Towards a Metric Suite Proposal to Quantify Confirmation Biases of Developers

Conference or Workshop Item

2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement

Towards a Metric Suite Proposal to Quantify Confirmation Biases of Developers

Gul Calikli and Ayse Bener
Data Science Laboratory
Mechanical and Industrial Engineering
Ryerson University
Toronto, ON, Canada
{gcalikli, ayse.bener}@ryerson.ca

Turgay Aytac
Prescience Inc.
West New York, NJ, USA
taytac@scidesktop.org

Ovunc Bozcan
Turkcell
Istanbul, Turkey
ovunc.bozcan@turkcellteknoloji.com.tr

***Abstract*—The goal of software metrics is the identification and measurement of the essential parameters that affect software development. Metrics can be used to improve software quality and productivity. Existing metrics in the literature are mostly product or process related. However, thought processes of people have a significant impact on software quality as software is designed, implemented and tested by people. Therefore, in defining new metrics, we need to take into account human cognitive aspects. Our research aims to address this need through the proposal of a new metric scheme to quantify a specific human cognitive aspect, namely "confirmation bias". In our previous research, in order to quantify confirmation bias, we defined a methodology to measure confirmation biases of people. In this research, we propose a metric suite that would be used by practitioners during daily decision making. Our proposed metric set consists of six metrics with a theoretical basis in cognitive psychology and measurement theory. Empirical sample of these metrics are collected from two software companies that are specialized in two different domains in order to demonstrate their feasibility. We suggest ways in which practitioners may use these metrics to improve software development process.**

I. Introduction

Software metrics provide a quantitative basis for the development and validation of software development process. More accurate schedule and cost estimates, better quality software products, and higher productivity require effective software management, which can be facilitated by the improved use of software metrics.

Product metrics may measure the complexity of the software design, the size of the final program (either source or object code), or the number of pages of documentation produced. [1]. Among most widely studied product metrics are LOC (Lines of Code), Halstead's metrics [2], McCabe's cyclomatic complexity [3] and function points [4]. These metrics are critized due to the lack of a universal agreement about what these metrics measure [1], [5]. Another set of widely used product metrics consist of theoretically grounded metrics of the Object-Oriented (OO) design by Kemerer and Chidamber [6].

Process Metrics are measures of the software development process such as overall development time, type of methodology used or development history (i.e., churn) metrics.

Although the three pillars of software development are Product, Process and People (3Ps), people-related metrics are rarely addressed. In the literature, various people-related metrics have been used to build learning-based models to identify defect-prone parts of software. among such metrics are organizational metrics [8], number of developers [9], [10], developer experience [11], [12] and social interaction between developers [13], [14]. However, these metrics are not directly related with the thought processes of people or other cognitive aspects. On the other hand, thought processes and cognitive aspects of people are one of the fundamental concerns of software development [21], [22].

In our previous research, we wanted to understand the impact of the thought processes of people in determining software defect proneness [15], [16], [17]. We focussed on a specific cognitive aspect of people, namely "confirmation bias", which is defined as the tendency of people to seek evidence to verify hypotheses rather than seeking evidence to refute them. In order to measure confirmation bias, we defined a methodology that is based on Wason's grounded work in cognitive psychology literature, namely "Wason's Selection Task" and "Wason's Rule Discovery Task". Having discovered a correlation between confirmation bias levels of developers and software defect density, we later used confirmation bias values of developers to learn defect prediction models [18]. Our empirical results demonstrated that the thought processes of people have a significant impact on the defect proneness of software.

Due to confirmation bias, developers perform tests to make their program work rather than to break their code. There are similarities between Wason's rule discovery task and functional (black-box) testing performed by developers. Moreover, logical reasoning skills, which Wason's selection task aims to assess are essential during unit testing activities, such as white-box testing.

