ON THE EVOLUTIONARY COUPLING AND ITS MEASUREMENT

by

Serkan Kırbaş

B.S., Computer Engineering, Middle East Technical University, 2003

M.S., Computer Engineering, Middle East Technical University, 2007

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Computer Engineering

Boğaziçi University

2017

ON THE EVOLUTIONARY COUPLING AND ITS MEASUREMENT

APPROVED BY:

Assoc. Prof. Alper Şen (Thesis Supervisor)	.....
Prof. Tracy Hall (Thesis Co-supervisor)	.....
Prof. Fikret Gürgen	.....
Prof. Pınar Yolum	.....
Assoc. Prof. Feza Buzluca	.....
Assoc. Prof. Hasan Sözer	.....

DATE OF APPROVAL: 18.05.2017

ACKNOWLEDGEMENTS

I would like to express my special appreciation and thanks to my supervisors Assoc. Prof. Alper Sen and Prof. Tracy Hall, you have been a tremendous mentor for me. I would like to thank you for encouraging my research and for your patience in this seven years-long journey. I would also like to thank Prof. Ayse Bener, Dr. Bora Caglayan, Rasim Mahmutogullari, Prof. Steve Counsell, and Dr. David Bowes for their contributions on our empirical research and its publications.

A special thanks to my family. Words cannot express how grateful I am to my wife Sermin Ildirar, my daughters Maya and Ada, who are by-products of my thesis:), my mother Tenzile Kirbas, my father Recep Kirbas, and my mother-in-law Suheyla Keklik for all of the sacrifices that you've made on my behalf. I would also like to thank all of my friends who supported me in this long journey. I would like to thank and commemorate my grandparents, Fehime Demirbilek and Saban Demirbilek, who passed away during my thesis. I also would like to thank my grandparents Mehmet Kirbas and Ismigul Kirbas, who always encouraged me in studying and shared my excitement.

ABSTRACT

ON THE EVOLUTIONARY COUPLING AND ITS MEASUREMENT

Evolutionary Coupling (EC) is the implicit relationship between the artifacts or parts of the software system that are frequently changed together during evolution of a system. Understanding the EC in software systems is important, as it has been shown to provide insight into architectural problems, cross-cutting concerns, software defects and the impact of change. Today, the increasing size and coupling within and between software increases the importance of work on EC.

In the first part of this thesis we analyse large commercial systems which have rarely been empirically studied to understand the relation between EC and defects. We explore the reasons for the contradicting results in the literature about the relationship between EC and defects. No studies exist to explain these contradictory findings. Our results show that the explanatory power of EC measures varies depending on defect types and module features such as size and developer activity.

In the second part of the thesis we develop EC measurement evaluation criteria by using measurement theory and metrology principles. We show the weaknesses and strengths of current EC measures based on measurement theory principles. We provide recommendations for practitioners and researchers about what EC measure to use and not to use as well as when to use these measures. Furthermore, we develop a meta-model for EC concepts, which are essential in understanding how the measure is derived and how to interpret it. To the best of our knowledge, this is the first work that applies measurement theory and metrology principles to EC measurement.

ÖZET

EVRİMSEL BAĞLAŞIM VE ÖLÇÜMÜ ÜZERİNE

Evrimsel Bağlaşım (EB), bir sistemin gelişimi sırasında birlikte sıkça değişen yazılım sisteminin parçaları (artifact) arasındaki örtülü ilişkidir. Yazılım sistemlerinde EB'yi anlamak önemlidir çünkü mimari problemler, çapraz kesim, yazılım hataları ve etki analizi hakkında bilgi verir. Günümüzde, yazılım içindeki ve yazılımlar arasındaki artan boyut ve bağlaşım, EB üzerinde çalışmanın önemini arttırmaktadır.

Bu tezin ilk bölümünde, evrimsel bağlaşım ile yazılım hataları arasındaki ilişkiyi anlamak için ampirik olarak nadiren incelenmiş olan büyük endüstriyel yazılım sistemlerini analiz ettik. EB ile hatalar arasındaki ilişki hakkında literatürdeki çelişkili sonuçların nedenlerini araştırdık. Sonuçlarımız, EB ölçülerinin açıklayıcı gücünün, yazılım boyutu ve yazılım geliştirici faaliyetleri gibi hata tiplerine ve modül özelliklerine bağlı olarak değiştiğini göstermektedir.

Tezin ikinci bölümünde ölçüm teorisi ve metroloji ilkelerini kullanarak EB ölçme değerlendirme kriterleri geliştirdik. Mevcut EB ölçütlerinin zayıf yönlerini ve güçlü yanlarını, ölçüm teorisi ilkelerine dayanarak gösterdik. Uygulayıcılara ve araştırmacılara, hangi EB ölçütlerini ne zaman kullanmaları gerektiği konusunda tavsiyeler sunduk. Ayrıca, EB ölçütlerinin nasıl türetildiğini ve nasıl yorumlanacağını anlamak için gerekli olan EB kavramları için bir meta model geliştirdik. Bu tez ölçüm teorisi ve metroloji ilkelerini EB ölçümlemesine uygulayan ilk çalışmadır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

n	Number of files in a module
p	Significance of correlation analysis result
z	z-value for the regression coefficient
ρ	Correlation coefficient or correlation strength (rho)
$\sum$	Summation

LIST OF ACRONYMS/ABBREVIATIONS

ARM	Association Rule Mining
BIPM	International Bureau of Weights and Measures
CA	Computer Associates
CA SCM	CA Software Change Manager
CBO	Coupling between Objects
CI	configuration Item
CK	Chidamber and Kemerer
CMDB	Configuration Management Database
CRM	Customer Relationship Management
DD	Defect Density
EC	Evolutionary Coupling
ERS	Empirical Relational System
IR	Information Retrieval
JCL	Job Control Language
LOC	Lines of Code
MR	Modification Request
MSR	Mining Software Repositories
MVC	Model-View-Controller
NoCommits	Number of Commits
NoD	Number of Defects
NoDevs	Number of Developers
NoECF	The total number of Evolutionary Coupled Files
NoECFMR	Sum of the number of Evolutionary Coupled Files for all MRs
NRS	numerical relational system
OO	object-oriented
OR	odds ratio
RC	Representation Condition
RFC	Response for Class

RMT	Representational Measurement Theory
SCM	Software Configuration Management
SDLC	Software Development Life Cycle
SECF	The set of Evolutionary Coupled Files
SECFMR	Set of Evolutionary Coupled Files in the scope of a MR
SZZ	Sliwerski-Zimmerman-Zeller
TDD	Test-Driven Development
TFSC	Total File Size Change in LOC
TNDVLP	Total Number of DeVeLoPers
TNEFC	Total Number of File Couplings
TNF	Total Number of Files
TNFR	Total Number of File Revisions
VIF	Variance Inflation Factors
WMO	World Meteorological Organization

1. INTRODUCTION

Today software is inevitable part of our lives. We depend on software systems in various domains such as air traffic control, finance, power, telecommunication and many others. Software systems are dynamic and they change due to several factors including competition, innovation, cost reduction and regulations. In other words, they should evolve to respond the changing environment and to maintain user satisfaction. Software repositories keep the details of these changes on evolving software systems. Software configuration management tools (SCM) and bug/issue tracking systems are now considered fundamental for software development life cycle (SDLC). Therefore it is now possible to answer various questions easily such as Five Ws: What happened?, Who is involved?, Where did it take place?, When did it take place?, Why did that happen?.

A new concept emerging from this new software development environment is evolutionary coupling (EC) [1] . EC is the implicit relationship between the artifacts or parts of the software system that are frequently changed together during evolution of a system. Understanding the EC in software systems is important, as it has been shown to provide insight into architectural problems [6–10], cross-cutting concerns [11–13], software defects [14–19] and the impact of change [20, 21]. Today, the increasing size and coupling within and between software increases the importance of work on EC.

EC information is generally extracted from the commit history of SCM tools. It is based on the assumption that artifacts committed together are logically coupled. This makes EC relatively simple to calculate compared to other types of coupling. For example, structural [22] and semantic [23] coupling are both measured based on the static and text analysis of source code. Often this source code is difficult to obtain from closed source developers. Dynamic coupling [24] analyses execution traces and so requires the software to be executed. EC requires access to only the version control system and is thus a relatively easy way to measure coupling, particularly for industrial

[1] Evolutionary coupling is also known as change coupling or logical coupling.

closed source systems.

EC has previously been shown to indicate architectural and design problems. Gall et al. [7, 8] showed that EC can discover design flaws such as God classes or Spaghetti code, without analysing the source code. Gall et al.'s results also identified architectural weaknesses such as poorly designed inheritance hierarchies and blurred interfaces between modules and submodules. Breu and Zimmermann [11] showed that EC information and data mining techniques could detect cross-cutting concerns in software systems. Such cross-cutting concerns emerging over time may contain functionality which does not align with its architecture. Furthermore, Eaddy et al. [12] argued that cross-cutting concerns made implementation and code change diffucult as multiple locations in the source code had to be found and updated simultaneously. Their study suggested that increased cross-cutting concerns may actually cause or contribute to defects. Our own previous study of EC in a banking system [3] suggested that EC does impact defects. In the first part of this dissertation, we present a detailed background about EC and its measurement.

Conversely, Graves et al. [18] showed that module level EC measures were a poor predictor of defect-proneness. Knab et al. [19] also found that EC did not predict defects in the Mozilla project; no studies exist to explain these contradictory findings. In the second part of this dissertation, we explore the reasons for these contradicting results of EC studies by analysing the role of the context and defect types. We have performed empirical studies on two large industrial systems: a legacy financial system and a modern telecommunications system. We investigated the effect of EC on the defect-proneness of large industrial software systems and explain why the effects vary. We collected historical data for 7 years from 5 different software repositories containing 176 thousand files. We applied correlation and regression analysis to explore the relationship between EC and software defects and we analysed defect types, size and process metrics to explain different effects of EC on defects through correlation.

While performing these empirical studies we have also spotted problems about EC measurement, which can also cause contradictory and unreliable results. Although

there has been considerable work in establishing a measurement theory basis for software measurement [25–28], it is not clear that these measurement principles have been used in EC measurement. Measures that do not adhere to these principles may be flawed. Empirical results obtained using flawed measures are likely to be unreliable and decisions and conclusions based on these results could be misleading. Such results lead to waste of time, money and resources as well as to the generation of contaminated scientific knowledge. This problem has been addressed in the third part of this dissertation. We evaluate the measurement of EC in software artefacts from a measurement theory perspective. We define 19 evaluation criteria based on the principles of measurement theory and metrology. We evaluate previously published EC measures by applying these criteria.

1.1. Our Thesis

The goal of our thesis work is to create an EC evaluation approach that supports EC measurement and its use in empirical studies.

First we investigate the effect of EC on defect proneness of a large legacy software in an industrial software development environment. We analysed the correlations between EC measures and defect measures such as the number of defects and defect density. We also used EC measures in regression models to explain defects. Module characteristics and defect types are used to explain why different results are reported by different studies in the literature.

Second we analyse measurement of EC. EC is measured based on the co-changes of software artefacts in the version history. There are a variety of different co-change characteristics such as locality and change type. One flaw of current EC measures is treating these different characteristics as equivalent during the aggregation of co-change measurement results. For example; in terms of locality the distance between co-changes, specifically cross-system vs. within-system co-changes, there is evidence [16] that a cross-system EC is not equal to a within-system EC in the context of defect prediction. In particular cross-system co-changes are more valuable for defect

prediction. Unreliable results are possible where EC measurement does not take such characteristics into account, and the validity of these measures is difficult to establish.

In addition, there is no consistent definition of EC with studies using a variety of different measures. This is problematic as it means that the phenomenon that one study is reporting as EC may be very different from another study. Some studies [7, 17, 20] require a minimum number of evolutionary coupled artefacts while others [6, 21, 29] do not. This inconsistency is problematic because it is hard to compare the results of different studies.

In this dissertation we define EC measurement evaluation criteria. We use measurement theory and metrology principles in the development of our criteria. We survey the state of the art in software measurement and metrology. Based on the survey, we extract five principles for EC measurement in Table 1.1. Measurement must start with an objective [26, 28, 30, 31]. Objectives put measurement in a context and determine the design of measures and interpretation of the measurement results. So by following this basic principal of the measurement theory we first find the objectives of EC measurement. This constitutes our first principle shown in Table 1.1. Measurement captures information about attributes of entities [26,28,31]. So both the entity and the attribute to be measured should be identified clearly. This basic principle [26, 28, 31] is the base of our second principle in Table 1.1. We characterise the concept of EC by specifying the entities to be measured and the characteristics (attributes) that should be taken into account. We develop a meta-model of EC and its sub-concepts including the relationships across them as suggested by Abran [28]. In order to be valid, a measurement method must satisfy the representation condition of measurement theory [25, 26, 31]. Based on this principle and the meta-model developed in the previous step, we define the empirical relations and concerns to be considered in establishing a sound empirical relation system, which captures all generally accepted ideas about EC. We formulate our third principle in Table 1.1 based on this principle of measurement theory. We check whether EC measures preserve the empirical relations while mapping the empirical relation system into the numerical one. Abran in his software metrology work [28] points out that mature engineering disciplines have well established mea-

surement methods and detailed procedures with a large international consensus. For example, the World Meteorological Organization (WMO) defines the temperature measurement setup in a very detailed and quantitative manner such as that thermometers should be positioned between 1.25m and 2m above the ground. Well-defined, standard and detailed measurement method and procedures are mandatory to ensure the accuracy, repeatability, and repetitiveness of measurement results. Based on this principle, we formulate our fourth principle to cover the detailed procedures and practicalities of EC measurement. Determining the scale type of measures and doing mathematical validation are also one of the basic practices of measurement. Scale types are very important to determine the limitations on the kind of mathematical manipulations that can be performed on measurement results. This forms our fifth principle.

We evaluate current EC measures based on these principles and criteria formulated. We reveal the picture of existing EC measurement practices and provide recommendations to be used in future uses of EC measurement.

Table 1.1. Software measurement requirements.

No	**Principle**
1	Identify Measurement Objectives
2	Identify Entities and Attributes to be Measured
3	Construct Sound Empirical Relation Systems
4	Define Measurement Method and Procedures
5	Use Right Scale Types and Validate Mathematically

1.2. Contributions

The contributions of this dissertation can be classified in two categories: empirical studies & tools and measurement.

1.2.1. Empirical Studies and Tools (first published in [1–4])

(i) We analyse large commercial systems which have rarely been empirically studied to understand the relation between EC and defects. Most previous studies are based on the analysis of open source systems.

(ii) We explore the reasons for the contradicting results in the literature about the relationship between EC and defects. No studies exist to explain these contradictory findings.

(iii) The explanatory power of EC measures varies depending on defect types and module features such as size and developer activity.

(iv) We implemented a novel MSR tool to mine various software repositories such as SCM, issue tracking and HR systems and to detect EC.

1.2.2. Measurement (first published in [5])

(i) We show the weaknesses and strengths of current EC measures based on measurement theory principles.

(ii) We provide recommendations for practitioners and researchers about what EC measure to use and not to use as well as when to use these measures.

(iii) We develop a meta-model for EC concepts, which are essential in understanding how the measure is derived and how to interpret the behaviour of the numerical values when returned to the real world.

(iv) To the best of our knowledge, this is the first work that applies measurement theory and metrology principles to EC measurement.

1.3. Organization

The roadmap of our work can be found in Figure 1.1. The chapters in this dissertation and our publications are also provided in this roadmap.

This dissertation is organised as follows: In the next chapter, we summarise background. Chapter 3 presents the empirical study regarding the relationship between

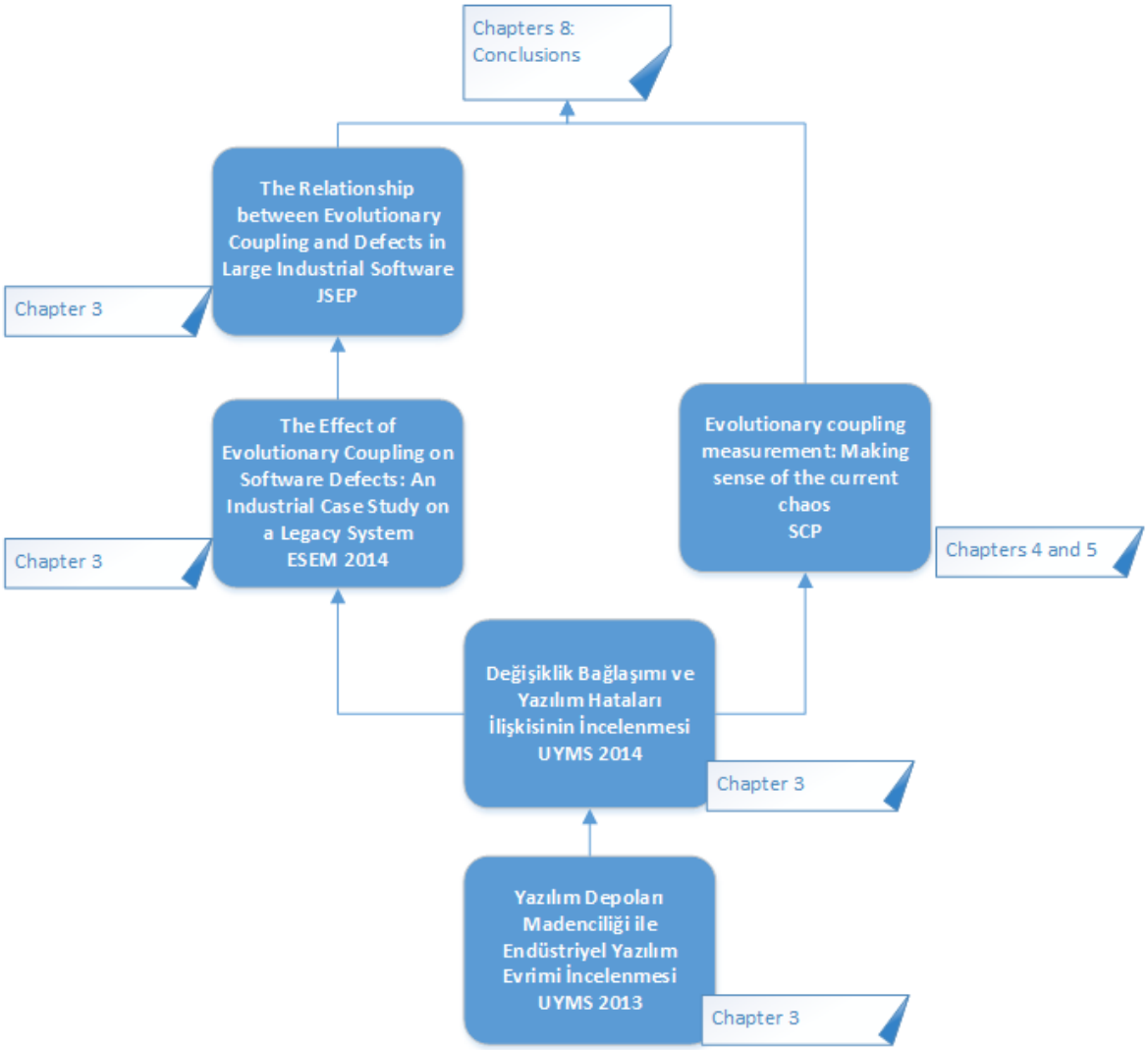


Figure 1.1. The roadmap of our work.

EC and defects. In Chapter 4, we present our methodology for study selection and EC measurement evaluation. Chapter 5 shows the results of applying our methodology to the EC measures. Chapter 6 summarises the related work. Finally, in Chapter 7, we summarise and present our conclusions.

2. BACKGROUND

In this chapter, we present some background information on EC, software measurement and EC measurement. We rely on this background information to explain our EC empirical studies and measurement evaluation in the thesis.

2.1. Evolutionary Coupling (EC)

Evolutionary coupling is the implicit dependency between two or more software artifacts that have been observed to frequently change together during the evolution of a system. It is also called change or logical coupling. This concept was first introduced in 1997 by Ball and Eick [6]. They presented a visualisation of co-changed classes as a graph. They applied a clustering algorithm to calculate the relative distances between these evolutionary coupled classes. They suggested that classes in the same cluster were semantically related. EC is one of the four coupling types, which are structural coupling, semantic coupling and dynamic coupling.

2.1.1. Other Types of Couplings

Definition 2.1. (Structural coupling): It is the first coupling type introduced in the software measurement literature in 1974 by Stevens et al. [32]. Most of the coupling measures are structural, which captures the relations between software artefacts such as method calls, inheritance, data sharing and access, etc. Coupling Between Objects (CBO) and Response for a Class (RFC) are two object-oriented (OO) coupling metrics of Chidamber and Kemerer (CK) metrics suite [33]. Shyam R Chidamber and Chris F. Kemerer proposed the first and most commonly used OO metrics in 1994. Definitions of two coupling metrics of the CK suite are provided below:

Definition 2.2. (Coupling between Objects): CBO is a count of the number of classes that are coupled to a particular class through method calls or variable accesses

of the other. CBO is the size of the set of classes that a class references and those classes that reference this class.

Definition 2.3. (Response for Class): RFC is the size of the response set of a class. Chidamber and Kemerer define the response set of a class as the set of methods that can potentially be executed in response to a message received by an object of that class.

Definition 2.4. (Semantic/Conceptual coupling): It is used to measure semantic similarity between software artefacts. In semantically coupled artefacts, same or semantically similar terms are present in their comments, identifiers, etc. This coupling type was introduced by Poshyvanyk et al. [34] in 2006. Information Retrieval (IR) techniques are used on the source code lexicon to calculate semantic coupling. Let us take two classes c1 and c2 with the following methods: $c1 = \{m1, m2\}, c2 = \{m3, m4, m5\}$. Semantic similarity ($SemSim$) between classes c1 and c2 is computed as the average of semantic similarities between methods of c1 and all other methods in class c2: $SemSim(c1, c2) = (SemSim(m1, c2) + (SemSim(m2, c2))/2$. Semantic similarity between the method m1 and the class c2 is calculated as:

$$SemSim(m1, c2) = (SemSim(m1, m3) + SemSim(m1, m4) + SemSim(m1, m5))/3$$

Semantic similarity between two methods takes a value between zero and one. One indicates perfect similarity whereas zero indicates no similarity. Parts of identifiers and comments in two methods are compared based on their similarity.

Definition 2.5. (Dynamic coupling): It calculates the coupling between software artefacts based on the call relationships occurring during program execution. This coupling type was introduced in 2004 by Arisholm et al. [24]. For example, let object x be an instance of class X, which is inherited from ancestor Y and object a be an instance of class A, which is inherited from ancestor B. Furthermore, Y implements

the method mY and B implements mB. When object x sends the message mB to object a, this causes a dynamic coupling between classes X and A.

There are also two coupling measures for software package which were proposed by Robert Cecil Martin [35]:

Definition 2.6. (Afferent couplings): It is a measure which indicates a package's responsibility in a software system. It is the count of classes in other packages which depend on classes within the package measured. Afferent couplings signal inward.

