University of Alberta

USING CITATION INFLUENCE AND SOCIAL NETWORK ANALYSIS TO PREDICT SOFTWARE DEFECTS

by

Wei Hu

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

Fall 2013
Edmonton, Alberta

Abstract

Resource constraints, e.g. lack of time and human resources, is a major issue in software testing practice. In short, testers have limited time to test software systems. Therefore, managers are expected to spend more resources on software components that are likely to contain many defects. To help managers make better decisions of selective testing, it is beneficial to identify defect-prone software components before the actual testing.

In this thesis, we propose a model for software defect prediction. The proposed model combines the topological properties of the software dependency network and the textual information in source code to predict defect-prone software components. We evaluate our model on data from Eclipse, Netbeans, and Gnome projects at different levels of granularity. The evaluation results are encouraging, showing that our model achieves higher prediction accuracy than prior work.

Acknowledgements

I would like to thank my supervisor Dr. Kenny Wong for his time, patience and guidance. It would not be possible to finish this thesis without his support and help. I would also like to express my appreciation to my fellow students and friends for their feedback and suggestions: Haiming Wang, Xiaochao Fan, Yining Wang, and Ronghong Li. The most special thanks goes to my parents who have loved and supported me for all my life.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Defects in software systems are costly. According to a report by the U.S. National Institute of Standards and Technology, due to inadequate testing, the annual cost of software defects on U.S. economy is estimated to be 59.5 billions [52]. There have been numerous efforts in developing software testing tools and techniques to identify and remove software defects [3]. However, as resources, *e.g.* time and human effort, in the software development process are limited, it is infeasible to conduct thorough tests on the entire software system [15]. Therefore, managers are typically expected to allocate these resources on the parts that are likely to contain more defects. To help managers make better decisions on selective testing and resource allocation, it is beneficial to predict defect-prone software components automatically.

In order to predict defect-prone software components, researchers propose to use program dependency relations. Previously, program dependency relations are widely used for software testing [44] and debugging [2]. Zimmermann *et al.* claim that central software components are more likely to be defect-prone and hence the topological properties of the *software dependency network* could be used to assist the defect prediction task [64]. They use program dependency relations to construct the *software dependency network* for binaries (DLL files) in Windows Server 2003, perform a social network analysis on the dependency network to measure the centrality of a particular software component in the network, and subsequently use the network metrics as predictors of post-release defects. They show that social network measures on the software dependency network are good predictors of post-release defects.

Bird *et al.* combine the *software dependency network* [64] and the *developer contribution network* [40] to construct a *socio-technical network* [4]. They argue that the *socio-technical network* considers both overt program dependency relations and latent co-development relations (software components that are developed by the same developer are implicitly related). Figure 2.1 shows an example of the *socio-technical network*. Component A depends on component C and component C depends on component B, D, and E. Developer Bob contributes to component A, B, and C and developer Don contributes to component C, D, and E. The latent co-development relations that both A and B are developed by Bob is captured by the path A-Bob-B and B-Bob-A. Bird's experiment

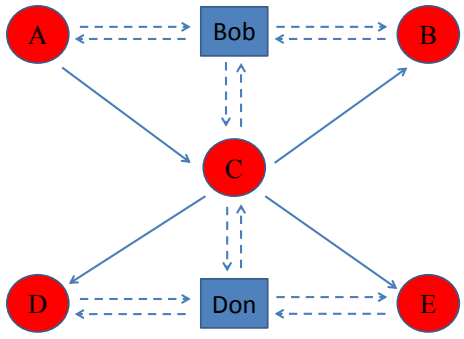


Figure 1.1: Socio-technical network example

shows that to predict post-release defects in Eclipse plugins, the *socio-technical network* approach achieves a dramatically higher accuracy than the *software dependency network* and the *developer contribution network* separately.

We note that both the *software dependency network* approach [64] and the *socio-technical network* approach [4] perform a social network analysis without considering dependency strength information [62].

We argue that dependency strength information can be used together with network topological properties to predict whether a software component is defect prone. The intuition is that dependency strength reveals how tightly software entities are coupled and the degree of coupling indicates software design quality [25]. Specifically, tight coupling between software components is generally taken as poor design as it makes modification and parallel development complex [53]. Therefore, we propose that different degrees of dependency strength (loose coupling and tight coupling) have different contributions to defect-proneness and hence should be analyzed separately.

There are coupling metrics that quantify coupling by considering method invocation [14], message passing [31], and co-changes [63]. Gethers *et al.* [22] argue that these metrics lack the ability to identify implicit relations embedded in comments and identifiers. They further use case studies to demonstrate the efficacy of using topic models [12] to measure entity couplings. Moreover, we note that, except for the topic model based approach [22], all these approaches work by studying code structure and hence cannot measure the relation strength between software components and non-software entities like developers [14, 31]. Therefore, in this work, in order to leverage both overt software dependency relations and latent co-development relations, we use a topic model to quantify coupling degrees between software components and between developers and software components. In the remainder of this thesis, we refer to **dependency strength** as the strength of the inter-component dependency relations and **relation strength** as the strength of either the inter-component dependency relations or the developer-component contribution relations.

In this work, we further leverage the dependency strength information and build a prediction model of defect-proneness. First, we use a topic model [18] to determine relation strengths among components and developers. Second, according to the strength, we classify relations into **weak relations** and **strong relations**. While weak relations correspond to loose entity couplings, *e.g.*, a software component depends slightly on another software component, strong relations abstract tight entity couplings, *e.g.* a developer is very dedicated to a particular software component. Third, according to the level of relation strength, either strong (tight coupling) or weak (loose coupling), we construct two modified *socio-technical networks* for different levels of entity coupling. Fourth, we conduct social network analyses on the *socio-technical networks* separately. These metric values are used to measure the centrality of software components, and are then used as predictor variables in a prediction model of defect-proneness.

We evaluate our approach on seven major releases of the Eclipse IDE (Integrated Development Environment), six major releases of the Netbeans IDE and six major releases of Gnome. We build a regression model with social network metric values as predictor variables, and use data from the previous release to predict whether a software component in the current release is defect prone. The experiment results show that our approach achieves markedly higher precision and recall than prior work.

Thesis Statements

By conducting social network analysis on *socio-technical networks* of different levels of relation strength separately, although the edges between elements within one network are still assigned unit weight, we gain extra information of relation strength. The separation of relations comes from following statements.

TS1: *Strong relations and weak relations have different correlations with the number of defects.*

TS2: *Strong relations and weak relations have different contributions to the predictive power of defect prediction models.*

TS3: *The separation of weak and strong relations improves the predictive power of defect-prone software components.*

TS4: *Social network metrics that are calculated from the weak network and the strong network can be used as predictors to predict defect-prone software components across releases.*

We verify **TS1** with the correlation results in section 4.2, and subsequently use the regression results in section 4.3 to verify **TS2**, **TS3**, and **TS4**.

Thesis Contributions

In this thesis, we propose a prediction model of defect-prone software components, based on social network analysis. Main contributions of this thesis are as follows.

- Combine network topological properties and code textual information to predict software defects.
- Demonstrate how to use the relation strength information to gain defect prediction accuracy.
- Compare our approach with a prior social network analysis based prediction method on a finer-granular data set.
- Compare our approach with a prior social network analysis based prediction method on three existing software systems.
- Build predictors of defect-prone software components, based on the social network analysis.

Most of this material has been published in the Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13) [28].

Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 introduces prior work of using social network analysis to predict software defects and the application of topic models in software engineering research. Chapter 3 sketches how we use dependency strength information to predict whether a software component is defect-prone. Chapter 4 presents our experimental results and demonstrates the improvement of prediction accuracy by using our prediction model. Chapter 5 discusses the threats to validity. Chapter 6 describes related work of using various metrics to predict software defects. Finally, Chapter 7 concludes this thesis with ideas for future work.

Chapter 2

Background

In this chapter, we elaborate on two concepts: the *socio-technical network* [4] and the *citation influence* topic model [18].

2.1 Socio-technical Network

Bird *et al.* argued that dependency relations and contribution history should be used together [4]. They constructed the *socio-technical network* by combining the *software dependency network* (dependency relations) and the *developer contribution network* (contribution history). To better explain the *socio-technical network*, we first introduce the *software dependency network* and the *developer contribution network*.

2.1.1 Software Dependency Network

Zimmermann *et al.* demonstrated that central software components are more likely to face defects [64]. They built the *software dependency network* to capture the centrality of a particular software component within the software system.

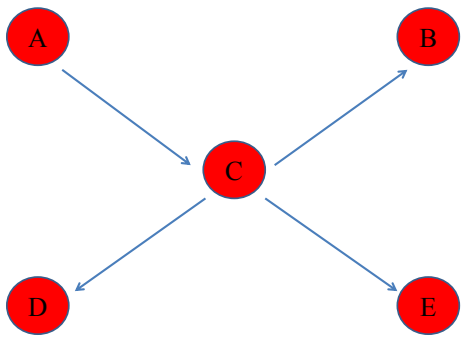


Figure 2.1: Software dependency network example

In the *software dependency network*, nodes denote software components and directed edges between components denote the dependency relations between them. As an example, in Figure 2.1, component A depends on component C, component C depends on component D, B, and E. In this example, component C is more central than other components. Such centrality is included in the network topology and can be extracted by social network analysis.

2.1.2 Developer Contribution Network

Pinzer *et al.* also claimed that network centrality measures are good indicators of defect-prone components [40]. Instead of measuring the centrality of software components within the software system, they constructed the *developer contribution network* and measured the centrality of software components in the context of the development process.

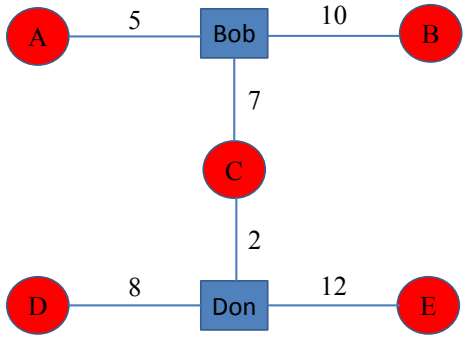


Figure 2.2: Developer contribution network example

Figure 2.2 depicts an example of a *developer contribution network*. Circles denote software components and rectangles denote developers. Undirected edges between a component and a developer represent that the component has contributions from the developer. The edges are all weighted by the number of commits. In this example, component C has contributions from developer Bob and developer Don, and is more central than other components in the development process.

Since all the edges are weighted in the *developer contribution network*, Pinzer *et al.* used a set of social network metrics that can operate on a weighted network to measure the centrality of software components, and used these metrics as predictors of defect-prone components.

2.1.3 Socio-technical Network

Bird *et al.* claimed that the centrality of a software component within the software system (*software dependency network*) and the centrality in the development process (*developer contribution network*) should be used together [4]. They constructed the *socio-technical network* by combining the *software dependency network* and the *developer contribution network*.

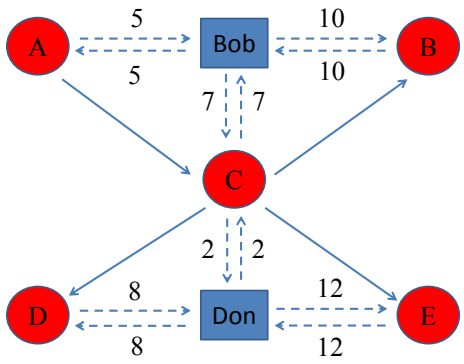


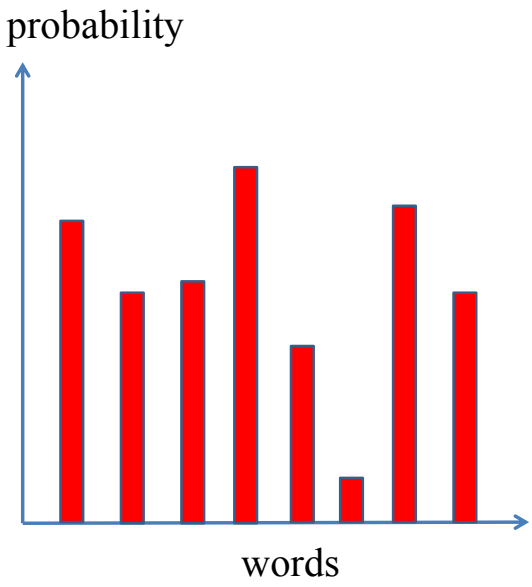
Figure 2.3: Socio-technical network example

The *socio-technical network* [4] can be formally defined as: let $\mathcal{C}$ be the set of software components and $\mathcal{D}$ be the set of developers. The socio-technical network $\mathcal{G}$ is defined as: $\mathcal{G} = (\mathcal{V}, \mathcal{E}_{\mathcal{C}} \cup \mathcal{E}_{\mathcal{D}})$, where $\mathcal{V} = \mathcal{C} \cup \mathcal{D}$, $\mathcal{E}_{\mathcal{C}} \subset \mathcal{C} \times \mathcal{C}$ is the set of unit edges (c, c') that component c depends on c' and $\mathcal{E}_{\mathcal{D}} \subset (\mathcal{D} \times \mathcal{C}) \cup (\mathcal{C} \times \mathcal{D})$ is the set of weighted edges $(c, d), (d, c)$ such that developer d contributed to software component c. Edges $(c, d), (d, c)$ is weighted by the number of commits the developer d made to software component c.

Figure 2.3 presents an example *socio-technical network* that is constructed by combining a *software dependency network* (Figure 2.1) and a *developer contribution network* (Figure 2.2). In Figure 2.3, component A depends on component C and component C depends on component B, D, and E. Developer Bob made five commits to component A, ten commits to component B, and seven commits to component C, and developer Don made two commits to component C, eight commits to component D, and twelve commits to component E.

As shown in Figure 2.3, in the *socio-technical network*, symmetric pairs of directed edges between software components and developers are weighted by the number of commits, whereas directed edges between software components are assigned unit weight. In other words, although the *socio-technical network* considers the strength of contribution relations (number of commits), it ignores potentially useful information about dependency strength.

Some of the social network metrics that Bird *et al.* use (like *betweenness*) can only be calculated on an unweighted network, however, the *socio-technical network* contains both weighted edges and unweighted edges. To remedy the issue of weights, Bird *et al.* convert the *socio-technical network* into an unweighted network. For metrics that can be calculated on both unweighted and weighted networks, Bird *et al.* use both these unweighted networks and the original weighted ones.



word	probability
RED	0.201
PINK	0.169
RENDER	0.153
WHITE	0.110
SCREEN	0.087
BLUE	0.076
BLACK	0.052
PURPLE	0.017
COLOR	0.017
BRIGHT	0.008
...	...

Figure 2.4: Topic: distribution of words

2.2 Citation Influence Model

The *citation influence model* [18] is a topic model that measures the influence of citations on the citing paper in a document network. It extends the Latent Dirichlet Allocation (LDA) model [6] by considering citations. In this section, we first introduce topic models and then give a formal definition of the *citation influence model*.

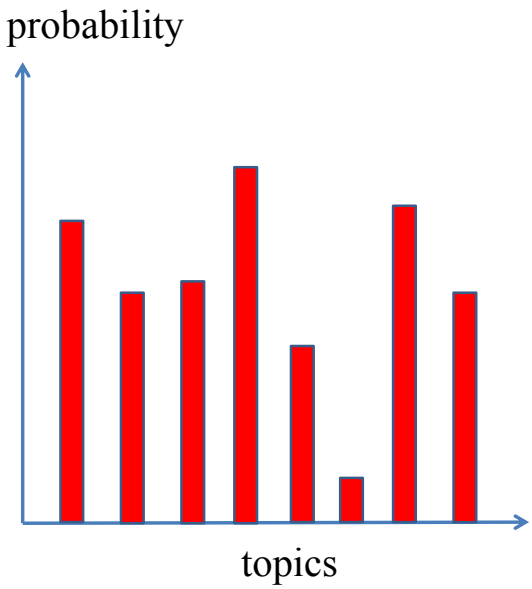
2.2.1 Topic Models

A topic model is a *generative model* that specifies a probabilistic generative process for generating documents. It assumes that a document is a mixture of topics, and a topic is a probability distribution over words.

Figure 2.4 shows an example topic. We can see that a topic is modelled by the probability of each word under that topic. According to the top ten words with the highest probability, we can conclude that this topic is related to colors.

Similarly, a document can be modelled by a mixture of topics. Figure 2.5 depicts an example document as a distribution of topics under that document.

A topic model abstracts the process of generating documents. In short, all the documents are assumed to be generated by a particular generative process, and a topic model seeks to recover the generative process given the documents. Figure 2.6 depicts the generative process for generating documents. In Figure 2.6, the corpus is the collection of all the documents, and the vocabulary contains all the distinct words. To generate a document, for each word, one first selects a topic from



topic	probability
Topic 1	0.301
Topic 2	0.119
Topic 3	0.088
Topic 4	0.071
Topic 5	0.055
Topic 6	0.043
Topic 7	0.032
Topic 8	0.017
Topic 9	0.009
Topic 10	0.008
...	...

Figure 2.5: Document: mixture of topics

its topic mixture according to the probability of each topic under that document, and subsequently selects a word according to the probability of each word under the selected topic.

Note that the generative process seeks to generate documents with given topic mixtures (distribution over topics) and topics (distribution over words). However, to facilitate text analysis, one often needs to infer the set of topics and the topic mixtures that were used to generate a collection of documents, which is opposite to the generative process. Statistical techniques like Gibbs Sampling [41] can be used to invert this process. The inferred topics and the topic mixture under a document are used to model that particular document. Informally, Gibbs Sampling can be described as an iterative process: first randomly select topic mixtures for each document and word distributions for each topic, and then iteratively adjust the topic mixtures and word distributions given the already generated documents, until the adjusted topic mixtures and word distributions converge.

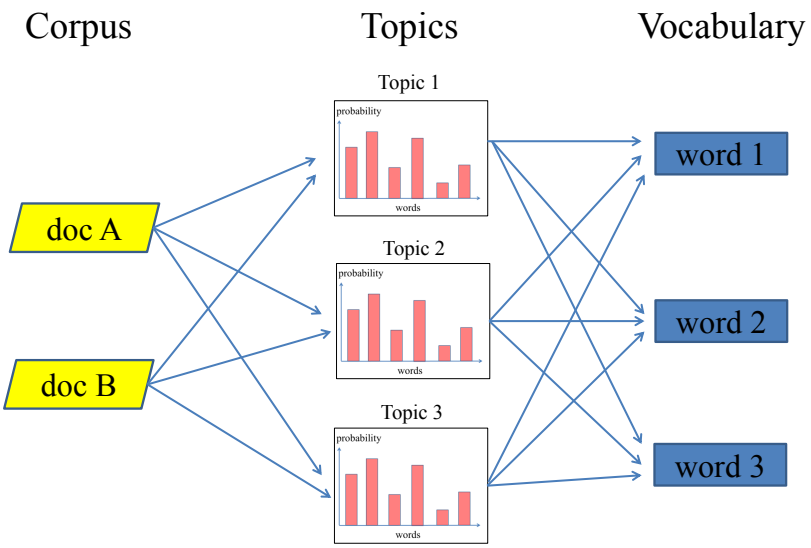


Figure 2.6: Generative process of topic models

2.2.2 Citation Influence Model

The *citation influence model* [18] extends the LDA model [6] by considering citations among documents. Specifically, when generating words, the *citation influence model* assumes that the citing document can either select words from topics of its own topic mixture or from topics that are 'borrowed' from cited documents. The percentage of words that are selected from 'borrowed' topics of a cited document reveals the citation influence of that particular cited document over the citing document.

The generative process of the *citation influence model* is given in Algorithm 1 and depicted in Figure 2.7. In Figure 2.7, solid lines represent generative process of tokens that are selected from a document's own topic mixture, while dash lines represent generative process of tokens that are selected from topics that are 'borrowed' from cited documents. Table 2.1 describes all the symbols used in the algorithm. We refer readers to [18] for detailed definition of the *citation influence model*.

As a topic model, the *citation influence model* can be used to measure the coupling between software components and non-software entities like developers. Subsequently, it can be applied to the *socio-technical network* to quantify both the overt software dependency relations and the latent co-development relations. Therefore, in this thesis, we apply the *citation influence model* rather than other code structure based coupling measures [14, 31].

In this work, we use the *citation influence model* to quantify the strength of the inter-component dependency relations and the strength of the developer-component contribution relations. Simply put, we transform the *socio-technical network* into a document network, use the *citation influence model* to quantify the citation influence and take the strength of citation influence as the strength of

Table 2.1: Notation used in the *citation influence model*.

Symbol	Description
m	the number of documents in corpus
k	the number of topics
v	the number of unique words in the vocabulary
c	cited document
d	citing document
t_i, t, t'	topic
w, w'	word
ϕ_i	distribution of topic t_i over v words
θ_j	distribution of cited document c_j over k topics
ψ_j	distribution of citing document d_j over k topics
$\alpha_\phi, \alpha_\theta, \alpha_\psi, \alpha_\gamma$	Dirichlet parameters of the multinomial distribution
$\phi_t, \phi_{t'}$	word distribution for topic t, t'
θ_c, θ_{c_l}	topic distribution for cited documents c, c_l
ψ_d	innovative topic distribution for citing document d
γ_d	parameter of the distribution of citation influences of citing document d
λ_d	parameters of the coin flip of citing document d
$s_{d,w}$	indicator of whether the word w in d is selected from a 'borrowed' topic

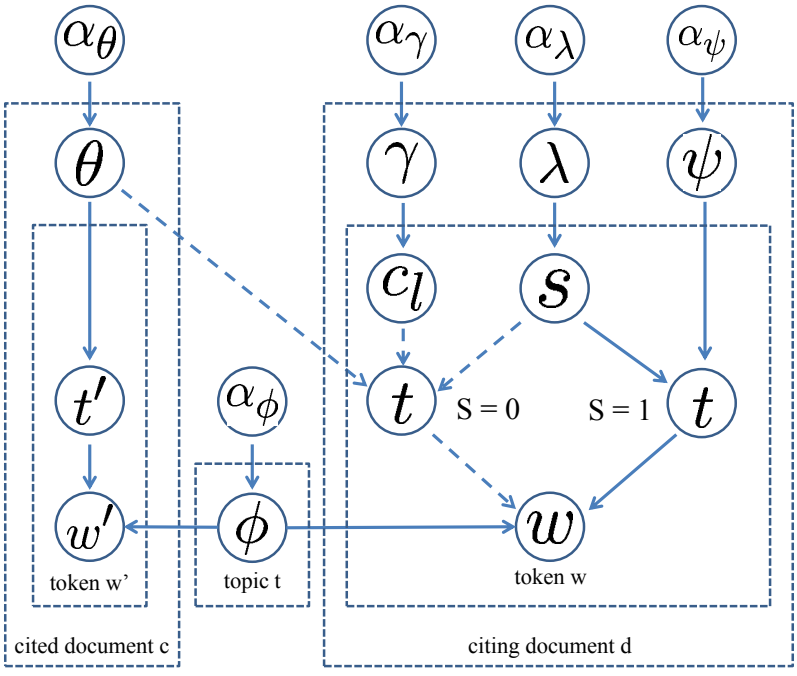


Figure 2.7: Generative process of the *citation influence model*

Algorithm 1 Generative process of the *citation influence model*

for all topic t **do**
  Select $\phi_t \sim dirichlet(\alpha_\phi)$
end for
for all cited document c **do**
  Select $\theta_c \sim dirichlet(\alpha_\theta)$
  for all each word w' **do**
    Select a topic $t' \sim multinomial(\theta_c)$
    Select a word $w' \sim multinomial(t')$
  end for
end for
for all citing document d **do**
  Select $\psi_d \sim dirichlet(\alpha_\psi)$ for its own topic mixture
  for all word w **do**
    Toss a coin $s_{d,w} \sim bernoulli(\lambda_d)$
    if $s_{d,w} = 0$ **then**
      // this word is selected from a 'borrowed' topic of a cited document
      Select a cited document $c_l \sim multinomial(\gamma_d)$ from citations $c_1 \dots c_N$
      Select a topic $t \sim multinomial(\theta_{c_l})$
      Select a word $w \sim multinomial(\phi_{t'})$
    else
      // this word is selected from its own topic mixture
      Select $t \sim multinomial(\psi_d)$
      Select a word $w \sim multinomial(\phi_t)$
    end if
  end for
end for

dependency relations and contribution relations. A detailed description of this approach is discussed in Chapter 3.

Chapter 3

Methodology

The intuition of using social network metrics as predictors of defect-prone components is that central software components (either central in the software structure or central in the development process) are more likely to have many defects [40, 64, 4]. A central software component can be interpreted as a component that coordinates many other components or is modified by many developers. These previous studies do not look into the reason why central software components are more defect-prone, but we conjecture that central components in the software system are more exposed to other entities, and hence are more likely to be found to contain defects.

We observe that neither the *software dependency network* [64] nor the *socio-technical network* [4] considers the strength of dependency relations. Specifically, although the *socio-technical network* considers the strength of dependency relations to some extent by weighting the developer-component edges by the number of commits, it does not consider the strength of the inter-component dependency relations. However, dependency strength reveals the degree of coupling between software entities and the degree of coupling signals design quality [25]. Therefore, we argue that relations of different level of strength have different contributions to defects and hence should be analyzed separately.

In this work, we separate tight coupling relations away from loose coupling relations and analyze them separately. Major phases in our approach are:

1. Apply the *citation influence model* to the original *socio-technical network* to quantify the strength of relations.

2. Classify the relations into two classes: weak and strong.

3. Build two *equal level socio-technical networks* for weak relations and strong relations respectively.

4. Calculate social network metrics on the two networks and use the social network metrics as predictors of defect-proneness.

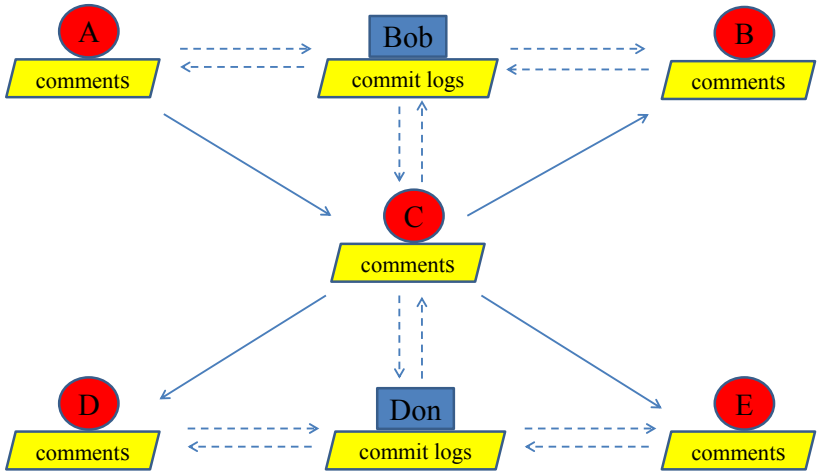


Figure 3.1: Document network

3.1 Phase I: Quantify Strength of Relations

The first step is to quantify the strength of relations in the *socio-technical network*. There are various ways to evaluate the strength of relations like class inheritance [43] and parameter passing [17]. We note that all these measures can only evaluate the strength of relations between software components. Therefore, we use the *citation influence model* [18] to determine the strength of both the inter-component dependency relations and the developer-component contribution relations in the *socio-technical network*.

