Table 2 Configuration of experiment platform

Computer	Power meter	Power supply
Lenovo E47	HOIKI 3334	KA3005P DC
Intel Core i3-2310M 2.10 GHz 2 core	Sampling Frequency 74.4kHz	Voltage Range 0V-30V
512 kb L2 Cache 64 kb L1 Cache	Measurement accuracy $\pm 0.5\%$ rdg	Current Range 0A-5A
DDR 2GB \times 2 Frequency 1333MHz	Measurement Range 1.5000W-9.000kW	Setup Accuracy Voltage: $\leq 0.5\% + 20\text{mV}$
Cycle time 6 clocks		Current: $\leq 0.5\% + 10\text{mA}$



(a) Training error curve of BP neural network

(b) Prediction error of the test set

Figure 3 Training error of BP Neural network and Prediction error the test set

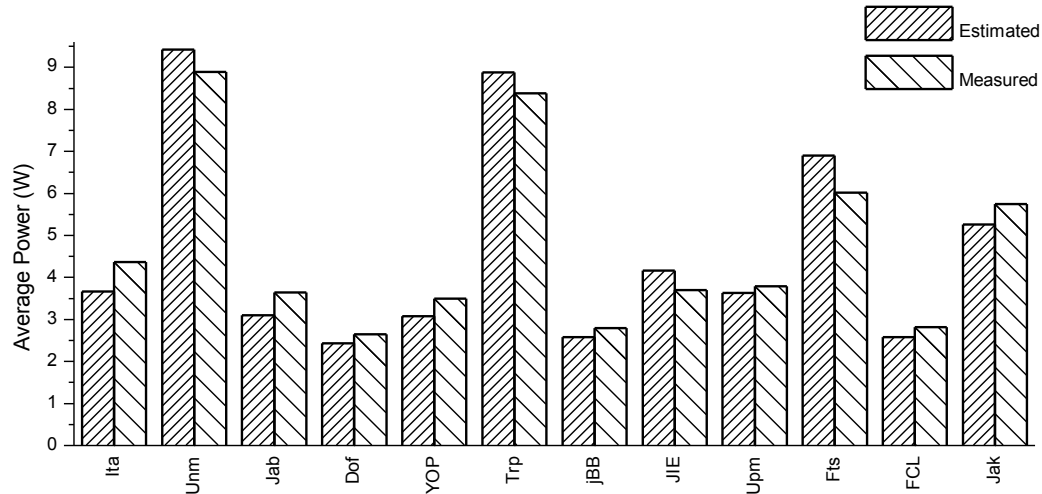


Figure 4 Estimate power value and measured power value of the test set

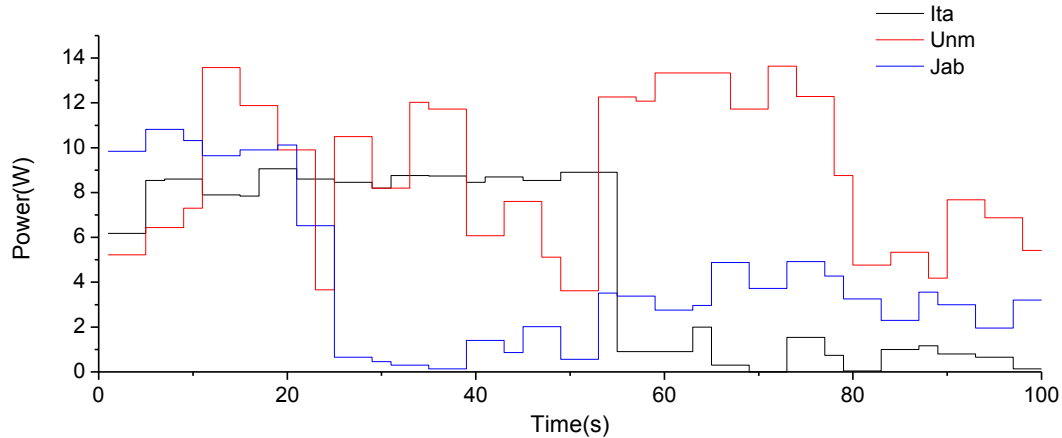


Figure 5 Instantaneous power consumption of Italc, UniversalMediaServer and Jabref.