Definition 2.7. (Efferent couplings): It is a measure which indicates a package's dependence on other packages. It is the count of classes in other packages which are referenced by classes of the package under measurement. Efferent couplings signal outward.

2.1.2. Concepts for understanding Evolutionary Coupling

Below we are providing descriptions of the concepts that are necessary to explain details of EC:

Definition 2.8. (Module): A module is part of a software system. A software system is composed of one or more independently developed modules. Similar functionality is contained within the same module and a module is generally composed of many source files. A module is generally owned by a specific team and the team members are responsible for its development and maintenance. In the systems analysed in this study, modules are also part of subsystems. There is a one-to-many relationship between subsystems and modules. A module can be part of only one subsystem and a subsystem may have many modules.

Definition 2.9. (Software Configuration Management): SCM is an essetial part of software development to track and control changes in the software [36]. It includes practices such as version control and creating baselines. SCM makes it possible to know all changes made to software, change details, traceability to requirements as well as people involved, etc.

Definition 2.10. (Version Control / Revision Control): A component of software configuration management. It is the management of changes to software and its documentation by creating and saving revisions [36].

Definition 2.11. (Commit): Commit is an atomic operation in version control systems. With a commit operation, a group of changes is made final and these changes become available to all users.

Definition 2.12. (Transaction): A transaction is an atomic SCM operation which involves commit of a set of files to a version control system together. Most of the modern version control systems such as Git and SVN keep track of transactions. Some version control systems such as CVS save each commit separately and does not keep transactions.

Definition 2.13. (Modification Request): A MR represents a conceptual software change which includes modification of one or more source code files by one or more software developers. These changes can be defect fixes or enhancements. Any changes made to the source code are generally made based on modification requests (MR). Changes without an associated MR may be performed as part of refactoring.

Definition 2.14. (Co-change): This term is used for changes performed together on two or more software artefacts in the scope of a commit or a modification request.

Definition 2.15. (Mining Software Repositories): MSR field analyses the rich data available in software repositories such as version control and issue tracking systems to uncover interesting and actionable information about software systems and projects.

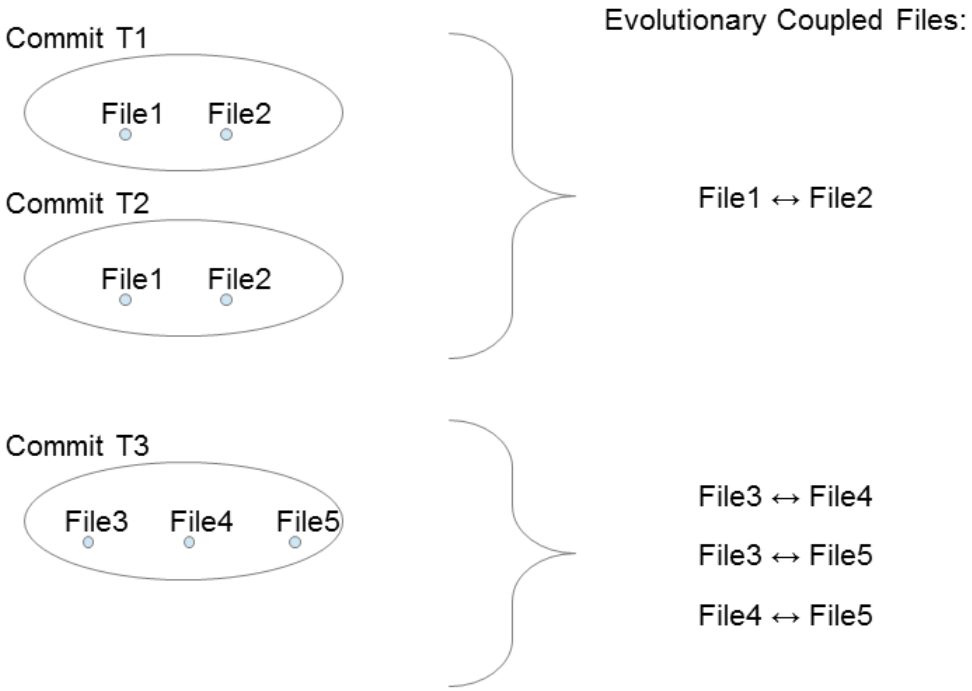


Figure 2.1. Identifying evolutionary coupling: An example.

2.1.3. Identifying Evolutionary Coupling

Two source code files File1 and File2 of a software project are evolutionary coupled if and only if they are committed together in a transaction at least once. Figure 2.1 presents a simple example. Ellipses represent commit transactions of a version control system. There are three commit transactions in our example and they are named as T1, T2 and T3.'File's in ellipses represent files changed in the scope of this particular commit. As illustrated by a bidirectional arrow in Figure 2.1, File1 and File2 are evolutionary coupled as they appear together in two commit transactions T1 and T2. Files

File3, File4 and File5 are also evolutionary coupled as they are changed together in the scope of commit transaction T3, although the strength of the coupling would be lower compared to the File1-File2 coupling. Strength here quantifies association established by a co-change from one file to another. The stronger the coupling between files, the more inter-related they are. Strength is calculated by using frequency of co-changes.

2.1.4. Association Rule Learning/Mining

Association Rule Mining (ARM, also known as Association Rule Learning) is used to calculate EC measures Support and Confidence, which are considered as the most popular EC measures [20, 21]. ARM is a rule-based machine learning method to discover relations between variables in large databases [37]. ARM was introduced by Agrawal et al. [37] in 1993. Let us apply ARM to the example illustrated in Figure 2.1 to explain ARM. The commit transactions in Figure 2.1 are first expressed as a set of items (Table 2.1). Each row in the table corresponds to a version control commit transaction, which has a unique identifier and a set of files changed in this commit transaction.

Table 2.1. Example transactions for applying association rule mining.

Transaction ID	**Items**
T1	{File1, File2}
T2	{File1, File2}
T3	{File3, File4, File5}

ARM is intended to uncover relationships between items and to represent them as association rules. For example, the following association rule can be extracted from our example: {File1} $\Rightarrow$ {File2}. {File1} and {File2} are item-sets, $\Rightarrow$ represents an association rule. This rule suggests a strong relationship between File1 and File2. Whenever File1 is changed in the scope of a commit transaction, it is very likely that File2 will also be changed in the same commit transaction. This can be regarded as the consequence of the analysis.

2.1.5. Evolutionary Coupling Measures

The two measures calculated with association rules are explained below:

Definition 2.16. (Support): The number of transactions the rule has been derived from. In other words, this is the total number of transactions, in which both files are committed.

In our example File1 was changed in two commit transactions. Both commit transactions also included changes of File2. So the support for the above rule is two.

Definition 2.17. (Confidence): It is the strength of the consequence of the rule.

In the above example, it determines how frequently File2 appears in transactions that contain File1. There are two commit transactions that contain File1. Two out of these two commit transactions contain also File2. The confidence for this rule is $2/2 = 1$. In other words, the confidence is 100%. The confidence of the rule is calculated by dividing the support count of $\{\text{File1}, \text{File2}\}$ by the support count of $\{\text{File1}\}$.

Some studies use Modification Requests (MRs) instead of transactions to group files and find EC. In this case, files changed in different transactions can still be found as evolutionary coupled if these different transactions are commited in the scope of the same MR. Based on this definition, we can use the following formula to calculate EC.

Let MR denote the set of modification requests, mr denote a specific modification request in MR, and f denote a source code file changed in the scope of mr. Based on these definitions, we calculate evolutionary coupled files and EC measures as follows:

Definition 2.18. (SECF): The set of Evolutionary Coupled Files of a file f is the set of files (except f) which are co-changed together with f in the scope of at least one

same MR. We provide its formal definition below:

$$SECF(f) = \{f_i | mr \in MR \wedge f_i \in mr \wedge f \in mr \wedge f_i \neq f\}$$

Definition 2.19. (NoECF): The total number of Evolutionary Coupled Files of a file f is defined as:

$$NoECF(f) = |SECF(f)|$$

Definition 2.20. (SECFMR): Set of Evolutionary Coupled Files of a file f in the scope of a modification request mr is defined as:

$$SECFMR(f, mr) = \{f_i | f_i \in mr \wedge f \in mr \wedge f_i \neq f\}$$

Definition 2.21. (NoECFMR): Sum of the number of Evolutionary Coupled Files of a file f for all mr's in MR:

$$NoECFMR(f) = \sum_{i=0}^{n} |SECFMR(f, mr_i)|$$

NoECF counts a coupling between two files as one even if they are coupled in the scope of multiple MRs. NoECFMR is different from NoECF in this respect. If two files are co-changed in the scope of five MRs, NoECFMR is calculated as five, whereas NoECF is calculated as one. NoECFMR considers the number of MRs, in which two files are coupled. We aim to use NoECFMR alongside NoECF to consider multiple MR co-changes, which may lead to stronger EC.

2.2. Software Measurement and Metrology

Fenton and Bieman [31] provide information about measurement fundamentals, data collection and data analysis as well as the mathematical and statistical background

of software measurement. Measurement is based on the following two concepts [31]:

(i) Entity: An object or an event in the real world such as a program, a building, maintenance phase of a program.
(ii) Attribute: A property or characteristic of an entity such as size of a program, age of a building, cost of maintenance phase of a program.

Measurement captures information about attributes of entities. Both the entity and the attribute to be measured should be specified for measurement, entities or attributes should not be used alone. There are three categories of entities which are commonly used in software measurement:

(i) Processes: Collections of software-related activities such as code review, testing and coding
(ii) Products: Any deliverables, artifacts or documents that are created in a process activity
(iii) Resources: Entities utilised by a process activity such as people

Within each category, there are two types of attributes:

(i) Internal Attributes: These are attributes which can be measured only by using the product, process or resource entity itself.
(ii) External Attributes: These are attributes of an entity that depend on the entity's context for measurement. For example, we need to involve users of software to measure usability of a software.

Fenton and Bieman [31] differentiate these two types of measurement:

(i) Direct (Base) Measurement: A direct quantification of an attribute of an entity such as the height or the weight of a person. No other attribute or entity is involved in the measurement.

(ii) Indirect (or Derived) Measurement: Direct measurement results are combined into a quantified item that reflects some the attribute of the entity to be measured. For example; velocity of an object can be measured only indirectly by using distance and time measurement. So base measurements of distance and time attributes are used to quantify velocity of the object.

Zuse [26] also contributed to the use of measurement theory in the area of software measurement. He extended classic measurement theory to the needs of software measurement [26]. Both Fenton and Zuse used the Representational Measurement Theory (RMT) approach. RMT [38] is a formal way to represent the empirical world of entities and their attributes and to capture their mathematics. Empirical relational system (ERS) captures the empirical knowledge. On the other hand, numerical relational system (NRS) captures the quantitative knowledge. Set algebra is used for both ERS and NRS. A measure is a link between ERS and NRS. These links should not cause inconsistencies. Representation Condition (RC) formalises this condition. In other words, RC asserts that a numerical model must always make sense in terms of the real world (empirical) model it is attempting to describe. For example, intuition about the weight of an object is quantified by different measures in kilograms, grams, pounds, ounces, etc.

We also use the software metrology perspective proposed by Abran [28]. Abran [28] discusses the software measurement from a measurement method point of view. Metrology is the science of measurement and includes all theoretical and practical aspects of measurement according to the definition by the International Bureau of Weights and Measures (BIPM) [39]. His work is complementary to the previous work on the measurement theory [26, 31]. Abran suggests the following steps for designing a measurement method [28]:

(i) Determination of the measurement objectives.
(ii) Characterization of the concept to be measured by specifying the entities to be measured and the characteristics (attributes) to take into account.

(iii) The setting up of the measurable construct, or of the meta - model, including the relationships across the concept to be measured (entity and attribute).
(iv) Definition of the numerical assignment rules.

Abran [28] indicates that not all software measures have strong designs and explains the quality criteria that should be expected from software measures. He describes what to look for when analysing a software measurement method. Cheikhi et al. [40] applies the metrology concepts to the Chidamber and Kemerer (CK) measures suite. They investigate how well CK measures address the metrology principles. They analyse all CK measures from metrology perspectives and provide their mapping to the metrology concepts. In this way it identifies which metrology concepts are not addressed by CK measures, which can guide improvement of the CK measures.

2.2.1. Measurement of Size

Size measurement is the most commonly used software measurement. It is used also for normalising derived measures besides quantifying size attribute of software. In the scope of this work we also measure software size and defects as well as measuring EC. Lines of Code (LOC) was chosen for size measurement. We used LOC to detect outliers in the data as well as to investigate file size as a possible confounding factor. We check for correlation between LOC and other measures.

2.2.2. Other Process Measures

EC measures are process measures. In this study we have also used some other process measures to investigate how unique the knowledge embedded in EC measures is compared to the other process measures. We used two basic process measures: number of commits (NoCommits) and number of developers (NoDevs). NoCommits is the total number of all commits in which a particular file is included. Similarly, NoDevs is the total number of all developers who changed a particular file.

In the scope of this work we also measure defects. Defect (or bug) is defined as a problem in a software product which does not meet a software requirement or end-user expectations [41]. We use the following measures for defects: Number of defects reported for a file (NoD) and Defect Density (DD). Defect Density is the number of defects found in software/module during a specific period of operation or development divided by the size of the software/module. Using defect density as normalised measure in our study mitigates the risk of size as a possible confounding factor. We employ the following formula for calculating defect density:

$$DD = NoD/LOC$$

2.3. Software Maintenance and Evolution

The concept of software evolution was first introduced by Manny Lehman in the 1970s. Along with research on IBM's OS 360 and OS 370 operating systems, he began to build eight commonly accepted laws known as Software Evolution Laws [42]. These laws are general observations for E-type systems, but not absolute laws like nature laws. E-type systems are written to perform some real-world activity. Their behaviour is strongly linked to the environment in which it runs. Furthermore it needs to adapt to varying requirements and circumstances in that environment. According to the classification of Lehman and Belady [42], the E-type systems are a part of the real world and they work integrated with it and therefore they are in constant change. The E-type system and its environment bring together an evolving feedback system. Eight laws that Lehman revealed are provided in Table 2.2 with the publication dates. According to these laws, software systems are subject to continuous changes to adapt to new and changing requirements. This leads to software ageing in the long term. In other words, the size and the complexity of systems increase, while their quality decreases.

Lehman used the term Software Evolution to emphasize the difference with the post-deployment activity of software maintenance. The ISO Standard 12207 (2008)

Table 2.2. Lehman's laws of software evolution.

No	**Date**	**Software Evolution Law**	**Description**
1	1974	Continuing Change	E-type systems should constantly adapt to changing environmental conditions otherwise they become progressively less satisfactory
2	1974	Increasing Complexity	Complexity increases over time as long as no work is done to maintain or reduce the complexity of the E-type systems
3	1980	Self Regulation	The evolution process of E-type systems is self-regulating with the distribution of product and process measures close to normal
4	1978	Conservation of Organisational Stability (invariant work rate)	The average net efficiency ratio during the evolution of an E-type system does not change over the life of the product
5	1978	Conservation of Familiarity	Throughout the evolution of the E-type system, all individuals involved must maintain the mastery of its content and behaviour for successful evolution.
6	1991	Continuing Growth	In order to maintain user satisfaction, the functional size of E-type systems must be constantly increased during their lifetime
7	1996	Declining Quality	The quality of E-type systems tends to decrease as long as the E-type systems are not adapted to the changes of the operational environment
8	1996	Feedback System	The evolutionary processes of E-type systems constitute multi-layered, multi-agent, multi-loop feedback systems

(Systems and software engineering — Software life cycle processes) [43] defines maintenance as follows: Software product undergoes modification to code and associated documentation due to a problem or the need for improvement. The objective of software maintenance is to modify the existing software while preserving its integrity. According to the 14764-2006 - IEEE ISO/IEC 14764:2006 [44] (Standard for Software Engineering - Software Life Cycle Processes - Maintenance), software maintenance involves any modification of a software product after its delivery to fix defects, to improve performance or other attributes, or to adapt the software to a changing environment.

Lientz and Swanson subdivided maintenance activities into four categories [45]: Corrective maintenance is any modification of a software product after its delivery to correct reported faults. This type of maintenance activity is reactive. On the other hand if a modification is performed for improving the software after its delivery, it is categorised as perfective maintenance. This type of activity is proactive. Sometimes we need to modify software because the environment in which the software runs changes. This type of maintenance is called adaptive maintenance. The last maintenance category is preventive maintenance. They are any software modifications performed to prevent problems before they emerge. This is also a proactive activity.

2.4. Statistical Methods for Empirical Research

In this subsection we provide background about some statistical methods which are used in our empirical research in this dissertation.

2.4.1. Correlation Analysis

Correlation analysis is commonly used to uncover relationship between two variables. It is very useful as it can be exploited in practice for predictions. Spearman and Pearson are two correlation analysis methods widely used in the scientific studies. Pearson correlation analysis requires normally distributed data. Since the data was not normally distributed in our study, we used Spearman's rank correlation analysis. Spearman correlation analysis was used to find the relationship between EC and defect

measures. Spearman's rank correlation analysis is a non-parametric test of correlation and assesses how well a monotonic function describes the association between variables. This is done by ranking the sample data separately for each variable. We used the Shapiro-Wilk test [46] to check for normality of the data. This test has the following null-hypothesis: the population is normally distributed. The p-value is checked to reject or accept the null-hypothesis. We reject the null-hypothesis if the p-value is less than the chosen alpha level (0.05). In other words, this suggests that the data tested is not normally distributed. Razali et al. [47] report that Shapiro-Wilk is the most powerful normality test.

We set the p-value (significance level) for Spearman correlation analysis to 0.05. If the data from the study results in a p-value of less than 0.05, we conclude that the correlation is significant. The correlation coefficient or correlation strength is represented by ρ (rho). It expresses the relationship between EC and software defects by a value between -1 and 1. ρ (rho) values of 1 or -1 indicate perfect positive or negative correlation, respectively. Values close to 0 indicate absence of correlation between measures. We considered ρ (rho) values less than 0.1 to be trivial, between 0.1 and 0.3 as low, between 0.3 and 0.5 as moderate, between 0.5 and 0.7 as high, between 0.7 and 0.9 as very high, and above 0.9 as almost perfect, as suggested by various studies in the literature [48, 49].

2.4.2. Multivariate Logistic Regression

Binary logistic model is used to estimate the probability of a binary dependent variable based on one or more independent variables. In our study, multivariate regression analysis is used to explore the relationship between EC (independent variable) and defects (dependent variable) to understand how helpful EC measures are in defect analysis compared to other process metrics (we build correlation models rather than prediction models). The following describes the steps taken to build a logistic regression model for the EC metrics, process metrics and the presence or absence of defects. The first step is to binarise the defect count such that a data-point is labelled defective if the defect count is greater than 0. Then we build a logistic regression model using

all terms and no interactions. Having built the model we test for multicollinearity to find any independent variables which are correlated. Then we build a model which includes interaction terms and identify terms which are correlated. Finally, we build an interaction model without correlated terms and apply stepwise reduction to remove terms which are not significant.

By using regression models, we aim to determine whether a particular independent variable really affects the dependent variable, and to estimate the magnitude of that effect, if any.

We diagnose collinearity through variance inflation factor (VIF) analysis [50]. We used 2.5 as the cut-off value for the simple model and 10 for the interaction model where collinearity naturally occurs by default. These cut-off values are commonly used and as a rule of thumb results greater than these cut-off values are considered as problematic. If a VIF value is greater than the cut-off value, the metric with the largest VIF is removed and the model re-built until all VIF values are less than the cut-off value.

3. RELATIONSHIP BETWEEN EC AND DEFECTS

Today our daily lives depend on large industrial software systems in various domains such as energy, finance, telecommunication, etc. Although it is very important to understand these software systems and their quality, large industrial systems have rarely been empirically studied to understand the relationship between EC and defects. The previous studies on EC focused mainly on open source projects. Also, most of these studies did not investigate large projects. Large industrial projects have very different software development processes and culture than the open source projects. Furthermore, they have also different quality characteristics such as low-defect density and different defect types. Therefore it is crucial to perform more empirical studies on large industrial systems to understand the relationship between EC and defects.

Software constantly changes for many reasons [42,51–53]. Studies have shown that changing software may be a defect-prone activity [18,54–56]. Code that is changed most frequently is likely to be the most defect-prone [55–58]. EC could explain some of this defect-proneness since when code with high EC is changed, a high number of changes must be made to related parts of the system. The locations of these related changes may be scattered within the application or even across applications in a software ecosystem. Correctly making related changes across these locations is likely to be challenging. Developers may miss some locations which should have been co-changed and this may cause unforeseen ripple effects and problems.

In this chapter, we analysed the correlation between EC measures and the number of defects and defect density in two large software systems in industrial software development environments. Correlation analysis is performed separately for each module. We also built logistic regression models. Multivariate regression analysis is used to explore the relationship between EC (independent variable) and defects (dependent variable) to understand how helpful EC measures are in defect analysis compared to other process metrics (we build correlation models rather than prediction models). We also analysed the relationship between EC and defect types. Our research questions

are as follows:

- (RQ1) What is the relationship between evolutionary coupling and software defects?

The results of our study showed that there was, in general, a relationship between EC and software defects in the software maintenance / evolution phase of the industrial software systems under study. We detected a positive correlation between EC measures and defects. Compared to other process measures such as the number of commits and the number of developers, EC measures seem to contain additional, sometimes important, information about defects: for every additional evolutionary coupled file in a module, the module is 8% more likely to be defective. However, correlation strength varied across modules and in some modules EC and defects were not correlated. Based on these findings, we added the following research question to our study:

- (RQ2) What factors explain why the relationship between evolutionary coupling and software defects is different for different modules?

Modules which were small in terms of Lines of Code (LOC) and developer numbers tended to be less correlated with EC. Fewer defects due to EC seem to occur in small modules. EC also appeared to be more highly correlated with some types of defects such as code implementation, acceptance criteria and analysis problems. Overall, regression analysis showed that EC may be useful for explaining defects in industrial systems.

We first published our results on the empirical studies at a large company in [2–4]. Then we expanded our empirical study to another large software company as well as the research scope and puslished in [1]. In particular, this chapter makes the following contributions:

(i) Firstly, we analyse large commercial systems which have rarely been empirically studied to understand the relation between EC and defects.
(ii) Secondly, we show that the effect of EC on defects varies depending on the module.
(iii) Thirdly, the explanatory power of EC measures varies depending on defect types and module features such as size and developer activity.
(iv) We implemented a novel MSR tool to mine various software repositories such as SCM, issue tracking and HR systems and to detect ECs.

This chapter is organised as follows: In the next section, we summarise related work. From section 3.1 to 3.5, we present our methodology including measures, data extraction and analysis methods. Section 3.6 shows the results of applying our methodology to two industrial systems. The discussions and threats to validity of this study are then addressed in sections 3.7 and 3.8, respectively. Finally, in Section 3.9, we summarise and present our conclusions.