To use the *citation influence model*, we transform the *socio-technical network* into a document network. Since the *socio-technical network* is composed of software components, developers and edges that represent inter-component dependency relations or contribution relations, we simply replace each software component and developer by an associated document as depicted in Figure 3.1:

- Software component: we extract and aggregate the comments of all the source files in a software component to be a document.
- Developer: we retrieve and aggregate the text in commit logs of a developer to be a document.

After the transformation, we apply the *citation influence model* to the document network to quantify the citation influence of a citation on the citing document and use the strength of citation influence as the strength of relations. The intuition is that text in source files and commit logs contain latent information about the code structure and development process, and thus citation influence reflects the strength of dependency relations and contribution relations.

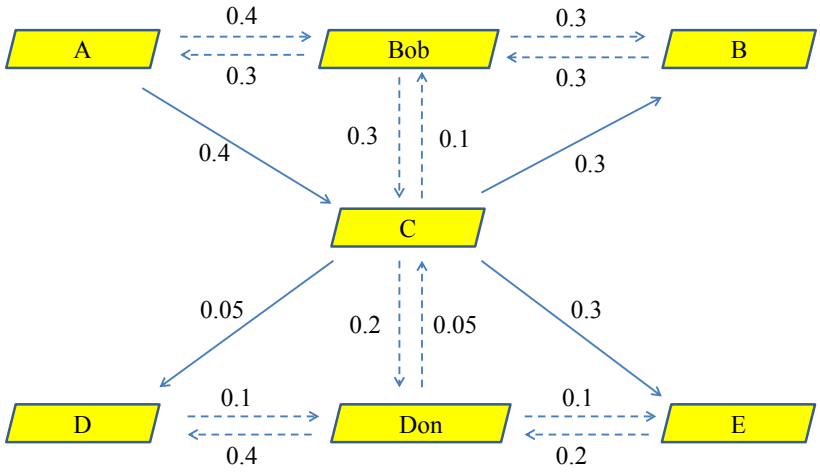


Figure 3.2: Citation influence model

As depicted in Figure 3.2, we transform developer Bob, Don, and component A, B, C, E, and E into documents, and the directed edges that connect these entities in the *socio-technical network* become directed edges that connect the corresponding documents in the document network. Document Bob cites document A, B, and C, and document Don cites document C, D, and E. The *citation influence model* may work out the influence of document C, D, and E on document Don by calculating, for example, that 5%, 40% and 10% of words in document Don are selected from topics that are 'borrowed' from document C, D, and E respectively. The influence score of document C, D, and E on Don is hence 0.05, 0.4 and 0.1. Since 45% of the words in document Bob are selected from its own topic mixture, the innovation (self-influence) score of document Don is 0.45. The influence score of document C, D, and E on document Don and the innovation score of Don sum to 1.0. Accordingly, based on the calculated citation influence in the document network, the strengths of the developer-component contribution relations of Don-C, Don-D, and Don-E in the *socio-technical network* are 0.05, 0.4, and 0.1, respectively.

Note that the *socio-technical network* combines the *software dependency network* and the *developer contribution network* to capture the centrality of software components in both the software architecture and the context of the development process [4]. Correspondingly, directed edges that are between the same two nodes have different weights and the strength of different types of edges conveys different information:

- *component-component*: the strength indicates how tightly a software component is coupled with another in the perspective of functionality.

- *component-developer*: the strength can be interpreted as how deeply a developer focuses on each software component. As is shown in Figure 3.1:
  - *component to developer*: the strength of edges C-to-Bob and C-to-Don reveals the level of contributions component C has received from developer Bob and Don and hence can be used to tell who is the major developer.
  - *developer to component*: the strength of edges Bob-to-A, Bob-to-B and Bob-to-C shows how dedicated developer Bob is to component A, B and C respectively.

3.2 Phase II: Classify Relations into Weak and Strong

After quantifying the strength of relations, we classify the strength of relations into two classes: weak and strong.

Since different levels of strengths of edges convey different information, we calculate a median of inter-component dependency relation strength per release (referred to as *inter-component median*) and a median of developer-component contribution relation strength per release (referred to as *developer-component median*). Moreover, as the developer-component contribution relations in the *socio-technical network* are weighted by the number of commits, we also calculate a median of the number of commits per release (referred to as the *commit median*).

For an inter-component dependency relation, as shown in Figure. 3.3, we simply classify it as weak if the citation influence value is below the *inter-component median*, or strong if the citation influence value is above the *inter-component median*. We use the *inter-component median* as the threshold because we want the number of classified weak relations and strong relations to be balanced. Since the classified relations are further analyzed in the experiments (Chapter 4) to compare their correlations with the number of defects, a balanced number of weak relations and strong relations would make the comparison less biased.

As for a developer-component contribution relation, there are various ways to classify the developer-component contribution relations, like using commit size as the threshold, however, we use a combination of the citation influence and the number of commits as the threshold in the classification. The reason of using the combination of the citation influence and the number of commits is that this combination considers both the centrality within the software system and the centrality in the development process. Moreover, commit size is incomparable when commits are made to components that are written in different languages (*e.g.* a commit of 3 LOC to a Python module against a commit of 10 LOC to a C module). As depicted in Figure. 3.4, we classify it as weak if the citation influence value is below the *developer-component median* and the associated number of commits is below the *commit median*, or strong if the citation influence value is above the *developer-component median* and the associated number of commits is above the *commit median*. We discard edges below (over)

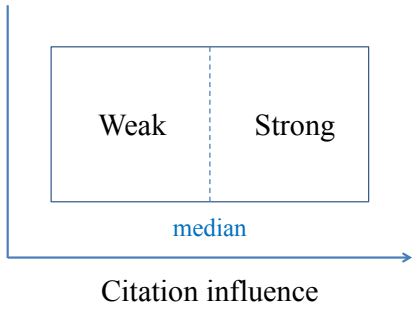


Figure 3.3: Inter-component dependency relations classification

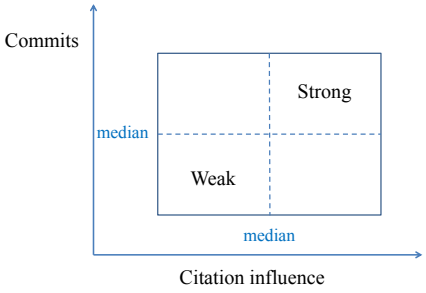


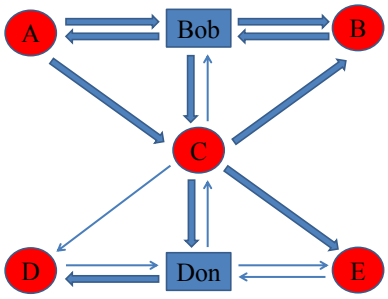
Figure 3.4: Developer-component contribution relations classification

the *developer-component median* but over (below) the *commit* median since logically these edges do not belong to any group.

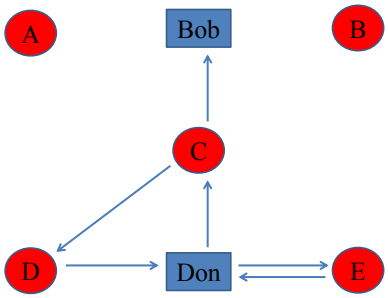
Figure 3.5a shows an example classification. Software component C is tightly coupled with A, B, and E but is only loosely coupled with D. Developer Don is more dedicated to component D than to C and E, whereas developer Bob contributes much to component A, B, and C.

3.3 Phase III: Construct *Equal-level Networks*

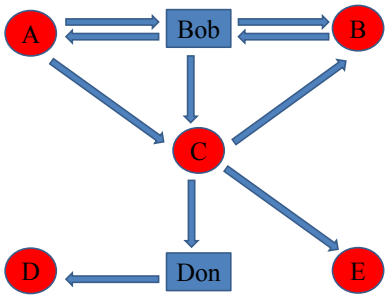
After the classification, we construct two *socio-technical networks* for different classes of relation strength (referred to as *equal-level networks*): one for strong relations and the other for weak relations. From Figure 3.5, the network in Figure 3.5b contains only weak relations (referred to as the *weak network*) and the network in Figure 3.5c (referred to as the *strong network*) contains only the strong relations. Both the *weak network* and the *strong network* are termed *equal-level networks* since they contain only relations with the same class of strength.



(a) Mixed strength of relations



(b) Weak relations only



(c) Strong relations only

Figure 3.5: Example of weak relations and strong relations. Thick edges denote weak relations and bold edges denote strong relations.

3.4 Phase IV: Conduct Social Network Analysis on *Equal-level networks*

Zimmermann *et al.* [64] recommends a set of social network metrics for building a prediction model of defect-proneness. These metrics are used in [64, 4, 39] and are shown to be effective. In our experiment (Chapter 4), we calculate these metrics on a per component basis and use them as the predictor variables in the prediction model. In this section, we briefly introduce these metrics.

All these social network metrics measure the centrality of nodes in the network. The metrics can be classified into two categories: global metrics and ego metrics. Global metrics are concerned about the role of a component within the entire network. Ego metrics measure the degree of domination of a component over its neighbours (dependencies, dependent components, and contributing developers).

Global Metrics

Global metrics measure the role a node plays within the entire network. In the context of a software system, global metrics evaluate the centrality of a software component within the system. A software component that is central to other components may be middleware or an adapter between software layers.

Degree centrality [21] is simply the sum of outgoing edges and incoming edges. In the context of the *socio-technical network*, high degree value indicates that a software component has many dependencies, dependent components, or contributing developers.

Betweenness centrality [20] measures how many times a node falls on the shortest path between others. If a software component is associated with a high degree of betweenness, this component has a potential for acting as a middleware or interface between two disconnected subsystems or has contribution from many developers.

Closeness centrality [47] evaluates the shortest distance from a node to all the other nodes in the network, with a high score of closeness meaning this node is close to other nodes. In the *socio-technical network*, if a software component is associated with a high closeness score, it is tightly coupled with other entities (either software components or developers).

Information centrality [51] is similar to *closeness centrality* in that it also evaluates the distance from a node to all the other nodes. The difference is that *information centrality* uses the harmonic mean length of the path as distance.

Distance weighted reach [59] measures the number of other nodes that can be reached from a node. It favours the case when many other nodes are reached within short steps.

Eigenvector centrality [45] suggests that the connection to a powerful neighbour (high values) increases a node's value more than a connection to a low valued neighbour. Each node's centrality value is the sum of the centrality values of that node's neighbours.

Bonacich's power [8] is a variant of degree centrality. A node is considered as central if it has connections to many other nodes or it has many neighbours that have connections to many other nodes.

Structure holes [11] measure the lack of connections between two parties. A *hole* on the path between two parties forces them to rely on a third party node for communication and that gives the third party node (a broker) an advantage over them.

- **Effective size** is the number of edges that are connected to a node minus the average number of edges that are connected to a node in the network.
- **Efficiency** is the effective size normalized by the size of the network.
- **Constraint** measures the disadvantage of a node that has limited options to reach other nodes.
- **Hierarchy** evaluates the concentration of the constraint value. When most of the constraint value is imposed by a single neighbour, the hierarchy value is high.

Ego Metrics

Ego metrics [59] are concerned about the role of a node (called the ego) within the network that is constructed by this node and its neighbours (called the ego network). In the context of the *socio-technical network*, ego metrics measure the role a software component with its dependencies, dependent components, and contributing developers.

In a directed network, an ego network can be constructed in different ways:

- *In-neighbourhood:* contains nodes that have an outgoing edge toward the ego.
- *Out-neighbourhood:* contains nodes that have an incoming edge from the ego.
- *InOut-neighbourhood:* a combination of the above sub-networks.

In the experiment, we calculate the following ego metrics on all three ego networks:
Size: number of nodes in the ego network.
Pairs: number of possible directed edges, $Size \times (Size - 1)$.
Ties: the number of directed edges in the ego network.
Density: ratio of possible ties that are present, $Ties/Pairs$.
Two step reach: the portion of nodes that can be reached within two steps.
Reach Efficiency: two step reach divided by size, $reach/size$.
Brokerage: number of pairs of nodes brokered by the ego.
Normalized brokerage: brokerage divided by pairs, $brokerage/pairs$.
Betweenness: betweenness centrality of the ego in the ego network.
Normalized betweenness: ego betweenness divided by pairs, $betweenness/pairs$.

Above are 11 global metrics and 10 ego metrics for each kind of ego network (*In-neighbourhood* ego network, *Out-neighbourhood* ego network and *InOut-neighbourhood* ego network). There are in total 41 social network metrics. In chapter 4, we calculate these 41 social network metrics for each software component on both the *weak network* and the *strong network* and use them as predictive variables in the prediction model.

Some of the above metrics can only be calculated on an unweighted network (*e.g. betweenness*) while others can be calculated on both an unweighted network and a weighted network (*e.g. degree centrality*). To deal with social network metrics that require an unweighted network, we simply remove the weights in each of the constructed weak and strong networks. For metrics that can be calculated on both unweighted and weighted networks, we use both these unweighted networks and the original weighted ones.

Chapter 4

Experiment and Discussions

In this chapter, we describe our data collection effort, present the preliminary correlation result, demonstrate how we predict defect-prone components, and discuss the prediction results. Briefly, we collect data from the Eclipse project, the Netbeans project, and the Gnome project, calculate social network metrics, train regression models on these metrics, and predict whether a software component is defect-prone.

4.1 Data Collection

We collect development data and post-release defect data from seven major releases of the Eclipse IDE (2.0 - 3.4), six major releases of the Netbeans IDE (6.1 - 7.0), and six major releases of the Gnome desktop suite (2.16 - 2.26). We use post-release defects as the measure of defect-proneness because they are often more problematic, as post-release defects can directly affect end-users and are often relatively more expensive to fix [7].

4.1.1 Project Similarities and Differences

We gather data from multiple releases of three open-source projects: the Eclipse IDE, the Netbeans IDE, and the Gnome desktop environment. All the three projects share a similarity of network structure and development process but differ in domains and languages.

Structural Similarities

All these projects can be decomposed into a network of components. The Eclipse IDE and the Netbeans IDE can be decomposed into a system of Java JAR files or a system of Java packages, whereas the Gnome desktop environment can be decomposed into a network of sub-projects.

Process Similarities

Eclipse, Netbeans, and Gnome are open-source projects with a long development history and stable release policy. All the three projects accept code contributions from volunteer developers and

hence communication between developers is made via electronic media like mailing lists and issue trackers.

Domain Differences

Both the Eclipse IDE and the Netbeans IDE are software development environments that have a set of development related functionality ranging from code management to compilation. Gnome is a desktop environment that provides a graphical desktop interface and a set of applications like a web browser and an XML library.

Language Differences

While the Eclipse IDE and the Netbeans IDE are written mostly in Java, Gnome is written in multiple languages like C++, Python, Vala, *etc.*

4.1.2 Component Granularity

To decompose Eclipse into software components, we use a Java package as the level of component granularity. We adopt this level of granularity because the number of defects associated with Java classes is too low to make the defect data useful for prediction. We use only the Java packages that exist in the Eclipse CVS repository and are shipped in the Eclipse IDE build for Linux x86/GTK 2. The number of packages per release we use ranges from 300 in Eclipse-2.0 to 900 in Eclipse-3.4.

As for Netbeans, however, due to the large number of Java packages, the number of defects associated with most packages is too low to make the defect data useful. Therefore, we use module JAR files (where most modules contains multiple JAR files, and most JAR files contain multiple Java packages) as the level of component granularity for Netbeans. We only use the module JAR files that are shipped in the Netbeans Java SE build. The number of module JAR files per release we use ranges from 500 in Netbeans-6.1 to 570 in Netbeans-7.0.

Since Gnome is written in multiple languages, decomposing it by language-specific units (*e.g.* Java packages) would not work. For simplicity, we only consider the Gnome sub-projects that are shipped with the Ubuntu releases. In an Ubuntu release, Gnome are decomposed and packed as multiple Ubuntu packages (referred to as Ubuntu Gnome packages). We take each Ubuntu gnome package as a single component. The number of Ubuntu Gnome packages per Ubuntu release we use ranges from 230 in Gnome-2.16 to 250 in Gnome-2.26.

4.1.3 Development and Defect Data

For each project, we mine the code repository to obtain the development history and mine the bug database to obtain the post-release defect data.

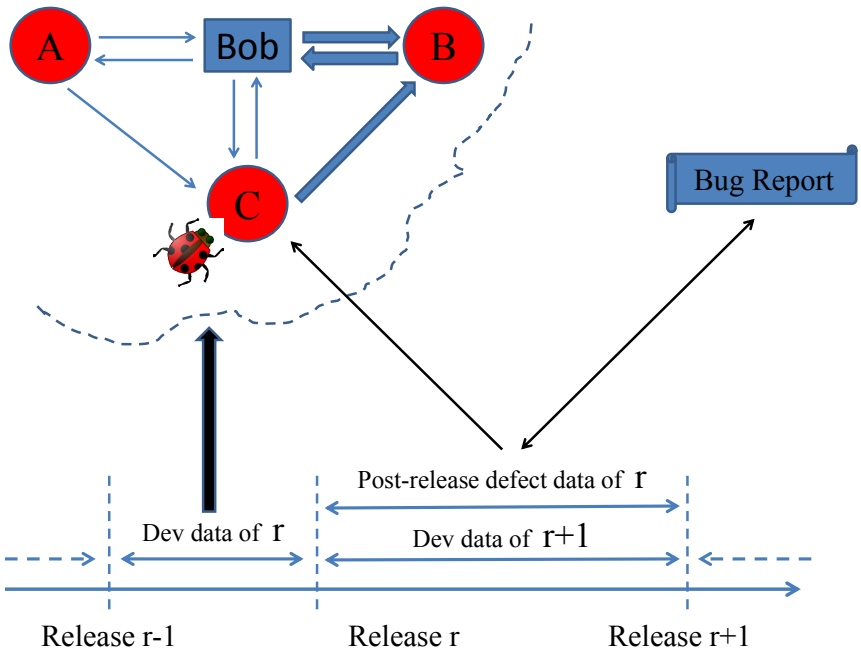


Figure 4.1: Mine development history and defect data

Figure 4.1 depicts the process of mining the development history and the defect data. For post-release defect data, we investigate the reported defects of release r that are opened after the release date of r and before the release date of $r + 1$.

Mine Development Data

We first retrieve the commit histories and source files from the public Eclipse Git repository, the Netbeans Mercurial repository, and the Gnome Git repository, and then extract development data from the commit histories.

As shown in Figure 4.2, to extract the commit data, we clone the mentioned code repositories to a local directory, convert the Netbeans mercurial repository into a Git repository mirror by using the **hg-fast-export** tool [1], and write a tool to parse commits from local Git repositories. For each commit in the code repository, we allocate it to a corresponding release and software component according to the commit date and commit path. In this way, we extract the pre-release commit logs from the code repositories.

Mine Defect Data

We identify defects by examining the commit logs to match commits with records from the bug report database, and link matched commits to corresponding source files [50].

[1] http://repo.or.cz/w/fast-export.git/tree

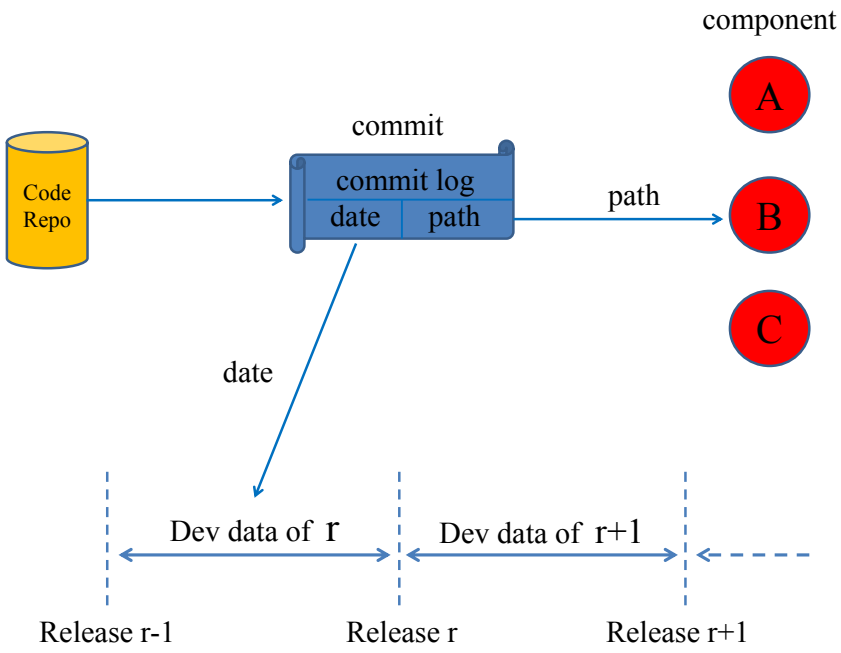


Figure 4.2: Mine development history

In order to identify post-release defects and assign defects to software components, we first mirror both the code repositories and the bug report databases. As Eclipse, Netbeans, and Gnome use bugzilla [48] as the bug tracker, we utilize its *Exporting to CSV* function to obtain all the defect data.

With the mirrored code repository and bug report database, as depicted in Figure 4.3, we use a similar matching heuristic as used in [50] to identify defect-fixing commits and extract the appropriate bug report identifiers by parsing the commit logs. The extracted bug report identifiers are subsequently used to search the associated bug reports in the bug database. After finding a defect-fixing commit and bug report pair, we first check whether this defect is reported after the date of the reported release and before the date of the next release by comparing the release date and the reported date written in the bug report. If this defect is determined to be dated post-release, we then use the commit path written in the commit log to assign this defect to concerned software components.

Note that our defect data mining approach collects the defect data only from the bugs that have fixing commits associated with them. We observe that there are bug reports left open in the bug tracker system, and no commit is made to fix those bugs. However, it is difficult to link those ignored bugs to the corresponding component, hence we only focus on those bugs that have fix commit associated with them. Moreover, we argue that those defects without associated fixing commits may be ignored by the developers on purpose, *e.g.*, they are just minor defects.

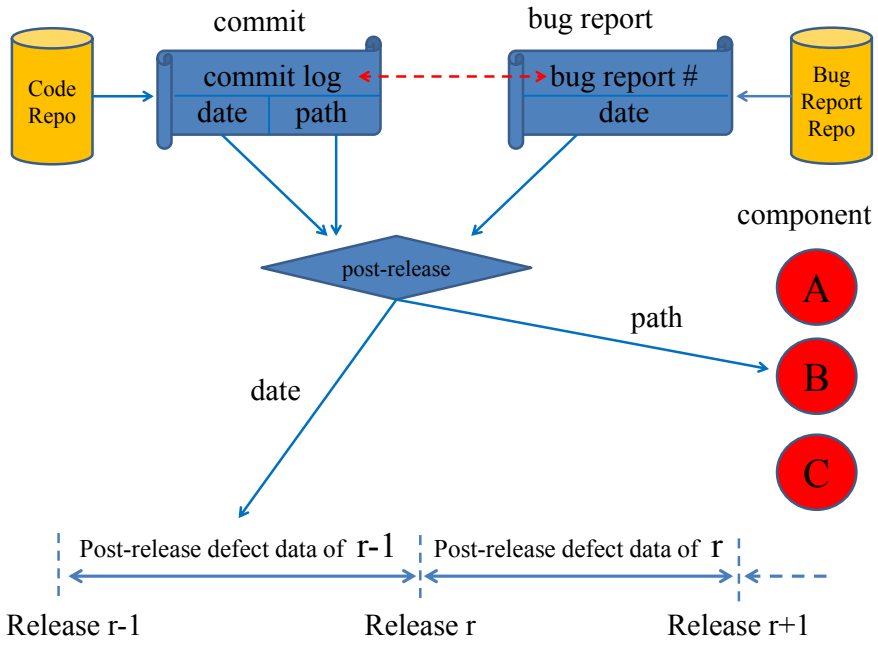


Figure 4.3: Mine defect data

4.1.4 Social Network Metrics

To construct the *equal-level network*, we need to identify software inter-component dependency relations and developer-component contribution relations.

As depicted in Figure 4.1, we retrieve the pre-release developer commit data to establish the developer-component contribution relations of Eclipse, Netbeans, and Gnome.

To identify the inter-component dependency relations of Eclipse and Netbeans, we obtain the Eclipse and Netbeans binaries from the public download site, and use the Java-bytecode analysis tool **DependencyFinder** [2] to extract the program dependencies. The dependency relations are determined at the class level. Consequently, we lift the dependency relations to the package level for Eclipse and to the JAR file level for Netbeans. As for the inter-component dependency relations of Gnome, since we only consider the sub-projects that are shipped with Ubuntu, we utilize the Ubuntu package management command **apt-rdepends** to find dependency relations between Gnome packages.

We use the above mentioned techniques (Section 4.1.3) to obtain the development history and the defect data from the Eclipse project, the Netbeans project, and the Gnome project, and subsequently construct the *socio-technical network*, the *weak network*, and the *strong network* for each project. These extracted data are used in the experiment to evaluate our methodology and to validate the hypotheses. Table 4.1 gives a detailed description of the extracted defect data and the constructed networks.

[2] http://depfind.sourceforge.net/

Table 4.1: The extracted post-release defect data and the constructed *socio-technical network*, *weak network* and *strong network* for Eclipse, Netbeans and Gnome. **D-C relations** denote the developer-component contribution relations, **C-C relations** denote the inter-component dependency relations.

(a) Eclipse at the level of Java packages.

Release	Network	# of components	# of developers	# of post-release defects	# of D-C relations	# of C-C relations
2.0	Socio-technical	362	57	1238	2212	4586
	Weak				1332	2198
	Strong				880	2388
2.1	Socio-technical	418	50	940	3150	5576
	Weak				1840	2572
	Strong				1310	3004
3.0	Socio-technical	636	66	2261	4088	8555
	Weak				2511	3731
	Strong				1577	4824
3.1	Socio-technical	725	66	1208	4490	10317
	Weak				2718	4681
	Strong				1772	5636
3.2	Socio-technical	906	68	1175	4500	13608
	Weak				2748	6458
	Strong				1752	7150
3.3	Socio-technical	1169	55	729	3248	17685
	Weak				1658	8773
	Strong				1590	8912
3.4	Socio-technical	1260	56	508	3118	18981
	Weak				1767	9295
	Strong				1351	9686

(b) Netbeans at the level of Java JAR files.

Release	Network	# of components	# of developers	# of post-release defects	# of D-C relations	# of C-C relations
6.1	Socio-technical	481	128	457	3052	2716
	Weak				1937	935
	Strong				1115	1781
6.5	Socio-technical	527	144	326	4154	3155
	Weak				2465	1252
	Strong				1689	1903
6.7	Socio-technical	500	130	188	3610	3049
	Weak				2293	1072
	Strong				1317	1977
6.8	Socio-technical	524	89	203	3264	3241
	Weak				1948	1256
	Strong				1316	1985
6.9	Socio-technical	568	76	431	2992	3561
	Weak				2022	1508
	Strong				970	2053
7.0	Socio-technical	540	73	318	2128	3482
	Weak				1429	1387
	Strong				699	2095

(c) Gnome at the level of Ubuntu packages.