Table 3 software network characteristic values of the test

	Ita	Unm	Jab	Dof	YOP	Trp	jBB	JIE	Upm	Fts	FCL	Jak
V	91	2784	5140	4290	2674	7266	222	1080	382	2028	185	4428
E	60	1980	4070	2884	3153	14196	159	819	201	1822	159	2267
L	3	10	17	8	9	13	5	7	5	8	5	13
C	0.104	0.254	0.198	0.205	0.5	0.397	0.295	0.227	0.163	0.294	0.295	0.194
<K>	1.429	3.492	3.84	2.931	3.134	3.876	1.632	3.112	1.712	3.562	1.632	3.612

In order to get accurate value for each program, we write automated test scripts interacting with the program being measured based on autorunner [49], which is simulated the scenario of normal user running the software. Also we set the test cycle to be 2 hours for each program to get its cumulative power consumption, and then calculate average power consumption of the program. In order to draw Figure 4, Figure 5 and Table 4 conveniently, we use a list of abbreviations (Ita, Unm, Jab, Dof, YOP, Trp, jBB, JIE, Upm, Fts, FCL and Jak) to replace the name of each test program (Italc, UniversalMediaServer, Jabref, Docfetcher, YOYOPlayer, TripleA, jBubbleBreaker, JIEplorer, Universal password manager, Ftpserver, FreeCell and Jajuk). Figure 3(b) present the prediction error between estimated power value and measured power value of the test set. Figure 4 shows the predicted value obtained by the trained BP network and measured value obtained by HOIKI of the test set and Figure 5 presents the instantaneous power consumption of

Italc, UniversalMediaServer and Jabref in first 100s sampling period.

For the test set, from Figure 3(b) and Figure 4 we find that the maximum error between predicted power value and the measured power value is 16.01%, minimum error is 4.1%, and average error is 11.2%, which is enough to meet the requirements of high level software power modeling. And the scope of average power consumption of common desktop applications is usually about 3W-12W, from Figure 4 we find that average power consumption of Unm is about 9W, which is the highest power consumption of the test set, and its instantaneous power consumption is presented in Figure 5, we can see that the power consumption is in the scope of 3.8W-13.6. In our training set, there are some applications with higher power consumption, thus our power model can also accurately predict higher power consumption. By observing the data in Table 3 and Figure 4, we can see that there is a certain correlation between power consumption and network characteristics of software, the most obvious characteristic is number of direct edges, in Table 3 programs with large E, generally their power consumption value is big, such as the points with large value in Figure 4, but there are also exceptions. Power consumption is not only related to E, but also influenced by other characteristic, that is the reason why the exception occurs, while the relation between other characteristics and power consumption cannot be easily seen from Figure 4. Thus we can conclude that there is big difference of each characteristic in the impact on software power consumption.

Through analysis of the experimental data, we can draw the following conclusions:

- (1) Our software power model at architecture level based on complex networks is effective, through comparison of the test set, average error of our model is maintained in the range of 11.2%, which is within the acceptable range.
- (2) It is proved that the rationality of our assumption that there is a nonlinear relation between the network characteristics of software and its power consumption and our method of measuring the network characteristics of software is effective.
- (3) BP neural network as an approximation tool is effectively reflecting the nonlinear relation.
- (4) There is a certain correlation between each characteristic and software power consumption, each characteristic has different impact on software power consumption.

7. Conclusion

In this paper, we model software systems as complex networks at architecture level, assuming that there is a nonlinear relation between network characteristics of the software and its power consumption. Based on this assumption, we extract five network characteristics of the software, analyze the impact of each characteristic on power consumption and put forward measurement method of each characteristic, and then we propose our software power model. Next we evaluate the validity of our model and the rationality of our assumption by experiments. Currently, we are doing some research in optimization methods at architecture level through node consolidation, node split, node replacement, architecture change and combination of these methods to reduce the power consumption of software.

Acknowledgments

This work was supported in part by the State Key Program of National Natural Science Foundation of China under Grant No.61332001; The National Natural Science Foundation of China under Grant No. 61272104 and 61472050; the Science and Technology Planning Project of Sichuan Province under Grant No. 2014JY0257, 2015GZ0103 and 2014-HM01-00326-SF.