3.1. Study Context

We performed our study on two large industrial systems. One of the systems was a large financial legacy system that had evolved for over 25 years to support the back-end business processes of a large financial institution (henceforward known as 'Company 1'). Much of the code was written in PL/I and COBOL, but there were also files written in JCL (Job Control Language), a scripting language used on mainframes to develop a batch job. The system consisted of 20 subsystems and 274 modules. The company started using a version control system in 2009. We analysed all subsystems and modules between 2009 and 2013 and the total size of the system was 87 million LOC, consisting of 150K individual files. The applications analysed in this study are back-end banking applications. Company 1 uses a 'modified' waterfall model for software development.

The other system studied was a large telecommunications system written in Java (henceforward known as 'Company 2'). The company had used a version control system (SVN) since 2006; we analysed 4 subsystems and 11 modules between 2006 and 2013.

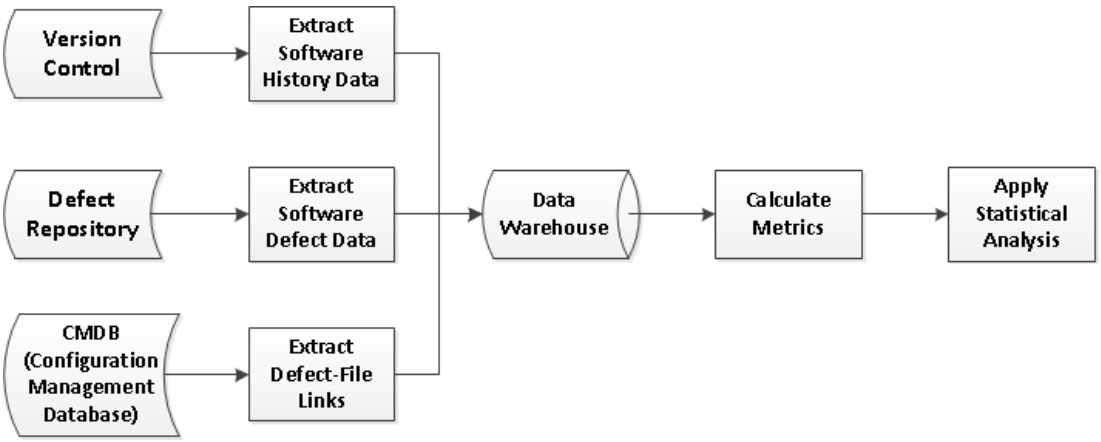


Figure 3.1. Data collection overview.

Company 2 uses an agile methodology as well as Test-Driven Development (TDD). The applications analysed in this study are web applications. Detailed information about the systems under study is given in Table 3.1.

Table 3.1. Summary about industrial systems under study.

	Company 1	**Company 2**
Programming Languages	Cobol,PL/I,Jcl	Java
Domain	Finance	Telecommunications
Versioning System	CA SCM	SVN
Defect Tracking System	Developed in-house	JIRA
Number of Software Sub-Systems	20	4
Number of Software Modules	274	11
Total Number of Developers	460	25
Total Software Size (LOC)	87 M	310 K
Total Number of Files	150 K	26 K
Total Number of File Versions	192 K	180 K
Total Number of Commits	50 K	24 K
Analysis Period	2009-2013	2006-2013
Percentage of Files Changed	%11	%15

3.2. Data Collection

We collected source code data from SVN and CA SCM version control systems, defect data from JIRA and in-house developed defect repositories and the link between source code and defects from CMDB at Company 1 and 2.

Figure 3.1 presents an overview of our approach to mine the data. We developed adapters for the three different data sources: version control, defect repository and CMDB. The output of adapters containing the data retrieved from the data source for the specified period are stored in a database. For source code data, we fetch all versions created during the specified period in the version control system and store them in the database. We applied static code analysis on each file revision providing method- and program-level static metrics (discussed in the next sub-section). Commit information such as the developer id that created the version, date of creation and the related problem/request/project id were available from the source code repository. We applied filtering to remove large commits that may have contained logically irrelevant changes. Commits containing more than 30 files were ignored and were not considered while calculating EC measures.

In CMDB, each software product is defined as a separate configuration item (CI) and each change is recorded and linked to the corresponding CI. In our study, we collected all source-code related changes performed in the scope of defect fixing or enhancement on the software product analysed over the defined period. In CMDB and JIRA, two different sources for a change were defined: Problem or Request. We could therefore distinguish between bug-fixing and enhancement.

3.3. Data Sources

3.3.1. Code Repositories:

Source code repositories are primarily used for storing and managing changes to source code artifacts. The full history of changes, the owner of the change, date of

the change and even the corresponding requirement or project task can be extracted. The tool used at Company 1 for managing the source code repository was a product of Computer Associates (CA), called CA Software Change Manager (CA SCM) [59]. CA SCM provides change management in addition to its version control functionality. A developer must make changes in CA SCM in a change package, similar to the notion of "change set" in other SCM tools. A package groups and keeps all related changes together, corresponding to the same defect fix or enhancement. The versioning system used at Company 2 was Apache Subversion (SVN) [60]. SVN is a popular versioning tool and is used by nearly half of all open source projects according to openhub.net [2] .

3.3.2. Defect Repositories: Company 1 (Finance):

We mined the defect repository to collect defect data reported. The defect repository at Company 1 was developed in-house by the company.

Mapping between Defects and Source Code: We followed different approaches for the two companies for finding a mapping between defects and source code. For Company 1, we used the Configuration Management Database (CMDB) for this purpose. For both companies, we assumed that files involved in a defect fix contained the defect. This assumption holds for most of the cases at both companies. Alternativelly, SZZ (Sliwerski-Zimmerman-Zeller) [54] method might be used to locate fix-inducing changes by linking version control to defect repository.

Configuration Management Database (CMDB): Many companies store information related to components of their information system in a CMDB which contains data describing the following entities [61]:

- managed resources such as computer systems and application software,
- process artifacts such as incident, problem and change records,
- relationships among managed resources and process artifacts.

[2]The Open Hub tracks more than 650,000 free and open source software (FOSS) repositories and generates several statistics on the hosted source.

In our study, we used the CMDB system at Company 1 to extract data about the relationship between Modification Requests (MR) and source code. The CMDB system was developed in-house by the company.

3.3.3. Defect Repository - Company 2 (Telecommunications):

Company 2 used JIRA [62], a proprietary defect tracking product, developed by Atlassian.

Mapping between Defects and Source Code: For Company 2, we used the defect IDs provided in the SVN commit comments by developers and the revision numbers provided in JIRA issues. For both companies, we assumed that files involved in a defect fix contained the defect. The percentage of fixed bugs linked to version control is changed between 73% and 79% yearly. The bugs which are not linked to version control include the defect fixes which do not require source code change and version control commit such as database related fixes. The mappings from SVN commit to defect and from JIRA issue to SVN commit were generally consistent.

3.4. Descriptions of Measures

Table 3.2 lists all the measures used in this study. These measures are defined in the background chapter of this dissertation. The following sections will set the context for this study.

3.4.1. EC Measures:

In the companies under study, any changes made to the source code were made based on modification requests (MR). A MR represents a conceptual software change which includes modification of one or more source code files by one or more software developers. These changes can be defect fixes or enhancements. We used an MR-based approach to calculate EC. In this study we used two EC measures NoECFMR and NoECF defined in the Background chapter.

Table 3.2. Summary of measures used in the study.

Measure	**Granularity**	**Description**
NoECF	File level	The number of unique Evolutionary Coupled Files of a file
NoECFMR	File level	Sum of the number of co-changed files (in the scope of an MR) of a file
TNEFC	Module level	Total Number of Evolutionary File Couplings in a Module
NoD	File level	Number of defects reported for a file
DD	File level	Defect Density
TND	Module level	Total Number of Defects of a Module
NoCommits	File level	Number of Commits - Process Metric for comparison purposes
NoDevs	File level	Number of Developers - Process Metric for comparison purposes
TNF	Module level	Total Number of Files in a Module
TNFR	Module level	Total Number of File Revisions in a Module
TNDVLP	Module level	Total Number of DeVeLoPers contributing to a Module
LOC	File level	Lines of Code, Size measure
TFSC	Module level	Total File Size Change in LOC for a Module

The following three issues were considered in the calculation of EC measures: i) the level at which measures are taken, ii) the approach for grouping files, and iii) the boundary for finding coupled files. We calculated EC measures at file level; we chose file level, since defects are mapped to files in the companies under study. One approach for grouping file changes is using commit transactions of versioning systems that are the unique commit operations of a developer. In this approach it is assumed that developers commit logically coupled files within a transaction. The system at Company 1 was a legacy system and developers rarely committed more than one file in one transaction. Therefore, we found that a transaction-based approach was not

appropriate to detect EC. We followed an MR-based approach and grouped the file changes according to the associated MR numbers [18]. In our approach, file changes spanning multiple transactions were grouped together if they were associated with the same MR. The third issue considered for EC calculation was the boundary for finding coupled files. We chose module level to find coupled files that resided in the same module. In other words, we consider EC only within module boundaries. Alternative module boundaries could be subsystem or system level, which considers cross module couplings. In this study, we ignore any cross-module ECs.

3.4.2. Defect Types:

We used the defect types listed in Table 3.3 and provide their descriptions. This defect type classification was used by Company 2 and each defect reported was tagged by one or more defect types (the defect repository stored the defect type data for each defect). The defect types in Table 3.3 are ordered based on the defect type codes used by the company.

3.5. Analysis Method

3.5.1. Analysis Method for Answering RQ1:

Spearman correlation analysis was used to find the relationship between EC and defect measures. Since the data is not normally distributed, we apply Spearman's rank correlation analysis. Spearman's rank correlation analysis is a non-parametric test of correlation and assesses how well a monotonic function describes the association between variables. This is done by ranking the sample data separately for each variable. We used the Shapiro-Wilk test [46] to check for normality of the data. This test has the following null-hypothesis: the population is normally distributed. The p-value is checked to reject or accept the null-hypothesis. We reject the null-hypothesis if the p-value is less than the chosen alpha level (0.05). In other words, this suggests that the data tested is not normally distributed.

Table 3.3. Defect type list.

No	Defect Root Causes	Descriptions
1	Bad Data	defects caused by invalid / unexpected data at persistence storage (databases)
2	Wrong Properties	defects caused by properties set incorrectly for the application like timeout, thread pool size, etc.
3	Default Value	defects caused by default values of the deployed application
4	Process Failure	defects caused by problems in software processes such as insufficient communication between teams
5	3rd Party System defect	defects caused by problems occurred at other systems running in the same environment
6	Database Upgrade Failure	defects caused by incomplete/unsuccessful database schema updates / migrations
7	Data Fix Errata	defects caused by incorrect scripts that are added to fix the data at persistence storage as part of another defect
8	CRM defect	defects caused by problems at Customer Relationship Management (CRM) system
9	Functionality Not Implemented Yet	defects caused by API methods or functionalities which are not implemented yet or not deployed with the existing software version
10	Unexpected Functionality	defects experienced by users as an unexpected functionality such as response failures and performance problems
11	Incorrect Environment	defects caused by non-satisfied prerequisites at the running environment of the application
12	Acceptance Criteria Impl	defects caused by missing / incomplete / incorrect automated acceptance tests
13	Database disconnect/reconnect error	defects caused by database (persistence storage) connection problems
14	Analysis	defects caused by missing / incomplete/ incorrect requirements / user stories
15	Incorrect Config	defects caused by incorrect application configuration such as versions of feeds / services pointed
16	Infrastructure Issues	defects caused by application infrastructure problems such as exhausted server swap memory or incorrect load balancing
17	Acceptance Criteria	defects caused by missing / incomplete / incorrect acceptance criteria
18	Missing or incomplete data migration	defects caused by incomplete / missing data migrations affecting a specific environment (test, qa, etc.)
19	User Error	defects or unexpected behaviour due to a invalid usage of the application
20	Not An Issue	defects which are not interpreted as defects (not supported scenario, no longer required behaviour, already fixed, etc.)
21	Test Implementation	defects caused by missing / incomplete / incorrect automated (unit / integration) tests
22	Code Implementation	defects caused by the defects inserted during implementation of new features or defect fixing

We set the p-value (significance level) for Spearman correlation analysis to 0.05. If the data from the study results in a p-value of less than 0.05, we conclude that the correlation is significant. The correlation coefficient or correlation strength is represented by ρ (rho). It expresses the relationship between EC and software defects by a value between -1 and 1. ρ (rho) values of 1 or -1 indicate perfect positive or negative correlation, respectively. Values close to 0 indicate absence of correlation between measures. We considered ρ (rho) values less than 0.1 to be trivial, between 0.1 and 0.3 as low, between 0.3 and 0.5 as moderate, between 0.5 and 0.7 as high, between 0.7 and 0.9 as very high, and above 0.9 as almost perfect, as suggested by various studies in the literature [48, 49].

Correlation analysis was applied on each module separately to obtain rho, p and $StdErr$ values for each. We used histograms to summarise the correlation results and the SPSS [63] tool was used for the statistical analysis.

After correlation analysis was performed, we applied multivariate logistic regression and multicollinearity analysis with basic process metrics such as number of commits, number of developers and prior number of defects as well as EC metrics. With this analysis, we are aiming to identify the relationship between metrics, as well as metrics that do not add any new knowledge about defects.

3.5.2. Analysis Method for Answering RQ2:

We used box plots to determine differences between the modules where significant correlation was or was not observed. We drew box plots for the following measures:

- TNF: Total Number of Files in a Module
- TNEFC: Total Number of Evolutionary File Couplings in a Module
- TNFR: Total Number of File Revisions in a Module
- TNDVLP: Total Number of Developers contributing to a Module
- TND: Total Number of Defects of a Module
- TFSC: Total File Size Change in LOC for a Module

These measures were chosen based on availability and their power to reflect different attributes of modules characterising size, developer activity and defects. Many studies in the literature suggest that size is generally an important factor. Since EC is dependent on developer activity, we have also added it as a factor. To check whether the difference is statistically significant, we apply a t-test if data is parametric and a Mann-Whitney test if non-parametric. We again take a significance level of 0.05.

To check the role of defect types, we repeated the correlation analysis between EC and defect measures, but this time for each 22 defect type. We aimed to find defect types that were likely to be related to EC and we checked the distribution of defect types for each module.

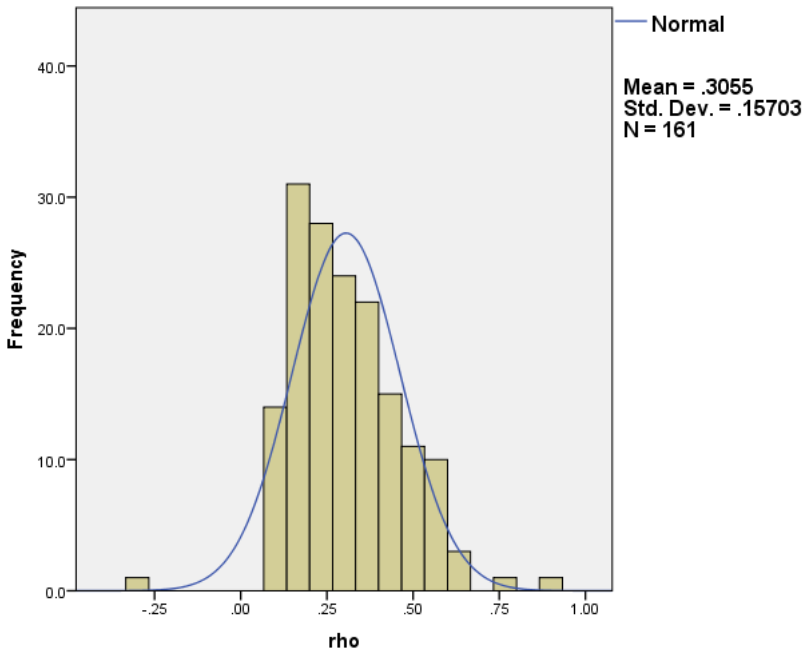


Figure 3.2. Company 1: Histogram of Spearman *rho* values for correlation between EC (NoECF measure) and number of defects (NoD).

3.6. Results

3.6.1. RQ1: Correlation Analysis Results

For 161 out of 274 (59%) software modules analysed at Company 1 and 6 out of 11 software modules at Company 2, we observed significant correlation ($p < 0.05$)

between the number of defects (NoD) and EC measures using Spearman's analysis. A Shapiro-Wilk test indicated that data distribution was not normal ($p = 0.0 < 0.05$) and so consequently, we used Spearman's analysis. For 32 out of 113 modules at Company 1 for which no significant correlation was observed, the number of commit values were either zero or low values ($<= 10$). EC measures need a lead time period (Zimmerman et al. [21]) and sufficient version control activity (prerequisite for EC measurement). Otherwise, they may not be useful. For Company 2, the modules for which no significant correlation is observed were all small in size and the number of defects were also low for these small modules.

The distribution of *rho* values of these 161 modules at Company 1 can be seen in the histogram in Figure 3.2. This figure only shows the histogram of Spearman *rho* values for correlation between NoECF and NoD. The histogram for correlation between NoECFMR and NoD is very similar to the former one, which can be seen in Figure 3.3. The correlation observed was generally low and moderate. For 21 modules, high and very high correlation was observed. Figures 3.4(a) and 3.4(b) show the distribution of *rho* values on the histogram for Company 2. The correlation values do not seem to be high but while interpreting these results we should consider that we are only analysing one factor among many factors, which can have a relationship with defects. From this perspective, having 59% of modules with significant correlation and low to moderate correlation strength is a very important result.

If we compare the analysis results of the two companies, we observe that Company 2 has relatively fewer modules with high correlation values. The practices such as agile and TDD used by Company 2 may have affected this result. Such practices may lead to lower coupling in systems. This result may also be due to the different architectures used by these two systems. Company 2 used the Model-View-Controller (MVC) architectural pattern in its projects. MVC splits software application into three layers and separates presentation, data and flow control. Whereas the architecture in the Company 1 systems is more ad hoc since these legacy systems have been evolved over a long period. Organizational structure of the companies may also have impact on the design and coupling of the systems analysed as suggested by Conway's law [64].

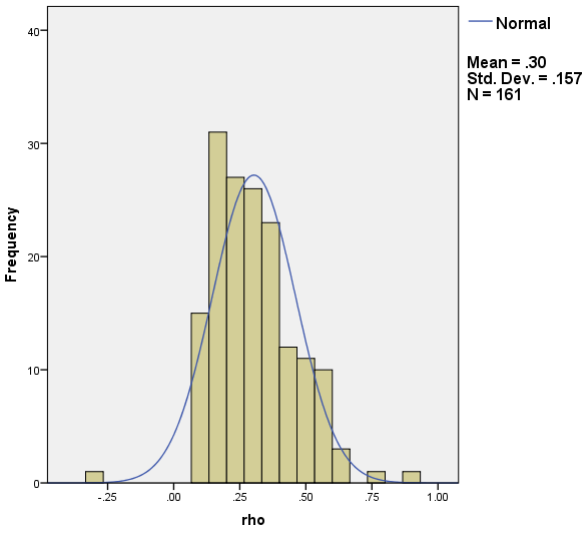
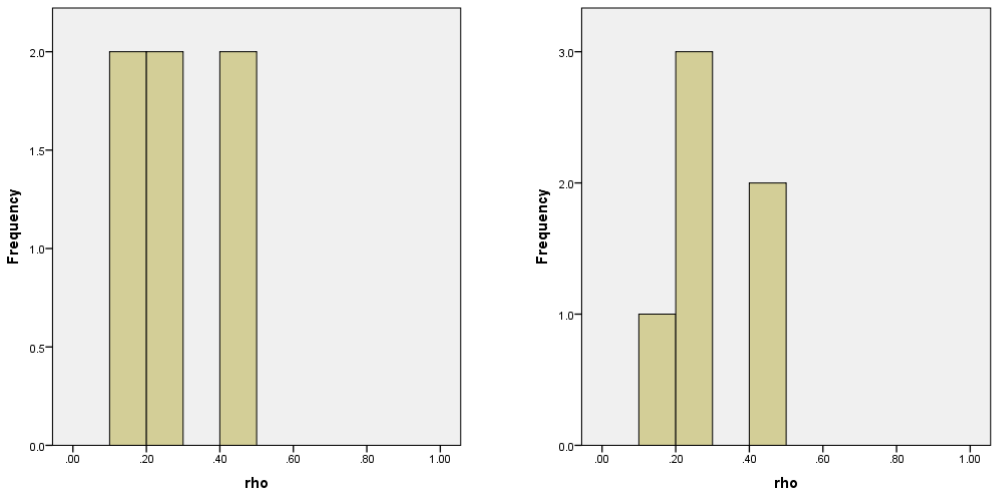


Figure 3.3. Company 1: Histogram of Spearman *rho* values for correlation between EC (NoECFMR measure) and number of defects.

However, this should be investigated further.



(a) Correlation between NoECF and NoD. (b) Correlation between NoECFMR and NoD.

Figure 3.4. Company 2: Histogram of Spearman *rho* values for correlation between EC measures and number of defects (NoD).

We also applied Spearman's analysis for EC measures and defect density (DD). For 147 out of 274 software modules analysed at Company 1, we observed significant correlation ($p < 0.05$) between DD and EC measures by using Spearman's analysis. The distribution of *rho* values can be seen in the histogram in Figure 3.5. Although there are slightly fewer modules identified as significant compared with the previous analysis,

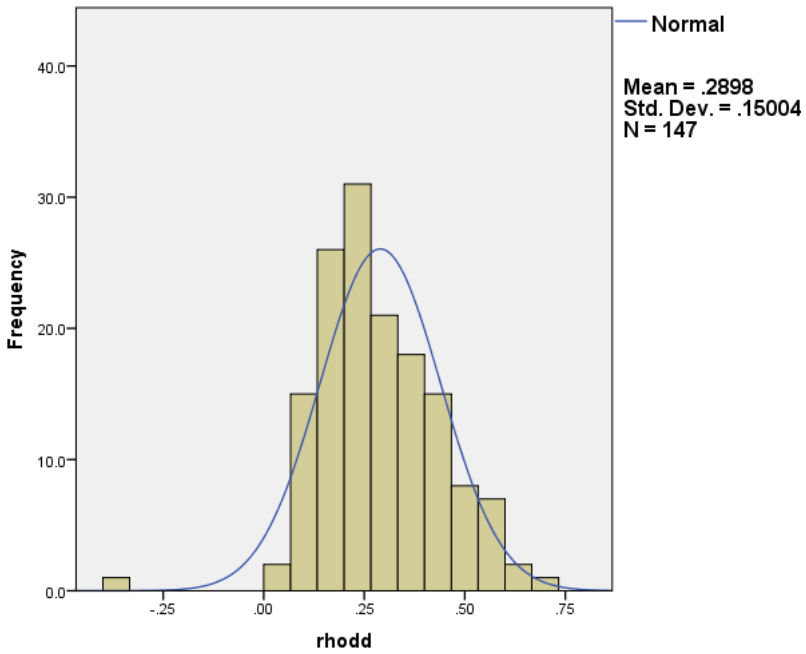
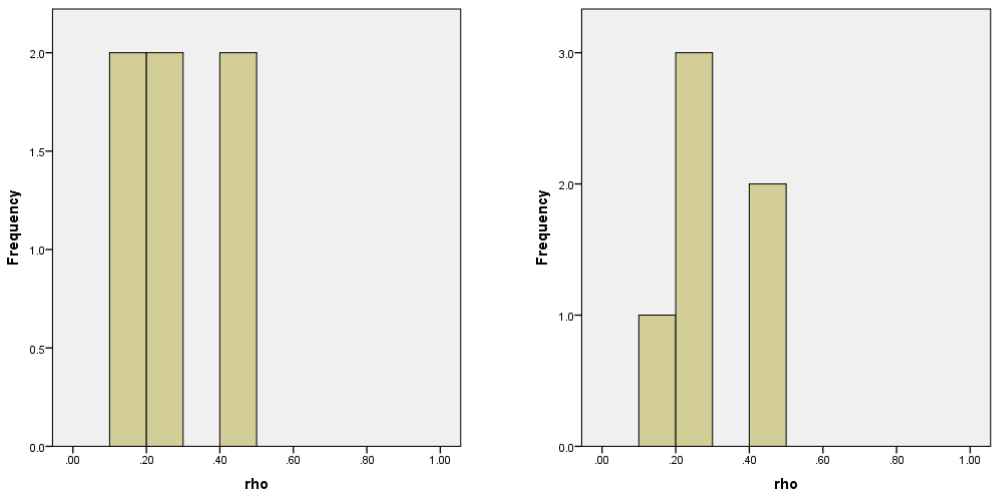


Figure 3.5. Company 1: Histogram of Spearman *rho* values for correlation between EC (NoECF measure) and defect density (DD).

the distribution of *rho* values shown in the two histograms shows great similarity. In keeping with the previous analysis results, the correlation observed was generally low and moderate; for a small number of modules, high correlation was observed. The results for Company 2 were similar to the previous analysis results as shown in Figure 3.6.