Release	Network	# of components	# of developers	# of post-release defects	# of D-C relations	# of C-C relations
2.16	Socio-technical	164	314	14151	6516	152
	Weak				3289	37
	Strong				3227	115
2.18	Socio-technical	170	333	8987	5682	158
	Weak				2865	41
	Strong				2817	117
2.20	Socio-technical	175	353	3846	6054	160
	Weak				3137	46
	Strong				2917	114
2.22	Socio-technical	186	381	4819	6636	160
	Weak				3332	37
	Strong				3304	123
2.24	Socio-technical	189	375	1316	6816	155
	Weak				3490	41
	Strong				3326	114
2.26	Socio-technical	186	359	1482	7238	150
	Weak				3716	43
	Strong				3522	107

We use the **Jung library** [3] to calculate the aforementioned social-network metrics (Section 3.4). These metrics are calculated on a component basis on both the original *socio-technical network* and the *equal-level networks*. Specifically, for each component, we calculate three sets of social metric values for it within the *socio-technical network*, the *weak network*, and the *strong network*.

4.2 Correlation with the Number of Post-Release Defects

In this section, we present the preliminary correlation results of Eclipse, Netbeans, and Gnome, and use the correlation results to validate **TS1**. Briefly, for each release, we calculate the above mentioned social network metrics (section 4.1.4) on the *socio-technical network*, the *weak network*, and the *strong network* on a component basis, and correlate the social network metrics that are calculated from the different networks with the number of post-release defects. We use Java package, Java JAR file, and Ubuntu package as the level of component granularity for Eclipse, Netbeans, and Gnome, respectively. Figure 4.4 depicts the correlation process.

Since the data is not normally distributed, we use Spearman's correlation. The correlation value is between -1 and 1. The closer this value is to -1 or 1, the stronger the relationship between the metric value and the number of defects. We only present a sample of the correlation results of Eclipse 3.4, Netbeans 7.0, and Gnome 2.26 in Table 4.2. The complete correlation results of the investigated Eclipse releases (2.0 - 3.4), Netbeans releases (6.1 - 7.0), and Gnome releases (2.16 - 2.26) are presented in Table A.1, Table A.2 and Table A.3. Correlation significant at 99% and 95% with Wilcoxon signed-rank tests are marked by (**) and (*) respectively.

[3] http://jung.sourceforge.net/index.html

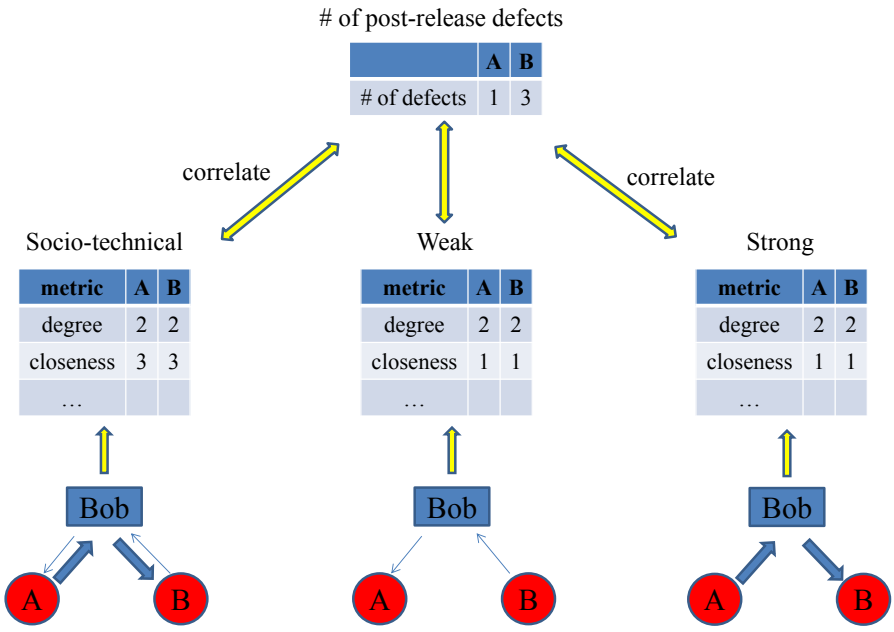


Figure 4.4: Correlation with the number of post-release defects

Note that we omit the correlation results of the **Ego betweenness** metric, the **Ego normalized betweenness** metric, the **Ego brokerage** metric, and the **Ego normalized brokerage** for the *in-neighbour ego network* and the *out-neighbour ego network* (Section 4.1.4). The reason is that, by definition, these four metrics always generate zero values on the *in-neighbour ego network* and the *out-neighbour ego network*, and hence are not meaningful for correlation purposes.

Based on the correlation results, we make the following observations:

- Most social network metrics calculated on the *weak network* have a higher correlation score than that calculated on the *strong network*.

For instance, as is shown in Table 4.2a, most of the social network metrics calculated on the *weak network* have higher correlation scores than that calculated on the *strong network*, with only three exceptions: *Degree centrality*, *Hierarchy*, and *Ego density*. This finding holds across all investigated Eclipse releases (2.0 - 3.4), Netbeans releases (6.1 - 7.0) and Gnome releases (2.16 - 2.26). It suggests that, more often, the weak relations have a stronger relationship with the number of defects. This finding supports **TS1** that weak relations and strong relations have different correlations with the number of defects.

As mentioned, tight coupling reveals bad design in practice [53] and is expected to contribute more to the defects. However, our finding suggests that, in terms of social network metrics, loose coupling (weak relations) have a stronger relationship with the number of defects. A possible explanation is that most of the defects are caused by developers inexperienced in subtle dependencies.

Table 4.2: Correlation of social network metrics with the number of post-release defects. Weak and Strong denote the *weak network* and the *strong network* respectively. Bold values indicate that the correlation on that network is higher than that on other networks. Correlation significant at 99% and 95% with Wilcoxon signed-rank tests [60] are marked by (**) and (*) respectively.

(a) Eclipse 3.4 at Java package level

Global Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Degree centrality	**0.508(**)**	0.482(**)	0.473(**)
Betweenness centrality	0.365(**)	0.267(**)	**0.439(**)**
Closeness centrality	0.188(**)	-0.098(**)	**0.368(**)**
Information centrality	0.398(**)	0.174(**)	**0.428(**)**
Distance weighted reach	0.218(**)	-0.094(**)	**0.378(**)**
Eigenvector centrality	0.322(**)	0.312(**)	**0.452(**)**
Effective size	**0.417(**)**	0.255(**)	0.414(**)
Efficiency	**0.249(**)**	0.134(**)	0.144(**)
Constriant	0.218(**)	0.220(**)	**0.349(**)**
Hierarchy	0.302(**)	**0.331(**)**	0.160(**)

InOut-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.365(**)	0.207(**)	**0.382(**)**
Ego ties	**0.369(**)**	0.197(**)	0.369(**)
Ego pairs	0.365(**)	0.207(**)	**0.382(**)**
Ego density	**-0.286(**)**	-0.164(**)	-0.193(**)
Ego reach	0.228(**)	0.064	**0.388(**)**
Ego reach efficiency	-0.183(**)	-0.123(**)	**0.244(**)**
Ego brokerage	0.420(**)	0.279(**)	**0.448(**)**
Ego normalized brokerage	0.278(**)	0.239(**)	**0.414(**)**
Ego betweeness	0.401(**)	0.292(**)	**0.440(**)**
Ego Normalized betweeness	0.387(**)	0.316(**)	**0.449(**)**

In-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.328(**)	0.289(**)	**0.344(**)**
Ego ties	**0.323(**)**	0.273(**)	0.321(**)
Ego pairs	0.328(**)	0.289(**)	**0.344(**)**
Ego density	**-0.127(**)**	-0.069(*)	-0.077(*)
Ego reach	0.228(**)	0.064	**0.388(**)**
Ego reach efficiency	-0.117(**)	-0.158(**)	**0.261(**)**

Out-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.353(**)	0.082(*)	**0.420(**)**
Ego ties	0.349(**)	0.063	**0.418(**)**
Ego pairs	0.353(**)	0.082(*)	**0.420(**)**
Ego density	-0.207(**)	-0.111(**)	**0.308(**)**
Ego reach	0.228(**)	0.064	**0.388(**)**
Ego reach efficiency	-0.185(**)	-0.009	**0.318(**)**

(b) Netbeans 7.0 at Java JAR file level

Global Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Degree centrality	**0.507(**)**	0.502(**)	0.492(**)
Betweenness centrality	0.366(**)	0.360(**)	**0.438(**)**
Closeness centrality	0.259(**)	0.098(*)	**0.368(**)**
Information centrality	0.432(**)	0.365(**)	**0.461(**)**
Distance weighted reach	0.286(**)	0.136(**)	**0.376(**)**
Eigenvector centrality	0.441(**)	0.439(**)	**0.455(**)**
Effective size	0.414(**)	0.315(**)	**0.427(**)**
Efficiency	0.045	**0.064**	-0.017
Constriant	0.253(**)	0.325(**)	**0.376(**)**
Hierarchy	0.247(**)	**0.285(**)**	0.147(**)

InOut-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.396(**)	0.261(**)	**0.415(**)**
Ego ties	0.421(**)	0.267(**)	**0.425(**)**
Ego pairs	0.396(**)	0.261(**)	**0.415(**)**
Ego density	**-0.238(**)**	-0.185(**)	-0.008
Ego reach	0.301(**)	0.221(**)	**0.402(**)**
Ego reach efficiency	-0.010	-0.088(*)	**0.276(**)**
Ego brokerage	0.433(**)	0.349(**)	**0.465(**)**
Ego normalized brokerage	0.250(**)	0.287(**)	**0.358(**)**
Ego betweeness	0.412(**)	0.364(**)	**0.457(**)**
Ego Normalized betweeness	0.393(**)	0.377(**)	**0.455(**)**

In-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.321(**)	0.285(**)	**0.365(**)**
Ego ties	0.315(**)	0.283(**)	**0.358(**)**
Ego pairs	0.321(**)	0.285(**)	**0.365(**)**
Ego density	-0.030	**0.105(*)**	0.045
Ego reach	0.301(**)	0.221(**)	**0.402(**)**
Ego reach efficiency	-0.019	-0.103(*)	**0.291(**)**

Out-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.396(**)	0.202(**)	**0.442(**)**
Ego ties	0.386(**)	0.181(**)	**0.442(**)**
Ego pairs	0.396(**)	0.202(**)	**0.442(**)**
Ego density	-0.239(**)	-0.150(**)	**0.247(**)**
Ego reach	0.301(**)	0.221(**)	**0.402(**)**
Ego reach efficiency	0.051	0.068	**0.331(**)**

(c) Gnome 2.26 at Ubuntu package level

Global Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Degree centrality	0.616(**)	0.583(**)	**0.619(**)**
Betweenness centrality	0.489(**)	0.446(**)	**0.620(**)**
Closeness centrality	0.379(**)	-0.163(*)	**0.622(**)**
Information centrality	0.588(**)	0.270(**)	**0.662(**)**
Distance weighted reach	0.400(**)	-0.161(*)	**0.626(**)**
Eigenvector centrality	0.591(**)	0.534(**)	**0.663(**)**
Effective size	0.616(**)	0.582(**)	**0.619(**)**
Efficiency	**-0.267(**)**	-0.198(**)	0.028
Constriant	-0.453(**)	-0.392(**)	**0.525(**)**
Hierarchy	**-0.575(**)**	-0.565(**)	0.293(**)

InOut-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	**0.623(**)**	0.615(**)	0.618(**)
Ego ties	0.608(**)	0.588(**)	**0.612(**)**
Ego pairs	**0.623(**)**	0.615(**)	0.618(**)
Ego density	-0.607(**)	**-0.618(**)**	-0.257(**)
Ego reach	0.561(**)	0.274(**)	**0.645(**)**
Ego reach efficiency	-0.260(**)	**-0.426(**)**	0.341(**)
Ego brokerage	0.618(**)	0.570(**)	**0.670(**)**
Ego normalized brokerage	0.026	0.100	**0.500(**)**
Ego betweeness	0.612(**)	0.567(**)	**0.667(**)**
Ego Normalized betweeness	0.592(**)	0.509(**)	**0.658(**)**

In-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.613(**)	**0.621(**)**	0.613(**)
Ego ties	0.609(**)	**0.624(**)**	0.612(**)
Ego pairs	0.613(**)	**0.621(**)**	0.613(**)
Ego density	**-0.412(**)**	-0.398(**)	-0.210(**)
Ego reach	0.561(**)	0.274(**)	**0.645(**)**
Ego reach efficiency	-0.402(**)	**-0.462(**)**	0.365(**)

Out-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.605(**)	0.346(**)	**0.652(**)**
Ego ties	0.590(**)	0.324(**)	**0.645(**)**
Ego pairs	0.605(**)	0.346(**)	**0.652(**)**
Ego density	**-0.550(**)**	-0.334(**)	0.356(**)
Ego reach	0.561(**)	0.274(**)	**0.645(**)**
Ego reach efficiency	-0.258(**)	-0.046	**0.385(**)**

Since loosely coupled software components are expected to share little similarity in terms of functionality, *e.g.* network and memory management, developers dedicated to one component may lack experience or knowledge to handle rarely referenced dependencies.

We conclude that *for the investigated Eclipse releases, Netbeans releases, and Gnome releases, there is no evidence to reject* ***TS1*** that strong relations and weak relations have different correlations with the number of defects.

- Most social network metrics calculated on the *weak network* have a higher correlation score than that calculated on the original *socio-technical networks*.

For investigated Eclipse releases (2.0 - 3.4), Netbeans releases (6.1 - 7.0), and Gnome releases (2.16 - 2.26), we observe that most of the social network metrics have higher correlation scores on the *weak networks* than on the original *socio-technical network*. As shown in Table 4.1c, most of the metrics on the *weak networks* are more correlated with the number of defects than that on the *socio-technical network*, with only a few exceptions like *Efficiency Hierarchy*, and *Ego Density*. This finding is persistent across all investigated Eclipse releases (2.0 - 3.4), Netbeans releases (6.1 - 7.0), and Gnome releases (2.16 - 2.26). It implies that, for Eclipse, Netbeans, and Gnome, the weak relations alone have a stronger relationship with the number of defects than mixed with the strong relations as in the *socio-technical network*.

In previous work [64, 39, 58, 4], social network metrics are used to measure the centrality of software components, and are used as predictors of software defects. These metrics are calculated on a network mixed with both weak relations and strong relations. However, our findings suggest that weak relations and strong relations have different correlations with the number of defects, and hence should be analyzed separately. Therefore, we conjecture that the separation of weak relations (loose coupling) from strong relations (tight coupling) will increase the predictive power of a defect prediction model.

4.3 Logistic Regression

As with previous work [64, 4], to predict whether a software component is defect-prone, we calculate the social network metrics and use the metric values as the predictor variables in logistic regressions. In this section, we present the regression results of Eclipse, Netbeans, and Gnome, and use the regression results to validate **TS2**.

4.3.1 Preprocess with PCA

Before training regression models, as proposed by [64, 4], we preprocess the metric values with standardization and principal component analysis (PCA) [29]. As suggested by [64, 4], some of these metrics are correlated with each other, *e.g. Ego size* calculated on the *in-neighbourhood network*

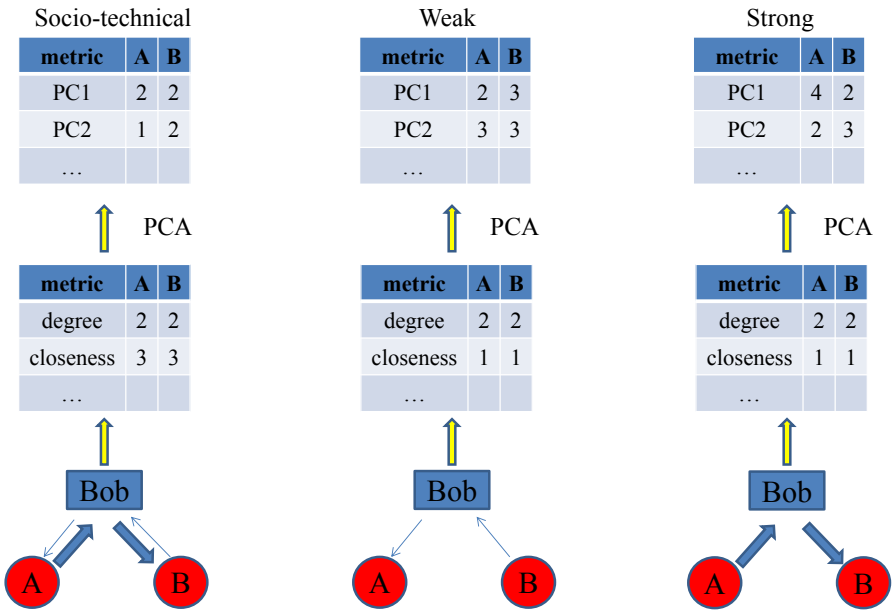


Figure 4.5: Compute principal components

and the *inout-neighbourhood network*. Since such correlation violates the assumption of independent predictor variables in regressions, we use PCA to address this *multicollinearity* issue. PCA transforms the set of metric values into a set of uncorrelated principal components (PC). A principal component is a linear combination of original metric values. Each principal component is independent of the others. We retain the minimum number of principal components that accounts for 95% of the variance in the original data and select the retained principal components as the predictor variables in regressions.

As depicted in Figure 4.5, we calculate the social network metrics on a component basis on the *socio-technical network*, the *weak network*, and the *strong network*, and put the calculated metric values together to form a metric value matrix. We subsequently apply the PCA to the metric value matrix to compute principal components for all the software component. These principal components are then used as predictors in the logistic regression model to predict whether a software component is defect-prone.

4.3.2 Evaluation Metrics

As with prior work [64, 4], we use four information retrieval metrics to evaluate predictions from the logistic regression model: precision, recall, F score, and Area under ROC curve (AUC) [42].

Let TF be true positives, FP be false positives and FN be false negatives.

$$Precision = \frac{TP}{TP + FP}$$

Precision measures type I error by calculating the proportion of the components that are classified as defect-prone are truly defect-prone components. The closer the *precision* is to 1, the less type I error this model makes in prediction.

$$Recall = \frac{TP}{TP + FN}$$

Recall measures type II error by calculating the proportion of the truly defect-prone components that are identified as defect-prone components. The closer the *recall* is to 1, the less type II error this model makes in prediction.

$$F\text{-}score = \frac{2 \times precision \times recall}{precision + recall}$$

F score considers both *precision* and *recall*, it can be calculated by computing the harmonic mean of *precision* and *recall*.

Area under ROC curve (AUC) is used to measure the classification ability of a prediction model. It is measured by calculating the area under the Receiver Operating characteristic (ROC) curve. ROC is used to illustrate the performance of a binary classifier system like logistic regression as its discrimination threshold is varied. This curve is created by plotting the value of true positive rate against false positive rate. The closer *AUC* value is to 1, the better the prediction is [10].

In addition to the information retrieval metrics, we use the *Nagelkerke coefficient of determination* [38], also known as Nagelkerke's pseudo R^2 , to measure how well the built model fits the training data. Similar to R^2, this metric evaluates how well the model explains the variance of the data but does not tell how good the prediction is.

4.3.3 Cross Validation

To verify whether the separation of weak relations and strong relations increases predictive power, as presented in Figure 4.6, we calculate metrics on the original *socio-technical network*, the *weak network*, and the *strong network*, apply PCA to the metrics to compute principal components, and use the principal components as predictors to build logistic regression models. We subsequently use K-fold cross validation [9] to measure the predictive power of these models.

On the Eclipse, Netbeans, and Gnome datasets, we take a software component as ***defect-prone*** if this component is ranked in the top X % by the most number of defects. We use 10, 15, 20, 25, 30 as thresholds for X. For the different thresholds for defect-proneness, we subsequently train four logistic models:

1. *Socio-technical model*: built with metrics calculated on the *socio-technical network*.
2. *Weak model*: built with data from the *weak network*.
3. *Strong model*: built with data from the *strong network*.

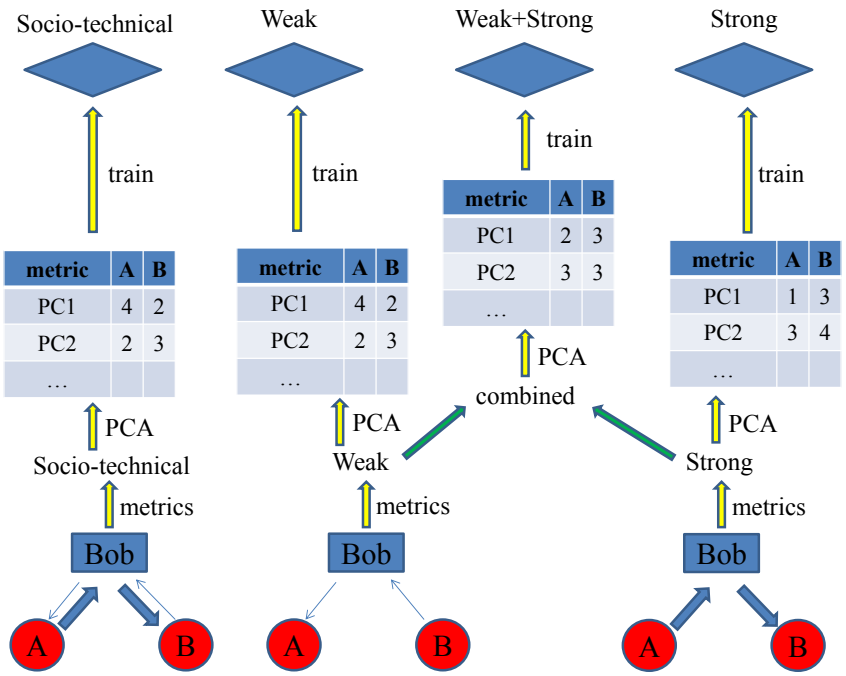


Figure 4.6: Train logistic regression models

4. *Weak+Strong model*: built with both data from the *weak network* and data from the *strong network*.

To clarify, for the *Weak+Strong model*, we use metric values from both the *weak network* and the *strong network* to form a combined metric value matrix, and apply PCA to the combined metric value matrix to compute principal components as predictors.

In total, there are $5 \times 4 = 20$ different prediction models per project release. We use repeated K-fold cross validation [9] to evaluate the predictive power of these models. In short, K-fold cross validation works by dividing the data into K disjoint groups with nearly equal size, training and testing the model K times, each time using K-1 groups for training and the remaining one group for testing. Accordingly, we compute the evaluation metrics (4.3.2) for these K testings and use average values of evaluation metrics to measure how accurate these models are. We refer readers to [9] for a detailed definition of K-fold cross validation.

Since researchers find from empirical experience and simulation studies that 10-fold cross validation is likely to achieve a good evaluation accuracy [16], we set $K = 10$ in our experiment. We repeat the 10-fold cross validation process 20 times and calculate the average values of evaluation metrics. A sample of the cross validation results of predicting top 30% defect-prone Eclipse Java packages, Netbeans Java JAR files, and Gnome Ubuntu packages is presented in Table 4.3. The complete cross validation results of the investigated Eclipse (2.0 - 3.4), Netbeans releases (6.1 - 7.0), and Gnome releases (2.16 - 2.26) are presented in Table B.1, Table B.2, and Table B.3.

Based on the cross validation results, we make the following observations:

Table 4.3: Results of repeated 10-fold cross validation of Eclipse, Netbeans and Gnome. Given data from current release, predict whether a software component is defect-prone after current release. Bold values indicate that they are higher than values calculated on other networks.

(a) Predict top **30%** defect-prone Eclipse Java packages

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
2.0	Socio-technical	0.714	0.526	0.592	0.855	0.440
	Weak	**0.782**	**0.546**	**0.629**	0.851	0.447
	Strong	0.780	0.489	0.587	0.836	0.354
	Weak+Strong	0.774	0.537	0.621	**0.873**	**0.504**
2.1	Socio-technical	0.751	**0.570**	**0.637**	**0.864**	**0.472**
	Weak	0.742	0.454	0.549	0.835	0.388
	Strong	0.747	0.487	0.579	0.823	0.409
	Weak+Strong	**0.754**	0.527	0.608	0.856	0.448
3.0	Socio-technical	**0.832**	0.598	0.689	0.882	0.508
	Weak	0.789	0.597	0.673	0.870	0.500
	Strong	0.798	0.573	0.660	0.877	0.516
	Weak+Strong	0.807	**0.656**	**0.717**	**0.901**	**0.586**
3.1	Socio-technical	0.792	0.654	0.711	0.898	0.554
	Weak	0.777	0.588	0.663	0.872	0.505
	Strong	0.785	0.579	0.660	0.867	0.489
	Weak+Strong	**0.811**	**0.666**	**0.726**	**0.905**	**0.579**
3.2	Socio-technical	0.777	**0.686**	**0.726**	0.844	0.452
	Weak	0.772	0.626	0.689	0.817	0.394
	Strong	0.754	0.609	0.670	0.813	0.374
	Weak+Strong	**0.791**	0.661	0.718	**0.845**	**0.463**
3.3	Socio-technical	0.727	0.571	0.637	0.813	0.377
	Weak	0.719	0.517	0.598	0.779	0.307
	Strong	0.730	0.590	0.650	0.804	0.347
	Weak+Strong	**0.745**	**0.620**	**0.674**	**0.820**	**0.391**
3.4	Socio-technical	0.738	0.429	0.537	0.802	0.348
	Weak	0.701	0.420	0.518	0.775	0.298
	Strong	0.723	0.446	0.547	0.792	0.321
	Weak+Strong	**0.738**	**0.512**	**0.600**	**0.816**	**0.382**

(b) Predict top **30%** defect-prone Netbeans Java JAR files

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
6.1	Socio-technical	0.719	**0.596**	**0.644**	0.827	0.388
	Weak	**0.735**	0.568	0.634	0.799	0.332
	Strong	0.646	0.575	0.600	0.807	0.358
	Weak+Strong	0.706	0.585	0.634	**0.846**	**0.432**
6.5	Socio-technical	0.666	0.411	0.498	0.842	0.386
	Weak	0.665	0.399	0.485	0.828	0.356
	Strong	0.606	0.397	0.468	0.837	0.385
	Weak+Strong	**0.683**	**0.474**	**0.548**	**0.857**	**0.421**
6.7	Socio-technical	**0.698**	0.347	0.448	0.845	0.369
	Weak	0.668	0.328	0.426	0.828	0.357
	Strong	0.688	0.379	0.471	0.840	0.365
	Weak+Strong	0.685	**0.406**	**0.494**	**0.848**	**0.385**
6.8	Socio-technical	0.674	0.396	0.485	0.865	0.416
	Weak	0.687	0.358	0.454	0.838	0.361
	Strong	0.617	0.312	0.399	0.842	0.360
	Weak+Strong	**0.714**	**0.461**	**0.546**	**0.880**	**0.459**
6.9	Socio-technical	0.769	0.436	0.548	0.852	0.433
	Weak	0.764	0.394	0.509	0.829	0.375
	Strong	0.765	0.425	0.536	0.830	0.386
	Weak+Strong	**0.779**	**0.481**	**0.586**	**0.857**	**0.445**
7.0	Socio-technical	0.686	0.396	0.488	0.861	0.418
	Weak	0.673	0.396	0.485	0.832	0.367
	Strong	**0.702**	0.353	0.454	0.851	0.385
	Weak+Strong	0.691	**0.426**	**0.513**	**0.883**	**0.461**

(c) Predict top **30%** defect-prone Gnome Ubuntu packages

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
2.16	Socio-technical	0.784	**0.698**	**0.713**	0.904	0.588
	Weak	**0.806**	0.650	0.696	0.889	0.570
	Strong	0.756	0.676	0.694	0.898	0.596
	Weak+Strong	0.796	0.650	0.691	**0.905**	**0.598**
2.18	Socio-technical	0.801	0.626	0.677	0.876	0.531
	Weak	0.778	0.697	0.716	0.880	0.535
	Strong	0.803	0.635	0.689	**0.904**	0.583
	Weak+Strong	**0.832**	**0.734**	**0.762**	0.903	**0.593**
2.20	Socio-technical	0.811	0.791	0.787	0.926	0.653
	Weak	0.807	0.689	0.726	0.866	0.557
	Strong	0.801	0.805	0.790	0.924	0.666
	Weak+Strong	**0.861**	**0.823**	**0.830**	**0.934**	**0.674**
2.22	Socio-technical	0.714	0.688	0.682	0.903	0.573
	Weak	**0.855**	0.700	0.747	0.877	0.564
	Strong	0.787	0.655	0.693	**0.907**	**0.586**
	Weak+Strong	0.829	**0.733**	**0.763**	0.877	0.581
2.24	Socio-technical	0.806	**0.691**	0.730	**0.901**	0.580
	Weak	**0.852**	0.658	0.728	0.885	0.556
	Strong	0.842	0.683	**0.736**	0.894	**0.606**
	Weak+Strong	0.824	0.655	0.713	0.897	0.596
2.26	Socio-technical	0.818	**0.777**	0.783	0.935	0.694
	Weak	0.842	0.729	0.768	0.908	0.653
	Strong	**0.876**	0.752	**0.794**	0.937	**0.705**
	Weak+Strong	0.823	0.759	0.775	**0.938**	0.703

- For the investigated Eclipse, Netbeans, and Gnome releases, the predictive power of the *Strong* model and the *Weak* model varies with releases.