During our previous research, we conducted field studies in large and medium-scale software companies both in Canada and Turkey. As a result, we have become aware of the problems encountered by practitioners during software development process. During daily software engineering activities, practitioners need metrics, which are simple and comprehensible enough to guide them while making decisions under uncertainty. Therefore, in this study, we propose a confirmation bias metric set to facilitate practitioners' decision making process about software development activities. We can summarize the contributions of

978-0-7695-5056-5/13 $26.00 © 2013 IEEE
DOI 10.1109/ESEM.2013.47

this work as follows:

- A set of metrics that are constructed with a firm basis in theoretical concepts in cognitive psychology. and measurement theory.
- Presentation of empirical data from commercial projects to illustrate the characteristics of these metrics on real applications.

The rest of the paper is organized as follows: In Section II, we present theoretical background about confirmation bias and measurement theory. Details of our empirical study such as data collection and the employed methodology are explained in Section III. In Section IV we present the empirical study results that aim to illustrate the characteristics of data from software companies. We explain the practical implications of the proposed metrics suite in Section V. Threats to validity are addressed in Section VI. Finally, we mention future work and conclude in Section VII.

II. Theoretical Background

A. Confirmation Bias

In cognitive psychology, confirmation bias is defined as the tendency of people to seek for evidence that could verify their hypotheses rather than seeking for evidence that could falsify them. The term confirmation bias was first used by Peter Wason in his rule discovery experiment [19] and later in his selection task experiment [20].

1) Wason's Rule Discovery Task: The experimental procedure of Wason's Rule Discovery Task is as follows: Initially, the subject is given a record sheet on which the triple of numbers "2 4 6 is written and (s)he is told that 2 4 6 conforms to a rule, which (s)he is supposed to discover. In order to discover the rule, the subject is asked to write down triples together with the reasons of his/her choice on the record sheet. After each instance, the examiner tells the subject whether the instance conforms to the rule or not. The subject can announce the rule only when (s)he is highly confident. If the subject fails to discover the rule at the first attempt, (s)he can continue giving instances together with reasons for his/her choice. This procedure continues iteratively until either the subject discovers the rule or (s)he wishes to give up. However, if the subject cannot discover the rule in 45 min, the experimenter aborts the procedure.

Wason designed this experiment in such a way that once the subject sees the triple "2 4 6", a set of hypotheses is formed in his/her mind. Most popular ones of such hypotheses are as follows: "even numbers in ascending order", "numbers ascending with increments of two" , "numbers ascending with constant increments". [19], [15], [16], [17], [18]. Unless the subject tries to refute such hypotheses after having gained some confidence about the rule, it is unlikely to discover the correct rule, which is "ascending numbers".

2) Wason's Rule Discovery Task in Relation to Developer Performance: There are similarities between Wason's rule discovery task and functional (black-box) testing that are performed by software developers to test the functional units of their codes during unit testing [21]. According to the findings of Wason's rule discovery task, the subjects have a tendency to select many triples (i.e., test cases) that are consistent with their hypotheses and few tests that are inconsistent with them. For instance, choosing triples such as "5 10 15", "1 3 5" and "3 6 9" would not be an effective way of testing the validity of the hypothesis "numbers ascending with constant increments". One can test the validity of such an hypothesis only by giving examples that have the potential to refute that hypothesis (i.e., "3 7 19", "4 2 16", "1 100 102", etc.). Similarly, program testers may select many test cases consistent with the program specifications (positive tests) and a few that are inconsistent with them (negative tests). Moreover, the number of possible test cases is either infinite or too large to be tested within a limited amount of time. Consequently, a strategic approach must be followed that covers both positive and negative test cases while trying to make the code fail during testing in order to find as many defects as possible.

3) Wason's Selection Task: In the original task, the subject is given four cards, where each card has a letter on one side and a number on the other side. These four cards are placed on a table showing D, K, 3, 7, respectively. Given the rule Every card that has a D on one side has a 3 on the other side, the subject is asked which card(s) must be turned over to find out whether the rule is true or false [20].