References

- [1] Mingay, Simon. "Green IT: the new industry shock wave." Gartner RAS Research Note G 153703 (2007): 2007.
- [2] Kumon, Kouichi. "Overview of Next-Generation Green Data Center." Fujitsu Sci. Tech. J 48.2 (2012): 177-183.
- [3] Roy, Kaushik, Saibal Mukhopadhyay, and Hamid Mahmoodi-Meimand. "Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits." Proceedings of the IEEE 91.2 (2003): 305-327.
- [4] Wei, Liqiong, et al. "Design and optimization of low voltage high performance dual threshold CMOS circuits." Proceedings of the 35th annual Design Automation Conference. ACM, 1998.
- [5] Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and Kirk W. Cameron. Powerpack: Energy profiling and analysis of high performance systems and applications. IEEE Trans. Parallel Distrib. Syst., 21(5):658–671, 2010.
- [6] Russ Joseph, David Brooks, and Margaret Martonosi. Live, runtime power measurements as a foundation for evaluating power/performance tradeoffs. In In Workshop on Complexity Effectice Design WCED, held in conjunction with ISCA-28. Jun 2001, June 2001.
- [7] Shoaib Kamil, John Shalf, and Erich Strohmaier. Power efficiency in high performance computing. In Parallel and Distributed Processing, 2008.IPDPS 2008. IEEE International Symposium on, pages 1 –8, Apr. 2008
- [8] V. Tiwari, S. Malik, and A. Wolfe. Power Analysis of Embedded Software: A First Step towards Software Power Minimization. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2(4):437–445, 1994.
- [9] V. Tiwari, S. Malik, A. Wolfe, and M. Tien-Chien Lee. Instruction Level Power Analysis and Optimization of Software. The Journal of VLSI Signal Processing,13(2):223–238, 1996.
- [10] M Lee, V Tiwari , S Malik et al. Power Analysis and Minimization Techniques for Embedded DSP Software[J].IEEE Transactions on VLSI Systems, 1997; 5(1) : 123~135
- [11] Nikolaidis, S., and Th Laopoulos. "Instruction-level power consumption estimation of embedded processors for low-power applications." Computer Standards & Interfaces 24.2 (2002): 133-137.
- [12] Leite, Alessandro, et al. "A Fine-grained Approach for Power Consumption Analysis and Prediction." Procedia Computer Science 29 (2014): 2260-2271
- [13] Blume, Holger, et al. "Hybrid functional-and instruction-level power modeling for embedded and heterogeneous processor architectures." Journal of Systems Architecture 53.10 (2007): 689-702.

- [14] Lovic Gauthier, Tohru Ishihara. Processor energy characterization for compiler-assisted software energy reduction. *Journal of Electrical and Computer Engineering*, 2012, 10(8): 1~13.
- [15] Su, Ching-Long, Chi-Ying Tsui, and Alvin M. Despain. "Low power architecture design and compilation techniques for high-performance processors." *Compcon Spring'94, Digest of Papers.. IEEE*, 1994
- [16] Brandolese, Carlo. "Source-level estimation of energy consumption and execution time of embedded software." *Digital System Design Architectures, Methods and Tools*, 2008. DSD'08. 11th EUROMICRO Conference on. IEEE, 2008.
- [17] Šimunić, Tajana, et al. "Source code optimization and profiling of energy consumption in embedded systems." *Proceedings of the 13th international symposium on System synthesis. IEEE Computer Society*, 2000.
- [18] Julien, Nathalie, et al. "Power consumption estimation of a C-algorithm: a new perspective for software design." *LCR. ACM*, 2002.
- [19] Li, D. and S. Hao, et al. (2013). Calculating source line level energy information for Android applications. *ISSTA, ACM*: 78 - 89.
- [20] Potkonjak, Miodrag, and Jan M. Rabaey. "Power minimization in DSP application specific systems using algorithm selection." *Acoustics, Speech, and Signal Processing*, 1995. ICASSP-95., 1995 International Conference on. Vol. 4. IEEE, 1995.
- [21] Hannah Bayer. Evaluating Algorithms according to their Energy Consumption http://www.wagak.cs.uni-kl.de/downloads/papers/Evaluating_Algorithms_according_to_their_Energy_Consumption.pdf
- [22] Hao, S. and D. Li, et al. (2013). Estimating mobile application energy consumption using program analysis. *ICSE, IEEE Press*: 92 - 101.
- [23] Estimating Android Applications' CPU Energy Usage via Bytecode Profiling Shuai Hao, Ding Li, William G.J. Halfond, Ramesh Govindan *ICSE-Greens 201*
- [24] An Empirical Study of the Energy Consumption of Android Applications .Ding Li, Shuai Hao, Jiaping Gui, William Gj Halfond
- [25] Seo, Chiyong, Sam Malek, and Nenad Medvidovic. "An energy consumption framework for distributed Java-based software systems." Submitted to *ACM SIGSOFT* (2006).
- [26] Seo, Chiyong, Sam Malek, and Nenad Medvidovic. "Estimating the energy consumption in pervasive java-based systems." *Pervasive Computing and Communications*, 2008. PerCom 2008. Sixth Annual IEEE International Conference on. IEEE, 2008.
- [27] Do, Thanh, Suhil Rawshdeh, and Weisong Shi. "pTop : A process-level power profiling tool." *Proceedings of the 2nd workshop on power aware computing and systems (HotPower'09)*. 2009.
- [28] Chen, Hui, Youhuizi Li, and Weisong Shi. "Fine-grained power management using process-level profiling." *Sustainable Computing: Informatics and Systems 2.1* (2012): 33-42.