For example, regarding Table 4.3a of Eclipse, F score on the *Weak* model is higher than that on the *Strong* model for Eclipse-3.0 while F score on the *Weak* model is lower than that on the *Strong* model for Eclipse-3.4. Across all the thresholds of defect-prone software components (top 5%, 10%, 15%, 20%, 25%, 30%), we observe the trend is that F score on the *Weak* model is persistently higher than that on the *Strong* model for Eclipse-2.1, Eclipse-3.0, Eclipse-3.1, Eclipse-3.2, Netbeans-6.8, and Netbeans-7.0, while F score on the *Weak* model is persistently lower than that on the *Strong* model for Eclipse-3.3, Eclipse-3.4, Netbeans-6.1, Netbeans-6.9, Gnome-2.16, and Gnome-2.26. Therefore, we can conclude that the prediction power of the *Strong* model and the *Weak* model changes across releases.

This observation contradicts the correlation results by Section 4.2 that most social network metrics calculated on the *weak network* is higher than that calculated on the *strong network*. A possible explanation is that many metrics with high correlation scores on the *weak network* are correlated with each other and hence are less useful when used as predictors of defects in regression analysis.

The above observation shows that, for different releases, weak relations and strong relations as predictors have different contributions to the predictive power of defects, which supports **TS2** that weak relations and strong relations have different contributions to the prediction power of defect prediction models. Therefore, based on the above observations, we conclude that there is no evidence to reject **TS2**.

- The predictions made by the *Weak+Strong* model are more accurate than predictions made by the *socio-technical* model.

Taking Table 4.2b as an example, on most Netbeans releases, F score and Recall of the *Weak+Strong* model is higher than that on the *socio-technical* model. This observation is persistent over all the Eclipse models and most of the Netbeans models, and Gnome models.

While the data we use to train the *socio-technical* model is computed on a *socio-technical network* mixed with weak relations and strong relations, the *Weak+Strong* model is trained by data that is computed from the *equal-level networks* which contains only weak relations or strong relations. Therefore, this observation supports **TS3** that the separation of weak relations and strong relations could improve the prediction power of defect models. Note that even if Table 4.3 shows that weak relations and strong relations play a different role in the prediction models of defects, it is not clear which kind of relation plays a significant role on a particular release, *e.g.*, the *weak* model for Netbeans-6.1 and *strong* model for Netbeans-6.8. Therefore, we combine the weak relations and strong relations together to build the *Weak+Strong* model and verify that the separation of weak relations from strong relations could increase the prediction power of defect models. For the

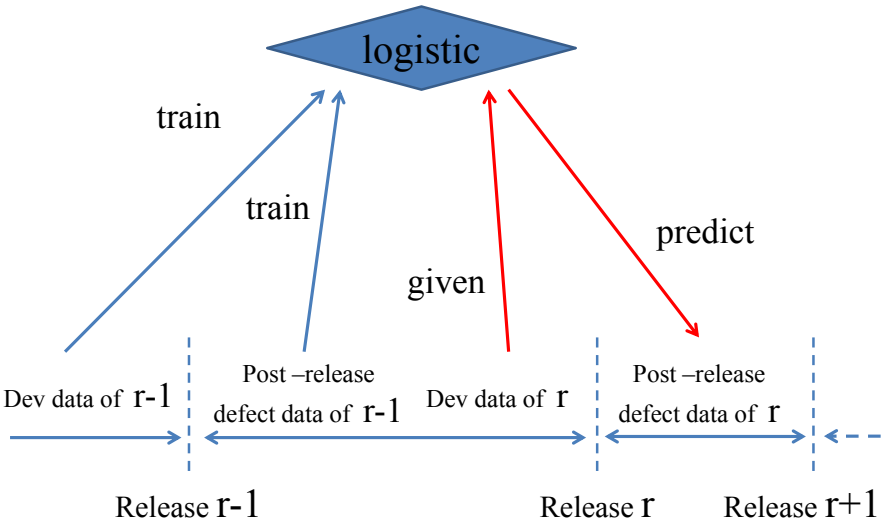


Figure 4.7: Prediction across releases

investigated Eclipse releases, Netbeans releases, and Gnome releases, we conclude that there is no evidence to reject **TS3**.

4.3.4 Prediction Across Releases

We also evaluate our approach in a more realistic scenario, namely using data from the previous release to predict the software components of the current release that are most likely to have post-release defects. As is shown in Figure 4.7, we first train a logistic regression model with development data and post-release defect data of release $r-1$, and then use the development data of release r to predict whether a software component is defect-prone after release r.

Note that social metric predictors are sensitive to the size of the network. As an example, since the *degree centrality* of a node is the sum of outgoing edges and incoming edges, as the size of the network grows over releases, the *degree centrality* value increases accordingly since the number of neighbours increases. Thus, the difference of network size makes metric values of the same component in two releases incomparable. To address the size issue, we normalize the metric values before the PCA preprocess (Section 4.3.1). Specifically, we normalize the metric values using the standard score [30]:

$$z_{mi} = \frac{v_{mi} - \mu_m}{\sigma_m}$$

For metric m, we center the metric value v_{mi} of component i by subtracting the mean of v_{mi} and dividing it by the standard deviation. The resulting distribution of metric values has zero mean and unit standard deviation.

Table 4.4: Results of using data from a previous release to predict whether a software component is defect-prone after release. Bold values indicate that they are higher than values calculated on other networks.

(a) Predict top 30% defect-prone Eclipse Java packages

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
2.1	Socio-technical	0.769	0.439	0.559	0.826	0.440
	Weak	0.758	0.480	0.587	0.829	0.388
	Strong	0.723	0.480	0.577	0.801	0.292
	Weak+Strong	**0.792**	**0.582**	**0.671**	**0.864**	**0.489**
3.0	Socio-technical	**0.767**	0.462	0.577	0.830	0.472
	Weak	0.750	0.500	0.600	0.851	0.425
	Strong	0.746	0.412	0.531	0.783	0.299
	Weak+Strong	0.719	**0.561**	**0.631**	**0.877**	**0.501**
3.1	Socio-technical	0.742	0.608	0.669	0.870	0.508
	Weak	0.780	0.561	0.653	0.860	0.474
	Strong	0.702	0.497	0.582	0.819	0.376
	Weak+Strong	**0.791**	**0.620**	**0.695**	**0.889**	**0.533**
3.2	Socio-technical	**0.843**	0.480	0.612	0.809	**0.554**
	Weak	0.759	0.536	0.628	0.857	0.453
	Strong	0.715	0.531	0.609	0.835	0.392
	Weak+Strong	0.792	**0.649**	**0.714**	**0.896**	0.554
3.3	Socio-technical	0.674	0.625	0.649	0.789	**0.452**
	Weak	**0.788**	0.566	0.659	0.814	0.372
	Strong	0.716	0.566	0.632	0.750	0.251
	Weak+Strong	0.787	**0.638**	**0.705**	**0.833**	0.420
3.4	Socio-technical	0.599	0.534	0.565	0.772	0.377
	Weak	**0.757**	0.484	0.591	0.774	0.295
	Strong	0.690	0.550	0.612	0.764	0.273
	Weak+Strong	0.747	**0.605**	**0.669**	**0.825**	**0.410**

Table 4.4 presents a sample results of using data from previous release to predict top 30% defect-prone Eclipse Java packages, Netbeans Java JAR files, and Gnome Ubuntu packages in next release. The complete prediction results of the investigated Eclipse releases (2.0 - 3.4), Netbeans releases (6.1 - 7.0), and Gnome releases (2.16 - 2.26) are presented in Table C.1, Table C.2, and Table C.3.

Based on the results, we make the following observations:

- The predictions made using the *Weak* model are more accurate than predictions made on the *socio-technical* model.

As shown in Table 4.4, in most cases, the *Weak* model achieves higher precision, recall, F score, AUC, and Nagelkerke's pseudo R^2 values. Taking Table 4.4a of Eclipse as an example, on average, the precision and recall of the *Weak* model are higher than that of the *socio-technical* model by 0.1 and 0.15. The results show that social network metrics calculated on the *weak network* are better indicators of defective components than that calculated on the *socio-technical networks*.

- The predictions made using the *Weak+Strong* model are more accurate than predictions made by the other models.

(b) Predict top 30% defect-prone Netbeans Java JAR files

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
6.5	Socio-technical	0.667	0.211	0.321	0.808	0.401
	Weak	0.650	0.317	0.426	0.861	0.392
	Strong	0.651	0.341	0.448	0.848	0.367
	Weak+Strong	**0.673**	**0.427**	**0.522**	**0.889**	**0.461**
6.7	Socio-technical	0.636	0.308	0.415	0.799	0.386
	Weak	0.662	0.398	0.497	0.829	0.356
	Strong	0.615	0.390	0.478	0.837	0.385
	Weak+Strong	**0.679**	**0.463**	**0.551**	**0.853**	**0.421**
6.8	Socio-technical	0.674	0.305	0.420	0.821	0.369
	Weak	**0.698**	0.330	0.448	0.828	0.357
	Strong	0.681	0.352	0.464	0.841	0.365
	Weak+Strong	0.685	**0.407**	**0.510**	**0.848**	**0.385**
6.9	Socio-technical	**0.830**	0.321	0.463	0.825	0.416
	Weak	0.680	0.358	0.469	0.835	0.355
	Strong	0.625	0.316	0.420	0.843	0.359
	Weak+Strong	0.726	**0.474**	**0.573**	**0.883**	**0.459**
7.0	Socio-technical	0.456	**0.514**	0.483	0.774	0.433
	Weak	0.750	0.394	0.517	0.830	0.375
	Strong	0.760	0.416	0.538	0.830	0.387
	Weak+Strong	**0.776**	0.482	**0.595**	**0.857**	**0.445**

(c) Predict top 30% defect-prone Gnome Ubuntu packages

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
2.18	Socio-technical	0.800	0.604	0.688	0.867	0.588
	Weak	**0.865**	0.653	**0.744**	0.849	0.519
	Strong	0.786	**0.673**	0.725	**0.916**	**0.616**
	Weak+Strong	0.805	**0.673**	0.733	0.897	0.601
2.20	Socio-technical	0.800	**0.721**	**0.759**	0.883	0.531
	Weak	0.786	0.623	0.695	0.832	0.470
	Strong	0.800	0.679	0.735	0.877	0.514
	Weak+Strong	**0.833**	0.660	0.737	**0.896**	**0.583**
2.22	Socio-technical	0.614	**0.879**	0.723	0.858	**0.653**
	Weak	**0.837**	0.590	0.692	0.863	0.514
	Strong	0.783	0.770	**0.777**	0.896	0.576
	Weak+Strong	0.815	0.721	0.765	**0.919**	0.635
2.24	Socio-technical	0.516	0.493	0.504	0.742	0.573
	Weak	**0.829**	0.586	0.687	0.815	0.481
	Strong	0.796	**0.672**	**0.729**	0.898	**0.579**
	Weak+Strong	0.804	0.638	0.712	**0.900**	0.577
2.26	Socio-technical	0.786	**0.733**	0.759	**0.923**	0.580
	Weak	**0.904**	0.701	**0.790**	0.898	0.616
	Strong	0.810	0.701	0.752	0.892	0.590
	Weak+Strong	0.845	0.731	0.784	0.909	**0.631**

As an instance, regarding Table 4.3b of Netbeans, on average, F score of the *Weak+Strong* model outperforms other models (*Weak*, *Strong*, and *socio-technical*) by about 0.1. Also, the Nagelkerke's pseudo R^2 of the *Weak+Strong* model in all Netbeans releases are higher than the other models. Furthermore, we note that on the Eclipse, Netbeans, and Gnome datasets, on average, the precision and recall of our *Weak+Strong* model outperforms the *socio-technical* model by 0.1 and 0.15. Considering the substantial number of Java packages shipped in the Eclipse IDE and Java JAR files shipped in the Netbeans IDE, the improvement is significant in terms of the number of components predicted to have defects. Take the prediction of top 25% defect-prone Eclipse-3.2 Java packages as an example, 0.15 higher recall means we can identify $25\% \times 700 \times 0.15 = 25$ more defect-prone Java packages.

For Gnome, however, we observe that the *Weak+Strong* model is not necessarily more accurate than the other models. In Table 4.3c, for all releases, the F score of the *Weak+Strong* model is not higher than all the other models. For Gnome-2.18, Gnome-2.22, Gnome-2.24, and Gnome-2.26, the precision of the *Weak+Strong* model is lower than the *Weak* model. Although the prediction made using the *Weak+Strong* model is not the most accurate, the *Weak+Strong* model achieves higher precision and F score than the *socio-technical* model on most Gnome releases. Therefore, we can still consider metrics calculated from the *weak network* and the *strong network* to be good predictors of defect-prone software components.

Therefore, based on above findings, *for the investigated Eclipse, Netbeans and Gnome releases*, we conclude that *there is no evidence to reject* ***TS4*** that social network metrics calculated from the *weak network* and the *strong network* can be used as predictors to predict defect-prone software components across releases.

Chapter 5

Limitations

This chapter discusses the threats to validity of the results of this thesis and the limitations of my approach.

Threats to Validity

The main concern of validity is the accuracy of the extracted defect data and the reliability of the evaluated relation strength.

The technique we use to identify defects is based on keyword matching in the commit log [50]. Due to the flexible nature of open source software projects, commit logs in Eclipse, Netbeans, and Gnome that concern bugs do not necessarily follow a common phrasing pattern. Therefore, the matching heuristic cannot guarantee full coverage of possible defect related keywords and hence the defect data we extract from the bug report database and repository are likely to be incomplete. Furthermore, the relation strength is measured by the *citation influence model* [18]. This measure is affected by the text quality in commit logs and source file comments. Namely, lack of comments in source files and varying vocabulary could affect the use of textual measures. We argue that text quality of the commit logs and the coverage of comments are common issues in exploring open source projects [5] and we expect that the general trend in our findings will still be preserved with better data.

In addition to the text quality issue, the choice of optimal topic model parameters could be a difficult task [24]. In our experiments, we simply use the default settings as in [18]. Namely, we set $K = 20$ as the number of topics. Further analysis is required to infer the optimal parameters. However, even with default parameter $K = 20$, we show that the weak relations and strong relations have different correlations with the number of defects and optimal parameters will only improve the results. Therefore, we can expect that the trend in our findings will hold with optimal parameters.

Another threat to validity comes from the fact that the bug reports we use to extract defect data is filed by users. First of all, users are likely to report duplicated bugs [46]. Secondly, users may not encounter or forget to report failures in the software even if a component is faulty. Thirdly, bugs that

are found in a particular release by users may root from an earlier release but are mistakenly marked as found in a later release. Therefore, the defect data we extract from the bug report databases are likely to be incomplete and inaccurate. However, as with the text quality issue, the reliability of user filed bug reports is a common issue in exploring open source projects [5]. We expect that the general trend in our findings will still be preserved with better data.

General Limitations

In this thesis, we propose to use the *citation influence model* to determine the strength of relations in the *socio-technical network* so as to construct the *weak network* and the *strong network*, and apply social network analysis to the two networks to calculate metrics as predictors of defect-prone software components. Our approach can only operate on a software system that is able to handle topic models and social network analysis.

First of all, the investigated software system and the version control system must preserve enough information of the development process (*e.g.* comments in source code and commit logs) so that the *citation influence model* is able to extract reliable topics and the relations of topics from the logged text. It requires the developers to be disciplined to write down necessary documents and meaningful commit logs. Therefore, out approach may only work on a mature software system with professional developers and documentation policy.

Secondly, the investigated software project must be big enough (the number of components and the number of involved developers) to construct a network of considerable size. As we calculate social network metrics on the network, metrics calculated on a small network might not be an accurate indicator of centrality of software components.

Chapter 6

Related Work

In this chapter, we discuss two lines of relate work: using various attributes to predict software defects and the application of topic models in mining software repositories.

6.1 Predict Software Defects with Various Attributes

Other than using social network metrics to predict defect-prone software components, there have been considerable efforts of using various attributes to predict software defects. Most of these attributes can be classified into two categories: 'non-structural' and 'structural'. While 'non-structural' attributes capture the internal features of a single software component, like code complexity metrics [35], 'structural' attributes are concerned with the high level software system structures and the centrality of a particular software component within the global software system, *e.g.*, software dependency network metrics [64].

6.1.1 Non-structural Attributes

Nagappan *et al.* used code complexity metrics to build a prediction model of software defects [35]. Shin *et al.* further proposed to combine method-calling structures and code complexity metrics to predict software defects [49]. In addition to the code attributes, Nagappan *et al.* demonstrated that development history could be used to predict defect-prone software components. They built a prediction model of software defects with code churn metrics and software dependency counts [37]. Apart from that, Nagappan *et al.* also demonstrated the efficacy of using change bursts (repeated changes) as defect predictors [36]. Giger *et al.* further proposed a set of finer-grained code change metrics. They subsequently used experiments on the Eclipse dataset to show that their metrics are good predictors of software defects [23]. Note that the employment of 'non-structural' attributes does not exclude the use of 'structural' attributes. Zimmermann *et al* combined 'non-structural' attributes (code complexity metrics) and 'structural' attributes (software dependency network) together to predict defect-prone software components [64]. Therefore, the *equal-level networks* (Sec-

tion 3.3) we construct in this thesis could be used together with 'non-structural' attributes to predict defect-prone software components.

6.1.2 Structural Attributes

In addition to code and history metrics, previous studies showed that social factors in both the *software dependency network* and the development team had a significant effect on software quality. Zimmermann *et al.* [64], Nguyen *et al.* [39], and Tosun *et al.* [58] constructed the software dependency network at the binary level (Windows DLL files), the Java package level, and the Java JAR file level, respectively, and used social network metrics to predict post-release defects. Wolf *et al.* [61] showed that developer communication is highly correlated with integration failures. Pinzger *et al.* [40] constructed a *developer contribution network* by linking a developer to a Windows Vista binary that this developer has contributed to. They further used social network analysis on the *developer contribution network* to demonstrate the efficacy of social network measures as predictors for post-release defects. In this thesis, we also utilize social factors to predict software defects. However, rather than exploring new metrics and new social factors, we seek to enhance the existing *socio-technical network* approach [4] by considering the strength of social relations within the software system and within the development team.

6.2 Topic Models in Mining Software Repositories

Topic models are widely used in the Mining Software Repositories field [57] to recover software artifact traceability [26], analyze software evolution [54, 55], and study software defects [13]. While this thesis uses a novel topic model called the *citation influence model*, according to a survey conducted by Thomas *et al.*, the *Latent Semantic Indexing* (LSI) model [19] and the *Latent Dirichlet Allocation* (LDA) model [6] are the two most heavily used topic models in the software engineering community [57].

6.2.1 Software Artifact Traceability Recovery

Topic models could be used to trace related software artifacts, like linking requirements documents to relevant source files. Such linking helps managers and developers to trace requirement (requirements documents) or concerns (bug reports) to the implementation (source files) [57]. Marcus *et al.* used LSI [19] to measure the similarity between source files and external documents (*i.e.*, user manuals) , and subsequently linked documents to similar source files [33]. Asuncion *et al.* used LDA [6] during the software development process to link documents, *e.g.*, wiki pages and bug reports, to relevant software modules [1]. McMaillan *et al* indirectly linked pairs of relevant documents by combining the links of documents to source files and source files to source files [34]. They used LSI to trace relevant source files for the given document. Hindle *et al.* related code commits to the topics extracted from requirements documents, and further validated the relevance between extracted

topics and development efforts [26]. The assumption of using topic models to link source code to related artifacts is that topic models could discover the latent textual relations between source code and relevant documents. In this work, we use the *citation influence model* [18] to measure the strength of textual relations between source code and relevant documents (Section 3.1).

6.2.2 Software Evolution Analysis

Researchers have proposed to apply the topic model to source files of different software releases to monitor the changes of topics over time. Linkstead *et al.* applied the LDA topic model to source files from different Eclipse releases and ArgoUML releases to study the trends in the topics over time [32]. Hindle *et al.* applied the LDA topic model to developer commit logs to identify the trend of topics in the development process [27]. Thomas *et al.* assessed the validity of using topic models to analyze software evolution [54]. They used experiments on JHotDraw to demonstrate that most changes in topics are caused by changes in software development activities. Thomas *et al.* further introduced the *Diff* topic model and proposed to use *Diff* to analyze software topics evolution [56]. The *Diff* model is derived from LDA and could extract topics from source files in an incremental way. In this thesis, we apply the *citation influence model* to different Eclipse, Netbeans, and Gnome releases to evaluate the strength of relations. However, instead of studying the change of strength of relations over time, we utilize the strength of relations to predict defect-prone software components over time.

Chapter 7

Conclusions and Future Work

In this thesis, we demonstrated how to leverage the relation strength information to assist social network analysis for predicting defects. In the context of the Eclipse, Netbeans, and Gnome datasets, we validated the following claims:

- Weak relations have a stronger relationship with the number of defects.
- In terms of social network analysis on the dependency network, weak relations and strong relations have different correlations with the number of post-release defects.
- Strong relations and weak relations have different contributions to the predictive power of defect prediction models.
- The separation of weak relations and strong relations in social network analysis improves the predictive accuracy for defect-prone components.

We evaluated the above claims for the Eclipse, Netbeans, and Gnome data sets at the granularity of Java packages, Java JAR files, and Ubuntu packages, respectively. The experiment showed that the separation of weak relations and strong relations significantly improves the post-release defect prediction accuracy.

In practical uses, our approach could help managers and testers better test a software system. For example, managers and testers could use our approach to predict whether a software component is defect-prone before the actual testing, and subsequently allocate more resources on the software components that may contain many defects.

Regarding future work, we plan to apply our approach to other software systems to test the efficacy in different domains. Also, we can divide the software system into subsystems (*e.g.*, Eclipse platform and Eclipse plugin development environment) and compare the efficacy of our approach in different subsystems. Apart from that, we note that lots of heterogeneous software systems have external dependencies, and some defects are rooted in external dependencies or are caused by misuse of external dependencies. Therefore, to predict these inter-software defects, we plan to merge the *equal-level networks* of related software systems and conduct social network analysis on the

united *equal-level network* to predict defects across the software border. Last but not least, we plan to investigate the optimal threshold for top X% rank for logistic regression. In the experiments, we numerate different thresholds (10%, 15%, 20%, 25%, and 30%) for logistic regression without selecting a best threshold. We could put more efforts on finding the optimal threshold given the required recall and precision.

Bibliography

[1] Hazeline U Asuncion, Arthur U Asuncion, and Richard N Taylor. Software traceability with topic modeling. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 95–104. ACM, 2010.

[2] Jean-Francois Bergeretti and Bernard A. Carré. Information-flow and data-flow analysis of while-programs. *ACM Trans. Program. Lang. Syst.*, 7(1):37–61, January 1985.

[3] Antonia Bertolino. Software testing research: Achievements, challenges, dreams. In *Future of Software Engineering, 2007. FOSE'07*, pages 85–103. IEEE, 2007.

[4] C. Bird, N. Nagappan, H. Gall, B. Murphy, and P. Devanbu. Putting it all together: Using socio-technical networks to predict failures. In *Software Reliability Engineering, 2009. ISSRE '09. 20th International Symposium on*, pages 109 –119, nov. 2009.

[5] Christian Bird, Adrian Bachmann, Eirik Aune, John Duffy, Abraham Bernstein, Vladimir Filkov, and Premkumar Devanbu. Fair and balanced? bias in bug-fix datasets. In *Proceedings of the the Seventh joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, 2009.

[6] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[7] B.W. Boehm. Software engineering economics. *Software Engineering, IEEE Transactions on*, (1):4–21, 1984.

[8] Phillip Bonacich. Power and centrality: A family of measures. *American Journal of Sociology*, 92(5):pp. 1170–1182, 1987.

[9] Simone Borra and Agostino Di Ciaccio. Measuring the prediction error. a comparison of cross-validation, bootstrap and covariance penalty methods. *Computational statistics & data analysis*, 54(12):2976–2989, 2010.

[10] Andrew P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30:1145–1159, 1997.

[11] Ronald S. Burt. *Structural holes: The social structure of competition*. Harvard University Press, Cambridge, MA, 1992.

[12] Jonathan Chang. Relational topic models for document networks. In *In Proc. of Conf. on AI and Statistics (AISTATS*, page 2009.

[13] T. Chen, S. W. Thomas, M. Nagappan, and A. E. Hassan. Explaining software defects using topic models. In *Proceedings of the 9th Working Conference on Mining Software Repositories*, pages 189–198, 2012.

[14] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, 20(6):476–493, June 1994.

[15] Cynthia F. Cohen, Stanley J. Birkin, Monica J. Garfield, and Harold W. Webb. Managing conflict in software testing. *Commun. ACM*, 47(1):76–81, January 2004.

[16] Anthony Christopher Davison and David Victor Hinkley. *Bootstrap methods and their application*, volume 1. Cambridge university press, 1997.

[17] Harpal Dhama. Quantitative models of cohesion and coupling in software. *Journal of Systems and Software*, 29(1):65 – 74, 1995. ¡ce:title¿Oregon Metric Workshop¡/ce:title¿.

[18] Laura Dietz, Steffen Bickel, and Tobias Scheffer. Unsupervised prediction of citation influences. In *Proceedings of the 24th international conference on Machine learning*, ICML '07, pages 233–240, New York, NY, USA, 2007. ACM.