4) Wason's Selection Task in Relation to Developer Performance: Testing the correctness of conditional statements in the source code during white box testing also requires logical reasoning skills. In order to explain the analogy between Wason's selection task and white box testing, we extend a simple example given by Stacy an MacMillian [22] as follows: Suppose a developer wants to make sure that his/her program avoids de-referencing a null pointer by always checking before de-referencing. During unit testing, the developer would perform a test that could be thought of as checking the validity of the following hypothesis: "If a pointer is de-referenced, then it is checked for nullity. " (If p, then q). Logical reasoning allows us to categorize parts of the code that may need to be tested as follows:

- Category #1: Parts of the code where a pointer is checked for nullity. The pointer may or may not be de-referenced in these code parts.
- Category #2: Parts of the code where a pointer is not checked for nullity. The pointer may or may not be de-referenced in those parts.
- Category #3: Parts of the code where a pointer is de-referenced. The pointer may or may not have been checked for nullity.
- Category #4: Parts of the code where a pointer is not de-referenced. The pointer may or may not have been checked for nullity.

It is important to determine whether every pointer is checked for nullity in parts of the code, which belong to category $\#3$, since we know every pointer is dereferenced in those parts of the code. It is also important to know whether every pointer is dereferenced in parts of the code that belong to category $\#2$, since we know none of the pointers are checked for nullity in those parts. On the other hand, we don't need to check whether every pointer is dereferenced in parts of the code that belong to category $\#1$, since we know every pointer

is checked for nullity, and we don't need to check whether every pointer is checked for nullity in parts of the code that belong to category $\#4$, since we know none of the pointers are dereferenced in there.

We can arrive the conclusion mentioned above by thinking according to modus ponens: Given that p is true, "if p then q is true only if q is true. The logical statements "if p then q" and "if not-q then not-p" are equivalent. Therefore, given that not-q is true, "if not-q, then not-p" is true only if not-p is true. Based on these logical facts, one can conclude that parts of the code, which belong to category $\#2$ and category $\#3$ need to be checked in order to find out whether the program avoids de-referencing a null pointer.

B. Measurement Theory

Metrics developed in empirical software engineering literature have been criticised due to the lack of a theoretical base [1]. As a solution to this issue, Weyuker proposed a formal set of criteria to evaluate software metrics [5]. Kemerer and Chidamber later used Weyuker's proposed set of measurement principles in order to analytically evaluate their Object-Oriented (OO) design metrics [6]. Weyuker's formal list of desiderata for software metrics is based on the foundations of measurement theory, which is a branch of applied mathematics that is useful in measurement and data analysis. Measurement theory attempts to describe, categorize, and evaluate quality of the measurements, improve the usefulness, accuracy and meaningfulness of measurements and propose methods for developing new and better measurement instruments [23]. Weyuker's principles and hence the theoretical base of Kemerer and Chidamber's OO design metrics are based on *representational* measurement theory. In *representational* measurement theory, numbers are assigned to objects only if objects are "quantifiable" and in order to be able to measure an object, there must be a transformation from the *empirical relational system* to the *formal relational system*. For instance, if the objects are rigid rods, then lengths of rigid rods in the empirical relation system can be transformed to real numbers in the formal relational system. Hence, transformation of the empirical relation "Rod A is longer than rod B" to the formal relation becomes "8 inch rod $1 >$ 12 inch rod".

Representational measurement theory is suitable for Kemerer and Chidamber's OO design metrics. Since classes in an object oriented code can be associated with numbers (e.g. numbers of children of a class, depth of inheritance tree, etc.). Representational theory requires the object to be "quantifiable". However, psychological variables, such as attitude, ability, intelligence cannot be measured directly. For such variables, *classical* measurement theory can be employed. The *classical* measurement theory sustained the development of most quantitative theories in psychology [24], [25]. *Classical* measurement theory merely requires the attribute of the object to be quantifiable, rather than the object itself. Confirmation bias is also a psychological variable and hence it can be inferred from other variables that are directly measured through a mathematical model. Such variables are called "latent" variables. An example of a widely invoked latent variable is the concept of "intelligence". We cannot measure intelligence directly, however, we can imagine a quality that we call "general intelligence" to make sense of patterns in scores on tests that are believed to measure specific mental abilities [26]. Similarly, we measure confirmation bias through 6 metrics indirectly from the scores and outcomes of our confirmation bias test. Details of the confirmation test are given in Section III-B.