(a) Correlation between NoECF and DD. (b) Correlation between NoECFMR and DD.

Figure 3.6. Company 2: Histogram of Spearman *rho* values for correlation between EC measures and defect density (DD).

We have also applied Spearman's correlation analysis for basic process metrics such as number of commits, number of developers and prior number of defects for comparison purposes. Table 3.4 summarises the results. We have found only moderate correlation (0.57) between EC measures and two process measures (NoCommits and NoDevs). The correlation between EC measures and defects (NoD) was slighter higher than the correlation between defects and the other two process measures (NoCommits and NoDevs).

Table 3.4. Spearman correlation results of the process metrics.

	NoECF	**NoECFMR**	**NoD**	**NoCommits**
NoECFMR	ρ=.99 p=.000			
NoD	ρ=.28 p=.000	ρ=.28 p=.000		
NoCommits	ρ=0.57 p=.00	ρ=0.57 p=.00	ρ=.24 p=.00	
NoDevs	ρ=0.57 p=.00	ρ=0.57 p=.00	ρ=.24 p=.000	ρ=0.99 p=.00

3.6.2. RQ1: Regression Analysis Results

After correlation analysis, we applied multivariate logistic regression to build models which indicate files which are likely to be defective. First, we built a logistic regression model using all terms and no interactions (Table 3.5).

Table 3.5. First model with all terms and no interaction.

	Estimate	**Std. Error**	**z value**	**Pr(> |z|)**
(Intercept)	-2.9369	0.0122	-240.71	0.0000
NoECFMR	0.0287	0.0048	5.95	0.0000
NoECF	0.0274	0.0056	4.88	0.0000
NoCommits	0.0213	0.0059	3.63	0.0003
NoDevs	0.8340	0.0250	33.41	0.0000

Having built the model we test for multicollinearity to find any independent variables which are correlated (Table 3.6). We assess the variance inflation factors (VIF). A VIF > 2.5 is considered problematic requiring one or more variables to be removed. 'NoECFMR' and 'NoECF' are identified as being correlated and therefore we remove 'NoECFMR' from the model (Table 3.7). Multicollinearity analysis results and odds ratio (OR)[3] effect sizes after removing 'NoECFMR' are also provided in Table 3.8 respectively. The odds ratio results suggest a rather low relation between EC and defects, although slightly higher than that of the number of commits.

Table 3.6. Test for multicolinearity.

	VIF
NoECFMR	21.29
NoECF	21.12
NoCommits	1.87
NoDevs	2.10

Table 3.7. Model for all terms without NoECFMR.

	Estimate	**Std. Error**	**z value**	**Pr(> |z|)**
(Intercept)	-2.9432	0.0122	-241.67	0.0000
NoECF	0.0601	0.0014	44.02	0.0000
NoCommits	0.0247	0.0057	4.33	0.0000
NoDevs	0.8362	0.0247	33.88	0.0000

Table 3.8. Multicolinearity and odds ratio effect size (without NoECFMR).

	VIF
NoECF	1.29
NoCommits	1.85
NoDevs	2.09

	OR
(Intercept)	0.05
NoECF	1.06
NoCommits	1.03
NoDevs	2.31

[3]An Odds Ratio greater than 1.0 indicates that an increase in the variable will increase the propensity for the file to be defective.

Having identified individual variables which make a significant contribution to the logistic regression model, we built a model which includes interaction terms (Table 3.9) and identify terms which are correlated (Table 3.10). Again, VIF values are highly likely to be correlated because we are using interaction terms, therefore $VIF > 10$ is considered problematic (Table 3.10). Odds ratio effect sizes for this model are provided in Table 3.11.

Table 3.9. Model with interaction terms.

	Estimate	**Std. Error**	**z value**	**Pr(> |z|)**
(Intercept)	-3.0191	0.0130	-232.56	0.0000
NoECF	0.0817	0.0018	46.07	0.0000
NoCommits	0.0888	0.0116	7.63	0.0000
NoDevs	1.2002	0.0310	38.74	0.0000
NoECF:NoCommits	-0.0021	0.0007	-3.13	0.0017
NoECF:NoDevs	-0.0356	0.0019	-18.35	0.0000
NoCommits:NoDevs	-0.0576	0.0064	-8.94	0.0000
NoECF:NoCommits:NoDevs	0.0024	0.0003	7.40	0.0000

Table 3.10. Multicolinearity (with interaction terms).

	VIF
NoECF	2.52
NoCommits	8.20
NoDevs	3.79
NoECF:NoCommits	10.81
NoECF:NoDevs	7.34
NoCommits:NoDevs	10.21
NoECF:NoCommits:NoDevs	14.59

Next we built an interaction model without correlated terms and applied stepwise reduction to remove terms which were not significant (Table 3.12). The multicollinearity analysis results and odds ratio effect sizes for this model are also provided in Table

Table 3.11. Odds ratio effect size (with interaction terms).

	OR	**2.5 %**	**97.5 %**
(Intercept)	0.05	0.05	0.05
NoECF	1.09	1.08	1.09
NoCommits	1.09	1.07	1.12
NoDevs	3.32	3.12	3.53
NoECF:NoCommits	1.00	1.00	1.00
NoECF:NoDevs	0.96	0.96	0.97
NoCommits:NoDevs	0.94	0.93	0.96
NoECF:NoCommits:NoDevs	1.00	1.00	1.00

3.13, respectively. This analysis shows how unique the knowledge embedded in EC measures is compared to the other process metrics.

The final model includes the following significant terms: NoECF, NoCommits, NoDevs and the interaction of NoECF with NoDevs. All terms apart from the interaction term are greater than 1.0 showing that when the independent variable increases, the propensity of a file to be defective increases. The interaction term (NoECF:NoDevs 0.98) is slightly less than 1.0 indicating that as both increase together the linear model is adjusted to marginally decrease the increasing propensity of the model to predict a file as being defective.

To check the relationship between EC measures and defect-proneness of files from a different perspective, we drew box plots for EC measures of files with and without defects. A separate box plot for each module was created and for some of the modules these can be seen in Figure 3.7. 1 represents files with defects, 2 represents files without any defects. These box plots show a clear separation between files with defects and without defects.

We also performed manual analysis for some highly evolutionary coupled files and their defects to show how software defects were influenced by EC. In some defect

Table 3.12. Reduced model with interaction terms with no collinearity.

	Estimate	**Std. Error**	**z value**	**Pr(> |z|)**
(Intercept)	-2.9878	0.0126	-236.95	0.0000
NoECF	0.0747	0.0015	48.92	0.0000
NoCommits	0.0324	0.0054	5.95	0.0000
NoDevs	0.9823	0.0248	39.63	0.0000
NoECF:NoDevs	-0.0184	0.0009	-20.27	0.0000

Table 3.13. Multicolinearity and odds ratio effect size (final model).

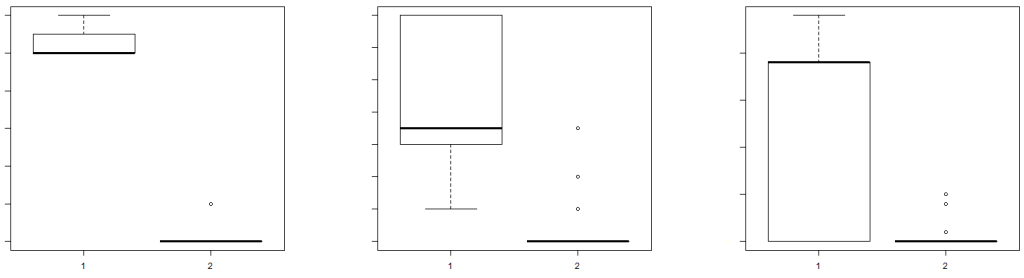
	VIF
NoECF	1.89
NoCommits	1.94
NoDevs	2.39
NoECF:NoDevs	2.40

	OR	**2.5 %**	**97.5 %**
(Intercept)	0.05	0.05	0.05
NoECF	1.08	1.07	1.08
NoCommits	1.03	1.02	1.04
NoDevs	2.67	2.54	2.80
NoECF:NoDevs	0.98	0.98	0.98

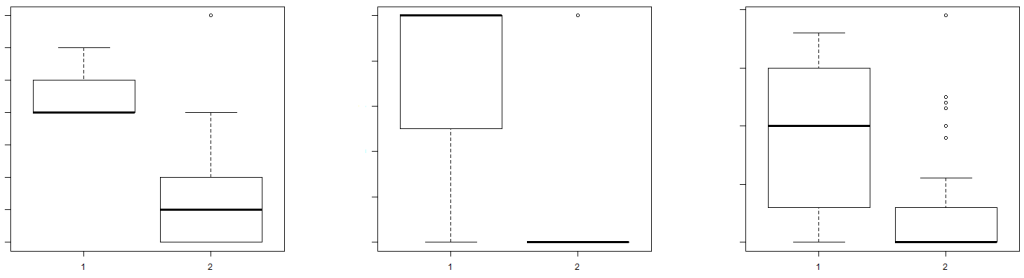
instances, a highly evolutionary coupled file was changed but this change was not accumulated to all coupled files correctly. This was the root cause of the fault. There was no structural or dynamic coupling between these files. We also observed similar instances but across different modules managed by different teams. A change made in a module was not accumulated to the evolutionary coupled modules. For some defect instances, a previous modification to a highly evolutionary coupled file caused some unanticipated behaviour in the coupled files.

3.6.3. RQ2: Box Plot Analysis Results

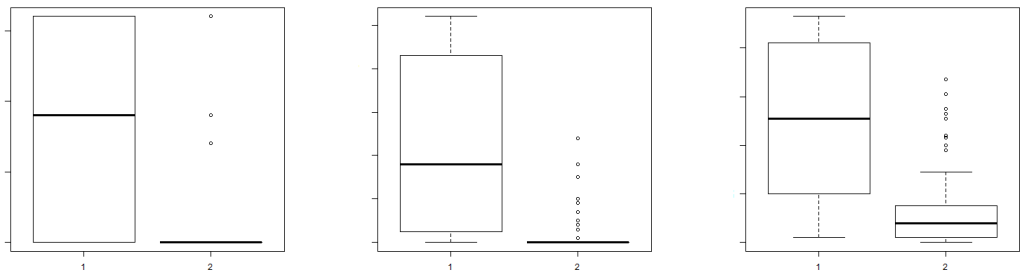
Figures 3.8, 3.9 and 3.10 show the box plots of module-level measures for modules where correlation is and is not detected. The y-axis of box plots is represented on a logarithmic scale and the range of measurement values in Figure 3.9 (for Company 2) is perfectly separated. Although there is overlap in the box plots for Company 1,



(a) Module 1. (b) Module 2. (c) Module 3.



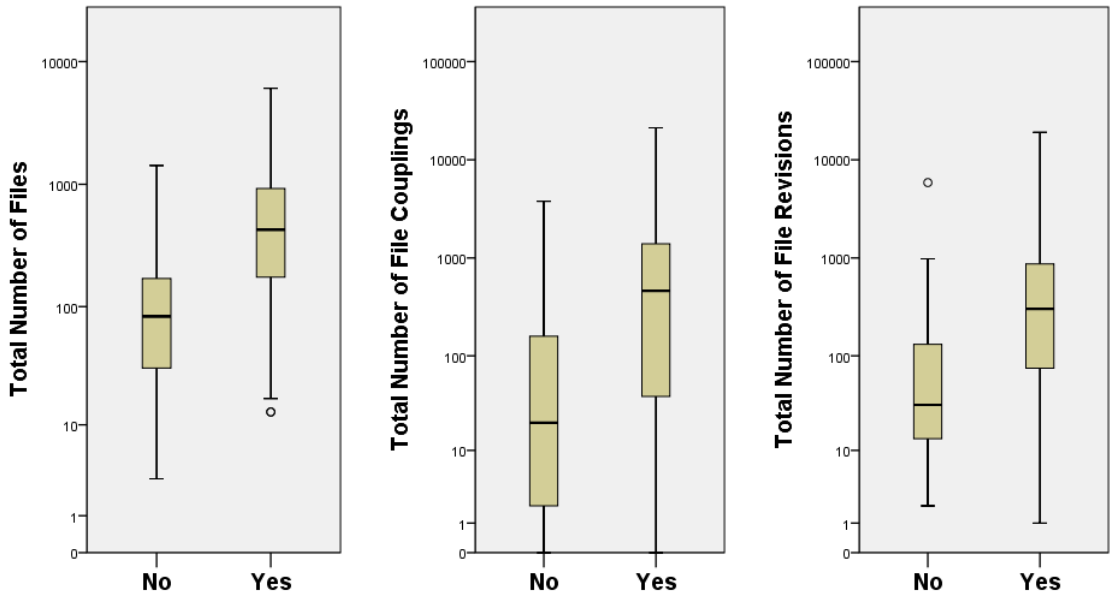
(d) Module 4. (e) Module 20. (f) Module 23.



(g) Module 38. (h) Module 55. (i) Module 64.

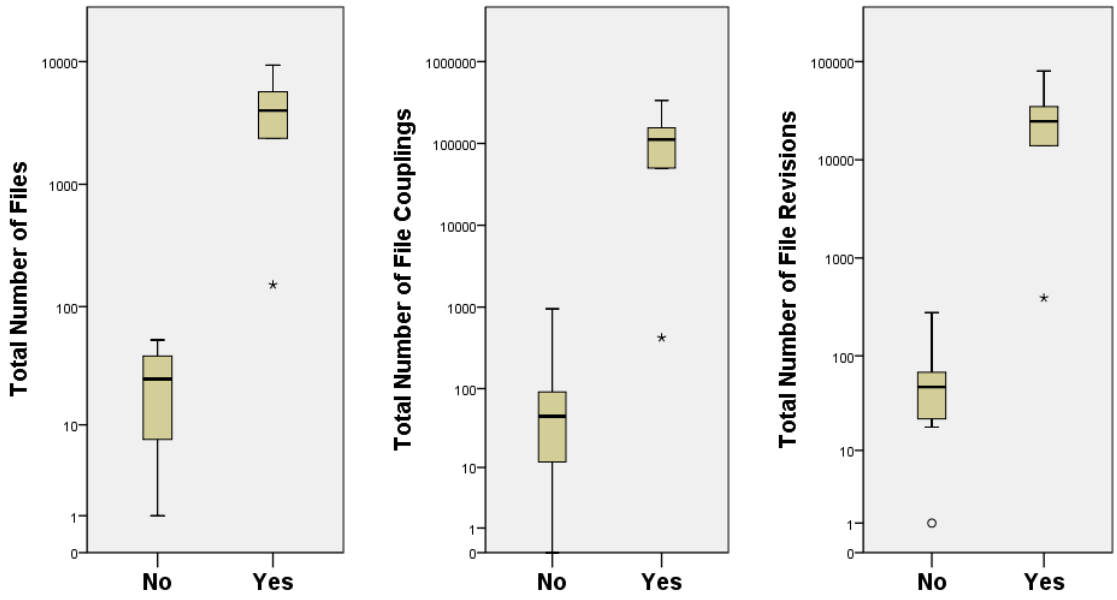
Figure 3.7. Box plots of EC measures for selected modules - files with defects vs. files without defects (1: represents files with defects, 2: represents files without any defects). Y-axis of box plots are removed to prevent revealing sensitive company data.

the difference is statistically significant (< 0.05) for both companies according to the Mann-Whitney test. All modules for which correlation is observed have high values for Total Number of Files (TNF), Total Number of Evolutionary File Couplings (TNEFC) and Total Number of File Revisions (TNFR). On the other hand, we did not observe a perfect separation of ranges for measures such as Total Number of Developers contributed (Figure 3.10(a)), Total Number of Defects (Figure 3.10(b)) and Total File Size



(a) Total number of files (TNF). (b) Total number of file couplings (TNEFC). (c) Total number of file revisions (TNFR).

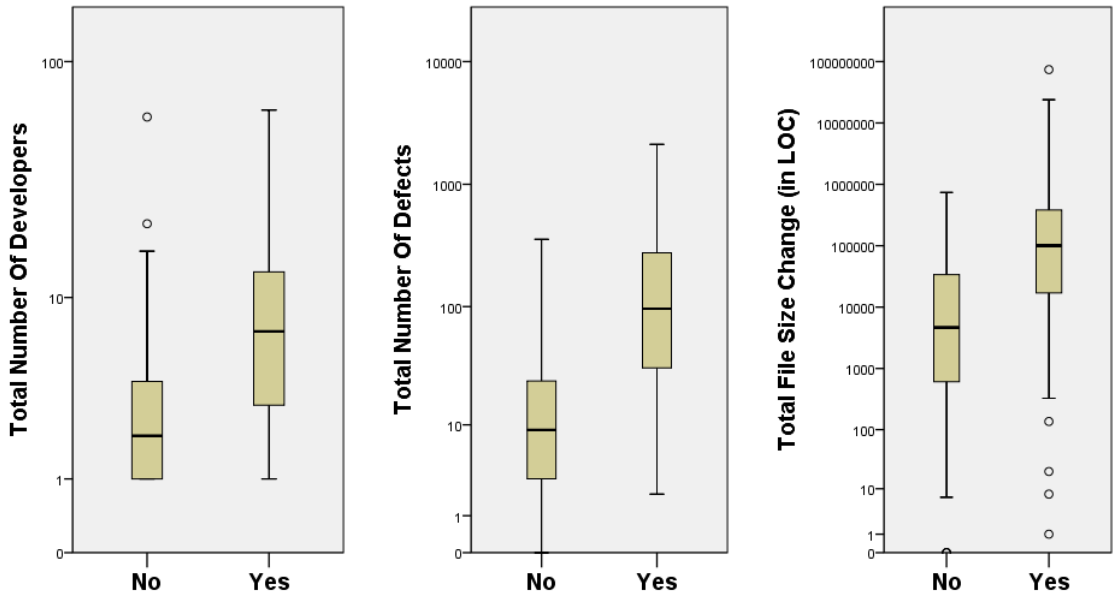
Figure 3.8. Company 1: Box plots of different measures for modules in which correlation between EC and defects detected (Yes) and not detected (No).



(a) Total number of files (TNF). (b) Total number of file couplings (TNEFC). (c) Total number of file revisions (TNFR).

Figure 3.9. Company 2: Box plots of different measures for modules in which correlation between EC and defects detected (Yes) and not detected (No).

Change in LOC for both companies (Figure 3.10(c)). However, the difference is still statistically significant (< 0.05) according to the Mann-Whitney test.



(a) Total number of developers. (b) Total number of defects. (c) Total file size change (in LOC).

Figure 3.10. Company 1: Box plots of different measures for modules in which correlation between EC and defects detected (Yes) and not detected (No).

We also checked how balanced the set of modules were in terms of defects with and without correlation and they were mostly unbalanced. There are generally more files without defects than those with defects.

We also analysed the relationship between module size and Spearman *rho* values for the correlation between EC (NoECF measure) and number of defects. The results can be seen in Figure 3.11 and Table 3.14. The correlation analysis showed a significant negative correlation (p=.005< 0.05 and *rho*=-0.218) between module size and *rho* value.

Table 3.14. Spearman correlation results.

	LOC
rho	ρ=-0.218**
	p=.005

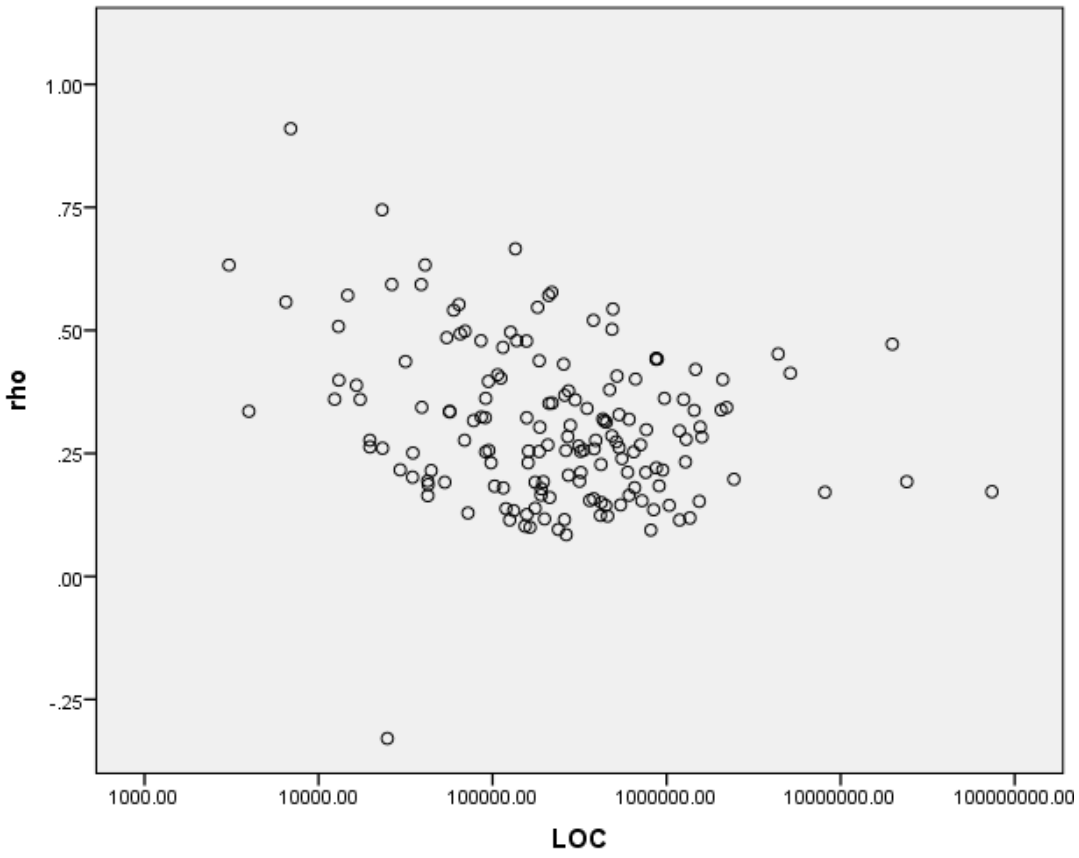


Figure 3.11. Module size vs. rho values of EC-defect correlation.