[19] Susan T Dumais, GW Furnas, TK Landauer, S Deerwester, SC Deerwester, et al. Latent semantic indexing. In *Proceedings of the Text Retrieval Conference*, 1995.

[20] L.C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40:35–41, 1977.

[21] Linton C. Freeman. Centrality in social networks conceptual clarification. *Social Networks*, page 215, 1978.

[22] M. Gethers and D. Poshyvanyk. Using relational topic models to capture coupling among classes in object-oriented software systems. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1 –10, sept. 2010.

[23] E. Giger, M. Pinzger, and H.C. Gall. Comparing fine-grained source code changes and code churn for bug prediction. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 83–92. ACM, 2011.

[24] S. Grant and J.R. Cordy. Estimating the optimal number of latent concepts in source code analysis. In *Source Code Analysis and Manipulation (SCAM), 2010 10th IEEE Working Conference on*, pages 65 –74, sept. 2010.

[25] S. Henry and D. Kafura. Software structure metrics based on information flow. *Software Engineering, IEEE Transactions on*, SE-7(5):510 – 518, sept. 1981.

[26] A. Hindle, C. Bird, T. Zimmermann, and N. Nagappan. Relating requirements to implementation via topic analysis: Do topics extracted from requirements make sense to managers and developers? In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 243 –252, sept. 2012.

[27] A. Hindle, M.W. Godfrey, and R.C. Holt. What's hot and what's not: Windowed developer topic analysis. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, pages 339 –348, sept. 2009.

[28] Wei Hu and Kenny Wong. Using citation influence to predict software defects. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 419–428, Piscataway, NJ, USA, 2013. IEEE Press.

[29] I.T. Jolliffe. *Principal Component Analysis*. Springer Verlag, 1986.

[30] Richard J Larsen and Morris L Marx. *An introduction to mathematical statistics and its applications; 5th ed.* Prentice Hall, Boston, MA, 2012. The book can be consulted by contacting: IT-ES-DNG: Abler, Daniel.

[31] Wei Li and Sallie Henry. Object-oriented metrics that predict maintainability. *J. Syst. Softw.*, 23(2):111–122, November 1993.

[32] Erik Linstead, Cristina Lopes, and Pierre Baldi. An application of latent dirichlet allocation to analyzing software evolution. In *Machine Learning and Applications, 2008. ICMLA'08. Seventh International Conference on*, pages 813–818. IEEE, 2008.

[33] Andrian Marcus and Jonathan I Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *Software Engineering, 2003. Proceedings. 25th International Conference on*, pages 125–135. IEEE, 2003.

[34] Collin McMillan, Denys Poshyvanyk, and Meghan Revelle. 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*, TEFSE '09, pages 41–48, Washington, DC, USA, 2009. IEEE Computer Society.

[35] N. Nagappan, T. Ball, and A. Zeller. Mining metrics to predict component failures. In *Proceedings of the 28th international conference on Software engineering*, pages 452–461. ACM, 2006.

[36] N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, and B. Murphy. Change bursts as defect predictors. In *Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on*, pages 309 –318, nov. 2010.

[37] Nachiappan Nagappan and Thomas Ball. Using software dependencies and churn metrics to predict field failures: An empirical case study. In *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*, ESEM '07, pages 364–373, Washington, DC, USA, 2007. IEEE Computer Society.

[38] N. J. D. Nagelkerke. A note on a general definition of the coefficient of determination. *Biometrika*, 78(3):691–692, September 1991.

[39] T.H.D. Nguyen, B. Adams, and A.E. Hassan. Studying the impact of dependency network measures on software quality. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1 –10, sept. 2010.

[40] Martin Pinzger, Nachiappan Nagappan, and Brendan Murphy. Can developer-module networks predict failures? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, SIGSOFT '08/FSE-16, pages 2–12, New York, NY, USA, 2008. ACM.

[41] Ian Porteous, David Newman, Alexander Ihler, Arthur Asuncion, Padhraic Smyth, and Max Welling. Fast collapsed gibbs sampling for latent dirichlet allocation. In *KDD08: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 569–577, 2008.

[42] David M. W. Powers. Evaluation: From precision, recall and f-factor to roc, informedness, markedness & correlation. Technical Report SIE-07-001, School of Informatics and Engineering, Flinders University, Adelaide, Australia, 2007.

[43] C. Rajaraman and M.R. Lyu. Reliability and maintainability related software coupling metrics in c++ programs. In *Software Reliability Engineering, 1992. Proceedings., Third International Symposium on*, pages 303–311, 1992.

[44] S. Rapps and E.J. Weyuker. Selecting software test data using data flow information. *Software Engineering, IEEE Transactions on*, SE-11(4):367 – 375, april 1985.

[45] Britta Ruhnau. Eigenvector-centrality - a node-centrality? *Social Networks*, 22(4):357 – 365, 2000.

[46] Per Runeson, Magnus Alexandersson, and Oskar Nyholm. Detection of duplicate defect reports using natural language processing. In *Proceedings of the 29th international conference on Software Engineering*, ICSE '07, pages 499–510, Washington, DC, USA, 2007. IEEE Computer Society.

[47] Gert Sabidussi. The centrality index of a graph. *Psychometrika*, 31(4):581–603, December 1966.

[48] N. Serrano and I. Ciordia. Bugzilla, itracker, and other bug trackers. *Software, IEEE*, 22(2):11–13, 2005.

[49] Y. Shin, R. Bell, T. Ostrand, and E. Weyuker. Does calling structure information improve the accuracy of fault prediction? In *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on*, pages 61–70. IEEE, 2009.

[50] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? *SIGSOFT Softw. Eng. Notes*, 30(4):1–5, May 2005.

[51] Karen Stephenson and Marvin Zelen. Rethinking centrality: Methods and examples. *Social Networks*, 11(1):1 – 37, 1989.

[52] Gregory Tassey. The economic impacts of inadequate infrastructure for software testing. *National Institute of Standards and Technology, RTI Project*, (7007.011), 2002.

[53] C. Taube-Schock, R. Walker, and I. Witten. Can we avoid high coupling? *ECOOP 2011–Object-Oriented Programming*, pages 204–228, 2011.

[54] Stephen W. Thomas, Bram Adams, Ahmed E. Hassan, and Dorothea Blostein. Validating the use of topic models for software evolution. In *Proceedings of the 2010 10th IEEE Working Conference on Source Code Analysis and Manipulation*, SCAM '10, pages 55–64, Washington, DC, USA, 2010. IEEE Computer Society.

[55] Stephen W. Thomas, Bram Adams, Ahmed E. Hassan, and Dorothea Blostein. Modeling the evolution of topics in source code histories. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, pages 173–182, New York, NY, USA, 2011. ACM.

[56] Stephen W Thomas, Bram Adams, Ahmed E Hassan, and Dorothea Blostein. Modeling the evolution of topics in source code histories. In *Proceedings of the 8th working conference on mining software repositories*, pages 173–182. ACM, 2011.

[57] S.W. Thomas. Mining software repositories with topic models. Technical report, Technical Report 2012-586, School of Computing, Queens University, 2012.

[58] Ayşe Tosun, Burak Turhan, and Ayşe Bener. Validation of network measures as indicators of defective modules in software systems. In *Proceedings of the 5th International Conference on Predictor Models in Software Engineering*, PROMISE '09, pages 5:1–5:9, New York, NY, USA, 2009. ACM.

[59] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[60] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83, 1945.

[61] Timo Wolf, Adrian Schroter, Daniela Damian, and Thanh Nguyen. Predicting build failures using social network analysis on developer communication. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 1–11, Washington, DC, USA, 2009. IEEE Computer Society.

[62] Eric S. K. Yu and John Mylopoulos. Understanding why in software process modelling, analysis, and design. In *Proceedings of the 16th international conference on Software engineering*, ICSE '94, pages 159–168, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[63] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl. Mining version histories to guide software changes. *Software Engineering, IEEE Transactions on*, 31(6):429 – 445, june 2005.

[64] Thomas Zimmermann and Nachiappan Nagappan. Predicting defects using network analysis on dependency graphs. In *Proceedings of the 30th international conference on Software engineering*, ICSE '08, pages 531–540, New York, NY, USA, 2008. ACM.

Appendix A

Correlation of Social Network Metrics with the Number of Post-release Defects

Table A.1: Correlation of social network metrics for Eclipse with number of post-release defects. Strong and Weak denote the *Strong network* and the *Weak network* respectively. Bold values indicate that the correlation on that network is higher than that on other networks. Correlation significant at 99% and 95% with Wilcoxon signed-rank tests are marked by (**) and (*) respectively.

(a) Eclipse 2.0 at Java package level

Global Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Degree centrality	**0.603(**)**	0.583(**)	0.570(**)
Betweenness centrality	0.298(**)	0.211(**)	**0.410(**)**
Closeness centrality	0.179(**)	-0.213(**)	**0.451(**)**
Information centrality	**0.491(**)**	0.212(**)	0.491(**)
Distance weighted reach	0.201(**)	-0.213(**)	**0.456(**)**
Eigenvector centrality	0.201(**)	**0.473(**)**	0.302(**)
Effective size	0.376(**)	0.165(**)	**0.423(**)**
Efficiency	0.123(*)	0.117(*)	**-0.155(**)**
Constriant	-0.151(**)	0.148(**)	**0.390(**)**
Hierarchy	-0.125(*)	0.169(**)	**0.240(**)**

InOut-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.309(**)	0.060	**0.391(**)**
Ego ties	0.331(**)	0.055	**0.394(**)**
Ego pairs	0.309(**)	0.060	**0.391(**)**
Ego density	**-0.205(**)**	-0.074	-0.190(**)
Ego reach	0.189(**)	-0.087	**0.474(**)**
Ego reach efficiency	-0.244(**)	-0.181(**)	**0.272(**)**
Ego brokerage	0.354(**)	0.173(**)	**0.479(**)**
Ego normalized brokerage	0.145(**)	0.131(*)	**0.345(**)**
Ego betweeness	0.330(**)	0.221(**)	**0.468(**)**
Ego normalized betweeness	0.300(**)	0.267(**)	**0.462(**)**

In-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.230(**)	0.169(**)	**0.280(**)**
Ego ties	0.225(**)	0.139(*)	**0.279(**)**
Ego pairs	0.230(**)	0.169(**)	**0.280(**)**
Ego density	-0.029	**-0.084**	0.040
Ego reach	0.189(**)	-0.087	**0.474(**)**
Ego reach efficiency	-0.039	-0.195(**)	**0.373(**)**

Out-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.399(**)	0.030	**0.498(**)**
Ego ties	0.424(**)	0.000	**0.493(**)**
Ego pairs	0.399(**)	0.030	**0.498(**)**
Ego density	-0.110(*)	-0.116(*)	**0.173(**)**
Ego reach	0.189(**)	-0.087	**0.474(**)**
Ego reach efficiency	**-0.463(**)**	-0.180(**)	0.204(**)

(b) Eclipse 2.1 at Java package level

Global Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Degree centrality	**0.625(**)**	0.606(**)	0.584(**)
Betweenness centrality	0.424(**)	0.361(**)	**0.446(**)**
Closeness centrality	0.260(**)	-0.106(*)	**0.426(**)**
Information centrality	**0.580(**)**	0.288(**)	0.477(**)
Distance weighted reach	0.311(**)	-0.112(*)	**0.438(**)**
Eigenvector centrality	**0.562(**)**	0.474(**)	0.474(**)
Effective size	**0.488(**)**	0.314(**)	0.443(**)
Efficiency	0.086	0.149(**)	**-0.208(**)**
Constriant	**-0.339(**)**	0.212(**)	0.313(**)
Hierarchy	**-0.306(**)**	0.179(**)	0.262(**)

InOut-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.446(**)	0.236(**)	**0.455(**)**
Ego ties	0.450(**)	0.221(**)	**0.458(**)**
Ego pairs	0.446(**)	0.236(**)	**0.455(**)**
Ego density	**-0.383(**)**	-0.197(**)	-0.152(**)
Ego reach	0.208(**)	0.129(*)	**0.396(**)**
Ego reach efficiency	**-0.344(**)**	-0.141(**)	0.239(**)
Ego brokerage	0.495(**)	0.349(**)	**0.501(**)**
Ego normalized brokerage	0.232(**)	0.239(**)	**0.342(**)**
Ego betweeness	**0.476(**)**	0.356(**)	0.475(**)
Ego normalized betweeness	0.401(**)	0.389(**)	**0.438(**)**

In-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	**0.386(**)**	0.343(**)	0.369(**)
Ego ties	0.378(**)	0.318(**)	**0.381(**)**
Ego pairs	**0.386(**)**	0.343(**)	0.369(**)
Ego density	-0.113(*)	**-0.124(*)**	0.081
Ego reach	0.208(**)	0.129(*)	**0.396(**)**
Ego reach efficiency	**-0.283(**)**	-0.206(**)	0.276(**)

Out-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.445(**)	0.109(*)	**0.483(**)**
Ego ties	0.432(**)	0.037	**0.482(**)**
Ego pairs	0.445(**)	0.109(*)	**0.483(**)**
Ego density	-0.206(**)	**-0.290(**)**	0.266(**)
Ego reach	0.208(**)	0.129(*)	**0.396(**)**
Ego reach efficiency	**-0.337(**)**	0.040	0.241(**)

(c) Eclipse 3.0 at Java package level

Global Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Degree centrality	**0.636(**)**	0.611(**)	0.630(**)
Betweenness centrality	0.385(**)	0.322(**)	**0.505(**)**
Closeness centrality	0.178(**)	-0.161(**)	**0.446(**)**
Information centrality	**0.536(**)**	0.345(**)	0.493(**)
Distance weighted reach	0.223(**)	-0.108(*)	**0.456(**)**
Eigenvector centrality	0.253(**)	0.446(**)	**0.465(**)**
Effective size	**0.551(**)**	0.356(**)	0.504(**)
Efficiency	0.118(**)	0.157(**)	**-0.204(**)**
Constriant	-0.346(**)	0.247(**)	**0.359(**)**
Hierarchy	**-0.301(**)**	0.170(**)	0.128(**)

InOut-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	**0.535(**)**	0.308(**)	0.526(**)
Ego ties	**0.559(**)**	0.322(**)	0.510(**)
Ego pairs	**0.535(**)**	0.308(**)	0.526(**)
Ego density	**-0.337(**)**	-0.183(**)	-0.261(**)
Ego reach	0.166(**)	0.028	**0.430(**)**
Ego reach efficiency	**-0.461(**)**	-0.375(**)	0.203(**)
Ego brokerage	**0.551(**)**	0.392(**)	0.525(**)
Ego normalized brokerage	0.224(**)	0.260(**)	**0.366(**)**
Ego betweeness	0.492(**)	0.389(**)	**0.519(**)**
Ego normalized betweeness	0.414(**)	0.394(**)	**0.498(**)**

In-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.421(**)	0.372(**)	**0.444(**)**
Ego ties	0.413(**)	0.365(**)	**0.429(**)**
Ego pairs	0.421(**)	0.372(**)	**0.444(**)**
Ego density	**-0.174(**)**	-0.092(*)	-0.049
Ego reach	0.166(**)	0.028	**0.430(**)**
Ego reach efficiency	-0.235(**)	**-0.304(**)**	0.276(**)

Out-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	**0.521(**)**	0.184(**)	0.503(**)
Ego ties	**0.536(**)**	0.151(**)	0.504(**)
Ego pairs	**0.521(**)**	0.184(**)	0.503(**)
Ego density	-0.151(**)	-0.194(**)	**0.323(**)**
Ego reach	0.166(**)	0.028	**0.430(**)**
Ego reach efficiency	**-0.520(**)**	-0.324(**)	0.279(**)

(d) Eclipse 3.1 at Java package level

Global Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Degree centrality	**0.632(**)**	0.619(**)	0.607(**)
Betweenness centrality	0.438(**)	0.361(**)	**0.538(**)**
Closeness centrality	0.244(**)	0.162(**)	**0.487(**)**
Information centrality	**0.575(**)**	0.327(**)	0.529(**)
Distance weighted reach	0.274(**)	0.137(**)	**0.489(**)**
Eigenvector centrality	0.223(**)	0.380(**)	**0.500(**)**
Effective size	**0.598(**)**	0.415(**)	0.532(**)
Efficiency	0.116(**)	0.160(**)	**-0.186(**)**
Constriant	**-0.463(**)**	0.250(**)	0.402(**)
Hierarchy	**-0.410(**)**	0.137(**)	0.181(**)

InOut-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	**0.584(**)**	0.380(**)	0.549(**)
Ego ties	**0.597(**)**	0.376(**)	0.546(**)
Ego pairs	**0.584(**)**	0.380(**)	0.549(**)
Ego density	**-0.416(**)**	-0.265(**)	-0.303(**)
Ego reach	0.198(**)	0.111(**)	**0.481(**)**
Ego reach efficiency	**-0.525(**)**	-0.386(**)	0.279(**)
Ego brokerage	**0.599(**)**	0.444(**)	0.568(**)
Ego normalized brokerage	0.233(**)	0.288(**)	**0.419(**)**
Ego betweeness	0.541(**)	0.426(**)	**0.560(**)**
Ego normalized betweeness	0.439(**)	0.404(**)	**0.542(**)**

In-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.472(**)	0.395(**)	**0.480(**)**
Ego ties	0.464(**)	0.386(**)	**0.472(**)**
Ego pairs	0.472(**)	0.395(**)	**0.480(**)**
Ego density	**-0.206(**)**	-0.123(**)	-0.068
Ego reach	0.198(**)	0.111(**)	**0.481(**)**
Ego reach efficiency	-0.266(**)	-0.263(**)	**0.338(**)**

Out-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	**0.545(**)**	0.228(**)	0.538(**)
Ego ties	**0.542(**)**	0.180(**)	0.536(**)
Ego pairs	**0.545(**)**	0.228(**)	0.538(**)
Ego density	-0.293(**)	-0.306(**)	**0.352(**)**
Ego reach	0.198(**)	0.111(**)	**0.481(**)**
Ego reach efficiency	**-0.558(**)**	-0.330(**)	0.342(**)

(e) Eclipse 3.2 at Java package level

Global Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Degree centrality	**0.593(**)**	0.565(**)	0.586(**)
Betweenness centrality	0.417(**)	0.278(**)	**0.501(**)**
Closeness centrality	0.330(**)	0.149(**)	**0.513(**)**
Information centrality	0.518(**)	0.182(**)	**0.530(**)**
Distance weighted reach	0.364(**)	-0.087(*)	**0.520(**)**
Eigenvector centrality	0.205(**)	0.282(**)	**0.454(**)**
Effective size	**0.536(**)**	0.253(**)	0.515(**)
Efficiency	**0.190(**)**	0.148(**)	-0.117(**)
Constriant	-0.308(**)	0.189(**)	**0.403(**)**
Hierarchy	-0.232(**)	**0.281(**)**	0.116(**)

InOut-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.507(**)	0.221(**)	**0.511(**)**
Ego ties	**0.515(**)**	0.209(**)	0.500(**)
Ego pairs	0.507(**)	0.221(**)	**0.511(**)**
Ego density	**-0.383(**)**	-0.199(**)	-0.318(**)
Ego reach	0.277(**)	-0.023	**0.485(**)**
Ego reach efficiency	**-0.389(**)**	-0.288(**)	0.264(**)
Ego brokerage	0.528(**)	0.269(**)	**0.535(**)**
Ego normalized brokerage	0.299(**)	0.224(**)	**0.413(**)**
Ego betweeness	0.498(**)	0.290(**)	**0.527(**)**
Ego normalized betweeness	0.445(**)	0.303(**)	**0.516(**)**

In-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.375(**)	0.257(**)	**0.410(**)**
Ego ties	0.370(**)	0.251(**)	**0.400(**)**
Ego pairs	0.375(**)	0.257(**)	**0.410(**)**
Ego density	**-0.159(**)**	-0.081(*)	-0.034
Ego reach	0.277(**)	-0.023	**0.485(**)**
Ego reach efficiency	-0.146(**)	-0.216(**)	**0.332(**)**

Out-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.508(**)	0.082(*)	**0.538(**)**
Ego ties	0.506(**)	0.038	**0.537(**)**
Ego pairs	0.508(**)	0.082(*)	**0.538(**)**
Ego density	-0.329(**)	-0.255(**)	**0.354(**)**
Ego reach	0.277(**)	-0.023	**0.485(**)**
Ego reach efficiency	**-0.466(**)**	-0.251(**)	0.309(**)

(f) Eclipse 3.3 at Java package level

Global Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Degree centrality	**0.563(**)**	0.547(**)	0.515(**)
Betweenness centrality	0.371(**)	0.242(**)	**0.474(**)**
Closeness centrality	0.183(**)	-0.128(**)	**0.441(**)**
Information centrality	0.461(**)	0.168(**)	**0.468(**)**
Distance weighted reach	0.209(**)	-0.131(**)	**0.448(**)**
Eigenvector centrality	0.302(**)	0.259(**)	**0.475(**)**
Effective size	**0.454(**)**	0.245(**)	0.433(**)
Efficiency	**0.221(**)**	0.184(**)	0.035
Constriant	0.200(**)	0.138(**)	**0.421(**)**
Hierarchy	0.310(**)	**0.350(**)**	0.162(**)

InOut-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	**0.414(**)**	0.205(**)	0.411(**)
Ego ties	**0.424(**)**	0.198(**)	0.404(**)
Ego pairs	**0.414(**)**	0.205(**)	0.411(**)
Ego density	**-0.307(**)**	-0.182(**)	-0.243(**)
Ego reach	0.217(**)	-0.027	**0.451(**)**
Ego reach efficiency	-0.279(**)	-0.261(**)	**0.308(**)**
Ego brokerage	0.451(**)	0.284(**)	**0.483(**)**
Ego normalized brokerage	0.271(**)	0.256(**)	**0.417(**)**
Ego betweeness	0.421(**)	0.290(**)	**0.482(**)**
Ego normalized betweeness	0.392(**)	0.312(**)	**0.485(**)**

In-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	**0.350(**)**	0.286(**)	0.348(**)
Ego ties	**0.343(**)**	0.262(**)	0.338(**)
Ego pairs	**0.350(**)**	0.286(**)	0.348(**)
Ego density	**-0.167(**)**	-0.144(**)	-0.049
Ego reach	0.217(**)	-0.027	**0.451(**)**
Ego reach efficiency	-0.159(**)	-0.232(**)	**0.320(**)**

Out-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.418(**)	0.057	**0.470(**)**
Ego ties	0.413(**)	0.036	**0.469(**)**
Ego pairs	0.418(**)	0.057	**0.470(**)**
Ego density	-0.272(**)	-0.123(**)	**0.364(**)**
Ego reach	0.217(**)	-0.027	**0.451(**)**
Ego reach efficiency	-0.337(**)	-0.190(**)	**0.363(**)**

(g) Eclipse 3.4 at Java package level

Global Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Degree centrality	**0.508(**)**	0.482(**)	0.473(**)
Betweenness centrality	0.365(**)	0.267(**)	**0.439(**)**
Closeness centrality	0.188(**)	-0.098(**)	**0.368(**)**
Information centrality	0.398(**)	0.174(**)	**0.428(**)**
Distance weighted reach	0.218(**)	-0.094(**)	**0.378(**)**
Eigenvector centrality	0.322(**)	0.312(**)	**0.452(**)**
Effective size	**0.417(**)**	0.255(**)	0.414(**)
Efficiency	**0.249(**)**	0.134(**)	0.144(**)
Constriant	0.218(**)	0.220(**)	**0.349(**)**
Hierarchy	0.302(**)	**0.331(**)**	0.160(**)

InOut-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.365(**)	0.207(**)	**0.382(**)**
Ego ties	**0.369(**)**	0.197(**)	0.369(**)
Ego pairs	0.365(**)	0.207(**)	**0.382(**)**
Ego density	**-0.286(**)**	-0.164(**)	-0.193(**)
Ego reach	0.228(**)	0.064	**0.388(**)**
Ego reach efficiency	-0.183(**)	-0.123(**)	**0.244(**)**
Ego brokerage	0.420(**)	0.279(**)	**0.448(**)**
Ego normalized brokerage	0.278(**)	0.239(**)	**0.414(**)**
Ego betweeness	0.401(**)	0.292(**)	**0.440(**)**
Ego normalized betweeness	0.387(**)	0.316(**)	**0.449(**)**

In-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.328(**)	0.289(**)	**0.344(**)**
Ego ties	**0.323(**)**	0.273(**)	0.321(**)
Ego pairs	0.328(**)	0.289(**)	**0.344(**)**
Ego density	**-0.127(**)**	-0.069(*)	-0.077(*)
Ego reach	0.228(**)	0.064	**0.388(**)**
Ego reach efficiency	-0.117(**)	-0.158(**)	**0.261(**)**

Out-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.353(**)	0.082(*)	**0.420(**)**
Ego ties	0.349(**)	0.063	**0.418(**)**
Ego pairs	0.353(**)	0.082(*)	**0.420(**)**
Ego density	-0.207(**)	-0.111(**)	**0.308(**)**
Ego reach	0.228(**)	0.064	**0.388(**)**
Ego reach efficiency	-0.185(**)	-0.009	**0.318(**)**

Table A.2: Correlation of social network metrics for Netbeans with number of post-release defects. Strong and Weak denote the *Strong network* and the *Weak network* respectively. Bold values indicate that the correlation on that network is higher than that on other networks. Correlation significant at 99% and 95% with Wilcoxon signed-rank tests are marked by (**) and (*) respectively.