TABLE I. PROPERTIES OF DATASETS

Dataset (Project)	Number of Active Files	Defect Rate	Number of Developers
Telecom1	826	0.11	10
Telecom2	1828	0.03	14
ERP	3199	0.07	6

III. EMPIRICAL STUDY

A. Data Collection

In this study, we used datasets from three different projects. In Table I, the total number of maintained/developed files and defect rates are listed for each dataset as well as the total number of developers in each project group. The defect rate is estimated as the ratio of the number of defective files to the number of total active files. All three datasets contain JAVA and JSP files, while dataset Telecom 2 also contains PL/SQL files besides JAVA and JSP files.

Datasets Telecom1 and Telecom2 belong to the largest wireless telecom operator (GSM) company in Turkey (and the third largest GSM company in Europe). Project group Telecom1 consists of 10 developers who are responsible for the development of a software product that is used to launch new campaigns. On average, 545 java files exist in a single version, and developers make modifications to 206 files per version (also on average). The total number of active files and defect rates , which are listed in Table I for dataset Telecom1 belong to the four versions that were released between the dates 07/02/2009 and 08/13/2009. Telecom1 project group releases a new version of the software product every 10 days on average.

Dataset Telecom2 comes from the billing and charging software package, which has been developed and maintained since the inception of the GSM company in 1994. Telecom2 project group consists of 14 developers. In Table I, information about defect rates and total number of active files for project Telecom2 correspond to four versions of the billing and charging software that were released between the dates 03/06/2011 and 05/08/2011. Software release period of project Telecom2 is every month on average.

Dataset ERP belongs to a project group that consists of 6 developers who are employees of the largest ISV (independent software vendor) in Turkey. The software developed by this project group is an enterprise resource planning (ERP) software. The snapshot of the ERP software, which dates from March 2011, was retrieved from the version management system in order to estimate the corresponding defect rate and total number of active files as shown in Table I.

B. Methodology

1) Preparation of Confirmation Bias Tests: Confirmation bias test consists of written questions and an interactive question. Interactive question is Wason's Rule Discovery Task itself [19], while written test is based on Wason's Selection

Task [20]. Written test consists of two parts 7 abstract and 7 thematic questions. Abstract questions require logical reasoning skills to be answered correctly, while real life experience and/or memory cueing [27] can help to answer thematic questions correctly.

2) Administration of Confirmation Bias Tests: Initially, we administered confirmation test to a pilot group consisting of 28 Computer Engineering PhD candidates. Half of the participants in the pilot group had at least two years of experience in commercial software product development. The pilot study was followed by the administration of the test to software engineers in various large scale companies and Small Medium Enterprises (SMEs). So far, we have administered the confirmation bias test to 199 software engineers (129 developers, 26 testers, 32 analysts and 12 project managers) fom 4 large scale companies and 3 SMEs.

In order to collect confirmation bias metrics in a controlled manner, we administered the confirmation bias test, which consists of the interactive question and the written question set, under a predefined standard procedure. The environment where the confirmation bias test was administered was isolated from noise and had adequate lighting.

In Wason's studies related to his selection task, real packs of cards were used. Most recent studies employ different procedures such as describing the cards and the pictorial representations of their visible sides with pencil and paper or on a computer screen. Employing such procedures have made insignificant differences in the results of the experiments [27]. Moreover, it is possible to administer this part of the confirmation bias test to a group of participants (instead of individually). Therefore, we preferred to use the pencil-and-paper approach rather than the traditional approach to administer the part of the confirmation bias test, which is based on Wason's Selection Task.

During the interactive part of the confirmation bias test, each participant answered the question in a separate room, and there was one examiner to guide and give feedback to each participant. Before the whole procedure started, the participants were asked to consent to have their responses recorded during the session. The goal of the voice recording was to catch every detail about the way a participant thought to discover the correct rule. Before starting the test, detailed information was given about the procedure to discover the correct rule.