3.6.4. RQ2: Defect Type Analysis Results

The results of correlation analysis for each defect type are summarised in Table 3.15. The columns of the table show the Spearman correlation strength (*rho* values) between two EC measures and defect measures. Rows of the table represent different defect types. The list of defect types and their descriptions are provided in Table 3.3 in Section 3.4.2. In the table, we include only the defect types that have at least one significant correlation result. Code Implementation is the top correlated defect type and moderate correlation was observed here. One interpretation is that developers tend to make coding errors while they work on source files which are highly evolutionary coupled and they should take into account more relations with more files when coding these files. For the defect types in the table, we observed low correlation, although they are significant. Defect types such as Acceptance Criteria and Analysis can be associated with external EC to other modules and applications. Involvement of more

Table 3.15. Spearman correlation analysis results (EC measures - defect types).

	NoECFMR vs NoD (Spearman Corr.)	**NoECF vs NoD (Spearman Corr.)**
Code Implementation	ρ (rho)=.176** p=.000	ρ (rho)=.182** p=.000
Acceptance Criteria	ρ (rho)=.111** p=.000	ρ (rho)=.113** p=.000
Functionality Not Implemented Yet	ρ (rho)=.088** p=.000	ρ (rho)=.091** p=.000
Analysis	ρ (rho)=.083** p=.000	ρ (rho)=.085** p=.000
Not An Issue	ρ (rho)=.052** p=.000	ρ (rho)=.045** p=.000
Test Implementation	ρ (rho)=.060** p=.000	ρ (rho)=.061** p=.000
3rd Party System defect	ρ (rho)=.047** p=.000	ρ (rho)=.045** p=.000
Bad Data	ρ (rho)=.091** p=.000	ρ (rho)=.093** p=.000

** Correlation is significant at the 0.01 level (2-tailed).

modules and applications may make analysis and defining acceptance test criteria more difficult. We can interpret the correlation with Test Implementation type in a similar way to Code Implementation. We have checked the defects of Not An Issue type with the project members. They explained that this defect type was generally used for deployment problems. Correlation between defects of this defect type and EC may be explained as that deploying highly evolutionary coupled files and modules may be more error-prone due to more dependencies to be considered and deployed together.

For the following defect types, correlation values observed were trivial and are therefore ignored: Wrong Properties, Defect Value, Process Failure, Database Upgrade Failure, Data Fix Errata, Unexpected Functionality, Incorrect Config, Infrastructure Issues, Missing or Incomplete Data Migration, Acceptance Criteria Implementation.

For the following defect types, no significant correlation was detected: Incorrect Environment, CRM Bug, User Error, Database Disconnect-Reconnect Error.

3.7. Discussion

Our findings give insights to future researchers and practitioners on the effect of EC on defects.

3.7.1. (RQ1) What is the relationship between evolutionary coupling and software defects?

Our results suggest that there is, in general, a significant positive correlation between EC measures and defects. This finding is consistent with the general opinion that low coupling is an important principle to follow for a high quality software design and that high coupling can be related to defects [22, 65, 66]. Fewer interconnections between elements reduce the chance that changes in one element cause problems in other elements. Fewer interconnections between elements is also reported to reduce programmer time [67]. It is essential to keep the effect of a change in one element on another element low. However, our study shows that correlation strength between EC and defects varies across modules. Correlation strength had a wide range of values from zero to 0.8. Furthermore, there are also modules in which EC and software defects are not correlated. This is an important finding, since it highlights that the effect of EC is likely to vary depending on the module analysed. It is likely that the context of each module affects the risk that making change will create unanticipated changes within other elements. In other words, a change made in source code may have different manifestations on defects based on the module context (e.g., development process characteristics). Some modules carry higher risk than others. Therefore it is important to consider the EC - defect relationship in the context of related modules.

The contradictory findings reported by previous EC studies such as Graves et al. [18] and Knab et al. [19] may be partially explained in terms of the different systems and modules used in these studies. As shown by existing studies [68] the context

has an important impact on studies. We have shown that EC has different effects on defects across different modules, because EC seems to manifest differently in different modules. The characteristics of the modules in individual systems must be accounted for in future studies as also suggested by existing studies [68]. This finding also provides practitioners with valuable information on detecting defects and problematic code hotspots. However, as for all other code measures in relation to defects, EC does not contribute to defects equally for every module in a system, so EC use is not consistently helpful. We recommend that practitioners use EC for assessing the quality of their software design but also in conjunction with other module characteristics. That way practitioners will get the best of both worlds.

3.7.2. (RQ2) What factors explain why the relationship between evolutionary coupling and software defects is different for different modules?

We also tried to explain possible reasons for the different effects of EC on software defects. We considered this issue from two perspectives: module characteristics and defect types. We found that EC was less likely to have an effect on software defects for modules with fewer files and where fewer developers contributed. This may be explained by fewer defects being caused by EC in relatively small modules. Potentially, there are fewer interconnections between elements in a small module. Let n denote the number of files in a module. The potential number of interconnections in a module is calculated as $\frac{n*(n-1)}{2} = \frac{n^2-n}{2}$. Interconnections between files in a module can grow quadratically with the number of files. The more inter-related the files are, the more difficult these modules are to comprehend and perform changes. This may eventually lead to defects. An alternative explanation at least for the non-density models would be that such files typically have fewer defects.

We also recommend that practitioners add EC measures to their metric suite for software design evaluation. We recommend that researchers report process and size metrics of modules in their EC studies to account for the possible effect of context in their results. Furthermore, we found that EC may be more related to some defect types such as Code Implementation, Acceptance Criteria, Test Implementation and

less related to others such as Unexpected Functionality, Infrastructure Issues, Missing or Incomplete Data Migration, Incorrect Environment, User Error.

We believe that defect types may be used to explain the contradictory findings reported by previous EC studies in the literature. The different systems and modules used in these studies have different defect types and EC has different relationships with different defect types. It is more likely that high EC will cause Code Implementation and Test Implementation defects, because a high number of changes must be made to related parts of the system when code with high EC is changed. The locations of these related changes may be scattered within the application or even across applications in a software ecosystem; making related changes across these locations is likely to be challenging and this can increase the cognitive load of developers [69]. Moreover, developers may miss some locations which should be co-changed and this may cause unforeseen code and test implementation problems. On the other hand, EC is unlikely to contribute to defects whose root cause is user error or infrastructure issues.

3.8. Threats to Validity

3.8.1. Construct Validity

Threats to construct validity relate to whether we measure what we intend to measure. When calculating EC measures, there are two ways in which to group file revisions in the source code repository: MR-based and Transaction-based. EC measures calculated on a transaction basis for the system under study do not reflect the coupling relations between files; therefore, we preferred a MR-based approach for their calculation. The reason is that changes for a single MR were frequently split across multiple commit transactions for the systems under study. In contrast to open source systems previously analysed, in this study we had good defect linking. That enabled us to use a MR-based approach.

Another threat is the potential overlap in knowledge of EC with existing process metrics. To mitigate this threat, we applied multivariate regression and multicollinear-

ity analysis to understand the overlap and the unique knowledge embedded in EC measures.

The EC metrics used in our study do not consider the age and temporal aspects of EC. ECs that are temporary or no longer valid due to refactorings or restructurings could not be detected in our study. This is a limitation of the study.

We assume that any change made to source code is committed to code repositories. The software processes in both companies place check points (at compilation, moving to test/production) to guarantee this assumption. Practitioners should be careful in using EC measures, since they may not be reliable for some modules if version control systems (VCSs) are not used by their developers (or there is a low utilisation of VCS). EC measures make sense if the VCS is used long enough and consistently. In this study, some modules had low utilisation for VCS and this may be an indication of problems with VCS adaptation for some projects in the company. VCS was introduced to some projects at Company 1 a few years ago. As a consequence, we recommend excluding these types of modules or systems when calculating EC measures and using EC measures in defect models.

All the files committed in a particular commit operation might not be logically coupled. This threat is mitigated by ignoring commit transactions having more than 30 files. Therefore large transactions, which may possibly include files from more than one MRs (e.g., the merge of a branch), are not used to calculate EC. Furthermore, we also performed a manual analysis of randomly chosen commits and MRs and checked the validity of this assumption. When huge commit transactions are removed, there are very few exceptions to this assumption. Another point is that there are differences between industrial and open source software development regarding this assumption. Companies generally place controls on MRs in the application life cycle (e.g., mandatory MR numbers during check-in, allowing only files associated with an MR to move to the production, etc.) and companies usually rigorously follow such conventions, unlike many open source projects [70].

3.8.2. Internal Validity

In this study, we used configuration items from the Configuration Management Database (CMDB) (for Company 1 only) attached to problem records and related requests (move to production, code review, move to test, etc.) with which to match defects to source code files. Two assumptions were made at this stage:

(i) Configuration items defined at CMDB correspond to source files changed in the scope of the resolution of a defect.
(ii) Configuration items of the source files changed in the scope of the resolution of a defect are linked to the problem record of the defect.

The validity of these two assumptions can be guaranteed for certain record types (move to production, code review), but in general these cannot be guaranteed, that for each defect, all related source files are detected.

Another assumption is that developers commit source files changed in the scope of the same MR to the same package in the code repository. This assumption is used in the calculation of EC measures. We rely on the data collected from versioning systems and any project which is not managed in the versioning system (or any file which is not committed to versioning systems) are not considered in our study.

The measures and defect types chosen for answering RQ2 are not exhaustive and do not cover all characteristics of a module and all defect types which can exist. An exhaustive examination may have revealed other factors that have a greater effect on defects and which may be confounding our results. In our study, we investigated file size (LOC) as a possible confounding factor. We observed that code size correlated with number of defects in some modules. Defect density (DD) however had either no significant correlation or only minor negative correlation. Using defect density in our study mitigates the risk of size as a possible confounding factor. We are planning future investigations to explore the effect size of a large number of factors related to defects.

3.8.3. External Validity

External validity relates to the generalisation of our study results. We only studied two industrial software systems. These systems may not be representative of the way developers develop systems more generally. We mitigate this risk by choosing two systems from different domains and with different technologies. In future work, we would like to extend this study by including more commercial systems and projects.

3.9. Conclusions

In this chapter, we presented a study on the relationship between EC and software defects in two large industrial software systems. We reported a positive correlation between EC and defect measures in the software maintenance / evolution phase of systems from two different companies. Our results indicated low, moderate and high level correlation, with varied correlation strength across modules. Our regression analysis results indicated that EC measures could be useful for explaining defects.

The box plots drawn for each module separately showed the potential of EC measures to distinguish defective and non-defective files. We also observed that the company using practices such as agile and TDD had relatively fewer modules with high EC-defect correlation values. However, this finding needs to be further investigated on more companies for generalisable conclusions.

We also tried to understand the reasons for variation of the observed effect of EC on software defects for different modules. We found that modules which were small in size in terms of file and developer numbers, tended to be less correlated with EC. Interconnections between files in a module can grow quadratically with the number of files. Increasing number of inter-related files make a module more difficult to comprehend and perform changes. This complexity may eventually lead to defects and this may be one of the reasons for variation across modules. Furthermore, we observed that EC measures showed higher correlation with some types of defects (based on root causes) such as code implementation, acceptance criteria and analysis problems. The disper-

sion of these defect types could be another reason for these varying effects. Different modules have different defect types and EC has different relationships with different defect types.

Module characteristics and defect types may also explain why different results are reported by different studies in the literature. Different applications or modules analysed may have different characteristics and defect types. We recommend that researchers report characteristics and defect types of modules in their EC studies to account for the possible effect of the context in their results. We also recommend that practitioners add EC measures to their metric suite for software design evaluation and consider the characteristics and defect types of their modules in their evaluation.

4. EC MEASUREMENT EVALUATION METHODS

Although there has been considerable work in establishing a measurement theory basis for software measurement [25–28], it is not clear that these measurement principles have been used in EC measurement. Measures that do not adhere to these principles may be flawed. Empirical results obtained using flawed measures are likely to be unreliable and decisions and conclusions based on these results could be misleading. Such results may lead to time, money and resources being waste as well as contaminated scientific knowledge generated.

In this chapter we present EC measurement evaluation criteria. We use measurement theory and metrology principles in the development of our criteria. Measurement must start with an objective [26, 28, 30, 31]. Objectives put measurement in a context and determine the design of measures and interpretation of the measurement results. So by following this basic principal of the measurement theory we first find the objectives of EC measurement. This constitutes our first research question (RQ1) in Table 4.1. Measurement captures information about attributes of entities [26, 28, 31]. So both the entity and the attribute to be measured should be identified clearly. This basic principle [26, 28, 31] is the base of our second research question (RQ2) in Table 4.1. We characterise the concept of EC by specifying the entities to be measured and the characteristics (attributes) that should be taken into account. We develop a meta-model of EC and its sub-concepts including the relationships across them as suggested by Abran [28]. In order to be valid, a measurement method must satisfy the representation condition of measurement theory [25, 26, 31]. Based on this principle and the meta-model developed in the previous step, we define the empirical relations and concerns to be considered in establishing a sound empirical relation system, which captures all generally accepted ideas about EC. We formulate our third question (RQ3) in Table 4.1 based on this principle of measurement theory. We check whether EC measures preserve the empirical relations while mapping the empirical relation system into the numerical one. Abran in his software metrology work [28] points out that mature engineering disciplines have well established measurement methods and

the detailed procedures with a large international consensus. For example, the World Meteorological Organization (WMO) defines the temperature measurement setup in a very detailed and quantitative manner such as that thermometers should be positioned between 1.25m and 2m above the ground. Well-defined, standard and detailed measurement method and procedures are mandatory to ensure the accuracy, repeatability, and repetitiveness of measurement results. Based on this principle, we formulate our fourth research question (RQ4) to cover the detailed procedures and practicalities of EC measurement. Determining the scale type of measures and doing mathematical validation are also one of the basic practices of measurement. Scale types are very important to determine the limitations on the types of mathematical manipulations that can be performed on measurement results. This forms our fifth research question (RQ5).

We evaluate current EC measures based on these research questions and criteria formulated for each research question. We reveal the picture of existing EC measurement practices and provide recommendations to be used in future uses of EC measurement.

Table 4.1. Research questions.

No	**Research Question**
RQ1	What are the objectives of EC measurement?
RQ2	Do existing EC studies identify entities and attributes to be measured?
RQ3	Do existing EC studies use sound empirical relation systems?
RQ4	Do existing EC studies define the measurement method and procedures?
RQ5	Do existing EC studies use scale type and mathematical validation?

Our evaluation methods are based on the software measurement literature. From the literature drawn on the principles of measurement theory [26, 31] and on software metrology [28] we develop evaluation criteria that will allow us to answer our research questions. Our evaluation criteria are based on established software measure-

ment theory (Fenton [31] and Zuse [26]) and, in particular, are based on representation conditions, the mathematical properties of the manipulation of numbers and the proper conditions for such manipulations. Our evaluation criteria also draw on software metrology [28] by including criteria which evaluate the measurement method.

The set of evaluation criteria that we develop are not of equal importance. Consequently, we weigh the importance of each criteria using three gradations (3 or 2 or 1, from high to low). These weights have been arrived at by the authors interpreting the literature on software measurement and applying to the measurement of EC.

We first published our results on the EC measurement evaluation methods in [5]. In particular, this chapter makes the following contributions:

(i) We develop a meta-model for EC concepts, which helps to understand how the measure is derived and how to interpret the numerical values of measurement results.
(ii) To the best of our knowledge, this is the first work that applies measurement theory and metrology principles to EC measurement.

The following subsections detail the principles of measurement theory and software metrology that we have used in the development of our evaluation criteria within the context of our Research Questions.

4.1. Study Selection Methods

To identify the set of EC studies to evaluate we used a cut-down Systematic Literature Review (SLR) approach based on that proposed by Kitchenham et al. [71]. SLR is a process of identifying, assessing and interpreting all available research evidence to provide answers for particular research question. We did not include all stages of an SLR, for example we did not include an explicit quality assessment, as the objectives of our study did not require this. We were aiming to identify all studies that use EC measurement, in order to evaluate that EC measurement. It could be argued that this

evaluation is an extended quality check.

First we prepared a protocol for our literature review which defines the process and methods to be used in our specific systematic review. Such a pre-defined protocol reduces the possibility of researcher bias and allows validation and replication. Without a protocol, researcher expectations are more likely to impact the study selection and evaluation. We now summarise our protocol.

We identified previous studies of EC using the search facilities in the SCOPUS tool. SCOPUS is a general indexing system, which indexes most of the main internationally recognised software engineering journals that regularly publish empirical studies. The following search string is used in our searches:

("evolutionary coupling" OR "change coupling" OR "logical coupling") AND (software) anywhere in study

We used these terms as our preliminary reading shows that three different terms are used to refer to EC: Evolutionary Coupling [21], Change Coupling [17] and Logical Coupling [7]. Our initial searches returned many false positives from other domains such as Genetics, Molecular Biology, Electronics, etc. We added Software as a keyword to reduce false positives from other domains. We follow Kitchenham et al.'s recommendation of using fairly simple search strings based on the main topic [71]. Simple strings are more likely to work on a variety of different digital libraries without extensive refinement.

Our searches covered 1997 to 2014. We chose 1997 as the start date since there is general agreement [17, 21] that EC is first introduced in 1997 by Ball et al. [6]. After obtaining the potentially relevant primary studies by digital library and manual searches, each must be assessed for relevance. The following inclusion criteria (a paper must be) and exclusion criteria (a paper must not be) were identified in the SLR protocol:

Inclusion criteria:

- A study measuring Evolutionary Coupling
- Published in English
- A Journal paper or Conference proceedings

Exclusion Criteria:

- Papers not subject to peer-review
- Grey literature such as technical reports

We assessed each candidate paper by applying the inclusion/exclusion criteria on all primary studies found. Then a randomly selected three papers were also assessed by the two other senior researchers, plus one other researcher. Disagreements among assessors were resolved with a meeting. This process identified fifteen included papers.

4.2. Research Question 1: What are the objectives of EC measurement?

This research question was motivated by Abran [28] and Fenton and Bieman [31] emphasising the importance of determining measurement objectives. It is important to know the intended use of the measure and the intended users of the measurement results (software user, software designer, etc.). As a measure can be very useful for a specific measurement objective while the same measure may not be appropriate for another measurement objective. Consequently our first evaluation criteria is:

Table 4.2. Criteria related with RQ1.

ID	**Criteria**	**Category**	**Weight**	**RQ**
C0	Objectives and Users of Measurement	Measurement Objectives	3	RQ1

This criteria is essential to all studies and so has a weighting of 3.

4.3. Research Question 2: Do existing EC studies identify entities and attributes to be measured?

This research question was included because to evaluate a base measure the concept to be measured must be first clearly defined as stated by Fenton [31] and by Abran [28]. An (empirical) operational definition of the entity and the attribute must be given; that is, the concept must be characterised. The first step of the characterisation is the decomposition of the concept into its sub-concepts as suggested by Abran [28]. The definitions of sub-concepts and their relation to the concept are identified in this decomposition process. We start the decomposition by analysing the definition of EC:

- The implicit relationship between two or more software artefacts which are frequently changed together [7, 8, 17].

The sub-concepts extracted from this definition are i) "software artefact", ii) "relationship" and iii) "co-change" (changed together). We further decompose these three concepts into their sub-concepts (following) and identify our detailed evaluation criteria.

4.3.1. EC sub-concept 1: Software artefact

We decompose this sub-concept into different types of software artefact: System, Module, Package, File, Class, Method. The granularity of software artefacts increases from System to Method level. Different EC studies in the literature use different types of software artefacts in their studies: System [7, 16], Module [7, 9, 16], Package [72], File [3, 10, 20, 29, 73, 74], Class [6, 8, 11, 17, 21] and Method [11, 21]. Studies show the importance of granularity in measuring EC. For example, [21] showed that information on the coupling between methods (low granularity) rather than classes or modules

(higher granularity) is more useful for developers during impact analysis. Consequently, we add the following evaluation criterion:

- Granularity of Software Artefact: The unit of code granularity of coupled items must be reported.

As granularity is fundamental to the interpretation of any results we assign a high weight (3) to this criterion, as below.

Table 4.3. Criteria related with RQ2.

ID	**Criteria**	**Category**	**Weight**	**RQ**
C1	Granularity of the Entity	Identifying Entities and Attributes	3	RQ2

4.3.2. EC sub-concept 2: Co-Change

Next, we decompose the sub-concept co-change by considering various characteristics of co-change. As these co-change characteristics are candidates for different base measurement as discussed later in the dissertation, we assign a high weight (3) to these criteria. The characteristics identified are:

- Grouping Characteristic: EC measurement starts with the direct measurement of co-change of software artefact pairs. To identify pairs of software artefacts, it must be clear how these pairs are related as this will make a difference to the conclusion that can be drawn. Three different grouping approaches are common, pairs based on: Modification Requests (MR), Commit Transactions or Tasks. Using different grouping approaches can result in measuring different attributes of entities. Consequently we introduce a criteria requiring that the grouping approach should be reported (C2 in Table 4.4).
- Locality Characteristics: The distance between co-changes is important. Co-changes that include files from different subsystems have been shown to result in more bugs than co-changes that include files only from the same subsystem

Table 4.4. Criteria related with RQ2 and RQ3.

ID	Criteria	Category	Weight	RQ
C2	Grouping Approach - Entity Level	Identifying Entities and Attributes	3	RQ2
C3	Locality Aspects	Sound Empirical Relation Systems	3	RQ3
C4	Change Scope	Sound Empirical Relation Systems	3	RQ3
C5	Change Type	Sound Empirical Relation Systems	3	RQ3
C6	Distinct Committers	Sound Empirical Relation Systems	3	RQ3
C7	Temporal Aspect-Regularity	Sound Empirical Relation Systems	3	RQ3
C8	Temporal Aspect-Recency	Sound Empirical Relation Systems	3	RQ3
C9	Change Size	Sound Empirical Relation Systems	3	RQ3

[16]. Alali et al. [75] also show that the distance among co-changed artefacts can increase the detection accuracy of EC, suggesting that most ECs occur within the same sub-folder and remain localised. Consequently we include the locality of changes in our evaluation criteria (C3 in Table 4.4).