(a) Netbeans 6.1 at Java JAR file level

Global Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Degree centrality	**0.505(**)**	0.475(**)	0.474(**)
Betweenness centrality	0.485(**)	0.425(**)	**0.522(**)**
Closeness centrality	0.465(**)	0.255(**)	**0.500(**)**
Information centrality	**0.523(**)**	0.478(**)	0.494(**)
Distance weighted reach	0.475(**)	0.278(**)	**0.504(**)**
Eigenvector centrality	0.413(**)	0.392(**)	**0.431(**)**
Effective size	**0.511(**)**	0.426(**)	0.485(**)
Efficiency	**0.096(*)**	0.083	0.094(*)
Constriant	0.176(**)	**0.302(**)**	0.283(**)
Hierarchy	0.147(**)	**0.172(**)**	0.160(**)

InOut-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	**0.475(**)**	0.364(**)	0.462(**)
Ego ties	**0.474(**)**	0.344(**)	0.454(**)
Ego pairs	**0.475(**)**	0.364(**)	0.462(**)
Ego density	**-0.373(**)**	-0.319(**)	-0.061
Ego reach	0.501(**)	0.331(**)	**0.503(**)**
Ego reach efficiency	0.203(**)	-0.020	**0.338(**)**
Ego brokerage	**0.530(**)**	0.442(**)	0.501(**)
Ego normalized brokerage	0.335(**)	0.346(**)	**0.381(**)**
Ego betweeness	**0.506(**)**	0.432(**)	0.493(**)
Ego normalized betweeness	**0.500(**)**	0.451(**)	0.494(**)

In-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	**0.384(**)**	0.342(**)	0.370(**)
Ego ties	0.344(**)	0.326(**)	**0.344(**)**
Ego pairs	**0.384(**)**	0.342(**)	0.370(**)
Ego density	**-0.098(*)**	0.021	-0.024
Ego reach	0.501(**)	0.331(**)	**0.503(**)**
Ego reach efficiency	0.181(**)	-0.017	**0.399(**)**

Out-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	**0.540(**)**	0.338(**)	0.516(**)
Ego ties	0.503(**)	0.278(**)	**0.511(**)**
Ego pairs	**0.540(**)**	0.338(**)	0.516(**)
Ego density	**-0.393(**)**	-0.349(**)	0.215(**)
Ego reach	0.501(**)	0.331(**)	**0.503(**)**
Ego reach efficiency	0.177(**)	0.077	**0.337(**)**

(b) Netbeans 6.5 at Java JAR file level

Global Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Degree centrality	**0.469(**)**	0.463(**)	0.464(**)
Betweenness centrality	0.413(**)	0.391(**)	**0.439(**)**
Closeness centrality	0.319(**)	0.174(**)	**0.426(**)**
Information centrality	0.431(**)	0.392(**)	**0.460(**)**
Distance weighted reach	0.336(**)	0.213(**)	**0.431(**)**
Eigenvector centrality	**0.459(**)**	0.444(**)	0.428(**)
Effective size	**0.463(**)**	0.423(**)	0.449(**)
Efficiency	0.132(**)	**0.182(**)**	-0.151(**)
Constriant	-0.244(**)	0.162(**)	**0.285(**)**
Hierarchy	**-0.220(**)**	0.206(**)	0.095(*)

InOut-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.436(**)	0.342(**)	**0.453(**)**
Ego ties	0.443(**)	0.317(**)	**0.454(**)**
Ego pairs	0.436(**)	0.342(**)	**0.453(**)**
Ego density	**-0.329(**)**	-0.313(**)	-0.235(**)
Ego reach	0.330(**)	0.287(**)	**0.429(**)**
Ego reach efficiency	-0.122(**)	-0.091(*)	**0.234(**)**
Ego brokerage	**0.487(**)**	0.447(**)	0.474(**)
Ego normalized brokerage	0.256(**)	0.259(**)	**0.313(**)**
Ego betweeness	0.446(**)	0.426(**)	**0.462(**)**
Ego normalized betweeness	0.424(**)	0.443(**)	**0.447(**)**

In-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.400(**)	**0.400(**)**	0.359(**)
Ego ties	0.346(**)	**0.359(**)**	0.335(**)
Ego pairs	0.400(**)	**0.400(**)**	0.359(**)
Ego density	**-0.252(**)**	-0.177(**)	-0.131(**)
Ego reach	0.330(**)	0.287(**)	**0.429(**)**
Ego reach efficiency	-0.111(*)	-0.168(**)	**0.284(**)**

Out-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.441(**)	0.304(**)	**0.465(**)**
Ego ties	0.413(**)	0.218(**)	**0.460(**)**
Ego pairs	0.441(**)	0.304(**)	**0.465(**)**
Ego density	-0.316(**)	**-0.325(**)**	0.220(**)
Ego reach	0.330(**)	0.287(**)	**0.429(**)**
Ego reach efficiency	-0.117(**)	-0.015	**0.262(**)**

(c) Netbeans 6.7 at Java JAR file level

Global Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Degree centrality	0.416(**)	0.414(**)	**0.424(**)**
Betweenness centrality	0.383(**)	0.348(**)	**0.448(**)**
Closeness centrality	0.323(**)	0.137(**)	**0.396(**)**
Information centrality	0.402(**)	0.376(**)	**0.444(**)**
Distance weighted reach	0.333(**)	0.148(**)	**0.399(**)**
Eigenvector centrality	**0.406(**)**	0.389(**)	0.398(**)
Effective size	0.413(**)	0.314(**)	**0.422(**)**
Efficiency	**0.149(**)**	0.121(**)	-0.060
Constriant	-0.163(**)	0.209(**)	**0.250(**)**
Hierarchy	**-0.170(**)**	0.152(**)	0.121(**)

InOut-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.391(**)	0.252(**)	**0.415(**)**
Ego ties	0.379(**)	0.217(**)	**0.414(**)**
Ego pairs	0.391(**)	0.252(**)	**0.415(**)**
Ego density	**-0.321(**)**	-0.252(**)	-0.154(**)
Ego reach	0.315(**)	0.188(**)	**0.394(**)**
Ego reach efficiency	-0.041	-0.091(*)	**0.219(**)**
Ego brokerage	0.429(**)	0.327(**)	**0.450(**)**
Ego normalized brokerage	0.243(**)	0.216(**)	**0.302(**)**
Ego betweeness	0.409(**)	0.347(**)	**0.442(**)**
Ego normalized betweeness	0.397(**)	0.367(**)	**0.437(**)**

In-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.333(**)	0.311(**)	**0.344(**)**
Ego ties	0.294(**)	0.290(**)	**0.309(**)**
Ego pairs	0.333(**)	0.311(**)	**0.344(**)**
Ego density	**-0.175(**)**	-0.069	-0.129(**)
Ego reach	0.315(**)	0.188(**)	**0.394(**)**
Ego reach efficiency	-0.031	-0.152(**)	**0.279(**)**

Out-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.394(**)	0.165(**)	**0.447(**)**
Ego ties	0.368(**)	0.109(*)	**0.446(**)**
Ego pairs	0.394(**)	0.165(**)	**0.447(**)**
Ego density	**-0.311(**)**	-0.223(**)	0.175(**)
Ego reach	0.315(**)	0.188(**)	**0.394(**)**
Ego reach efficiency	-0.045	0.105(*)	**0.220(**)**

(d) Netbeans 6.8 at Java JAR file level

Global Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Degree centrality	0.462(**)	**0.464(**)**	0.453(**)
Betweenness centrality	**0.447(**)**	0.342(**)	0.432(**)
Closeness centrality	**0.400(**)**	-0.080	0.394(**)
Information centrality	0.418(**)	0.361(**)	**0.452(**)**
Distance weighted reach	**0.411(**)**	0.072	0.394(**)
Eigenvector centrality	**0.449(**)**	0.360(**)	0.415(**)
Effective size	**0.426(**)**	0.359(**)	0.417(**)
Efficiency	**0.225(**)**	0.223(**)	-0.164(**)
Constriant	-0.315(**)	0.276(**)	**0.335(**)**
Hierarchy	**-0.313(**)**	0.151(**)	-0.101(*)

InOut-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.396(**)	0.267(**)	**0.396(**)**
Ego ties	**0.389(**)**	0.236(**)	0.378(**)
Ego pairs	0.396(**)	0.267(**)	**0.396(**)**
Ego density	**-0.363(**)**	-0.282(**)	-0.334(**)
Ego reach	0.385(**)	0.180(**)	**0.403(**)**
Ego reach efficiency	**-0.307(**)**	-0.066	0.293(**)
Ego brokerage	**0.443(**)**	0.393(**)	0.431(**)
Ego normalized brokerage	**0.351(**)**	0.299(**)	0.278(**)
Ego betweeness	**0.436(**)**	0.383(**)	0.432(**)
Ego normalized betweeness	0.417(**)	0.400(**)	**0.424(**)**

In-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.303(**)	**0.325(**)**	0.277(**)
Ego ties	0.263(**)	**0.318(**)**	0.235(**)
Ego pairs	0.303(**)	**0.325(**)**	0.277(**)
Ego density	-0.249(**)	0.040	**-0.264(**)**
Ego reach	0.385(**)	0.180(**)	**0.403(**)**
Ego reach efficiency	-0.219(**)	-0.143(**)	**0.336(**)**

Out-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.430(**)	0.163(**)	**0.454(**)**
Ego ties	0.388(**)	0.117(**)	**0.446(**)**
Ego pairs	0.430(**)	0.163(**)	**0.454(**)**
Ego density	**-0.413(**)**	-0.253(**)	0.122(**)
Ego reach	0.385(**)	0.180(**)	**0.403(**)**
Ego reach efficiency	**-0.338(**)**	0.062	0.278(**)

(e) Netbeans 6.9 at Java JAR file level

Global Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Degree centrality	**0.506(**)**	0.502(**)	0.504(**)
Betweenness centrality	0.386(**)	0.324(**)	**0.407(**)**
Closeness centrality	0.423(**)	0.187(**)	**0.459(**)**
Information centrality	0.458(**)	0.344(**)	**0.469(**)**
Distance weighted reach	0.429(**)	0.202(**)	**0.458(**)**
Eigenvector centrality	**0.495(**)**	0.238(**)	0.449(**)
Effective size	0.447(**)	0.349(**)	**0.457(**)**
Efficiency	0.161(**)	0.233(**)	**-0.283(**)**
Constriant	**-0.384(**)**	0.317(**)	0.202(**)
Hierarchy	**-0.362(**)**	0.185(**)	-0.249(**)

InOut-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.432(**)	0.245(**)	**0.440(**)**
Ego ties	0.436(**)	0.225(**)	**0.446(**)**
Ego pairs	0.432(**)	0.245(**)	**0.440(**)**
Ego density	**-0.382(**)**	-0.260(**)	-0.307(**)
Ego reach	0.400(**)	0.253(**)	**0.464(**)**
Ego reach efficiency	**-0.424(**)**	0.034	0.159(**)
Ego brokerage	0.450(**)	0.319(**)	**0.453(**)**
Ego normalized brokerage	0.310(**)	0.234(**)	**0.314(**)**
Ego betweeness	0.437(**)	0.327(**)	**0.443(**)**
Ego normalized betweeness	0.393(**)	0.347(**)	**0.417(**)**

In-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	**0.331(**)**	0.292(**)	0.323(**)
Ego ties	0.304(**)	0.290(**)	**0.319(**)**
Ego pairs	**0.331(**)**	0.292(**)	0.323(**)
Ego density	**-0.225(**)**	0.046	-0.160(**)
Ego reach	0.400(**)	0.253(**)	**0.464(**)**
Ego reach efficiency	**-0.303(**)**	-0.083(*)	0.228(**)

Out-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.430(**)	0.194(**)	**0.465(**)**
Ego ties	0.401(**)	0.114(**)	**0.459(**)**
Ego pairs	0.430(**)	0.194(**)	**0.465(**)**
Ego density	**-0.400(**)**	-0.355(**)	0.165(**)
Ego reach	0.400(**)	0.253(**)	**0.464(**)**
Ego reach efficiency	**-0.422(**)**	0.163(**)	0.173(**)

(f) Netbeans 7.0 at Java JAR file level

Global Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Degree centrality	**0.507(**)**	0.502(**)	0.492(**)
Betweenness centrality	0.366(**)	0.360(**)	**0.438(**)**
Closeness centrality	0.259(**)	0.098(*)	**0.368(**)**
Information centrality	0.432(**)	0.365(**)	**0.461(**)**
Distance weighted reach	0.286(**)	0.136(**)	**0.376(**)**
Eigenvector centrality	0.441(**)	0.439(**)	**0.455(**)**
Effective size	0.414(**)	0.315(**)	**0.427(**)**
Efficiency	0.045	**0.064**	-0.017
Constriant	0.253(**)	0.325(**)	**0.376(**)**
Hierarchy	0.247(**)	**0.285(**)**	0.147(**)

InOut-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.396(**)	0.261(**)	**0.415(**)**
Ego ties	0.421(**)	0.267(**)	**0.425(**)**
Ego pairs	0.396(**)	0.261(**)	**0.415(**)**
Ego density	**-0.238(**)**	-0.185(**)	-0.008
Ego reach	0.301(**)	0.221(**)	**0.402(**)**
Ego reach efficiency	-0.010	-0.088(*)	**0.276(**)**
Ego brokerage	0.433(**)	0.349(**)	**0.465(**)**
Ego normalized brokerage	0.250(**)	0.287(**)	**0.358(**)**
Ego betweeness	0.412(**)	0.364(**)	**0.457(**)**
Ego normalized betweeness	0.393(**)	0.377(**)	**0.455(**)**

In-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.321(**)	0.285(**)	**0.365(**)**
Ego ties	0.315(**)	0.283(**)	**0.358(**)**
Ego pairs	0.321(**)	0.285(**)	**0.365(**)**
Ego density	-0.030	**0.105(*)**	0.045
Ego reach	0.301(**)	0.221(**)	**0.402(**)**
Ego reach efficiency	-0.019	-0.103(*)	**0.291(**)**

Out-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.396(**)	0.202(**)	**0.442(**)**
Ego ties	0.386(**)	0.181(**)	**0.442(**)**
Ego pairs	0.396(**)	0.202(**)	**0.442(**)**
Ego density	-0.239(**)	-0.150(**)	**0.247(**)**
Ego reach	0.301(**)	0.221(**)	**0.402(**)**
Ego reach efficiency	0.051	0.068	**0.331(**)**

Table A.3: Correlation of social network metrics for Gnome with number of post-release defects. Strong and Weak denote the *Strong network* and the *Weak network* respectively. Bold values indicate that the correlation on that network is higher than that on other networks. Correlation significant at 99% and 95% with Wilcoxon signed-rank tests are marked by (**) and (*) respectively.

(a) Gnome 2.16 at Ubuntu package level

Global Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Degree centrality	0.557(**)	0.545(**)	**0.558(**)**
Betweenness centrality	0.533(**)	0.498(**)	**0.600(**)**
Closeness centrality	0.490(**)	0.228(**)	**0.617(**)**
Information centrality	0.537(**)	0.339(**)	**0.617(**)**
Distance weighted reach	0.497(**)	0.253(**)	**0.617(**)**
Eigenvector centrality	0.555(**)	0.539(**)	**0.603(**)**
Effective size	0.555(**)	0.531(**)	**0.557(**)**
Efficiency	**-0.165(*)**	-0.011	0.025
Constriant	-0.247(**)	-0.188(*)	**0.422(**)**
Hierarchy	**-0.473(**)**	-0.426(**)	0.397(**)

InOut-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	**0.557(**)**	0.546(**)	0.553(**)
Ego ties	0.547(**)	0.532(**)	**0.559(**)**
Ego pairs	**0.557(**)**	0.546(**)	0.553(**)
Ego density	-0.543(**)	**-0.550(**)**	-0.083
Ego reach	0.522(**)	0.276(**)	**0.618(**)**
Ego reach efficiency	-0.391(**)	**-0.439(**)**	0.361(**)
Ego brokerage	0.554(**)	0.537(**)	**0.617(**)**
Ego normalized brokerage	0.112	0.117	**0.446(**)**
Ego betweeness	0.546(**)	0.537(**)	**0.613(**)**
Ego normalized betweeness	0.538(**)	0.511(**)	**0.598(**)**

In-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.557(**)	**0.570(**)**	0.533(**)
Ego ties	0.552(**)	**0.565(**)**	0.533(**)
Ego pairs	0.557(**)	**0.570(**)**	0.533(**)
Ego density	**-0.210(**)**	-0.196(*)	-0.061
Ego reach	0.522(**)	0.276(**)	**0.618(**)**
Ego reach efficiency	-0.441(**)	**-0.458(**)**	0.401(**)

Out-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.546(**)	0.367(**)	**0.615(**)**
Ego ties	0.531(**)	0.337(**)	**0.614(**)**
Ego pairs	0.546(**)	0.367(**)	**0.615(**)**
Ego density	**-0.489(**)**	-0.365(**)	0.402(**)
Ego reach	0.522(**)	0.276(**)	**0.618(**)**
Ego reach efficiency	-0.375(**)	-0.119	**0.388(**)**

(b) Gnome 2.18 at Ubuntu package level

Global Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Degree centrality	0.538(**)	0.507(**)	**0.545(**)**
Betweenness centrality	0.476(**)	0.468(**)	**0.583(**)**
Closeness centrality	0.469(**)	0.158(*)	**0.587(**)**
Information centrality	-0.306(**)	0.360(**)	**0.605(**)**
Distance weighted reach	0.474(**)	0.173(*)	**0.591(**)**
Eigenvector centrality	0.512(**)	0.488(**)	**0.602(**)**
Effective size	0.531(**)	0.504(**)	**0.544(**)**
Efficiency	**-0.279(**)**	-0.192(*)	0.036
Constriant	-0.341(**)	-0.268(**)	**0.486(**)**
Hierarchy	**-0.487(**)**	-0.411(**)	0.314(**)

InOut-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	**0.546(**)**	0.520(**)	0.540(**)
Ego ties	0.534(**)	0.503(**)	**0.546(**)**
Ego pairs	**0.546(**)**	0.520(**)	0.540(**)
Ego density	-0.513(**)	**-0.515(**)**	-0.124
Ego reach	0.503(**)	0.212(**)	**0.602(**)**
Ego reach efficiency	-0.217(**)	**-0.370(**)**	0.369(**)
Ego brokerage	0.533(**)	0.494(**)	**0.616(**)**
Ego normalized brokerage	-0.008	0.070	**0.484(**)**
Ego betweeness	0.530(**)	0.495(**)	**0.613(**)**
Ego normalized betweeness	0.513(**)	0.461(**)	**0.601(**)**

In-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.529(**)	0.527(**)	**0.532(**)**
Ego ties	0.527(**)	**0.530(**)**	0.526(**)
Ego pairs	0.529(**)	0.527(**)	**0.532(**)**
Ego density	-0.325(**)	**-0.341(**)**	-0.090
Ego reach	0.503(**)	0.212(**)	**0.602(**)**
Ego reach efficiency	-0.377(**)	-0.387(**)	**0.387(**)**

Out-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.536(**)	0.305(**)	**0.603(**)**
Ego ties	0.515(**)	0.282(**)	**0.605(**)**
Ego pairs	0.536(**)	0.305(**)	**0.603(**)**
Ego density	**-0.442(**)**	-0.267(**)	0.422(**)
Ego reach	0.503(**)	0.212(**)	**0.602(**)**
Ego reach efficiency	-0.199(**)	-0.078	**0.417(**)**

(c) Gnome 2.20 at Ubuntu package level

Global Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Degree centrality	**0.645(**)**	0.616(**)	0.642(**)
Betweenness centrality	0.600(**)	0.579(**)	**0.674(**)**
Closeness centrality	0.531(**)	0.259(**)	**0.677(**)**
Information centrality	-0.374(**)	0.423(**)	**0.680(**)**
Distance weighted reach	0.551(**)	0.264(**)	**0.679(**)**
Eigenvector centrality	0.623(**)	0.595(**)	**0.681(**)**
Effective size	**0.642(**)**	0.614(**)	0.641(**)
Efficiency	**-0.265(**)**	-0.121	0.103
Constriant	-0.296(**)	-0.188(*)	**0.507(**)**
Hierarchy	**-0.529(**)**	-0.408(**)	0.408(**)

InOut-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	**0.651(**)**	0.639(**)	0.649(**)
Ego ties	0.640(**)	0.613(**)	**0.641(**)**
Ego pairs	**0.651(**)**	0.639(**)	0.649(**)
Ego density	-0.619(**)	**-0.642(**)**	-0.088
Ego reach	0.577(**)	0.323(**)	**0.684(**)**
Ego reach efficiency	-0.216(**)	**-0.412(**)**	0.402(**)
Ego brokerage	0.647(**)	0.610(**)	**0.696(**)**
Ego normalized brokerage	0.102	0.225(**)	**0.539(**)**
Ego betweeness	0.643(**)	0.608(**)	**0.689(**)**
Ego normalized betweeness	0.629(**)	0.582(**)	**0.671(**)**

In-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	**0.642(**)**	0.641(**)	0.631(**)
Ego ties	**0.636(**)**	0.634(**)	0.631(**)
Ego pairs	**0.642(**)**	0.641(**)	0.631(**)
Ego density	**-0.273(**)**	-0.245(**)	-0.055
Ego reach	0.577(**)	0.323(**)	**0.684(**)**
Ego reach efficiency	-0.416(**)	**-0.464(**)**	0.436(**)

Out-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.641(**)	0.398(**)	**0.691(**)**
Ego ties	0.627(**)	0.375(**)	**0.689(**)**
Ego pairs	0.641(**)	0.398(**)	**0.691(**)**
Ego density	**-0.538(**)**	-0.370(**)	0.487(**)
Ego reach	0.577(**)	0.323(**)	**0.684(**)**
Ego reach efficiency	-0.206(**)	-0.043	**0.472(**)**

(d) Gnome 2.22 at Ubuntu package level

Global Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Degree centrality	0.598(**)	0.568(**)	**0.610(**)**
Betweenness centrality	0.568(**)	0.512(**)	**0.695(**)**
Closeness centrality	0.439(**)	-0.139	**0.667(**)**
Information centrality	0.577(**)	0.305(**)	**0.702(**)**
Distance weighted reach	0.227(**)	-0.160(*)	**0.668(**)**
Eigenvector centrality	0.594(**)	0.570(**)	**0.697(**)**
Effective size	0.601(**)	0.571(**)	**0.609(**)**
Efficiency	**-0.209(**)**	-0.099	0.104
Constriant	-0.329(**)	-0.216(**)	**0.571(**)**
Hierarchy	**-0.524(**)**	-0.406(**)	0.395(**)

InOut-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.603(**)	0.584(**)	**0.606(**)**
Ego ties	0.591(**)	0.560(**)	**0.608(**)**
Ego pairs	0.603(**)	0.584(**)	**0.606(**)**
Ego density	**-0.585(**)**	-0.583(**)	-0.107
Ego reach	0.535(**)	0.393(**)	**0.695(**)**
Ego reach efficiency	-0.215(**)	-0.334(**)	**0.466(**)**
Ego brokerage	0.598(**)	0.575(**)	**0.707(**)**
Ego normalized brokerage	0.083	0.221(**)	**0.572(**)**
Ego betweeness	0.592(**)	0.576(**)	**0.703(**)**
Ego Normalized betweeness	0.574(**)	0.562(**)	**0.693(**)**

In-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.596(**)	**0.607(**)**	0.594(**)
Ego ties	0.590(**)	**0.608(**)**	0.587(**)
Ego pairs	0.596(**)	**0.607(**)**	0.594(**)
Ego density	**-0.306(**)**	-0.250(**)	-0.077
Ego reach	0.535(**)	0.393(**)	**0.695(**)**
Ego reach efficiency	-0.413(**)	-0.398(**)	**0.486(**)**

Out-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.595(**)	0.417(**)	**0.691(**)**
Ego ties	0.578(**)	0.389(**)	**0.694(**)**
Ego pairs	0.595(**)	0.417(**)	**0.691(**)**
Ego density	**-0.570(**)**	-0.446(**)	0.479(**)
Ego reach	0.535(**)	0.393(**)	**0.695(**)**
Ego reach efficiency	-0.212(**)	0.023	**0.498(**)**

(e) Gnome 2.24 at Ubuntu package level

Global Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Degree centrality	0.645(**)	0.614(**)	**0.647(**)**
Betweenness centrality	0.545(**)	0.518(**)	**0.639(**)**
Closeness centrality	0.582(**)	0.329(**)	**0.644(**)**
Information centrality	-0.246(**)	-0.423(**)	**0.662(**)**
Distance weighted reach	0.602(**)	0.337(**)	**0.647(**)**
Eigenvector centrality	0.625(**)	0.581(**)	**0.647(**)**
Effective size	**0.646(**)**	0.615(**)	0.644(**)
Efficiency	**-0.295(**)**	-0.174(*)	0.082
Constriant	-0.377(**)	-0.225(**)	**0.518(**)**
Hierarchy	**-0.574(**)**	-0.440(**)	0.452(**)

InOut-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	**0.654(**)**	0.639(**)	0.647(**)
Ego ties	0.650(**)	0.614(**)	**0.655(**)**
Ego pairs	**0.654(**)**	0.639(**)	0.647(**)
Ego density	-0.627(**)	**-0.648(**)**	-0.118
Ego reach	0.596(**)	0.361(**)	**0.666(**)**
Ego reach efficiency	-0.212(**)	**-0.390(**)**	0.351(**)
Ego brokerage	0.649(**)	0.621(**)	**0.673(**)**
Ego normalized brokerage	0.061	0.253(**)	**0.493(**)**
Ego betweeness	0.640(**)	0.615(**)	**0.669(**)**
Ego normalized betweeness	0.623(**)	0.567(**)	**0.653(**)**

In-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.636(**)	**0.641(**)**	0.641(**)
Ego ties	0.638(**)	0.642(**)	**0.649(**)**
Ego pairs	0.636(**)	**0.641(**)**	0.641(**)
Ego density	**-0.360(**)**	-0.254(**)	-0.099
Ego reach	0.596(**)	0.361(**)	**0.666(**)**
Ego reach efficiency	-0.383(**)	**-0.425(**)**	0.366(**)

Out-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.644(**)	0.408(**)	**0.667(**)**
Ego ties	0.634(**)	0.392(**)	**0.668(**)**
Ego pairs	0.644(**)	0.408(**)	**0.667(**)**
Ego density	**-0.550(**)**	-0.370(**)	0.435(**)
Ego reach	0.596(**)	0.361(**)	**0.666(**)**
Ego reach efficiency	-0.201(**)	0.048	**0.444(**)**

(f) Gnome 2.26 at Ubuntu package level

Global Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Degree centrality	0.616(**)	0.583(**)	**0.619(**)**
Betweenness centrality	0.489(**)	0.446(**)	**0.620(**)**
Closeness centrality	0.379(**)	-0.163(*)	**0.622(**)**
Information centrality	0.588(**)	0.270(**)	**0.662(**)**
Distance weighted reach	0.400(**)	-0.161(*)	**0.626(**)**
Eigenvector centrality	0.591(**)	0.534(**)	**0.663(**)**
Effective size	0.616(**)	0.582(**)	**0.619(**)**
Efficiency	**-0.267(**)**	-0.198(**)	0.028
Constriant	-0.453(**)	-0.392(**)	**0.525(**)**
Hierarchy	**-0.575(**)**	-0.565(**)	0.293(**)

InOut-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	**0.623(**)**	0.615(**)	0.618(**)
Ego ties	0.608(**)	0.588(**)	**0.612(**)**
Ego pairs	**0.623(**)**	0.615(**)	0.618(**)
Ego density	-0.607(**)	**-0.618(**)**	-0.257(**)
Ego reach	0.561(**)	0.274(**)	**0.645(**)**
Ego reach efficiency	-0.260(**)	**-0.426(**)**	0.341(**)
Ego brokerage	0.618(**)	0.570(**)	**0.670(**)**
Ego normalized brokerage	0.026	0.100	**0.500(**)**
Ego betweeness	0.612(**)	0.567(**)	**0.667(**)**
Ego normalized betweeness	0.592(**)	0.509(**)	**0.658(**)**

In-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.613(**)	**0.621(**)**	0.613(**)
Ego ties	0.609(**)	**0.624(**)**	0.612(**)
Ego pairs	0.613(**)	**0.621(**)**	0.613(**)
Ego density	**-0.412(**)**	-0.398(**)	-0.210(**)
Ego reach	0.561(**)	0.274(**)	**0.645(**)**
Ego reach efficiency	-0.402(**)	**-0.462(**)**	0.365(**)

Out-neighbourhood Ego Metrics			
Metric	**Socio-technical**	**Strong**	**Weak**
Ego size	0.605(**)	0.346(**)	**0.652(**)**
Ego ties	0.590(**)	0.324(**)	**0.645(**)**
Ego pairs	0.605(**)	0.346(**)	**0.652(**)**
Ego density	**-0.550(**)**	-0.334(**)	0.356(**)
Ego reach	0.561(**)	0.274(**)	**0.645(**)**
Ego reach efficiency	-0.258(**)	-0.046	**0.385(**)**

Appendix B

Cross Validation of Eclipse, Netbeans and Gnome Logistic Models

Table B.1: Results of repeated 10-fold cross validation of Eclipse. Given data from current release, predict whether a Eclipse Java package is defect-prone in current release. Bold values indicate that they are higher than values calculated on other networks.