3) Defining the metric set: We defined the metric set as a natural extension of our previous study [18]. As a result, we obtained a simple and comprehensible metric set that can guide software practitioners while making decisions during the software development process. In Table II, definition of the confirmation bias metrics are given. Ideal and worst-case values for each metric are given in Table III. Ideal value of a metric is among the indications of low confirmation bias and hence it is desirable that a confirmation bias metric is as close to the ideal value as possible. On the other hand, worst-case value of a metric is among the indications of high confirmation bias.

4) Calculation of Group based Metrics Values: In this study, the empirical results are group-based (i.e., the results are based on confirmation bias metrics values of developer groups). We define "developer group" as one or more developer(s) who create/update the same set of source code files. We used "average" operator to calculate group based confirmation bias metrics values. Assuming that A_{di} represents the i^{th} confirmation bias metric value of d^{th} developer, $d \in G_j$ means that d^{th} developer is among the group of developers who created and/or modified j^{th} source file, and finally, S^avg_{ji} represents the resulting i^{th} confirmation bias metric value of j^{th} source file when operator avg is applied. We can formulize the definition for the "avg" operator as follows:

$$S^{avg}_{ji} = \Sigma_d(A_{di}|\forall d \in G_j)/\Sigma_d(1|\forall d \in G_j) \quad (1)$$

TABLE II. CONFIRMATION BIAS METRIC SET

Metric	Explanation	Test Type
	Eliminative/enumerative index by Wason [19]	Interactive
$Ind_{elim/enum}$ $Inst_{Compatible}$	Total number of positive and compatible instances	Interactive
$Rules/Time$	Total number rules announced per unit time	Interactive
$Abs_{PartialInsight}$	Partial Insight in abstract questions	Written
S_{Th}	Score in thematic questions	Written
$avgL_{IR}$	Average length of immediate rule announcements	Interactive

TABLE III. IDEAL AND WORST-CASE VALUES FOR THE CONFIRMATION BIAS METRIC SET

Metric	Ideal Value	Worst-Case Value
$Ind_{elim/enum}$	$Ind^{max}_{elim/enum}$	0
$Inst_{Compatible}$	0	$Inst^{max}_{Compatible}$
$Rules/Time$	0.1	$(Rules/Time)^{max}$
$Abs_{PartialInsight}$	0	1
S_{Th}	1	0
$avgL_{IR}$	0	$avgLIR^{max}$

TABLE IV. DISTRIBUTION OF DEVELOPER GROUPS WITH RESPECT TO GROUP SIZE FOR EACH DATASET

	# of Developers in Developer Groups						
Dataset	1	2	3	4	5	6	**Total**
Telecom1	7	13	11	7	4	1	43
Telecom2	13	8	0	0	0	0	21
ERP	6	4	2	1	0	0	13

IV. EMPIRICAL STUDY RESULTS

In this section, we present our proposed metric suite by giving the definition, theoretical basis and empirical results of each metric. As mentioned previously, the empirical results are group-based (i.e., the results are based on confirmation bias metrics values of developer groups). We identified developer groups for each project by mining log files that are obtained from version management systems. Total number of developer groups for each project are given in Table IV. Distribution of developer groups with respect to group size (i.e., total number of developers in a group) are also given in Table IV.

In this section, we compare confirmation bias levels of developer groups of Telecom1, Telecom2 and ERP projects by using our proposed metric suite. Our empirical results show that confirmation bias metrics values of developer groups, which belong to projects Telecom2 and ERP are much closer to their corresponding ideal values compared to metrics values of Telecom1 developer groups. As it can be deduced from Table III, metrics $Abs_{PartialInsight}$, S_{Th} are in the range $0,1$ (i.e., $0 \leq Abs_{PartialInsight} \leq 1$ and $0 \leq S_{Th} \leq 1$). We also mapped the values of the rest of the metrics to the range $[0,1]$ in Figures, where we present

percentage distribution of developer groups with respect to each metrics values (i.e. Figures 1, 2, 3, 4, 5 and 6). For this purpose, we divided value of each metric (i.e., $Ind_{elim/enum}$, $Inst_{Compatible}$, $Rules/Times$, $avgLIR$) by its maximum value (i.e. $Ind^{max}_{elim/enum}$, $Inst^{max}_{Compatible}$, $Rules/Times^{max}$, $avgLIR^{max}$).