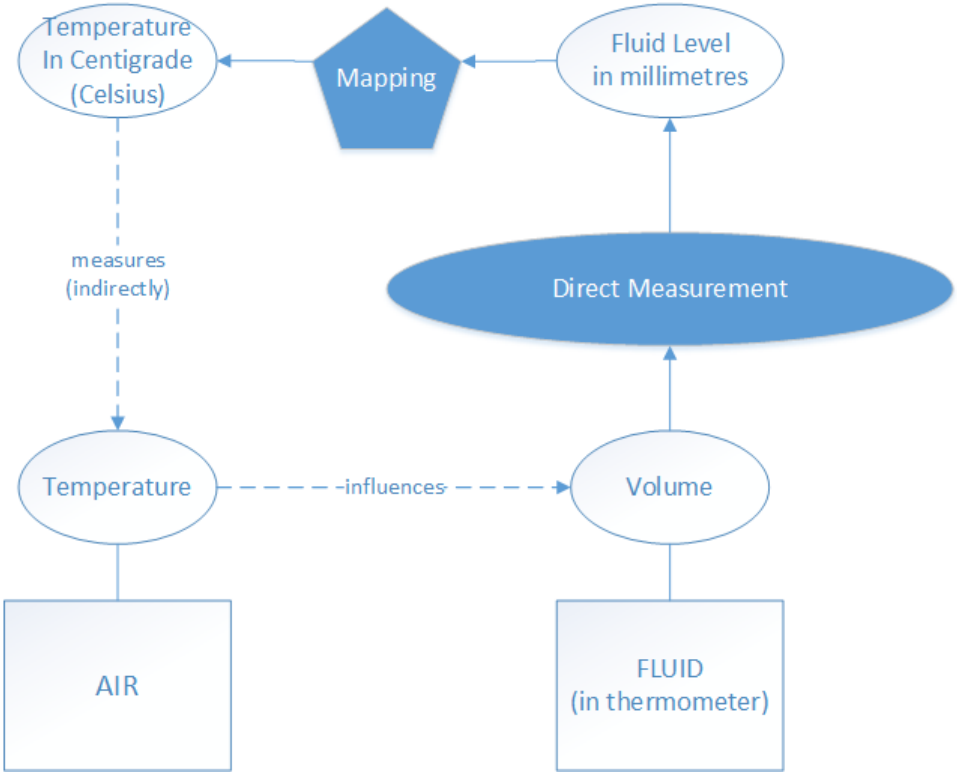
- Change Characteristics: The reason for a change may also affect EC: bug-fix or enhancement [76]. The process involved and the teams can be very different for these different changes with different scope. Therefore, we distinguish the reason for a change in our evaluation criteria (C4 in Table 4.4).
- Change Type Characteristics: There are three basic types of changes that can be made: addition, modification and deletion. These different changes have been shown to have impact on the results of studies. For example, [21] suggests that different types of changes enhance the ability to predict related or missing changes

during development. Consequently our evaluation criteria distinguish between change types (C5 in Table 4.4).

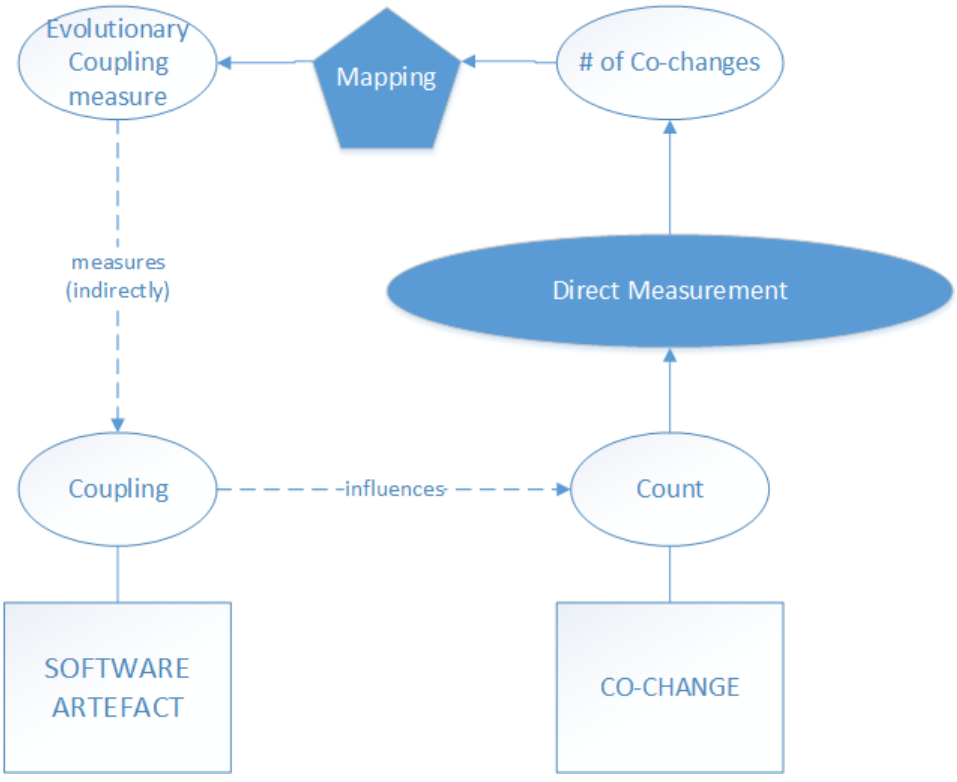
- Committer Characteristics: Madeyski et al. [77] found out that defect prediction models trained on a data set containing Number of Distinct Committers (NDC) were significantly better than the simple models without NDC. Distinct committers can also effect EC measures especially when they are used in defect prediction. Consequently our evaluation criteria include distinct committers (C6 in Table 4.4).
- Temporal Characteristics: The work of Alali et al. [75] highlights the temporal aspects in the detection of ECs. Their results show that the age of ECs can increase the detection accuracy of EC. They pointed also the importance of recency: there was a 40% chance that a pattern will occur again in a few days or less. Furthermore, it appears that the older a coupling is, the more rooted it is. Consequently our evaluation criteria include age, recency and regularity of EC (C7 and C8 in Table 4.4).
- Change Size Characteristics: The size of changes involved in EC is likely to affect the impact of that EC. Large changes are not likely to have the same impact on EC as small changes. Consequently our evaluation criteria includes the size of changes (C9 in Table 4.4).

4.3.3. EC sub-concept 3: Relationship

There are many different reasons for two artefacts to need changing at the same time, this means that there are a variety of reasons for EC such as structural, dynamic and semantic couplings. We suggest that EC is not a direct relationship between software artefacts, but it is a representation of other types of underlying direct relationships. This is very similar to temperature measurement as we generally do not measure air temperature directly. Instead it is measured via its effect on a fluid in a glass thermometer. The level of the fluid represents the air temperature as the strength of the EC represents the underlying direct couplings and relationships. We further detail this analogy below.



(a) Temperature measurement.



(b) Evolutionary coupling measurement.

Figure 4.1. Temperature analogy.

Temperature measurement relies on measuring some physical property of a liquid that changes with temperature. For example, a glass thermometer consists of a glass tube filled with mercury or some other liquid. The fluid expands with the increasing temperature, so we can determine the temperature by measuring the volume of the fluid. We can read the temperature simply by observing the level of the fluid in the thermometer. No one is interested in how many centimetres the mercury expands. The attribute of interest is temperature here and the entity is the air. In other words, the temperature attribute (or property) of the air is the underlying attribute to be measured by observing the level of the liquid. This is illustrated in Figure 4.1. For example, a two centimetre raise in the level of the liquid is mapped to two centigrades of temperature raise. In the same way direct overall coupling is the underlying attribute of a software artefact to be measured by calculating EC. Being co-changed is an indication of underlying couplings between software artefacts. So the term implicit relationship used to define EC can be replaced by explicit or direct couplings and relationships between software artefacts which constitute underlying attributes measured such as structural coupling, dynamic coupling, semantic coupling and cross-cutting concerns.

4.4. Research Question 3: Do existing studies use a sound empirical relation system?

4.4.1. Establishing a Sound Empirical Relation System

In order to be valid, a measurement method must satisfy the representation condition of measurement theory [25, 26, 31]. The representational theory of measurement provides a way to formalize our intuition about the way the world works. In other words, a measurement value should represent attributes of entities we observe and the manipulation of these values should preserve empirical relationships that we observe among entities. To map entities into numerical world and empirical relations into numerical relations we use a measurement mapping M.

Following Fenton's example we take the example of measuring the height of a person. In this specific case, person is the entity and height is an attribute of the

entity, which is to be measured. Let's take two instances of the people entity as Tyrion Lannister (the dwarf in the HBO series Game of Thrones) and Shaquille O'Neal (nicknamed Shaq, a famous NBA basketball player). If we take doesn't have the same height as the (binary) empirical relation to compare the heights of these two people, any measure aiming to measure the height attribute of a person entity must preserve the empirical relation as Tyrion does not have the same height as Shaq. This empirical relation can be mapped to the numerical relation $\neq$. In other words, $M(Tyrion) \neq M(Shaq)$ must hold whenever a measure maps these people into numbers. If we use metre as a base measure from the International System of Units (SI), we observe that the empirical relation is preserved: $M(Tyrion) = 1.35 \neq M(Shaq) = 2.16$.

Table 4.5. Empirical relations and attributes applied on.

Empirical Relation	**Attributes applied**	**Correspnd. Mathematical Operation**	**Nominal**	**Ordinal**	**Interval**	**Ratio**	**Absolute**
is not equal in strength	Coupling Strength	$\neq$	✓	✓	✓	✓	✓

To evaluate and compare EC measures from the empirical relation system point of view (which is addressed by our third research question, RQ3), we defined a basic empirical relation and its corresponding mathematical operations in Table 4.5. The not equal in strength empirical relation covers all scale types as shown in Table 4.5. So all existing EC measures can be covered by using this empirical relation. Users of our criteria can decide using stronger relationships such as greater than based on their specific measurement objectives and context. A condition holding for defect prediction may not be true for aspect detection. Or temporal aspect - recency can be interpreted quite differently depending on the measurement objectives. Therefore we chose to use the not equal in strength to cover all EC possible measures and measurement objectives. In the context of EC measurement, software artefact is the entity and coupling strength is the attribute of the entity, which is to be measured. EC measures are indirect measures as first co-change entity is measured and aggregated onto software

artefact entities. Therefore we need to focus on the direct measurement first: co-change measurement. As we detail previously, there are different characteristics of the co-change entity. A very important theoretical flaw of current EC measures is treating these different characteristics as equivalent during aggregating co-change measurement results and associating these with a software artefact. Any valid EC measure to be used should preserve the empirical relation. Based on the characteristics we previously describe for co-change, we similarly define the following empirical system evaluation criteria as provided in Table 4.4:

- Locality Aspect: cross-system EC is not equal in strength to within-system EC (C3 in Table 4.4). As discussed previously, different types of coupling may contribute differently to the overall coupling measurement result and so EC measurement should consider cross-module, cross-package or cross-application couplings.
- Change Scope: EC resulting from a Bug-fix is not equal in strength to EC resulting from an Enhancement (C4 in Table 4.4). As discussed previously, the relationships among files coupled through a Bug-fix can be quite different from the relationships among files coupled through an Enhancement.
- Change Type: EC resulting from additions/deletions is not equal in strength to EC resulting from modifications (C5 in Table 4.4). Treating addition, deletion, and modification as equivalent can be problematic in the same way that the CBO measure has also been said to be problematic (CBO incorrectly treats forward and backward links as equivalent [40, 78]). The relationships among files coupled through additions can be quite different from the relationships among files coupled through modifications.
- Distinct Committers: EC resulting from multiple distinct developers is not equal in strength to EC resulting from a single developer (C6 in Table 4.4). Co-changed artefacts can be changed by different committers. There are various previous studies suggesting that the number of committers impacts on the results of analysing software. For example, Madeyski et al. [77] found out that the defect prediction models trained on a data containing Number of Distinct Committers (NDC) were significantly better than the simple models without NDC.

Distinct committers are also likely to affect EC measures especially when they are used in defect prediction.

- Temporal Aspect-Regularity: EC resulting from regular changes is not equal in strength to EC resulting from non-regular changes (C7 in Table 4.4). The patterns of co-changes can be diverse such as short-lived, long-lived regular, and long-lived non-regular. Mondal et al. [79] showed that couplings based on regular co-changes can be more useful in some contexts such as impact analysis or change prediction.
- Temporal Aspect-Recency: EC resulting from recent changes is not equal in strength to EC resulting from non-recent changes (C8 in Table 4.4). If a project undergoes restructuring frequently, assignment of a higher weight to recent changes can be used to increase change prediction performance [21]. In such projects older changes are less likely to be relevant than recent changes as the older relationships between software artefacts may not be relevant any more.
- Change Size: EC resulting from larger change sizes is not equal in strength to EC resulting from smaller change sizes (C9 in Table 4.4). As discussed previously, the size of changes can be very different. For example, it can be a few LOC, hundreds of LOC or thousands of LOC for a file. The size of the change constitutes another concern that can affect EC measurement, as the size of changes on co-changed software artefacts can contribute to the strength of the coupling between software artefacts.

Which of these criteria are most important to consider will depend on the particular measurement objectives and the users of those measures. However it is important that each criteria and possibly its weighting is explicitly considered.

4.4.2. Mapping from Empirical Relation System to Numerical System

Measurement is formally defined as a mapping from the empirical world to the numerical world. In other words, the domain of the mapping is the real world and its range is the mathematical world. The mapping as part of measurement assigns an

Table 4.6. Examples for empirical relation system evaluation.

No	**Cases for not equal in strength Empirical Relation (L)**	**Validation Sample (Example Scenario)**	**Must Hold in Measurement Objective scope**
1	Coupling through co-changes of two software artefacts residing on the same system is not equal in strength to the one through co-changes from different system (Locality Aspect, Criterion 3)	Commit1: File1, File2 Commit2: File2, File3 (File1 and File2 reside in System 1) (File3 resides in System 2)	$M(File1 - File2) \neq M(File2 - File3)$
2	Being changed by multiple developers constitutes not equal coupling in strength compared to being changed by a single developer (Distinct Committers, Criterion 6)	Commit1: File1, File2 (Developer 1) Commit2: File1, File2 (Developer 1) Commit3: File4, File5 (Developer 1) Commit4: File4, File5 (Developer 2)	$M(File1 - File2) \neq M(File4 - File5)$
3	Being changed regularly yields not equal coupling in strength to being changed non-regularly (Temporal Aspect-Regularity, Criterion 7)	Commit1: File1, File2 (07/2014) Commit2: File1, File2 (07/2014) Commit3: File1, File2 (07/2014) Commit4: File4, File5 (01/2014) Commit5: File4, File5 (04/2014) Commit6: File4, File5 (07/2014)	$M(File1 - File2) \neq M(File4 - File5)$

entity a number or symbol to characterise an attribute. A measure must clearly define its mapping rules, its domain and range.

The behaviour of the measures in the numerical world should represent the corresponding attribute of the real world entity. Entities must be mapped to numbers and empirical relations to numerical relations via measurement mapping M so that the mapping preserves the relation. This rule is called the representation condition [31].

Table 4.6 provides several examples for the not equal in strength empirical relation. We define at least one case for each criteria listed in Table 4.4 to be able to check that numerical relations preserve empirical relations. One counter-example is enough to show that the representation condition is not satisfied for a given measure. Let us explain the first example in Table 4.6. There are two commit transactions: Commit1 and Commit2. Two software artefacts (File1 and File2) have been changed in Commit1. File2 and File3 have been changed in Commit2. If we consider only this information to measure the EC between files, the EC between File1 and File2 (expressed as M(File1-File2)) will be the same as the EC between File2 and File3 (expressed as M(File2-File3)). However, if we consider that File1 and File2 reside in System 1 while File3 resides in System 2, the EC between File2 and File3 as a cross-system coupling will be intuitively stronger compared to the EC between File1 and File2.

4.5. Research Question 4: Do existing studies define measurement method and procedures?

In software metrology, the measurement method is generally defined as a logical organisation of operations used in a measurement [28]. A base measurement of a specific attribute should have its specific measurement method designed. We summarise the EC measurement process based on the ISO 15939 measurement information model [80]. The measurement method links the attribute of an entity to a base measure. Function Point (FP) and COSMIC-FFP are two examples of measurement methods in software engineering used to measure the functional size of software. EC is not a base measure and not measured directly. Co-change measurement is the direct measurement and

co-change measures are base measures. Then, the values of two or more base measures are used to derive EC measures by using a measurement function. Therefore EC is a derived measure. EC as a derived measure is then used in the context of an analysis model to explain the relationship between EC and the information needed [80].

Measurement methods should be complemented with detailed measurement procedures, which are sets of operations, described specifically, used in the performance of particular measurements according to a given method [28]. When measurement method and procedures are not sufficiently well defined and standardised to ensure the accuracy, repeatability, and repetitiveness of measurement results, then, when the same entity (software) is measured by different measurers, the results can potentially be significantly different. By using the adapted ISO 15939 measurement information model and our own experience in measuring EC [3, 4] on large software we identified the following evaluation criteria as provided in Table 4.7:

- Normalisation: EC measures should be normalised for comparison with different studies (C10 in Table 4.7). Normalisation has a considerable impact on software measures [15, 81]. In the scope of EC measurement, product size or total number of couplings are potential candidates for normalisation of EC measurement results. A measure which is not normalised has little value out of the measurement context. EC measures hold this risk as well. Furthermore, size of the application can implicitly and unintentionally be included in EC measure if normalisation is not considered.
- Grouping Approach - Temporal (Entity Level: Software Artefact Pair): The period of time from which version history is used for measurement should be considered (whole life of an application or part of it such as only the maintenance phase, specific release, etc.) (C11 in Table 4.7). The commit history from version control systems (VCS) are essential for calculating EC measures. It should be specified which part of the commit history is or should be considered. Some EC measures need the version control history from a long period to yield meaningful results [21]. This type of measure will not be applicable especially in the initial

development phases.

- Large Merge Transactions (Measurement Method): Large merge transactions should be ignored in EC measurement as large merge transactions generally contain co-changes which aren't relevant. We may end up with EC values which do not accurately reflect the underlying coupling among software artefacts (C12 in Table 4.7). Development teams sometimes create different branches to implement different features or to support maintenance of different versions of a software. The merge from a branch to trunk may cause a single large commit transaction in the version history. This single commit has all the changes commited into the branch. Large merge transactions do not reflect the fine-grained relations among files. To detect EC within transactions, large merge transactions should be avoided. For example, some new features implemented on a branch can be then moved (merged) all together to the trunk and all the files changed in the scope of multiple features will be incorrectly interpreted as coupled in this merge transaction. On the other hand, large transactions can be useful in some other contexts such as mining cross-cutting concerns (aspect mining) [11].
- Branches - Aggregation of Base Measures (Measurement Function): Commit transactions for the same bug-fix/enhancement on different branches should not be considered multiple times (C13 in Table 4.7). If different versions of a product are alive and used by its customers, bug-fixes/enhancements should generally be addressed in multiple branches. Especially bug-fixes should be done on all live branches. The same set of files changed in the scope of the same bug-fix/enhancement should not be counted multiple times for different branches when calculating an EC measure. Otherwise the strength of EC among these files may be misleadingly measured higher than the actual strength.
- Aggregation of Base Measures (Measurement Function): Being changed in the scope of multiple MRs / Transactions should be considered (C14 in Table 4.7). Some software artefacts can be coupled through multiple MRs / Transactions. The number of different MRs / Transactions can be used in the measurement of EC. In some measurement contexts, the number of different MRs / Transactions involved can contribute to EC measurement.

- Single Artefact Commits - Aggregation of Base Measures (Measurement Function): Are commit transactions involving only single software artefact considered? (C15 in Table 4.7). In the scope of some commits, only one software artefact is committed. When calculating EC measures, it should be indicated whether this is considered or not. Previous studies show that about one third of all commits have only one file involved. With increased granularity, this ratio will increase even more. Involvement of single artefact commits is critical for the calculation of some EC measures such as Confidence [21].
- Refactorings - Software Artefacts (Measurement Method): Refactorings should be considered in the measurement (C16 in Table 4.7). Refactorings such as name changes, moving artefacts to different folders, packages, modules, etc. should be considered in EC measurement. It is very important especially for projects that are frequently restructured [21].
- Prerequisite: Adequateness of Version Control (VC) History: The version history should be good and rich enough to have meaningful measures. All software artefacts should be involved in at least one commit transaction / MR / Task (C17 in Table 4.7). EC is an indirect measure and representation of the coupling attribute of a software artefact. In most of its uses (analysis model [80]) it is used to represent the coupling and complexity of a software artefact. However, EC being a process measure we need to have enough version control (VC) activity in order to represent coupling, which is a product measure. If we have no VC activity for a particular software artefact, it might be wrong to conclude that it is coupled with no other software artefacts. Therefore it is important to check that all software artefacts are involved in at least one commit transaction, MR or Task as this will mean that each software artefact will reveal its couplings.
- Relative Strength (Measurement Function): All transactions in the system should be considered in the EC measurement of each software artefact (C18 in Table 4.7). Commit transaction density and co-change sizes can be different for different systems. It can be hard to interpret the measurement results. Considering all transactions in the calculation of individual EC calculations of each software artefact can provide measurement results which can be comparable across stud-

ies and projects. Considering all transactions in a system can provide a better relative coupling strength as Beyer et al. [10, 82] proposed in their visualisation work. It can also be used for normalisation purposes.

Table 4.7. Criteria related with RQ4.

ID	**Criteria**	**Category**	**Weight**	**RQ**
C10	Normalisation	Measurement Method and Procedure	1	RQ4
C11	Grouping Approach - Temporal	Measurement Method and Procedure	1	RQ4
C12	Large Merge Transactions	Measurement Method and Procedure	1	RQ4
C13	Branches	Measurement Method and Procedure	1	RQ4
C14	Aggregation of Base Measures	Measurement Method and Procedure	1	RQ4
C15	Single Artefact Commits - Aggregation of Base Measures	Measurement Method and Procedure	1	RQ4
C16	Refactorings of Software Artefacts	Measurement Method and Procedure	1	RQ4
C17	Adequateness of Version Control (VC) History	Measurement Method and Procedure	3	RQ4
C18	Relative Strength (Measurement Function)	Measurement Method and Procedure	1	RQ4

These evaluation criteria in Table 4.7 are part of our overall EC evaluation criteria and are used to answer our fourth research question (RQ4). The impact of these criteria on EC is not high as the criteria identified under other categories earlier. So we assign a lower weight (1) to these criteria. The only exception is C17. This is a prerequisite and it is assigned a high weight (3).