(a) Predict top **10%** defect-prone Eclipse Java packages

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
2.0	Socio-technical	0.547	0.274	0.341	0.896	0.417
	Weak	**0.610**	**0.352**	**0.417**	0.902	0.437
	Strong	0.406	0.178	0.233	0.871	0.335
	Weak+Strong	0.589	0.342	0.403	**0.913**	**0.475**
2.1	Socio-technical	0.678	**0.439**	0.500	**0.909**	**0.490**
	Weak	0.689	0.374	0.458	0.863	0.388
	Strong	**0.752**	0.415	**0.510**	0.887	0.461
	Weak+Strong	0.690	0.419	0.496	0.905	0.489
3.0	Socio-technical	**0.793**	0.436	0.531	0.910	0.471
	Weak	0.791	**0.494**	**0.582**	0.907	0.503
	Strong	0.617	0.371	0.438	0.923	0.477
	Weak+Strong	0.722	0.486	0.561	**0.930**	**0.559**
3.1	Socio-technical	0.772	0.468	0.559	0.929	0.553
	Weak	**0.831**	0.453	0.565	0.913	0.505
	Strong	0.740	0.390	0.488	0.917	0.511
	Weak+Strong	0.742	**0.518**	**0.591**	**0.948**	**0.613**
3.2	Socio-technical	0.720	0.461	0.545	0.935	0.561
	Weak	0.726	0.386	0.484	0.910	0.490
	Strong	0.746	0.420	0.521	0.904	0.474
	Weak+Strong	**0.801**	**0.518**	**0.610**	**0.939**	**0.576**
3.3	Socio-technical	0.718	**0.413**	0.510	0.881	0.444
	Weak	**0.760**	0.327	0.442	0.863	0.411
	Strong	0.743	0.349	0.462	0.864	0.400
	Weak+Strong	0.741	0.402	**0.510**	**0.893**	**0.477**
3.4	Socio-technical	0.767	0.356	0.475	0.845	0.383
	Weak	0.738	0.316	0.433	0.818	0.325
	Strong	**0.825**	0.349	0.479	0.845	0.381
	Weak+Strong	0.788	**0.449**	**0.561**	**0.876**	**0.454**

(b) Predict top **15%** defect-prone Eclipse Java packages

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
2.0	Socio-technical	0.722	0.418	0.500	0.884	0.458
	Weak	0.729	0.412	0.497	0.888	0.486
	Strong	0.639	0.332	0.409	0.877	0.401
	Weak+Strong	**0.765**	**0.524**	**0.595**	**0.909**	**0.549**
2.1	Socio-technical	**0.798**	**0.478**	**0.573**	0.883	0.481
	Weak	0.732	0.416	0.504	0.853	0.406
	Strong	0.752	0.391	0.496	0.870	0.441
	Weak+Strong	0.768	0.439	0.535	**0.893**	**0.488**
3.0	Socio-technical	0.746	0.512	0.588	0.929	0.558
	Weak	**0.787**	0.490	0.585	0.907	0.514
	Strong	0.731	0.494	0.574	0.927	0.542
	Weak+Strong	0.748	**0.573**	**0.636**	**0.941**	**0.619**
3.1	Socio-technical	0.695	0.455	0.534	0.914	0.524
	Weak	**0.773**	0.423	0.527	0.888	0.478
	Strong	0.683	0.399	0.485	0.900	0.492
	Weak+Strong	0.760	**0.546**	**0.621**	**0.925**	**0.573**
3.2	Socio-technical	**0.807**	0.543	**0.638**	0.921	0.564
	Weak	0.781	0.506	0.604	0.891	0.495
	Strong	0.804	0.441	0.560	0.889	0.477
	Weak+Strong	0.777	**0.547**	0.630	**0.923**	**0.580**
3.3	Socio-technical	0.756	0.481	0.579	0.877	0.473
	Weak	**0.841**	0.433	0.562	0.836	0.396
	Strong	0.712	0.385	0.492	0.857	0.413
	Weak+Strong	0.747	**0.496**	**0.588**	**0.890**	**0.498**
3.4	Socio-technical	0.771	0.359	0.478	0.845	0.383
	Weak	0.747	0.316	0.433	0.817	0.325
	Strong	**0.825**	0.351	0.482	0.845	0.381
	Weak+Strong	0.785	**0.445**	**0.559**	**0.876**	**0.454**

(c) Predict top **20%** defect-prone Eclipse Java packages

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
2.0	Socio-technical	0.732	0.569	0.624	0.891	0.506
	Weak	0.753	0.527	0.601	0.869	0.476
	Strong	0.748	0.470	0.560	0.852	0.394
	Weak+Strong	**0.812**	**0.583**	**0.664**	**0.901**	**0.570**
2.1	Socio-technical	0.765	**0.473**	**0.566**	**0.889**	**0.515**
	Weak	0.738	0.455	0.540	0.863	0.448
	Strong	**0.774**	0.430	0.532	0.861	0.453
	Weak+Strong	0.743	0.462	0.551	0.886	0.495
3.0	Socio-technical	0.797	0.558	0.644	0.916	0.569
	Weak	0.793	0.526	0.619	0.899	0.544
	Strong	0.757	0.519	0.601	0.909	0.549
	Weak+Strong	**0.841**	**0.621**	**0.704**	**0.934**	**0.621**
3.1	Socio-technical	0.724	0.497	0.579	0.895	0.521
	Weak	0.792	0.493	0.596	0.883	0.503
	Strong	0.747	0.459	0.558	0.886	0.484
	Weak+Strong	**0.815**	**0.615**	**0.691**	**0.915**	**0.575**
3.2	Socio-technical	0.748	0.531	0.615	0.883	0.502
	Weak	0.784	0.531	0.626	0.867	0.462
	Strong	0.775	0.472	0.579	0.861	0.456
	Weak+Strong	**0.795**	**0.578**	**0.663**	**0.897**	**0.544**
3.3	Socio-technical	0.754	0.481	0.580	0.878	0.473
	Weak	**0.836**	0.434	0.564	0.837	0.396
	Strong	0.713	0.386	0.493	0.857	0.413
	Weak+Strong	0.750	**0.495**	**0.590**	**0.889**	**0.498**
3.4	Socio-technical	0.738	0.429	0.538	0.802	0.348
	Weak	0.700	0.419	0.519	0.773	0.298
	Strong	0.722	0.446	0.546	0.791	0.321
	Weak+Strong	**0.740**	**0.513**	**0.602**	**0.815**	**0.382**

(d) Predict top **25%** defect-prone Eclipse Java packages

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
2.0	Socio-technical	0.730	0.561	0.619	0.886	0.506
	Weak	0.747	0.525	0.600	0.865	0.476
	Strong	0.755	0.474	0.564	0.854	0.394
	Weak+Strong	**0.825**	**0.590**	**0.673**	**0.904**	**0.570**
2.1	Socio-technical	0.751	**0.565**	**0.636**	**0.863**	**0.472**
	Weak	0.738	0.448	0.545	0.836	0.388
	Strong	0.755	0.492	0.581	0.827	0.409
	Weak+Strong	**0.757**	0.529	0.612	0.853	0.448
3.0	Socio-technical	**0.831**	0.598	0.688	0.881	0.508
	Weak	0.787	0.598	0.672	0.869	0.500
	Strong	0.794	0.573	0.658	0.876	0.516
	Weak+Strong	0.802	**0.652**	**0.715**	**0.900**	**0.586**
3.1	Socio-technical	0.795	0.654	0.712	0.899	0.554
	Weak	0.775	0.588	0.664	0.871	0.505
	Strong	0.786	0.582	0.663	0.869	0.489
	Weak+Strong	**0.811**	**0.664**	**0.723**	**0.903**	**0.579**
3.2	Socio-technical	0.778	**0.687**	**0.727**	0.844	0.452
	Weak	0.774	0.627	0.690	0.815	0.394
	Strong	0.754	0.610	0.671	0.814	0.374
	Weak+Strong	**0.791**	0.661	0.717	**0.846**	**0.463**
3.3	Socio-technical	0.729	0.572	0.638	0.812	0.377
	Weak	0.719	0.515	0.596	0.778	0.307
	Strong	0.728	0.590	0.648	0.805	0.347
	Weak+Strong	**0.747**	**0.622**	**0.676**	**0.822**	**0.391**
3.4	Socio-technical	0.737	0.430	0.538	0.802	0.348
	Weak	0.701	0.419	0.519	0.774	0.298
	Strong	0.725	0.446	0.547	0.791	0.321
	Weak+Strong	**0.740**	**0.515**	**0.602**	**0.817**	**0.382**

(e) Predict top **30%** defect-prone Eclipse Java packages

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
2.0	Socio-technical	0.714	0.526	0.592	0.855	0.440
	Weak	**0.782**	**0.546**	**0.629**	0.851	0.447
	Strong	0.780	0.489	0.587	0.836	0.354
	Weak+Strong	0.774	0.537	0.621	**0.873**	**0.504**
2.1	Socio-technical	0.751	**0.570**	**0.637**	**0.864**	**0.472**
	Weak	0.742	0.454	0.549	0.835	0.388
	Strong	0.747	0.487	0.579	0.823	0.409
	Weak+Strong	**0.754**	0.527	0.608	0.856	0.448
3.0	Socio-technical	**0.832**	0.598	0.689	0.882	0.508
	Weak	0.789	0.597	0.673	0.870	0.500
	Strong	0.798	0.573	0.660	0.877	0.516
	Weak+Strong	0.807	**0.656**	**0.717**	**0.901**	**0.586**
3.1	Socio-technical	0.792	0.654	0.711	0.898	0.554
	Weak	0.777	0.588	0.663	0.872	0.505
	Strong	0.785	0.579	0.660	0.867	0.489
	Weak+Strong	**0.811**	**0.666**	**0.726**	**0.905**	**0.579**
3.2	Socio-technical	0.777	**0.686**	**0.726**	0.844	0.452
	Weak	0.772	0.626	0.689	0.817	0.394
	Strong	0.754	0.609	0.670	0.813	0.374
	Weak+Strong	**0.791**	0.661	0.718	**0.845**	**0.463**
3.3	Socio-technical	0.727	0.571	0.637	0.813	0.377
	Weak	0.719	0.517	0.598	0.779	0.307
	Strong	0.730	0.590	0.650	0.804	0.347
	Weak+Strong	**0.745**	**0.620**	**0.674**	**0.820**	**0.391**
3.4	Socio-technical	0.738	0.429	0.537	0.802	0.348
	Weak	0.701	0.420	0.518	0.775	0.298
	Strong	0.723	0.446	0.547	0.792	0.321
	Weak+Strong	**0.738**	**0.512**	**0.600**	**0.816**	**0.382**

Table B.2: Results of repeated 10-fold cross validation of Netbeans. Given data from current release, predict whether a Netbeans Java JAR file is defect-prone in current release. Bold values indicate that they are higher than values calculated on other networks.

(a) Predict top **10%** defect-prone Netbeans Java JAR files

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
6.1	Socio-technical	0.530	0.266	0.332	0.893	0.403
	Weak	0.588	0.280	0.362	0.905	0.449
	Strong	0.616	0.292	0.369	0.874	0.373
	Weak+Strong	**0.674**	**0.479**	**0.536**	**0.919**	**0.495**
6.5	Socio-technical	0.670	0.414	0.498	0.841	0.386
	Weak	0.658	0.399	0.485	0.830	0.356
	Strong	0.606	0.388	0.459	0.836	0.385
	Weak+Strong	**0.679**	**0.471**	**0.546**	**0.854**	**0.421**
6.7	Socio-technical	**0.716**	0.354	0.459	0.846	0.369
	Weak	0.678	0.325	0.424	0.828	0.357
	Strong	0.699	0.384	0.475	0.842	0.365
	Weak+Strong	0.685	**0.405**	**0.489**	**0.848**	**0.385**
6.8	Socio-technical	0.679	0.397	0.487	0.865	0.416
	Weak	0.691	0.356	0.455	0.836	0.361
	Strong	0.655	0.320	0.412	0.843	0.360
	Weak+Strong	**0.727**	**0.468**	**0.549**	**0.884**	**0.459**
6.9	Socio-technical	0.722	0.451	0.537	0.899	0.477
	Weak	0.692	0.329	0.428	0.871	0.393
	Strong	**0.726**	0.356	0.458	0.873	0.403
	Weak+Strong	0.722	**0.488**	**0.561**	**0.910**	**0.511**
7.0	Socio-technical	0.684	0.397	0.488	0.862	0.418
	Weak	0.654	0.393	0.480	0.833	0.367
	Strong	**0.699**	0.352	0.450	0.853	0.385
	Weak+Strong	0.699	**0.428**	**0.517**	**0.884**	**0.461**

(b) Predict top **15%** defect-prone Netbeans Java JAR files

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
6.1	Socio-technical	0.660	0.383	0.468	0.867	0.401
	Weak	0.603	0.290	0.377	0.858	0.391
	Strong	0.635	0.347	0.430	0.850	0.369
	Weak+Strong	**0.661**	**0.422**	**0.501**	**0.889**	**0.461**
6.5	Socio-technical	0.673	0.415	0.502	0.841	0.386
	Weak	0.668	0.404	0.490	0.831	0.356
	Strong	0.602	0.390	0.460	0.836	0.385
	Weak+Strong	**0.689**	**0.472**	**0.549**	**0.854**	**0.421**
6.7	Socio-technical	0.704	0.351	0.452	0.845	0.369
	Weak	0.685	0.333	0.431	0.829	0.357
	Strong	**0.710**	0.388	0.483	0.841	0.365
	Weak+Strong	0.684	**0.407**	**0.495**	**0.846**	**0.385**
6.8	Socio-technical	0.681	0.400	0.488	0.867	0.416
	Weak	0.671	0.354	0.445	0.834	0.361
	Strong	0.643	0.323	0.408	0.846	0.360
	Weak+Strong	**0.714**	**0.460**	**0.544**	**0.883**	**0.459**
6.9	Socio-technical	0.764	0.434	0.543	0.852	0.433
	Weak	0.759	0.392	0.506	0.830	0.375
	Strong	0.764	0.424	0.534	0.830	0.386
	Weak+Strong	**0.777**	**0.482**	**0.585**	**0.856**	**0.445**
7.0	Socio-technical	0.685	0.394	0.487	0.860	0.418
	Weak	0.672	0.398	0.486	0.834	0.367
	Strong	**0.701**	0.353	0.454	0.853	0.385
	Weak+Strong	0.690	**0.418**	**0.506**	**0.881**	**0.461**

(c) Predict top **20%** defect-prone Netbeans Java JAR files

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
6.1	Socio-technical	0.719	**0.599**	**0.646**	0.830	0.388
	Weak	**0.732**	0.570	0.634	0.798	0.332
	Strong	0.649	0.572	0.601	0.805	0.358
	Weak+Strong	0.704	0.583	0.632	**0.846**	**0.432**
6.5	Socio-technical	0.674	0.416	0.505	0.841	0.386
	Weak	0.652	0.396	0.482	0.828	0.356
	Strong	0.609	0.391	0.465	0.837	0.385
	Weak+Strong	**0.681**	**0.472**	**0.546**	**0.853**	**0.421**
6.7	Socio-technical	**0.724**	0.353	0.454	**0.847**	0.369
	Weak	0.667	0.334	0.430	0.827	0.357
	Strong	0.695	0.380	0.475	0.841	0.365
	Weak+Strong	0.698	**0.405**	**0.497**	0.847	**0.385**
6.8	Socio-technical	0.681	0.401	0.486	0.866	0.416
	Weak	0.682	0.356	0.453	0.834	0.361
	Strong	0.643	0.319	0.410	0.844	0.360
	Weak+Strong	**0.718**	**0.460**	**0.547**	**0.883**	**0.459**
6.9	Socio-technical	0.764	0.436	0.545	0.855	0.433
	Weak	0.761	0.393	0.506	0.830	0.375
	Strong	0.758	0.422	0.531	0.830	0.386
	Weak+Strong	**0.776**	**0.479**	**0.582**	**0.857**	**0.445**
7.0	Socio-technical	0.688	0.396	0.489	0.860	0.418
	Weak	0.667	0.398	0.486	0.834	0.367
	Strong	**0.694**	0.353	0.454	0.852	0.385
	Weak+Strong	0.688	**0.426**	**0.510**	**0.883**	**0.461**

(d) Predict top **25%** defect-prone Netbeans Java JAR files

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
6.1	Socio-technical	0.721	**0.597**	**0.646**	0.831	0.388
	Weak	**0.733**	0.567	0.633	0.798	0.332
	Strong	0.651	0.577	0.603	0.805	0.358
	Weak+Strong	0.704	0.585	0.632	**0.846**	**0.432**
6.5	Socio-technical	0.672	0.419	0.504	0.843	0.386
	Weak	0.666	0.404	0.489	0.829	0.356
	Strong	0.600	0.389	0.462	0.837	0.385
	Weak+Strong	**0.681**	**0.476**	**0.549**	**0.856**	**0.421**
6.7	Socio-technical	**0.707**	0.355	0.456	**0.848**	0.369
	Weak	0.672	0.327	0.425	0.825	0.357
	Strong	0.698	0.380	0.475	0.840	0.365
	Weak+Strong	0.676	**0.406**	**0.492**	0.847	**0.385**
6.8	Socio-technical	0.688	0.399	0.490	0.864	0.416
	Weak	0.683	0.360	0.454	0.837	0.361
	Strong	0.639	0.313	0.404	0.844	0.360
	Weak+Strong	**0.713**	**0.468**	**0.549**	**0.885**	**0.459**
6.9	Socio-technical	0.770	0.439	0.548	0.854	0.433
	Weak	0.766	0.397	0.510	0.830	0.375
	Strong	0.769	0.425	0.537	0.830	0.386
	Weak+Strong	**0.773**	**0.480**	**0.582**	**0.856**	**0.445**
7.0	Socio-technical	0.684	0.395	0.488	0.860	0.418
	Weak	0.673	0.393	0.481	0.832	0.367
	Strong	**0.702**	0.352	0.455	0.851	0.385
	Weak+Strong	0.692	**0.420**	**0.510**	**0.881**	**0.461**

(e) Predict top **30%** defect-prone Netbeans Java JAR files

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
6.1	Socio-technical	0.719	**0.596**	**0.644**	0.827	0.388
	Weak	**0.735**	0.568	0.634	0.799	0.332
	Strong	0.646	0.575	0.600	0.807	0.358
	Weak+Strong	0.706	0.585	0.634	**0.846**	**0.432**
6.5	Socio-technical	0.666	0.411	0.498	0.842	0.386
	Weak	0.665	0.399	0.485	0.828	0.356
	Strong	0.606	0.397	0.468	0.837	0.385
	Weak+Strong	**0.683**	**0.474**	**0.548**	**0.857**	**0.421**
6.7	Socio-technical	**0.698**	0.347	0.448	0.845	0.369
	Weak	0.668	0.328	0.426	0.828	0.357
	Strong	0.688	0.379	0.471	0.840	0.365
	Weak+Strong	0.685	**0.406**	**0.494**	**0.848**	**0.385**
6.8	Socio-technical	0.674	0.396	0.485	0.865	0.416
	Weak	0.687	0.358	0.454	0.838	0.361
	Strong	0.617	0.312	0.399	0.842	0.360
	Weak+Strong	**0.714**	**0.461**	**0.546**	**0.880**	**0.459**
6.9	Socio-technical	0.769	0.436	0.548	0.852	0.433
	Weak	0.764	0.394	0.509	0.829	0.375
	Strong	0.765	0.425	0.536	0.830	0.386
	Weak+Strong	**0.779**	**0.481**	**0.586**	**0.857**	**0.445**
7.0	Socio-technical	0.686	0.396	0.488	0.861	0.418
	Weak	0.673	0.396	0.485	0.832	0.367
	Strong	**0.702**	0.353	0.454	0.851	0.385
	Weak+Strong	0.691	**0.426**	**0.513**	**0.883**	**0.461**

Table B.3: Results of repeated 10-fold cross validation of Gnome. Given data from current release, predict whether an Gnome Ubuntu package is defect-prone in current release. Bold values indicate that they are higher than values calculated on other networks.

(a) Predict top **10%** defect-prone Gnome Ubuntu packages

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
2.16	Socio-technical	**0.569**	**0.450**	**0.482**	0.921	0.507
	Weak	0.412	0.308	0.331	0.905	0.451
	Strong	0.468	0.410	0.408	0.928	0.509
	Weak+Strong	0.488	0.394	0.417	**0.935**	**0.553**
2.18	Socio-technical	0.568	0.523	0.513	0.941	0.582
	Weak	0.570	0.446	0.479	0.940	0.541
	Strong	0.566	0.492	0.502	0.934	0.519
	Weak+Strong	**0.628**	**0.549**	**0.566**	**0.961**	**0.660**
2.20	Socio-technical	0.559	0.504	0.502	0.944	0.570
	Weak	0.645	0.567	0.581	0.946	0.581
	Strong	0.662	0.553	0.579	0.958	0.646
	Weak+Strong	**0.687**	**0.618**	**0.634**	**0.966**	**0.732**
2.22	Socio-technical	**0.741**	**0.659**	**0.674**	**0.979**	**0.709**
	Weak	0.522	0.497	0.477	0.946	0.559
	Strong	0.572	0.485	0.497	0.955	0.616
	Weak+Strong	0.632	0.550	0.566	0.972	0.706
2.24	Socio-technical	0.640	0.548	0.565	0.969	0.667
	Weak	0.686	0.602	0.605	0.951	0.628
	Strong	0.746	0.654	0.673	0.982	0.755
	Weak+Strong	**0.840**	**0.786**	**0.799**	**0.992**	**0.862**
2.26	Socio-technical	0.686	**0.640**	**0.637**	0.978	0.715
	Weak	**0.695**	0.631	0.630	0.979	0.674
	Strong	0.671	0.561	0.590	0.977	0.725
	Weak+Strong	0.687	0.621	0.624	**0.991**	**0.799**

(b) Predict top **15%** defect-prone Gnome Ubuntu packages

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
2.16	Socio-technical	0.484	0.340	0.373	**0.851**	0.393
	Weak	0.480	0.308	0.355	0.828	0.325
	Strong	0.568	0.410	0.455	0.826	0.375
	Weak+Strong	**0.609**	**0.427**	**0.473**	0.848	**0.404**
2.18	Socio-technical	**0.677**	0.489	0.542	0.878	0.455
	Weak	0.610	0.428	0.472	0.851	0.426
	Strong	0.618	0.486	0.517	0.897	0.463
	Weak+Strong	0.667	**0.501**	**0.546**	**0.903**	**0.501**
2.20	Socio-technical	0.805	0.658	0.704	0.943	0.642
	Weak	0.731	0.599	0.636	0.947	0.654
	Strong	**0.828**	0.681	**0.724**	0.944	0.684
	Weak+Strong	0.783	**0.685**	0.707	**0.962**	**0.732**
2.22	Socio-technical	0.625	0.520	0.542	0.923	0.563
	Weak	0.695	0.508	0.557	0.920	0.571
	Strong	0.654	0.522	0.551	0.929	0.600
	Weak+Strong	**0.704**	**0.556**	**0.597**	**0.949**	**0.652**
2.24	Socio-technical	0.821	0.701	0.727	0.977	0.773
	Weak	0.763	0.663	0.688	0.964	0.720
	Strong	0.767	0.728	0.722	0.976	0.775
	Weak+Strong	**0.891**	**0.842**	**0.848**	**0.979**	**0.811**
2.26	Socio-technical	**0.713**	0.565	0.606	**0.962**	**0.693**
	Weak	0.672	0.627	0.619	0.939	0.583
	Strong	0.674	0.617	0.616	0.950	0.660
	Weak+Strong	0.701	**0.653**	**0.639**	0.956	0.687

(c) Predict top **20%** defect-prone Gnome Ubuntu packages

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
2.16	Socio-technical	0.718	0.581	0.612	**0.890**	0.489
	Weak	0.644	0.443	0.499	0.850	0.415
	Strong	**0.825**	**0.600**	**0.666**	0.875	0.489
	Weak+Strong	0.808	0.592	0.651	0.882	**0.505**
2.18	Socio-technical	0.624	0.543	0.546	0.884	0.457
	Weak	0.653	0.479	0.515	0.875	0.435
	Strong	0.607	0.523	0.534	0.885	0.492
	Weak+Strong	**0.702**	**0.552**	**0.582**	**0.909**	**0.523**
2.20	Socio-technical	0.816	0.720	0.738	0.939	0.672
	Weak	0.822	0.640	0.694	0.940	0.672
	Strong	0.716	0.692	0.684	0.940	0.686
	Weak+Strong	**0.893**	**0.816**	**0.837**	**0.967**	**0.796**
2.22	Socio-technical	0.723	0.591	0.619	0.903	0.539
	Weak	0.758	0.643	0.670	0.886	0.570
	Strong	0.710	0.589	0.609	**0.923**	0.585
	Weak+Strong	**0.815**	**0.692**	**0.722**	0.923	**0.628**
2.24	Socio-technical	0.761	0.698	0.700	**0.957**	0.689
	Weak	**0.821**	**0.751**	**0.760**	0.940	0.686
	Strong	0.738	0.685	0.680	0.949	0.669
	Weak+Strong	0.785	0.699	0.717	0.949	**0.718**
2.26	Socio-technical	0.842	0.763	0.778	0.963	0.739
	Weak	0.780	0.722	0.724	0.941	0.672
	Strong	0.802	0.794	0.768	0.958	0.706
	Weak+Strong	**0.871**	**0.798**	**0.810**	**0.964**	**0.744**

(d) Predict top **25%** defect-prone Gnome Ubuntu packages

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
2.16	Socio-technical	0.691	0.588	0.611	0.871	0.484
	Weak	**0.756**	**0.626**	**0.651**	0.854	0.454
	Strong	0.714	0.596	0.619	0.862	0.499
	Weak+Strong	0.712	0.583	0.614	**0.881**	**0.504**
2.18	Socio-technical	0.632	0.538	0.554	0.875	0.471
	Weak	0.666	0.568	0.580	0.872	0.468
	Strong	0.683	0.555	0.582	0.890	0.520
	Weak+Strong	**0.754**	**0.594**	**0.637**	**0.905**	**0.558**
2.20	Socio-technical	0.772	0.737	0.733	0.943	0.678
	Weak	0.776	0.678	0.698	0.894	0.565
	Strong	0.789	0.808	0.780	0.941	0.693
	Weak+Strong	**0.828**	**0.809**	**0.802**	**0.951**	**0.739**
2.22	Socio-technical	0.720	0.628	0.651	0.910	0.575
	Weak	**0.782**	0.626	**0.677**	0.892	0.566
	Strong	0.773	0.575	0.638	0.904	0.580
	Weak+Strong	0.745	**0.631**	0.662	**0.912**	**0.612**
2.24	Socio-technical	0.798	0.696	0.718	0.923	0.619
	Weak	**0.817**	**0.715**	**0.741**	0.899	0.598
	Strong	0.773	0.693	0.712	0.920	0.609
	Weak+Strong	0.801	0.694	0.723	**0.925**	**0.649**
2.26	Socio-technical	0.821	**0.789**	**0.784**	0.933	0.678
	Weak	0.819	0.710	0.743	0.897	0.637
	Strong	0.853	0.720	0.761	**0.940**	**0.723**
	Weak+Strong	**0.883**	0.725	0.779	0.934	0.701

(e) Predict top **30%** defect-prone Gnome Ubuntu packages

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
2.16	Socio-technical	0.784	**0.698**	**0.713**	0.904	0.588
	Weak	**0.806**	0.650	0.696	0.889	0.570
	Strong	0.756	0.676	0.694	0.898	0.596
	Weak+Strong	0.796	0.650	0.691	**0.905**	**0.598**
2.18	Socio-technical	0.801	0.626	0.677	0.876	0.531
	Weak	0.778	0.697	0.716	0.880	0.535
	Strong	0.803	0.635	0.689	**0.904**	0.583
	Weak+Strong	**0.832**	**0.734**	**0.762**	0.903	**0.593**
2.20	Socio-technical	0.811	0.791	0.787	0.926	0.653
	Weak	0.807	0.689	0.726	0.866	0.557
	Strong	0.801	0.805	0.790	0.924	0.666
	Weak+Strong	**0.861**	**0.823**	**0.830**	**0.934**	**0.674**
2.22	Socio-technical	0.714	0.688	0.682	0.903	0.573
	Weak	**0.855**	0.700	0.747	0.877	0.564
	Strong	0.787	0.655	0.693	**0.907**	**0.586**
	Weak+Strong	0.829	**0.733**	**0.763**	0.877	0.581
2.24	Socio-technical	0.806	**0.691**	0.730	**0.901**	0.580
	Weak	**0.852**	0.658	0.728	0.885	0.556
	Strong	0.842	0.683	**0.736**	0.894	**0.606**
	Weak+Strong	0.824	0.655	0.713	0.897	0.596
2.26	Socio-technical	0.818	**0.777**	0.783	0.935	0.694
	Weak	0.842	0.729	0.768	0.908	0.653
	Strong	**0.876**	0.752	**0.794**	0.937	**0.705**
	Weak+Strong	0.823	0.759	0.775	**0.938**	0.703

Appendix C

Prediction Across Releases

Table C.1: Results of using data from a previous release to predict whether an Eclipse Java package is defect-prone in next release. Bold values indicate that they are higher than values calculated on other networks.