In Section II-A, we explained the hypothetical relationship between developers' testing behavior, and Wason's Rule Discovery and Selection Tasks. In this section, we also explain what type of information each metric may provide us about developers' unit testing activities, which may lead to an increase in defect rates. In order to investigate the link between confirmation bias and defect rates through testing, we also interviewed with the project managers and developers of Telecom1, Telecom2 and ERP projects.

Telecom1 is currently having serious problems with their customers due to high amount of post-release defects. According to the interviews we conducted with developers and project manager of this project group, developers' testing activities mainly consist of executing the test cases, which were previously prepared as part of the requirements analysis document. Such type of tests can be classified as "positive tests", since during these tests developers try to validate whether specific parts of the software conform to the corresponding use-cases in the requirements document. After the unit tests, developers perform smoke tests (i.e., simple integration tests where developers just check whether the system under test returns normally and does not blow up when it is invoked). During the interviews, the developers indicated that they employed negative tests only when a new functionality is introduced to the software. Since Telecom1 project group launches a new release every 10 days, new functionalities are rarely introduced to the software. Therefore, tests which focus on "What might go wrong?" (i.e., negative tests) are rarely conducted.

Telecom2 and ERP projects have not experienced a software quality-related crisis, that is as serious as the crisis that is currently encountered by Telecom1 project. As it can be seen from the snapshots in Table I, pre-release defect rate of Telecom2 and ERP projects are lower than defect rate of Telecom1 project. Telecom2 and ERP projects are mission critical, since their software products contain modules such as billing, charging and revenue collection. Therefore, it is crucial for developers and project managers of Telecom2 and ERP that the testing strategies they employ are efficient and effective. During the interviews, Telecom2 and ERP project developers indicated that negative tests are important to explore "What might go wrong?". Negative tests are part of developers' testing routine in addition to the tests to validate use-cases in the requirements analysis documents. In some cases some negative test scenarios might be overlooked, since such test cases may seem too extreme to be encountered while the software is in use in real life settings. For instance, ERP developers prepared negative test cases where 100-line long bills were generated by the software module. However, problems were later encountered in the field (i.e., in the form of post-release defects), when a client of the ERP company attempted to generate 10000-line long bills. ERP project manager indicated that in order to minimize such incidents, developers take into account such experiences in the past while generating negative test scenarios for the next release of the software. Based on the outcomes of positive and negative tests, addition of new features or refactoring might be required. However, such modifications might not be feasible at the time, since modifications may lead to failures in other parts of the software.

TABLE V. SUMMARY STATISTICS FOR THE $Ind_{elim/enum}$ METRIC

Project (Dataset)	Metric	Mean$_{[0,1]}$	Mean	Median	Max
Telecom1	$Ind_{elim/enum}$	0.23	0.39	0.25	1.68
Telecom2	$Ind_{elim/enum}$	0.55	1.71	1.72	3.09
ERP	$Ind_{elim/enum}$	0.47	0.93	0.94	2.00

A. Metric 1: Eliminative/Enumerative Index ($Ind_{elim/enum}$)

1) Definition: $Ind_{elim/enum}$ metric is extracted from the interactive part of the confirmation bias test. This metric is used to estimate the proportion of the total number of instances that are incompatible with reasons the participant writes down during the interactive test to those that are compatible.

2) Theoretical Basis: The eliminative/enumerative index ($Ind_{elim/enum}$) was introduced by Wason to evaluate the results of his rule discovery task [19], which forms the basis of the interactive part of confirmation bias test. In his rule discovery task, Wason concluded that participants who announced the correct rule on the first try had higher $Ind_{elim/enum}$ values compared to the rest of the participants.