4.6. Research Question 5: Do existing studies use scale type and mathematical validation?

As stated by Zuse [26] and Fenton [31], scale types are very important to determine which mathematical and statistical operations can be applied on measurement results. For example, it is invalid to compute multiplication, division and ratios with nominal, ordinal and interval scales. The scale type limits the analysis. One can perform only the operations which are applicable to the scale of the given measure. The table in the online appendix [83] provides a complete list of the scale types and permissible operations for each type. In this study, we evaluate the scale types of EC and its sub-components. Furthermore, we evaluate the design of the measurement methods in terms of its scale types. If an EC measure uses incompatible scale types, numbers produced will be flawed and meaningless. If we use these values in analysis models, they will lead to misleading conclusions about software products and processes. For example, Zuse [26] has identified three concurrent scale types for Cyclomatic Complexity: ordinal, ratio and absolute. Typically, a measurement method should involve a single scale type.

We add a last evaluation criterion (Criterion 19) to consider the mathematical validation as part of research question five (RQ5). As this principle is very important for the soundness of a EC measure, we have assigned the highest weight which is 3.

Table 4.8. Criteria related with RQ5.

ID	**Criteria**	**Category**	**Weight**	**RQ**
C19	Scale Type and Mathematical Validation	Scale Type and Mathematical Validation	3	RQ5

5. EVALUATING EVOLUTIONARY COUPLING MEASURES

In this chapter, we evaluated existing EC measures according to the criteria provided in the previous chapter. We first published our evaluation results in [5]. In particular, this chapter makes the following contributions:

(i) We show the weaknesses and strengths of current EC measures based on measurement theory principles.
(ii) We provide recommendations for practitioners and researchers about what EC measure to use and not to use as well as when to use these measures.

The following subsections provide evaluation process details and evaluation results for each RQ separately.

5.1. Applying the Evaluation Criteria

Four researchers were involved in the assessment process. One researcher evaluated all papers against our evaluation criteria. Then three senior researchers also evaluated a randomly selected set of three papers independently to validate the assessments made by the first author. Disagreements among assessors were resolved with a meeting and the evaluation criteria clarified so that such disagreements could be avoided in the future. We have followed the following steps to randomize the papers. Papers have been assigned numbers according to the their order in the table. Then these numbers (from 1 to 15) have been written on small pieces of papers. Three numbers have been randomly chosen from these papers.

5.2. RQ1: What are the objectives of the EC measurement?

Table 5.1 summarises our findings on the objectives and users of EC measures of published studies. We have identified six different objectives in total. The most common objectives are defect prediction and change prediction. The less common objective is impact analysis. The intended users of EC measures are diverse covering managers, team leaders, developers, analysts, testers and architects.

Table 5.1. Objectives and users of EC measurement (RQ1).

Measurement Objective	**Intended Users**	**Studies**
Defect Prediction	Managers / Team Leaders	Gall et al. [7,8], Beyer and Hassan [10], D'Ambros et al. [17], Cataldo et al. [29], Steff and Russo [14], Kouroshfar [16], Kirbas et al. [3,4]
Impact Analysis	Analysts / Testers	Pirklbauer [84]
Change Prediction	Developers	Ying et al. [20], Zimmermann et al. [21], Kagdi et al. [73] [85], Zou et al. [72]
Bug Localisation	Developers	Tantithamthavorn et al. [15]
Visualisation	Architects / Managers / Team Leaders	Ball et al. [6], Pinzger et al. [9], Beyer and Hassan [10], Steff and Russo [14]
Cross-cutting concern (Aspect) Detection	Developers	Breu and Zimmermann [11], Eaddy et al. [12], Adams et al. [13]

5.3. RQ2: Do existing EC studies identify entities and attributes to be measured?

Table 5.2 summarises the evaluation results of criteria one and two, which are related to RQ2. All EC measures satisfy both criteria. So the answer to this research question is a clear Yes.

Table 5.2. EC measurement evaluation results for research question 2 (RQ2).

Criteria	**Ball et al. [6]**	**Gall et al. [7]**	**Gall et al. [8]**	**Ying et al. [20]**	**Zimmermann et al. [21]**	**Pinzger et al. [9]**	**Breu et al. [11]**	**Beyer et al. [10,82]**	**Kagdi et al. [73]**	**Kagdi et al. [85]**	**Zou et al. [72]**	**D'Ambros et al. [17]**	**Cataldo et al. [29]**	**Kouroshfar [16]**	**Mondal et al. [79]**	**SUM**
C1. Granularity of the Entity	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	**15/15**
C2. Grouping Approach - Entity Level	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	**15/15**
SCORE	**6/6**	**6/6**	**6/6**	**6/6**	**6/6**	**6/6**	**6/6**	**6/6**	**6/6**	**6/6**	**6/6**	**6/6**	**6/6**	**6/6**	**6/6**	

5.4. RQ3: Do existing EC studies use sound empirical relation systems?

Table 5.3 summarises the evaluation results of criteria from three to nine, which are related to RQ3. The last column of the table shows the total number of studies satisfying a criterion at each row. Each criterion is considered only by few studies (Max: 6, Min: 0 out of 15 studies). EC studies are relatively successful in addressing the criterion C8 (Temporal Aspect - Recency) which have been covered by 6 studies. No study satisfies all seven criteria related to the RQ3. Furthermore, no study provides an implicit definition of the empirical relation system and empirical relations used in their EC measure design. These results suggest that EC measures have problems with establishing a sound empirical relation system.

Table 5.3. EC measurement evaluation results for research question 3 (RQ3).

Criteria	**Ball et al. [6]**	**Gall et al. [7]**	**Gall et al. [8]**	**Ying et al. [20]**	**Zimmermann et al. [21]**	**Pinzger et al. [9]**	**Breu et al. [11]**	**Beyer et al. [10,82]**	**Kagdi et al. [73]**	**Kagdi et al. [85]**	**Zou et al. [72]**	**D'Ambros et al. [17]**	**Cataldo et al. [29]**	**Kouroshfar [16]**	**Mondal et al. [79]**	**SUM**
C3. Locality Aspects	X	✓	✓	X	X	✓	X	X	X	X	X	X	X	✓	X	**4/15**
C4. Change Scope	X	✓	✓	X	X	✓	X	X	X	X	X	X	X	X	X	**3/15**
C5. Change Type	X	X	X	X	✓	X	✓	X	X	X	✓	X	X	X	X	**3/15**
C6. Distinct Committers	X	X	X	X	X	X	✓	X	X	X	✓	X	X	X	X	**2/15**
C7. Temporal Aspect - Regularity	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	**0/15**
C8. Temporal Aspect - Recency	X	X	X	X	✓	X	✓	X	✓	✓	✓	✓	X	X	X	**6/15**
C9. Change Size	X	X	X	X	X	X	✓	X	X	X	X	X	X	X	X	**1/15**
SCORE	**0/21**	**6/21**	**6/21**	**0/21**	**6/21**	**6/21**	**12/21**	**0/21**	**3/21**	**3/21**	**9/21**	**3/21**	**0/21**	**3/21**	**0/21**	

5.5. RQ4: Do existing EC studies define measurement method and procedures?

Table 5.4 summarises the evaluation results of criteria from C10 to C18, which are related to RQ4. Each criterion is considered only by few studies (Max: 9, Min: 0 out of 15 studies). EC studies are relatively successful in addressing criterion C11 (Grouping Approach - Temporal) which has been covered by 9 studies.

No study satisfies all nine criteria related to RQ4. Furthermore, no study provides a detailed measurement procedure for EC measurement. These evaluation results suggest that measurement method and procedures for EC measurement are not sufficiently well-defined and standardized.

5.6. RQ5: Do existing EC studies use scale type and mathematical validation?

Another problematic area is mathematical validation (criterion 19). No EC measure published performed an explicit mathematical validation (Table 5.5). Furthermore, all existing EC measures apply summation to different potential base units in their EC calculations such as cross-system and within-system co-changes.

5.7. Limitations of the Study

The validity of our work is mainly related with the appropriateness of inferences made on the basis of our assessment, specifically whether our assessment can measure how well EC measures are currently measured. The main threats to validity arise from the subjective nature of the criteria identification process. The criteria used are based on our own experiences and those reported in the literature. To address this threat, we used some SLR practices [71] (a cut-down SLR approach) to identify a comprehensive set of studies proposing EC measures, which helps to identify a comprehensive set of criteria. Nevertheless, there might be some other criteria which could not be identified by this study. We hope that this will provide the foundations for further study of the

Table 5.4. EC measurement evaluation results for research question 4 (RQ4).

Criteria	**Ball et al. [6]**	**Gall et al. [7]**	**Gall et al. [8]**	**Ying et al. [20]**	**Zimmermann et al. [21]**	**Pinzger et al. [9]**	**Breu et al. [11]**	**Beyer et al. [10,82]**	**Kagdi et al. [73]**	**Kagdi et al. [85]**	**Zou et al. [72]**	**D'Ambros et al. [17]**	**Cataldo et al. [29]**	**Kouroshfar [16]**	**Mondal et al. [79]**	**SUM**
C10	✓	X	X	X	X	X	X	✓	X	X	✓	X	✓	X	✓	**5/15**
C11	X	✓	✓	✓	✓	✓	✓	X	X	X	✓	X	✓	✓	X	**9/15**
C12	X	X	X	✓	✓	X	X	X	✓	✓	X	✓	X	X	X	**5/15**
C13	X	X	X	X	✓	X	X	X	X	X	X	X	X	X	X	**1/15**
C14	X	X	X	X	X	X	✓	✓	X	X	X	✓	✓	X	X	**4/15**
C15	X	X	X	X	X	X	X	✓	X	X	✓	X	X	X	X	**2/15**
C16	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	**0/15**
C17	X	X	X	✓	X	X	X	X	X	X	X	✓	X	X	X	**2/15**
C18	X	X	X	X	X	X	X	✓	X	X	X	X	X	X	X	**1/15**
SCORE	**1/11**	**1/11**	**1/11**	**5/11**	**3/11**	**1/11**	**2/11**	**4/11**	**1/11**	**1/11**	**3/11**	**5/11**	**3/11**	**1/11**	**1/11**	

Table 5.5. EC measurement evaluation results for research question 5 (RQ5).

Criteria	**Ball et al. [6]**	**Gall et al. [7]**	**Gall et al. [8]**	**Ying et al. [20]**	**Zimmermann et al. [21]**	**Pinzger et al. [9]**	**Breu et al. [11]**	**Beyer et al. [10,82]**	**Kagdi et al. [73]**	**Kagdi et al. [85]**	**Zou et al. [72]**	**D'Ambros et al. [17]**	**Cataldo et al. [29]**	**Kouroshfar [16]**	**Mondal et al. [79]**	**SUM**
C19. Scale Type and Math. Validation	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	**0/15**
SCORE	**0/3**	**0/3**	**0/3**	**0/3**	**0/3**	**0/3**	**0/3**	**0/3**	**0/3**	**0/3**	**0/3**	**0/3**	**0/3**	**0/3**	**0/3**	

criteria expected from EC measurement.

Another threat to the validity of this study is that the studies used for formulating the Criterion 18 have been themselves assessed against this criteria. Although this is not ideal, all other criteria are driven by measurement theory and metrology principles. The general measurement principles guiding the formulation of the criteria are general. There can be also some overlap between these broad categories defined in the paper such as numerical assignment rules and measurement method as observed. However, most of the criteria like grouping approach (C2), locality aspects (C3), temporal aspect-regularity (C7), large merge transactions (C12), adequateness of VC history (C17), etc. are reflecting the EC measurement specifics. We used these general categories to guide our criteria formulation, brain storming and to help cover all aspects of measurement. In our criteria formulation we use measurement method & procedure category (addressed by RQ4) to capture the practicalities of EC measurement and whether EC measurement is well defined and standardised to ensure its repeatability, accuracy and repetitiveness. On the other hand, numerical assignment

rules are utilised in the scope of "sound empirical relation system" category (addressed by RQ3) to create concrete cases for the criteria to check whether numerical relations preserve empirical relations. Although there is overlap, the focus is different and they uncover different aspects of EC measurement.

Another threat is potential bias in the evaluation process. However we employed a validation exercise with three senior researchers to mitigate any potential bias.

External validity relates to the generalisation of our study results. We evaluated a number of EC measures. These may not be exhaustive and new EC measures might be proposed in the future.

5.8. Discussion

Our EC measure evaluations suggest that EC is currently not measured well. The particular weakness revealed by our research questions provide important information about the quality of existing EC measures and potential areas to improve. The EC measures of Breu & Zimmermann [11] and Zou et al. [72] are particularly worth to highlight as they have the highest overall scores. Our evaluations suggest that there is a need for new EC measures, which more clearly address the criteria we present. Furthermore, the criteria we presented can be used in the future to benchmark any newly developed measures of EC. We now discuss the answers to each of our research questions:

5.8.1. RQ1: What are the objectives of the EC measurement?

Our results show that EC measures are used for various purposes in software engineering research. We found six different objectives for EC measurement: Defect prediction, impact analysis, change prediction, bug localisation, visualisation and aspect detection. All of these areas of research could be improved in the future if EC measures were developed that satisfy our evaluation criteria.

5.8.2. RQ2: Do existing EC studies identify entities and attributes to be measured?

Our results reveal that all existing EC studies identify entities and attributes to be measured. Both criteria C1 and C2 of RQ2 are the only criteria satisfied by all existing studies. Existing studies do very well in this area. The level of granularity (C1) used by the studies vary and include files, classes and methods. The main grouping approach (C2) is VCS commits. This result provides a promising basis for improving other aspects of EC measurement.

5.8.3. RQ3: Do existing EC studies use sound empirical relation systems?

Breu & Zimmermann [11] and Zou et al. [72] satisfy the highest score, 12 and 9 respectively, in relation to this research question. The measurement objectives of these studies are aspect detection and change prediction respectively. Unfortunately some RQ3 criteria were satisfied by very few studies. For example the criterion 6 (distinct committers) is satisfied by only Breu & Zimmermann [11] and Zou et al. [72] and the criterion 9 (co-change size) is satisfied by only Breu & Zimmermann [11].

Overall our results suggest that existing EC measures need improvement. Evaluation criteria related to the empirical relation system (i.e. change locality, change scope, temporal, change size and change type) potentially lead to studies using different base units and so are important to the reliable measurement of EC. We suggest that practitioners and researchers measure each aspect identified in our evaluation criteria separately and use each in their analysis models as separate parameters. None of the studies used satisfied criterion C7. This suggests that the regularity of co-changes is not addressed in any of the EC measures that we assessed. Again, this may mean that the measures currently used are undermined.

5.8.4. RQ4: Do existing EC studies define measurement method and procedures?

D'Ambros et al. [17] and Ying et al. [20] satisifed the most criteria for their methods and procedures. In addition some criteria are satisfied by very few studies. For example, criterion 13 (branches) is only covered by Zimmermann et al. [21]; criterion 15 (single artefact commits) is only covered by Beyer & Hassan [10] and Zou et al. [72]; criterion 17 (adequateness of version history history) is only covered by Ying et al. [20] and D'Ambros et al. [17]; criterion 18 (relative strength) is only covered by Beyer et al. [10,82]. However none of the studies satisfied criterion 16 in EC measurement, which means that no study considered refactorings within or across projects is a potential candidate for EC measurement.

The studies we evaluate highlight many areas where improvement is needed. Overall the methods and procedures used to measure EC are not well-defined. Consequently measurement results of the same entity by different measurers can potentially be significantly different. This means that the usability and repeatability of existing EC measures is not strong which will make it difficult to replicate EC studies. It is essential that detailed measurement procedures should be described for EC measurement.

5.8.5. RQ5: Do existing EC studies use scale type and mathematical validation?

Our results suggest that this area is not addressed well by current EC measures. None of the existing studies declare the scale type of their EC measures or attempt to validate the measures mathematically. EC data is generally not normally distributed as reported by several studies [3, 4, 7, 17]. This limits the statistical methods that can be used on EC measures. For example, Pearson correlation as well as other parametric methods like ANOVA (Analysis of Variance) will not apply to EC measures. Nonparametric methods such as Spearman and Kendall correlations should be preferred over parametric methods such as Pearson correlation.

Furthermore, aggregation of different types of co-changes, especially RQ3 related criteria such as C3 (within vs. cross system) and C5 (deletion/creation vs. modification), may be also problematic as these can be potentially interpreted as different base units in their EC calculations. We suggest measuring different types of co-changes separately and aggregating them within these different types when calculating EC.

Overall our results suggest that the quality of measurement used in EC needs significant improvement. Even the studies that score highest in our evaluations miss important criteria. On the other hand, the criteria related to RQ1 (providing the objectives of measurement) and RQ2 (identifying entities and attributes) are satisfied by all studies. So the results we present do show that there is a base of good measurement practices from which EC measurement could be strengthened.

6. RELATED WORK

6.1. Evolutionary Coupling

EC was first identified in 1997 by Ball et al. [6]. Early studies [6–9] on EC focused on the relation between EC and architectural problems and EC was used as an indicator of architectural weaknesses and modularity problems. Classes that were frequently changed together during the evolution of a system were presented visually using EC information by Ball et al. [6]. Clusters of classes were identified according to EC measures. Ball et al. showed that classes belonging to the same cluster were semantically relevant. EC among different clusters was used as an indicator of ineffective class partitioning. Gall et al. analysed EC at a module-level and reported that EC provides useful insights into system architecture [7]. They identified potential structural shortcomings and detected modules and programs which should undergo restructuring or even reengineering. Another study by Gall et al. analysed EC at a class level on an industrial software system [8]. This study was important since it demonstrated that EC could be used to identify architectural weaknesses such as poorly designed interfaces and inheritance hierarchies. Pinzger et al. showed that candidate modules for refactoring could be detected by showing ECs between modules on Kiviat diagrams [9]. Beyer and Hassan explored EC data in the calculation of the distance between two files in a version control system (VCS) and displayed results as a series of animated panels [10]. They showed how the structure of a software system decayed or remained stable over time.

Besides detecting architectural problems, EC has also been used to predict possible co-changes and to recommend them to developers. In a study by Ying et al., an approach using data mining techniques was developed to recommend related source code parts to software developers assigned modification requests (MRs) [20]. Applying the approach to open source projects revealed important dependencies. A study by Zimmermann et al. presented a technique which predicted the parts of source code likely to change given the already changed parts of source code (at file, class, property,

method levels) [21]. Association rule mining was used to detect ECs.

EC has also proven useful in detecting the cross-cutting concerns scattered across systems. Breu et al. [11] leverage EC information to mine aspect candidates (identifying cross-cutting concerns). Eaddy et al. [12] argued that cross-cutting concerns are harder to implement and change consistently because multiple (possibly unrelated) locations in the code have to be found and updated simultaneously. Their study suggested that increased cross-cutting concerns may cause or even contribute to defects. Adams et al. [13] developed an aspect mining technique based on co-addition or co-removal of dependencies on program entities over time. They suggest that detailed knowledge about cross-cutting concerns in the source code is crucial for the cost-effective maintenance and successful evolution of large systems.

No study has looked at the quality of EC measurement and the way in whether EC is measured inconsistently across studies.

6.2. Relationship between Evolutionary Coupling and Defects

EC measures have also been used in defect prediction studies. These studies are related to our first research question (RQ1). First, we focus on the studies, which reported a relation between EC and defects. Steff and Russo created sequential commit graphs of evolutionary coupled classes [14]. They showed that the graphs could be used for defect prediction. A study by Tantithamthavorn et al. proposed improvements to existing defect localisation methods by utilising EC information [15]. The proposed method was applied and verified on two open source projects (Eclipse SWT and Android ZXing).

D'Ambros et al. [17] analysed three open source software systems and detected correlation between EC and software defects. This was the first study focusing explicitly on the relationship between EC and software defects, which corresponds to our first research question (RQ1). They found a positive correlation between EC and defects. Furthermore, they reported that defects with a high severity exhibited a correlation

with EC. This study considered only EC between classes within a project. Another study reported by Kouroshfar concluded that cross-subsystem EC measures are more related to defects than within-subsystem EC [16]. Kouroshfar's findings are related to our second research question (RQ2), as Kouroshfar proposed different kinds of EC (between sub-systems vs. within sub-systems) as a factor affecting the relationship between defects and EC.

Other studies using EC metrics suggest that EC does not contribute to defects and is not useful for identifying defects. These studies could not find any relationship between EC and defects, which is related to our RQ1. In a study conducted by Graves et al., various statistical models were developed to assess which features of the revision history of a module could be used for defect prediction [18]. Results from study showed that prediction performance of the models using EC measures were lower compared with other models. Another study by Knab et al. found that EC measures did not give good results for predicting defects [19]. In that study, the ability of EC to predict defect density was tested. In our previous studies of EC [3,4] we examined the effect of EC on software defects for an industrial legacy banking system. For some modules we observed significant correlation between EC and defect measures, whereas for others no relation was detected. This study is different from our previous studies in terms of companies involved and the analysis applied. In this study we analyse also a large modern telecommunications software and used different analysis such as multivariate regression analysis, defect type and module characteristic analysis.

The previous studies on EC focused on open source projects. Also, most of these studies did not investigate large projects. Our study is different in the sense that we investigate industrial projects, which have very different software development processes and culture than the open source projects. Moreover, the size of the projects we analysed are different to existing studies. For example, the size of the projects studied by D'Ambros et al. [17] were between 1K and 3K (number of classes). The size of the projects that we studied were 20K and 150K (as number of files). Large industrial systems have rarely been empirically studied to understand the relationship between EC and defects. This is an important contribution of our work on existing

knowledge of EC.

In contrast to previous studies, we also show that the relationship between EC and defects varies for different modules even in the same system. We provide the distribution of numerical values for EC - defect relationship as histograms. This introduces a more realistic and probabilistic model for the EC - defect relationship and can be also used to explain the contradictory results reported by different studies. Furthermore, we attempt to explain factors affecting the relationship between EC measures and defects, which has not been explicitly addressed by previous studies.

6.3. Software Measurement and Metrology

The groundwork for software measurement was established mainly in the 70s through various studies such as modularity by Parnas [86], Yourdon [87], complexity by McCabe [88] and Halstead [89], sofware metrics by Gilb [90] and Boehm [91], function point by Albrecht [92], GQM by Basili [30]. Based on these earlier work, measurement theory was introduced to software measurement in the 80s by Zuse, Fenton. Zuse [26] contributed to the use of measurement theory in the area of software measurement. He extended classic measurement theory to the needs of software measurement [26]. Fenton [31] provided information about measurement fundamentals, data collection and data analysis as well as the mathematical and statistical background of software measurement. Both Fenton and Zuse used the Representational Measurement Theory (RMT) approach. Furthermore, measurement paradigms such as GQM (Goal-Question-Metric) [30] by Basili et al. was also introduced in the mid 80s.

In the 90s, software measures on object-oriented programming (OOP) were developed by authors such as Chidamber and Kemerer, Briand, Morris, Bieman, Sharble. OOP measures proposed by Shyam R Chidamber and Chris F. Kemerer in 1994 are known as CK metrics. CK metrics are still the most commonly used OOP metrics. Briand and et al. proposed coupling and cohesion measures for OOP [22, 93]. In 1995 Kitchenham et al. [94] proposed a framework for validating software measurement by

identifying the elementary components of measures and the measurement process.