(a) Predict top 10% defect-prone Eclipse Java packages

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
2.1	Socio-technical	0.429	0.162	0.235	0.788	0.379
	Weak	0.700	0.233	0.350	0.855	0.324
	Strong	0.750	0.300	0.429	**0.899**	0.382
	Weak+Strong	**0.833**	**0.333**	**0.476**	0.890	**0.425**
3.0	Socio-technical	0.559	0.352	0.432	0.796	**0.484**
	Weak	0.643	0.243	0.353	0.873	0.388
	Strong	0.875	0.378	0.528	0.868	0.399
	Weak+Strong	**0.882**	**0.405**	**0.556**	**0.900**	0.480
3.1	Socio-technical	0.759	0.449	0.564	0.930	0.473
	Weak	0.733	0.407	0.524	0.912	0.478
	Strong	0.742	0.426	0.541	0.898	0.491
	Weak+Strong	**0.769**	**0.556**	**0.645**	**0.935**	**0.571**
3.2	Socio-technical	0.780	0.410	0.538	0.919	0.617
	Weak	0.833	0.510	0.633	0.935	0.568
	Strong	0.813	0.531	0.642	0.948	0.595
	Weak+Strong	**0.844**	**0.551**	**0.667**	**0.966**	**0.667**
3.3	Socio-technical	0.698	0.474	0.565	0.899	0.566
	Weak	**0.800**	0.462	0.585	0.915	0.519
	Strong	0.714	0.449	0.551	0.928	0.518
	Weak+Strong	0.796	**0.500**	**0.614**	**0.941**	**0.588**
3.4	Socio-technical	0.571	0.198	0.294	0.863	0.502
	Weak	**0.780**	0.410	0.538	0.905	0.482
	Strong	0.758	0.321	0.450	0.887	0.410
	Weak+Strong	0.766	**0.462**	**0.576**	**0.923**	**0.524**

(b) Predict top 15% defect-prone Eclipse Java packages

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
2.1	Socio-technical	**0.800**	0.271	0.405	0.775	0.458
	Weak	0.724	0.412	0.525	0.882	0.446
	Strong	0.714	0.196	0.308	0.837	0.297
	Weak+Strong	0.758	**0.490**	**0.595**	**0.937**	**0.594**
3.0	Socio-technical	0.642	0.400	0.493	0.857	0.481
	Weak	**0.774**	0.407	0.533	0.842	0.408
	Strong	0.722	0.220	0.338	0.837	0.333
	Weak+Strong	0.707	**0.492**	**0.580**	**0.910**	**0.533**
3.1	Socio-technical	0.632	0.453	0.528	0.867	0.558
	Weak	**0.796**	0.459	0.582	0.890	0.497
	Strong	0.743	0.306	0.433	0.866	0.420
	Weak+Strong	0.770	**0.553**	**0.644**	**0.928**	**0.577**
3.2	Socio-technical	0.735	0.466	0.570	0.876	0.524
	Weak	**0.755**	0.389	0.514	0.873	0.425
	Strong	0.694	0.358	0.472	0.871	0.414
	Weak+Strong	0.750	**0.505**	**0.604**	**0.922**	**0.554**
3.3	Socio-technical	**0.821**	0.406	0.544	0.846	**0.564**
	Weak	0.776	0.504	0.611	0.891	0.495
	Strong	0.651	0.214	0.322	0.833	0.332
	Weak+Strong	0.792	**0.580**	**0.670**	**0.917**	0.553
3.4	Socio-technical	0.556	0.473	0.511	0.818	0.473
	Weak	**0.839**	0.406	0.547	0.827	0.367
	Strong	0.701	0.318	0.437	0.813	0.326
	Weak+Strong	0.770	**0.505**	**0.610**	**0.882**	**0.491**

(c) Predict top 20% defect-prone Eclipse Java packages

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
2.1	Socio-technical	0.681	0.410	0.512	0.828	0.506
	Weak	0.717	0.475	0.571	0.857	0.437
	Strong	0.625	0.375	0.469	0.814	0.310
	Weak+Strong	**0.766**	**0.613**	**0.681**	**0.894**	**0.538**
3.0	Socio-technical	0.676	0.455	0.543	0.875	0.515
	Weak	0.733	0.423	0.537	0.858	0.449
	Strong	0.675	0.346	0.458	0.841	0.372
	Weak+Strong	**0.737**	**0.538**	**0.622**	**0.904**	**0.551**
3.1	Socio-technical	0.634	0.478	0.545	0.848	0.569
	Weak	**0.857**	0.545	0.667	0.887	0.529
	Strong	0.760	0.345	0.475	0.847	0.398
	Weak+Strong	0.829	**0.618**	**0.708**	**0.918**	**0.587**
3.2	Socio-technical	**0.812**	0.495	0.615	0.858	0.521
	Weak	0.762	0.455	0.570	0.866	0.449
	Strong	0.708	0.343	0.462	0.840	0.380
	Weak+Strong	0.777	**0.545**	**0.640**	**0.908**	**0.546**
3.3	Socio-technical	0.687	0.479	0.564	0.852	**0.502**
	Weak	0.744	0.500	0.598	0.859	0.439
	Strong	0.608	0.323	0.422	0.773	0.273
	Weak+Strong	**0.757**	**0.552**	**0.639**	**0.878**	0.488
3.4	Socio-technical	0.758	0.320	0.450	0.775	0.473
	Weak	**0.839**	0.406	0.547	0.827	0.367
	Strong	0.701	0.318	0.437	0.813	0.326
	Weak+Strong	0.770	**0.505**	**0.610**	**0.882**	**0.491**

(d) Predict top 25% defect-prone Eclipse Java packages

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
2.1	Socio-technical	0.647	0.423	0.512	0.829	0.508
	Weak	**0.765**	0.488	0.595	0.856	0.434
	Strong	**0.765**	0.488	0.595	0.855	0.410
	Weak+Strong	0.754	**0.575**	**0.652**	**0.899**	**0.554**
3.0	Socio-technical	0.685	0.385	0.493	0.852	**0.514**
	Weak	0.673	0.423	0.520	0.883	0.492
	Strong	0.735	**0.462**	0.567	0.855	0.432
	Weak+Strong	**0.766**	**0.462**	**0.576**	**0.888**	0.503
3.1	Socio-technical	0.645	0.597	0.620	0.862	0.533
	Weak	0.792	0.615	0.693	0.911	0.566
	Strong	**0.822**	0.569	0.673	0.890	0.521
	Weak+Strong	0.811	**0.662**	**0.729**	**0.930**	**0.634**
3.2	Socio-technical	0.793	0.479	0.597	0.858	0.517
	Weak	0.693	0.522	0.596	0.876	0.466
	Strong	0.711	0.440	0.544	0.864	0.443
	Weak+Strong	**0.824**	**0.597**	**0.693**	**0.912**	**0.574**
3.3	Socio-technical	0.657	0.490	0.561	0.858	0.501
	Weak	0.764	0.505	0.608	0.857	0.452
	Strong	0.758	0.521	0.617	0.870	0.469
	Weak+Strong	**0.777**	**0.599**	**0.676**	**0.899**	**0.554**
3.4	Socio-technical	0.580	0.473	0.521	0.828	0.474
	Weak	**0.771**	0.422	0.545	0.872	0.444
	Strong	0.719	0.427	0.536	0.856	0.420
	Weak+Strong	0.750	**0.500**	**0.600**	**0.887**	**0.496**

(e) Predict top 30% defect-prone Eclipse Java packages

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
2.1	Socio-technical	0.769	0.439	0.559	0.826	0.440
	Weak	0.758	0.480	0.587	0.829	0.388
	Strong	0.723	0.480	0.577	0.801	0.292
	Weak+Strong	**0.792**	**0.582**	**0.671**	**0.864**	**0.489**
3.0	Socio-technical	**0.767**	0.462	0.577	0.830	0.472
	Weak	0.750	0.500	0.600	0.851	0.425
	Strong	0.746	0.412	0.531	0.783	0.299
	Weak+Strong	0.719	**0.561**	**0.631**	**0.877**	**0.501**
3.1	Socio-technical	0.742	0.608	0.669	0.870	0.508
	Weak	0.780	0.561	0.653	0.860	0.474
	Strong	0.702	0.497	0.582	0.819	0.376
	Weak+Strong	**0.791**	**0.620**	**0.695**	**0.889**	**0.533**
3.2	Socio-technical	**0.843**	0.480	0.612	0.809	**0.554**
	Weak	0.759	0.536	0.628	0.857	0.453
	Strong	0.715	0.531	0.609	0.835	0.392
	Weak+Strong	0.792	**0.649**	**0.714**	**0.896**	0.554
3.3	Socio-technical	0.674	0.625	0.649	0.789	**0.452**
	Weak	**0.788**	0.566	0.659	0.814	0.372
	Strong	0.716	0.566	0.632	0.750	0.251
	Weak+Strong	0.787	**0.638**	**0.705**	**0.833**	0.420
3.4	Socio-technical	0.599	0.534	0.565	0.772	0.377
	Weak	**0.757**	0.484	0.591	0.774	0.295
	Strong	0.690	0.550	0.612	0.764	0.273
	Weak+Strong	0.747	**0.605**	**0.669**	**0.825**	**0.410**

Table C.2: Results of using data from a previous release to predict whether an Netbeans Java JAR file is defect-prone in next release. Bold values indicate that they are higher than values calculated on other networks.

(a) Predict top 10% defect-prone Netbeans Java JAR files

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
6.5	Socio-technical	0.438	0.159	0.233	0.818	0.487
	Weak	0.667	0.424	0.519	0.949	0.538
	Strong	**0.917**	0.333	0.489	0.921	0.449
	Weak+Strong	0.652	**0.455**	**0.536**	**0.958**	**0.581**
6.7	Socio-technical	0.154	0.059	0.085	0.682	0.434
	Weak	**0.773**	**0.386**	**0.515**	0.892	0.421
	Strong	0.750	0.341	0.469	0.908	0.446
	Weak+Strong	0.727	0.364	0.485	**0.933**	**0.510**
6.8	Socio-technical	0.471	0.200	0.281	0.877	0.409
	Weak	0.500	0.206	0.292	0.920	0.435
	Strong	0.385	0.147	0.213	0.923	0.410
	Weak+Strong	**0.526**	**0.294**	**0.377**	**0.946**	**0.504**
6.9	Socio-technical	0.500	0.239	0.324	0.902	0.460
	Weak	0.765	0.325	0.456	0.904	0.446
	Strong	0.706	0.300	0.421	0.906	0.402
	Weak+Strong	**0.773**	**0.425**	**0.548**	**0.939**	**0.556**
7.0	Socio-technical	0.150	0.354	0.211	0.719	0.514
	Weak	0.619	0.283	0.388	0.918	0.469
	Strong	**0.750**	0.391	0.514	0.926	0.492
	Weak+Strong	0.688	**0.478**	**0.564**	**0.948**	**0.570**

(b) Predict top 15% defect-prone Netbeans Java JAR files

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
6.5	Socio-technical	0.368	0.159	0.222	0.836	0.403
	Weak	0.667	0.286	0.400	0.908	0.449
	Strong	0.722	0.265	0.388	0.874	0.370
	Weak+Strong	**0.735**	**0.510**	**0.602**	**0.918**	**0.494**
6.7	Socio-technical	0.417	0.147	0.217	0.828	0.363
	Weak	0.706	0.273	0.393	0.875	0.379
	Strong	**0.750**	0.273	0.400	0.907	0.420
	Weak+Strong	**0.750**	**0.341**	**0.469**	**0.925**	**0.490**
6.8	Socio-technical	0.471	0.200	0.281	0.880	0.408
	Weak	**0.600**	0.265	0.367	0.920	0.421
	Strong	0.462	0.176	0.255	0.918	0.390
	Weak+Strong	0.579	**0.324**	**0.415**	**0.944**	**0.496**
6.9	Socio-technical	0.684	0.181	0.286	0.860	0.451
	Weak	0.765	0.325	0.456	0.904	0.451
	Strong	0.647	0.275	0.386	0.918	0.427
	Weak+Strong	**0.773**	**0.425**	**0.548**	**0.938**	**0.544**
7.0	Socio-technical	0.128	0.396	0.193	0.706	0.477
	Weak	0.727	0.333	0.457	0.869	0.393
	Strong	**0.743**	0.361	0.486	0.872	0.409
	Weak+Strong	0.720	**0.500**	**0.590**	**0.909**	**0.516**

(c) Predict top 20% defect-prone Netbeans Java JAR files

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
6.5	Socio-technical	0.282	0.250	0.265	0.820	0.401
	Weak	0.650	0.317	0.426	0.861	0.392
	Strong	0.651	0.341	0.448	0.848	0.367
	Weak+Strong	**0.673**	**0.427**	**0.522**	**0.889**	**0.461**
6.7	Socio-technical	0.583	0.077	0.136	0.759	0.363
	Weak	0.706	0.273	0.393	0.875	0.379
	Strong	**0.750**	0.273	0.400	0.907	0.420
	Weak+Strong	**0.750**	**0.341**	**0.469**	**0.925**	**0.490**
6.8	Socio-technical	0.674	0.305	0.420	0.821	0.369
	Weak	**0.698**	0.330	0.448	0.828	0.357
	Strong	0.681	0.352	0.464	0.841	0.365
	Weak+Strong	0.685	**0.407**	**0.510**	**0.848**	**0.385**
6.9	Socio-technical	0.642	0.472	0.544	0.874	0.416
	Weak	0.680	0.358	0.469	0.835	0.355
	Strong	0.625	0.316	0.420	0.843	0.359
	Weak+Strong	**0.726**	**0.474**	**0.573**	**0.883**	**0.459**
7.0	Socio-technical	0.128	0.396	0.193	0.706	0.477
	Weak	0.727	0.333	0.457	0.869	0.393
	Strong	**0.743**	0.361	0.486	0.872	0.409
	Weak+Strong	0.720	**0.500**	**0.590**	**0.909**	**0.516**

(d) Predict top 25% defect-prone Netbeans Java JAR files

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
6.5	Socio-technical	0.548	0.138	0.221	0.797	0.416
	Weak	**0.647**	0.268	0.379	0.858	0.387
	Strong	0.635	0.402	0.493	0.859	0.401
	Weak+Strong	0.654	**0.415**	**0.507**	**0.884**	**0.451**
6.7	Socio-technical	0.273	0.066	0.106	0.651	0.430
	Weak	0.645	0.398	0.492	0.836	0.381
	Strong	0.631	0.431	0.512	0.854	0.430
	Weak+Strong	**0.698**	**0.488**	**0.574**	**0.863**	**0.452**
6.8	Socio-technical	0.622	0.295	0.400	0.815	0.374
	Weak	0.667	0.330	0.441	0.832	0.359
	Strong	0.708	**0.374**	0.489	0.843	0.371
	Weak+Strong	**0.723**	**0.374**	**0.493**	**0.850**	**0.389**
6.9	Socio-technical	**0.849**	0.328	0.474	0.827	0.424
	Weak	0.706	0.379	0.493	0.836	0.354
	Strong	0.674	0.326	0.440	0.843	0.364
	Weak+Strong	0.688	**0.463**	**0.554**	**0.888**	**0.478**
7.0	Socio-technical	0.415	**0.532**	0.466	0.771	0.435
	Weak	**0.789**	0.409	0.538	0.830	0.376
	Strong	0.740	0.394	0.514	0.821	0.356
	Weak+Strong	0.756	0.474	**0.583**	**0.856**	**0.446**

(e) Predict top 30% defect-prone Netbeans Java JAR files

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
6.5	Socio-technical	0.667	0.211	0.321	0.808	0.401
	Weak	0.650	0.317	0.426	0.861	0.392
	Strong	0.651	0.341	0.448	0.848	0.367
	Weak+Strong	**0.673**	**0.427**	**0.522**	**0.889**	**0.461**
6.7	Socio-technical	0.636	0.308	0.415	0.799	0.386
	Weak	0.662	0.398	0.497	0.829	0.356
	Strong	0.615	0.390	0.478	0.837	0.385
	Weak+Strong	**0.679**	**0.463**	**0.551**	**0.853**	**0.421**
6.8	Socio-technical	0.674	0.305	0.420	0.821	0.369
	Weak	**0.698**	0.330	0.448	0.828	0.357
	Strong	0.681	0.352	0.464	0.841	0.365
	Weak+Strong	0.685	**0.407**	**0.510**	**0.848**	**0.385**
6.9	Socio-technical	**0.830**	0.321	0.463	0.825	0.416
	Weak	0.680	0.358	0.469	0.835	0.355
	Strong	0.625	0.316	0.420	0.843	0.359
	Weak+Strong	0.726	**0.474**	**0.573**	**0.883**	**0.459**
7.0	Socio-technical	0.456	**0.514**	0.483	0.774	0.433
	Weak	0.750	0.394	0.517	0.830	0.375
	Strong	0.760	0.416	0.538	0.830	0.387
	Weak+Strong	**0.776**	0.482	**0.595**	**0.857**	**0.445**

Table C.3: Results of using data from a previous release to predict whether a Gnome Ubuntu package is defect-prone in next release. Bold values indicate that they are higher than values calculated on other networks.

(a) Predict top 10% defect-prone Gnome Ubuntu packages

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
2.18	Socio-technical	0.571	0.471	0.516	0.888	0.507
	Weak	**0.778**	0.438	0.560	0.888	0.491
	Strong	0.636	0.438	0.519	0.910	0.507
	Weak+Strong	0.667	**0.500**	**0.571**	**0.945**	**0.600**
2.20	Socio-technical	0.500	**0.647**	0.564	0.908	**0.582**
	Weak	**0.778**	0.412	0.538	0.889	0.485
	Strong	0.727	0.471	0.571	0.922	0.526
	Weak+Strong	0.769	0.588	**0.667**	**0.944**	0.568
2.22	Socio-technical	0.800	0.444	0.571	0.919	0.570
	Weak	**1.000**	0.588	0.741	0.926	0.523
	Strong	0.769	0.588	0.667	0.962	0.622
	Weak+Strong	0.933	**0.824**	**0.875**	**0.978**	**0.829**
2.24	Socio-technical	0.424	**0.667**	0.519	0.837	**0.709**
	Weak	0.727	0.444	0.552	0.921	0.513
	Strong	**0.909**	0.556	**0.690**	0.958	0.698
	Weak+Strong	0.786	0.611	0.688	**0.968**	0.702
2.26	Socio-technical	0.407	0.611	0.489	0.924	0.667
	Weak	0.846	0.524	0.647	0.938	0.600
	Strong	**0.938**	**0.714**	**0.811**	0.958	0.669
	Weak+Strong	**0.938**	**0.714**	**0.811**	**0.962**	**0.739**

(b) Predict top 15% defect-prone Gnome Ubuntu packages

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
2.18	Socio-technical	0.600	0.480	0.533	0.840	0.393
	Weak	0.727	0.333	0.457	0.793	0.336
	Strong	**0.800**	0.500	0.615	0.832	0.398
	Weak+Strong	0.765	**0.542**	**0.634**	**0.843**	**0.428**
2.20	Socio-technical	**0.812**	0.500	**0.619**	**0.901**	0.455
	Weak	0.750	0.360	0.486	0.817	0.380
	Strong	0.765	**0.520**	**0.619**	0.879	0.452
	Weak+Strong	0.786	0.440	0.564	0.891	**0.486**
2.22	Socio-technical	0.679	0.655	0.667	0.900	0.642
	Weak	**0.882**	0.577	0.698	0.919	0.596
	Strong	0.864	**0.731**	**0.792**	0.959	0.726
	Weak+Strong	0.850	0.654	0.739	**0.960**	**0.732**
2.24	Socio-technical	0.263	0.357	0.303	0.832	0.563
	Weak	0.650	0.448	0.531	0.894	0.503
	Strong	0.708	**0.586**	0.642	0.923	0.573
	Weak+Strong	**0.739**	**0.586**	**0.654**	**0.937**	**0.615**
2.26	Socio-technical	0.559	0.679	0.613	0.866	0.773
	Weak	0.857	0.643	0.735	0.952	0.685
	Strong	0.750	**0.750**	0.750	0.960	0.710
	Weak+Strong	**0.909**	0.714	**0.800**	**0.979**	**0.785**

(c) Predict top 20% defect-prone Gnome Ubuntu packages

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
2.18	Socio-technical	0.613	0.543	0.576	0.862	0.489
	Weak	**0.800**	0.485	0.604	0.805	0.404
	Strong	0.731	0.576	0.644	0.875	0.505
	Weak+Strong	**0.800**	**0.606**	**0.690**	**0.880**	**0.523**
2.20	Socio-technical	**0.846**	**0.611**	**0.710**	**0.936**	0.457
	Weak	0.714	0.429	0.536	0.820	0.385
	Strong	0.643	0.514	0.571	0.868	0.455
	Weak+Strong	0.654	0.486	0.557	0.880	**0.467**
2.22	Socio-technical	0.619	0.703	0.658	0.854	0.672
	Weak	0.760	0.528	0.623	0.910	0.572
	Strong	**0.812**	**0.722**	**0.765**	0.943	0.682
	Weak+Strong	0.806	0.694	0.746	**0.964**	**0.723**
2.24	Socio-technical	0.750	0.474	0.581	0.899	0.539
	Weak	0.792	0.514	0.623	0.842	0.455
	Strong	0.733	**0.595**	0.657	0.900	0.536
	Weak+Strong	**0.808**	0.568	**0.667**	**0.915**	**0.580**
2.26	Socio-technical	0.714	**0.811**	**0.759**	**0.952**	0.689
	Weak	0.765	0.684	0.722	0.938	0.654
	Strong	0.758	0.658	0.704	0.919	0.585
	Weak+Strong	**0.812**	0.684	0.743	0.950	**0.699**

(d) Predict top 25% defect-prone Gnome Ubuntu packages

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
2.18	Socio-technical	0.647	0.524	0.579	0.858	0.484
	Weak	**0.767**	0.561	0.648	0.823	0.425
	Strong	0.722	**0.634**	0.675	**0.881**	0.508
	Weak+Strong	0.765	**0.634**	**0.693**	0.874	**0.511**
2.20	Socio-technical	**0.805**	**0.717**	**0.759**	**0.918**	0.471
	Weak	0.692	0.429	0.529	0.836	0.423
	Strong	0.697	0.548	0.613	0.875	0.473
	Weak+Strong	0.714	0.595	0.649	0.882	**0.495**
2.22	Socio-technical	0.560	**0.824**	0.667	0.844	**0.678**
	Weak	**0.800**	0.522	0.632	0.835	0.478
	Strong	0.786	0.717	**0.750**	0.925	0.623
	Weak+Strong	0.767	0.717	0.742	**0.939**	0.658
2.24	Socio-technical	0.450	0.574	0.505	0.780	**0.575**
	Weak	**0.795**	0.608	0.689	0.838	0.470
	Strong	0.767	**0.647**	**0.702**	**0.889**	0.550
	Weak+Strong	0.767	**0.647**	**0.702**	0.886	0.562
2.26	Socio-technical	0.761	**0.745**	0.753	0.912	0.619
	Weak	0.821	0.681	0.744	0.900	0.594
	Strong	0.805	0.702	0.750	0.898	0.553
	Weak+Strong	**0.854**	**0.745**	**0.795**	**0.916**	**0.641**

(e) Predict top 30% defect-prone Gnome Ubuntu packages

Release	Network	Precision	Recall	F-Score	AUC	Nagel.
2.18	Socio-technical	0.800	0.604	0.688	0.867	0.588
	Weak	**0.865**	0.653	**0.744**	0.849	0.519
	Strong	0.786	**0.673**	0.725	**0.916**	**0.616**
	Weak+Strong	0.805	**0.673**	0.733	0.897	0.601
2.20	Socio-technical	0.800	**0.721**	**0.759**	0.883	0.531
	Weak	0.786	0.623	0.695	0.832	0.470
	Strong	0.800	0.679	0.735	0.877	0.514
	Weak+Strong	**0.833**	0.660	0.737	**0.896**	**0.583**
2.22	Socio-technical	0.614	**0.879**	0.723	0.858	**0.653**
	Weak	**0.837**	0.590	0.692	0.863	0.514
	Strong	0.783	0.770	**0.777**	0.896	0.576
	Weak+Strong	0.815	0.721	0.765	**0.919**	0.635
2.24	Socio-technical	0.516	0.493	0.504	0.742	0.573
	Weak	**0.829**	0.586	0.687	0.815	0.481
	Strong	0.796	**0.672**	**0.729**	0.898	**0.579**
	Weak+Strong	0.804	0.638	0.712	**0.900**	0.577
2.26	Socio-technical	0.786	**0.733**	0.759	**0.923**	0.580
	Weak	**0.904**	0.701	**0.790**	0.898	0.616
	Strong	0.810	0.701	0.752	0.892	0.590
	Weak+Strong	0.845	0.731	0.784	0.909	**0.631**