- [29] Merkel, Andreas, and Frank Bellosa. "Balancing power consumption in multiprocessor systems." *ACM SIGOPS Operating Systems Review*. Vol. 40. No. 4. ACM, 2006.
- [30] Tan TK, Raghunathan AK, Lakishminarayana G, Jha NK. High-Level software energy macro-modeling. In: *Proc. of the 38th ACM Conf. on Design Automation*. 2001. 605-610. [doi: 10.1109/DAC.2001.156211]
- [31] Tan TK, Raghunathan AK, Jha NK. Software architectural transformations: A new approach to low energy embedded software. In: *Proc. of the Design Automation Test in Europe*. 2003. 1046-1051
- [32] Li, Ye, Bertan Bakkaloglu, and Chaitali Chakrabarti. "A system level energy model and energy-quality evaluation for integrated transceiver front-ends." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 15.1 (2007): 90-103.
- [33] A. Muttreja, A. Raghunathan, S. Ravi, N. K. Jha. Automated energy/performance macromodeling of embedded software. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2007, 26(3): 542-552.
- [34] V. Konstantakos, A. Chatzigeorgiou, S. Nikolaidis, T. Laopoulos. Energy consumption estimation in embedded systems. *IEEE Transactions on Instrumentation and Measurement*, 2008, 57(4): 797-804.
- [35] C. J. Xian, L. Cai, Y.H. Lu. Power measurement of software programs on computers with multiple I/O components. *IEEE Transactions on Instrumentation and Measurement*, 2007, 56(5): 2079-2086.
- [36] E. Saxe. Power-Efficient Software. *Communication of ACM*, 2010, 53(2): 44-48
- [37] Seungyong Oh, Jungsoo Kim, Seonpil Kim, Chong-Min Kyung. Task partitioning algorithm for intra-task dynamic voltage scaling. In *Proc. of the IEEE International Symposium on Circuits and Systems (ISCAS'2008)*, Seattle, 2008: 1228-1231.
- [38] M. E. Salehi, M. Samadi, M. Najibi, et al. Dynamic Voltage and Frequency Scheduling for Embedded Processors Considering Power/Performance Tradeoffs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2011, 19(10): 1931-1935.
- [39] Sheikh, Hafiz Fahad, and Ishtiaq Ahmad. "Dynamic task graph scheduling on multicore processors for performance, energy, and temperature optimization." *Green Computing Conference (IGCC), 2013 International*. IEEE, 2013.
- [40] Khan, Samee Ullah, and Ishfaq Ahmad. "A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids." *Parallel and Distributed Systems*, *IEEE Transactions on* 20.3 (2009): 346-360.
- [41] Sheikh H F, Tan H, Ahmad I, et al. Energy-and performance-aware scheduling of tasks on parallel and distributed systems. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 2012, 8(4): 32.

- [42] Šubelj, Lovro, and Marko Bajec. "Software systems through complex networks science: Review, analysis and applications." Proceedings of the First International Workshop on Software Mining. ACM, 2012.
- [43] Chatzigeorgiou, Alexander, and George Melas. "Trends in object-oriented software evolution: Investigating network properties." Software Engineering (ICSE), 2012 34th International Conference on. IEEE, 2012.
- [44] Jenkins, Samantha, and Steven R. Kirk. "Software architecture graphs as complex networks: A novel partitioning scheme to measure stability and evolution." Information Sciences 177.12 (2007): 2587-2601.
- [45] Chaikalis, Theodore, and Alexander Chatzigeorgiou. "Forecasting Java Software Evolution Trends employing Network Models." IEEE Transactions on Software Engineering 41.6 (2015): 582-602.
- [46] Chaikalis, Theodore, et al. "SEAgile: Effortless Software Evolution Analysis." Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on. IEEE, 2014.
- [47] Jenkins S, Kirk S R. Software architecture graphs as complex networks: A novel partitioning scheme to measure stability and evolution. Information Sciences. 2007, 177(12): 2587-2601.
- [48] Jin, Wen, et al. "The improvements of BP neural network learning algorithm." Signal Processing Proceedings, 2000. WCCC-ICSP 2000. 5th International Conference on. Vol. 3. IEEE, 2000.
- [49] AutoRunner. <http://www.spasvo.com/news/zhuanti/20131118/>
- [50] Eclipse Metrics Plugin. Available from: <http://metrics.sourceforge.net/>