In the 21st century, software measurement has seen massive standardisation effort such as ISO/IEC 15939:2002 on software measurement process, functional size measurement methods like COSMIC International Standard (ISO/IEC 19761:2011), FiSMA: ISO/IEC 29881:2008, IFPUG: ISO/IEC 20926:2009, Mark-II: ISO/IEC 20968:2002, NESMA: ISO/IEC 24570:2005. Another novelty was the introduction of metrology to the software measurement area. Abran [28] took a measurement method point of view for software measurement. Metrology is the science of measurement and includes all theoretical and practical aspects of measurement according to the definition by the International Bureau of Weights and Measures (BIPM) [39]. His work is complementary to the previous work on the measurement theory [26, 31]. Cheikhi et al. [40] applied the metrology concepts to the Chidamber and Kemerer (CK) measures suite. They investigated how well CK measures address the metrology principles. They analyse all CK measures from metrology perspectives and provide their mapping to the metrology concepts.

7. CONCLUSIONS AND FUTURE WORK

7.1. Relationship Between Evolutionary Coupling And Defects

In this dissertation, we studied the relationship between EC and software defects in two large industrial software systems. We reported a positive correlation between EC and defect measures in the software maintenance / evolution phase of systems from two different companies. Our results indicated low, moderate and high level correlation, with varied correlation strength across modules. Our regression analysis results indicated that EC measures could be useful for explaining defects. The box plots drawn for each module separately showed the potential of EC measures to distinguish defective and non-defective files. We also observed that the company using practices such as agile and TDD had relatively fewer modules with high EC-defect correlation values. However, this finding needs to be further investigated on more companies for generalisable conclusions.

We also studied the reasons for variation of the observed effect of EC on software defects for different modules. We found that modules which were small in size in terms of file and developer numbers, tended to be less correlated with EC. Interconnections between files in a module can grow quadratically with the number of files. Increasing number of inter-related files make a module more difficult to comprehend and perform changes. This complexity may eventually lead to defects and this may be one of the reasons for variation across modules. Furthermore, we observed that EC measures showed higher correlation with some types of defects (based on root causes) such as code implementation, acceptance criteria and analysis problems. The dispersion of these defect types could be another reason for these varying effects. Different modules have different defect types and EC has different relationships with different defect types. It is more likely that high EC will cause code and test implementation defects, because of the ripple effects due to high number of inter-related files. Module characteristics and defect types may also explain why different results are reported by different studies in the literature. Different applications or modules analysed may have different characteristics

and defect types. We recommend that researchers report characteristics and defect types of modules in their EC studies to account for the possible effect of the context in their results. We also recommend that practitioners add EC measures to their metric suite for software design evaluation and consider the characteristics and defect types of their modules in their evaluation.

7.2. Evolutionary Coupling Measurement Evaluation

In this dissertation, we evaluate EC measurement from a measurement theory perspective. We combine the software measurement ideas of Fenton, Zuse and Abran to develop 19 EC measurement evaluation criteria. We use these criteria to evaluate the current published studies on EC measurement. Despite the importance of EC and the published studies of EC, our results suggest that many basic principles of good software measurement have not been used in the measurement of EC. Scale type and mathematical validation stands out an area not addressed by EC studies. None of the existing EC studies declare their scale type nor attempt to validate their measures mathematically. Our evaluation results suggest that establishing a sound empirical relation system is also a problematic area. Existing studies fail to address the different types of co-changes such as locality (within vs. cross system) and change type (deletion/creation/modification). We also found that EC measurement methods and procedures are not well reported by existing studies. In addition no EC study presents measures that account for the temporal aspects of EC in terms of regularity or the impact of refactorings on EC. Overall our results suggest that there is much work to be done to put EC measurement on a firm footing that will enable the reliable measurement of EC and the accurate replication of EC measurement.

To the best of our knowledge, this is the first work that applies measurement theory and metrology principles to EC measurement. Although our results suggest significant weakness in the way that EC has previously been measured, it is highly likely that measurement in other areas of software engineering also requires significant improvement. Indeed Cheikhi et al. [40] show such measurement weakness in their analysis of OO metrics. We believe that the results and the criteria developed in this

study will guide future research in EC to improve how EC is measured. Indeed the criteria presented in this dissertation can be adapted and used in any EC related study to ensure measurement quality.

7.3. Future Work

In future work, we would like to extend our empirical study on the relationship between EC and defects by including more commercial systems and projects. Using defect density in our study mitigates the risk of size as a possible confounding factor. We are planning future investigations to explore the effect size of a large number of factors related to defects. Furthermore we are planning future investigations to explore the effect size of a large number of factors related to defects.

Another future direction of research can be modelling EC by using Bayesian networks. A Bayesian network as probabilistic directed acyclic graph can be used to model the influence of structural, semantic, dynamic couplings on EC. Given EC values, the network can be used to compute the probabilities of the presence of various couplings (structural, semantic, dynamic) between software artefacts.

Our evaluation on existing EC measures revealed that there are no EC measures satisfying all 19 criteria. One possible future research would be to develop new EC measures satisfying all or most of these criteria. The criteria developed in this study will guide future research in EC to improve how EC is measured. This research can be also combined with objectives. We have identified that an EC study aims one of these six objectives: defect prediction, impact analysis, change prediction, bug localisation , visualisation, cross-cutting concern (aspect) detection. For each EC objective potentially a new optimal EC measure can be researched. All of these areas of research could be improved in the future if EC measures were developed that satisfy our evaluation criteria. Furthermore, the criteria we presented can be used in the future to benchmark any newly developed measures of EC.

REFERENCES

1. Kirbas, S., B. Caglayan, T. Hall, S. Counsell, D. Bowes, A. Sen and A. Bener, "The relationship between evolutionary coupling and defects in large industrial software", *Journal of Software: Evolution and Process*, pp. e1842–n/a, 2017.

2. Kirbas, S. and A. Sen, "Yazılım Depoları Madenciliği ile Endüstriyel Yazılım Evrimi İncelemesi.", *Proceedings of the 7th Turkish National Software Engineering Symposium*, UYMS'13, Izmir, Turkey, 2013.

3. Kirbas, S., A. Sen, B. Caglayan, A. Bener and R. Mahmutogullari, "The Effect of Evolutionary Coupling on Software Defects: An Industrial Case Study on a Legacy System", *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '14, pp. 6:1–6:7, ACM, Torino, Italy, 2014.

4. Kirbas, S., A. Sen, B. Caglayan and A. Bener, "Değişiklik Bağlaşımı ve Yazılım Hataları İlişkisinin İncelenmesi", *Proceedings of the 8th Turkish National Software Engineering Symposium*, UYMS'14, Güzelyurt, KKTC, 2014.

5. Kirbas, S., T. Hall and A. Sen, "Evolutionary coupling measurement: Making sense of the current chaos", *Science of Computer Programming*, Vol. 135, pp. 4 – 19, 2017.

6. Ball, T., J.-M. Kim, A. A. Porter and H. P. Siy, "If your version control system could talk", *ICSE Workshop on Process Modelling and Empirical Studies of Software Engineering*, 1997.

7. Gall, H., K. Hajek and M. Jazayeri, "Detection of logical coupling based on product release history", *Proceedings of the International Conference on Software Maintenance*, pp. 190–198, IEEE, 1998.

8. Gall, H., M. Jazayeri and J. Krajewski, "CVS release history data for detecting logical couplings", *Proceedings of the Sixth International Workshop on Principles of Software Evolution*, pp. 13–23, IEEE, 2003.

9. Pinzger, M., H. Gall, M. Fischer and M. Lanza, "Visualizing multiple evolution metrics", *Proceedings of the 2005 ACM symposium on Software visualization*, pp. 67–75, ACM, New York, NY, USA, 2005.

10. Beyer, D. and A. E. Hassan, "Animated visualization of software history using evolution storyboards", *Proceedings of the 13th Working Conference on Reverse Engineering*, WCRE'06, pp. 199–210, IEEE, Benevento, Italy, 2006.

11. Breu, S. and T. Zimmermann, "Mining aspects from version history", *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*, ASE'06, pp. 221–230, IEEE, Tokyo, Japan, 2006.

12. Eaddy, M., T. Zimmermann, K. D. Sherwood, V. Garg, G. C. Murphy, N. Nagappan and A. V. Aho, "Do crosscutting concerns cause defects?", *IEEE Transactions on Software Engineering*, Vol. 34, No. 4, pp. 497–515, 2008.

13. Adams, B., Z. M. Jiang and A. E. Hassan, "Identifying Crosscutting Concerns Using Historical Code Changes", *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, ICSE '10, pp. 305–314, 2010.

14. Steff, M. and B. Russo, "Co-evolution of logical couplings and commits for defect estimation", *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, MSR'12, pp. 213–216, Zurich, Switzerland, 2012.

15. Tantithamthavorn, C., A. Ihara and K.-I. Matsumoto, "Using Co-change Histories to Improve Bug Localization Performance", *14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, SNPD'13, pp. 543–548, IEEE, Honolulu, HI, USA, 2013.

16. Kouroshfar, E., "Studying the effect of co-change dispersion on software quality", *Proceedings of the 35th International Conference on Software Engineering*, ICSE'13, pp. 1450–1452, IEEE, San Francisco, CA, USA, 2013.

17. D'Ambros, M., M. Lanza and R. Robbes, "On the relationship between change coupling and software defects", *Proceedings of the 16th Working Conference on Reverse Engineering*, WCRE'09, pp. 135–144, IEEE, Lille, France, 2009.

18. Graves, T. L., A. F. Karr, J. S. Marron and H. Siy, "Predicting fault incidence using software change history", *IEEE Transactions on Software Engineering*, Vol. 26, No. 7, pp. 653–661, 2000.

19. Knab, P., M. Pinzger and A. Bernstein, "Predicting defect densities in source code files with decision tree learners", *Proceedings of the 2006 international workshop on Mining software repositories*, pp. 119–125, ACM, Shanghai, China, 2006.

20. Ying, A. T., G. C. Murphy, R. Ng and M. C. Chu-Carroll, "Predicting source code changes by mining change history", *IEEE Transactions on Software Engineering*, Vol. 30, No. 9, pp. 574–586, 2004.

21. Zimmermann, T., A. Zeller, P. Weissgerber and S. Diehl, "Mining version histories to guide software changes", *IEEE Transactions on Software Engineering*, Vol. 31, No. 6, pp. 429–445, 2005.

22. Briand, L. C., J. W. Daly and J. K. Wust, "A unified framework for coupling measurement in object-oriented systems", *IEEE Transactions on Software Engineering*, Vol. 25, No. 1, pp. 91–121, 1999.

23. Poshyvanyk, D., A. Marcus, R. Ferenc and T. Gyimóthy, "Using information retrieval based coupling measures for impact analysis", *Empirical software engineering*, Vol. 14, No. 1, pp. 5–32, 2009.

24. Arisholm, E., L. C. Briand and A. Foyen, "Dynamic coupling measurement for

object-oriented software", *IEEE Transactions on Software Engineering*, Vol. 30, No. 8, pp. 491–506, 2004.

25. Fenton, N. and B. Kitchenham, "Validating software measures", *Software Testing, Verification and Reliability*, Vol. 1, No. 2, pp. 27–42, 1991.

26. Zuse, H., *A Framework of Software Measurement*, Walter de Gruyter & Co., Hawthorne, NJ, USA, 1997.

27. Briand, L. C., S. Morasca and V. R. Basili, "Property-based software engineering measurement", *IEEE Transactions on Software Engineering*, Vol. 22, No. 1, pp. 68–86, 1996.

28. Abran, A., *Software Metrics and Software Metrology*, Wiley-IEEE Computer Society Pr, 2010.

29. Cataldo, M., A. Mockus, J. Roberts and J. Herbsleb, "Software Dependencies, Work Dependencies, and Their Impact on Failures", *IEEE Transactions on Software Engineering*, Vol. 35, No. 6, pp. 864–878, November 2009.

30. Basili, V. R. and D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data", *IEEE Transactions on Software Engineering*, Vol. 10, No. 6, pp. 728–738, 1984.

31. Fenton, N. and J. Bieman, *Software metrics: a rigorous and practical approach*, CRC Press, 2014.

32. Stevens, W. P., G. J. Myers and L. L. Constantine, "Structured design", *IBM Systems Journal*, Vol. 13, No. 2, pp. 115–139, 1974.

33. Chidamber, S. R. and C. F. Kemerer, "A Metrics Suite for Object Oriented Design", *IEEE Transactions on Software Engineering*, Vol. 20, No. 6, pp. 476–493, June 1994.

34. Poshyvanyk, D. and A. Marcus, "The Conceptual Coupling Metrics for Object-Oriented Systems", *Proceedings of the 22nd IEEE International Conference on Software Maintenance*, Vol. 6 of *ICSM'06*, pp. 469–478, 2006.

35. Martin, R. C., *Agile software development: principles, patterns, and practices*, Prentice Hall PTR, 2003.

36. Pressman, R. S., *Software engineering: a practitioner's approach*, Palgrave Macmillan, 2005.

37. Agrawal, R., T. Imieliński and A. Swami, "Mining association rules between sets of items in large databases", *Acm sigmod record*, Vol. 22, pp. 207–216, ACM, 1993.

38. Roberts, F. S., *Measurement theory*, Cambridge University Press, 1985.

39. International Bureau of Weights and Measures (BIPM), *What is metrology?*, 2016, `http://www.bipm.org/en/worldwide-metrology/`, accessed at August 2016.

40. Cheikhi, L., R. E. Al-Qutaish, A. Idri and A. Sellami, "Chidamber and kemerer object-oriented measures: Analysis of their design from the metrology perspective", *International Journal of Software Engineering & Its Applications*, Vol. 8, No. 2, 2014.

41. (STF), S. T. F., *Web page of Software Testing Fundamentals*, 2016, `http://softwaretestingfundamentals.com/defect/`, accessed at December 2016.

42. Lehman, M. M., "Programs, life cycles, and laws of software evolution", *Proceedings of the IEEE*, Vol. 68, No. 9, pp. 1060–1076, 1980.

43. *ISO/IEC/IEEE 12207-2008: Standard for Systems and Software Engineering – Software Life Cycle Processes*, Standard, International Standards Organisation (ISO), 2008.

44. *ISO/IEC 14764 IEEE Std 14764-2006 Software Engineering – Software Life Cycle Processes – Maintenance*, Standard, International Standards Organisation (ISO), 2006.

45. Lientz, B. P. and E. B. Swanson, *Software Maintenance Management*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1980.

46. Shapiro, S. S. and M. B. Wilk, "An analysis of variance test for normality (complete samples)", *Biometrika*, pp. 591–611, 1965.

47. Razali, N. M. and Y. B. Wah, "Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests", *Journal of Statistical Modeling and Analytics*, Vol. 2, No. 1, pp. 21–33, 2011.

48. Hopkins, W. G., *A new view of statistics*, Will G. Hopkins, 1997.

49. Lu, H., Y. Zhou, B. Xu, H. Leung and L. Chen, "The ability of object-oriented metrics to predict change-proneness: a meta-analysis", *Empirical Software Engineering*, Vol. 17, pp. 200–242, 2012.

50. Montgomery, D. C., E. A. Peck and G. G. Vining, *Introduction to linear regression analysis*, John Wiley & Sons, 2012.

51. Bennett, K. H. and V. T. Rajlich, "Software maintenance and evolution: a roadmap", *Proceedings of the Conference on the Future of Software Engineering*, pp. 73–87, ACM, 2000.

52. Mens, T., *Introduction and roadmap: History and challenges of software evolution*, Springer, 2008.

53. Visser, J., "Change is the Constant", *ERCIM News*, Vol. 2012, No. 88, 2012.

54. Śliwerski, J., T. Zimmermann and A. Zeller, "When do changes induce fixes?", *ACM sigsoft software engineering notes*, Vol. 30, No. 4, pp. 1–5, 2005.

55. Moser, R., W. Pedrycz and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction", *Proceedings of the ACM/IEEE 30th International Conference on Software Engineering*, ICSE'08, pp. 181–190, IEEE, 2008.

56. Nagappan, N., A. Zeller, T. Zimmermann, K. Herzig and B. Murphy, "Change Bursts as Defect Predictors", *Proceedings of the IEEE 21st International Symposium on Software Reliability Engineering*, ISSRE'10, pp. 309–318, November 2010.

57. Nagappan, N. and T. Ball, "Use of relative code churn measures to predict system defect density", *Proceedings of the 27th International Conference on Software Engineering*, ICSE 2005, pp. 284–292, IEEE, 2005.

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

59. CASCM, *Web page of CA Software Change Manager*, 2013, `http://www.ca.com/us/products/detail/CA-Software-Change-Manager.aspx`, accessed at April 2013.

60. SVN, *Web page of Apache Subversion*, 2015, `http://subversion.apache.org/`, accessed at January 2015.

61. Carlisle, F., J. Eisinger, M. Johnson, V. Kowalski, J. Mukerji, D. Snelling, W. Vambenepe, M. Waschke and V. Wiles, *Configuration Management Database (CMDB) Federation Specification*, Distributed Management Task Force Inc. (DMTF), 2010.

62. JIRA, *Web page of JIRA*, 2015, `https://www.atlassian.com/software/jira`, accessed at January 2015.

63. SPSS, *Web page of SPSS*, 2013, `http://www-01.ibm.com/software/analytics/`

spss/, accessed at April 2013.

64. Conway, M. E., "How do committees invent", *Datamation*, Vol. 14, No. 4, pp. 28–31, 1968.

65. Briand, L. C., J. Wüst, J. W. Daly and D. V. Porter, "Exploring the relationships between design measures and software quality in object-oriented systems", *Journal of systems and software*, Vol. 51, No. 3, pp. 245–273, 2000.

66. Baldwin, C. Y. and K. B. Clark, *Design rules: The power of modularity*, Vol. 1, MIT Press, 2000.

67. Offutt, A. J., M. J. Harrold and P. Kolte, "A software metric system for module coupling", *Journal of Systems and Software*, Vol. 20, No. 3, pp. 295–308, 1993.

68. Hall, T., S. Beecham, D. Bowes, D. Gray and S. Counsell, "A systematic literature review on fault prediction performance in software engineering", *IEEE Transactions on Software Engineering*, Vol. 38, No. 6, pp. 1276–1304, 2012.

69. Sweller, J., "Cognitive Load During Problem Solving: Effects on Learning", *Cognitive Science*, Vol. 12, No. 2, pp. 257–285, 1988.

70. Herzig, K. and A. Zeller, "The Impact of Tangled Code Changes", *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR'13, pp. 121–130, 2013.

71. Kitchenham, B. and S. Charters, *Guidelines for performing systematic literature reviews in software engineering*, Tech. rep., EBSE-2007-01, 2007.

72. Zhou, Y., M. Wursch, E. Giger, H. Gall and J. Lu, "A bayesian network based approach for change coupling prediction", *15th Working Conference on Reverse Engineering*, WCRE'08, pp. 27–36, IEEE, 2008.

73. Kagdi, H., S. Yusuf and J. I. Maletic, "Mining Sequences of Changed-files from

Version Histories", *Proceedings of the 2006 International Workshop on Mining Software Repositories*, MSR '06, pp. 47–53, 2006.

74. Colaço, M., M. Mendonça and F. Rodrigues, "Mining software change history in an industrial environment", *XXIII Brazilian Symposium on Software Engineering*, SBES'09, pp. 54–61, IEEE, 2009.

75. Alali, A., B. Bartman, C. D. Newman and J. I. Maletic, "A Preliminary Investigation of Using Age and Distance Measures in the Detection of Evolutionary Couplings", *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pp. 169–172, 2013.

76. *ISO/IEC 14764: Software Engineering - Software Life Cycle Processes - Maintenance*, Standard, International Standards Organisation (ISO), 2006.

77. Madeyski, L. and M. Jureczko, "Which process metrics can significantly improve defect prediction models? An empirical study", *Software Quality Journal*, Vol. 23, No. 3, pp. 393–422, 2015.

78. Kitchenham, B., "What's up with software metrics?–A preliminary mapping study", *Journal of systems and software*, Vol. 83, No. 1, pp. 37–51, 2010.

79. Mondal, M., C. Roy and K. Schneider, "Improving the detection accuracy of evolutionary coupling by measuring change correspondence", *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE)*, pp. 358–362, February 2014.

80. *ISO/IEC 15939: Software Engineering - Software Measurement Process*, Standard, International Standards Organisation (ISO), 2007.

81. Ericsson, M., W. Lowe, T. Olsson, D. Toll and A. Wingkvist, "A Study of the Effect of Data Normalization on Software and Information Quality Assessment", *Proceedings of the 20th Asia-Pacific Software Engineering Conference*, Vol. 2 of

APSEC '13, pp. 55–60, December 2013.

82. Beyer, D. and A. Noack, *Mining Co-Change Clusters from Version Repositories*, Tech. rep., Ecole Polytechnique Fédérale de Lausanne (EPFL), 2005.

83. Appendix, O., *Mathematical Operations Supported by Scale Types*, 2016, `http://depend.cmpe.boun.edu.tr/scp/MathematicalOperationsSupportedbyScaleTypes.pdf`, accessed at August 2016.

84. Pirklbauer, G., C. Fasching and W. Kurschl, "Improving Change Impact Analysis with a Tight Integrated Process and Tool", *Seventh International Conference on Information Technology: New Generations (ITNG), 2010*, pp. 956–961, April 2010.

85. Kagdi, H. and J. I. Maletic, "Combining Single-Version and Evolutionary Dependencies for Software-Change Prediction", *Proceedings of the Fourth International Workshop on Mining Software Repositories*, MSR '07, pp. 17–, 2007.

86. Parnas, D. L., "On the criteria to be used in decomposing systems into modules", *Communications of the ACM*, Vol. 15, No. 12, pp. 1053–1058, 1972.

87. Yourdon, E., *Techniques of program structure and design*, Prentice-Hall, 1975.

88. McCabe, T. J., "A Complexity Measure", *IEEE Transactions on Software Engineering*, Vol. 2, No. 4, pp. 308–320, 1976.

89. Halstead, M. H., *Elements of software science*, Vol. 7, Elsevier New York, 1977.

90. Gilb, T., *Software metrics*, Winthrop Publishers, 1977.

91. Boehm, B. W., *Characteristics of software quality*, Vol. 1, North-Holland, 1978.

92. Albrecht, A. J., "Measuring application development productivity", *Proceedings of the joint SHARE/GUIDE/IBM application development symposium*, Vol. 10, pp.

83–92, 1979.

93. Briand, L. C., J. W. Daly and J. Wüst, "A unified framework for cohesion measurement in object-oriented systems", *Empirical Software Engineering*, Vol. 3, No. 1, pp. 65–117, 1998.

94. Kitchenham, B., S. L. Pfleeger and N. Fenton, "Towards a framework for software measurement validation", *IEEE Transactions on Software Engineering*, Vol. 21, No. 12, pp. 929–944, 1995.