If the value of $Ind_{elim/enum}$ is lower than 1, this implies that participants are more inclined to use triples of numbers (i.e., test cases) that are compatible with their hypotheses. As it was previously indicated by Teasley et al., there is a similarity between software testing and Wason's rule discovery task [21]. Developers with $Ind_{elim/enum}$ values lower than 1 are more likely to be inclined to select positive test cases to verify their code. This, in turn, leads to an increase in software defect density. On the other hand, effective unit testing can be achieved only by employing negative test scenarios in addition to positive tests.

3) Empirical Data: Percentage distribution of developer groups with respect to $Ind_{ElimEnum}$ metric values and summary statistics are shown in Figure 1 and Table V for all three projects Telecom1, Telecom2 and ERP.

4) Interpretation of Data: As shown in Figure 1, 41.86% of the developer groups in project Telecom1 have $Ind_{elim/enum}$ value that is in the range [0, 0.1] (i.e., $Ind_{Elim/Enum} \geq 0$ and $Ind_{Elim/Enum} \leq 0.1$). Moreover, the values of the eliminative/enumerative index is less than 0.5 for 79.07% of the Telecom1 developer groups. Distributions of the $Ind_{elim/enum}$ values among developer groups for the projects Telecom2 and ERP exhibit a tendency towards a more eliminative behavior. The difference among the distributions for all three projects is statistically significant ($\chi^2(2, N = 77) = 17.4, p = 1E-6$). The value of the $Ind_{elim/enum}$ metric for 71.43% and 61.54% of the developer groups belonging to projects Telecom2 and ERP, respectively is in the range [0.5, 1] (i.e., $Ind_{Elim/Enum} > 0.5$ and $Ind_{Elim/Enum} \leq 1$).

B. Metric 2: Total Number of Positive and Compatible Instances ($Inst_{Compatible}$)

1) Definition: $Inst_{Compatible}$ metric value is also extracted from the interactive part of the confirmation bias test. This

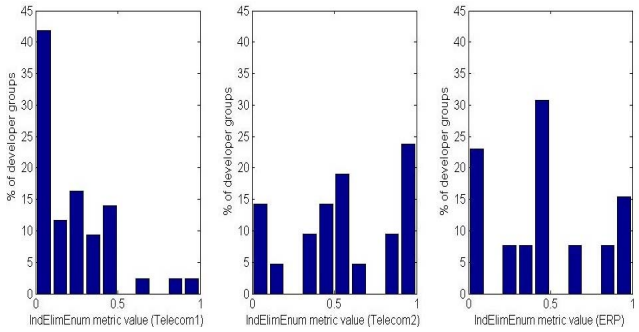

Fig. 1. Percentage distribution of developer groups for the $Ind_{ElimEnum}$ metric

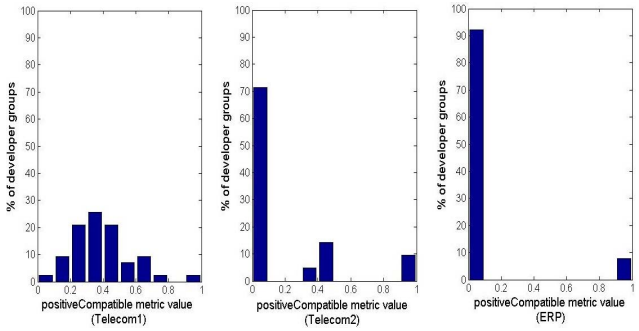

Fig. 2. Percentage distribution of developer groups for the $Inst_{Compatible}$ metric

metric gives the total number of compatible instances given by a participant after (s)he makes an incorrect rule announcement.

2) Theoretical Basis: Giving rules that conform to one or more of the previously announced incorrect rules is an indication of the participant's lack of tendency to refute the hypotheses in his/her mind even though (s)he has been informed about the incorrectness of these hypotheses (i.e., previously announced rules). In his rule discovery task experiment, Wason also observed that subjects, who failed to find the correct rule on the first announcement, gave examples of triples that are compatible with the rules, which they previously announced [19].

3) Empirical Data: Percentage distribution of developer groups with respect to $Inst_{Compatible}$ metric values and summary statistics are shown in Figure 2 and Table VI for all three projects Telecom1, Telecom2 and ERP.

4) Interpretation of Data: As it can be seen from the percentage distribution of developer groups for the $Inst_{Compatible}$ metric in Figure 2, compatible instances are rarely announced by developers that belong to the projects Telecom2 and ERP. There is a statistically significant difference among he three distributions ($\chi^2(2, N = 77) = 18.1, p = 1E-6$). The $Inst_{Compatible}$ metric values for the 71.43% and 92.31% of developer groups belonging to the projects Telecom2 and ERP are in the range $[0, 0.1]$. On the other hand, only 2.33% of the developer groups that belong to the project Telecom1 have $Inst_{Compatible}$ metric values that are in the range $[0, 0.1]$.

TABLE VI. SUMMARY STATISTICS FOR THE $Inst_{Compatible}$ METRIC

Project (Dataset)	Metric	Mean$_{[0,1]}$	Mean	Median	Max
Telecom1	$Inst_{Compatible}$	0.40	1.60	1.50	4.00
Telecom2	$Inst_{Compatible}$	0.18	0.18	0.00	1.00
ERP	$Inst_{Compatible}$	0.08	0.08	0.00	1.00

TABLE VII. SUMMARY STATISTICS FOR THE $Rules/Time$ METRIC

Project (Dataset)	Metric	Mean$_{[0,1]}$	Mean	Median	Max
Telecom1	$Rules/Time$	0.78	0.20	0.20	0.25
Telecom2	$Rules/Time$	0.30	0.20	0.13	0.67
ERP	$Rules/Time$	0.60	0.30	0.28	0.50

C. Metric 3: Total Number of Rules Announced per Unit Time ($Rules/Time$)

1) Definition: The metric $Rules/Time$ measures the Total number of rules announced per unit time during the interactive part of the confirmation bias test. $Rules/Time$ is estimated by dividing the total number of rules announced by the participant (i.e., N_a) to the total time (in minutes) it takes to complete/terminate the interactive part of the confirmation bias test.

2) Theoretical Basis: In his rule discovery task, Wason used a metric N_a measure the total number of rules announced by a participant throughout the whole experiment. As one of the outcomes of his rule discovery task, Wason presented a frequency distribution of participants with respect to the total number of rule announcements made [19]. In our research, we introduce the metric $Rules/Time$ by also taking "time" into consideration. A developer having a high $Rules/Time$ value (i.e., a developer who announces high number of rules in a short period of time) as the outcome of the interactive question has the tendency to deliver his/her code to the testing phase without making adequate unit testing. For such a developer, the compilation of his/her code is sufficient. In other words, high $Rules/Time$ is an indication of the developers rush to solve the interactive question correctly, mostly without checking the possibility of the alternative hypotheses in his/her mind by giving instances.

3) Empirical Data: Percentage distribution of developer groups with respect to $Rules/Time$ metric values and summary statistics are shown in Figure 3 and Table VII for all three projects Telecom1, Telecom2 and ERP.

4) Interpretation of Data: There is a statistically significant difference among the distributions of developer groups with respect to the $Rules/Time$ metric values for the three projects Telecom1, Telecom2 and ERP ($\chi^2(2, N = 77) = 26.7, p = 1E-6$). The $Rules/Time$ metric values for the 86.05% of the developer groups belonging to the project Telecom1 are in the range $[0.5, 1]$. On the other hand, the metric values for the 76.19% and 53.85% of the developer groups, which belong to the project groups Telecom2 and ERP, respectively are less than 0.5.

D. Metric 4: Partial Insight in Abstract Questions ($Abs_{PartialInsight}$)

1) Definition: The metric $Abs_{PartialInsight}$ measures the extent of having both verifying and falsifying tendencies during the written part of the confirmation bias test.

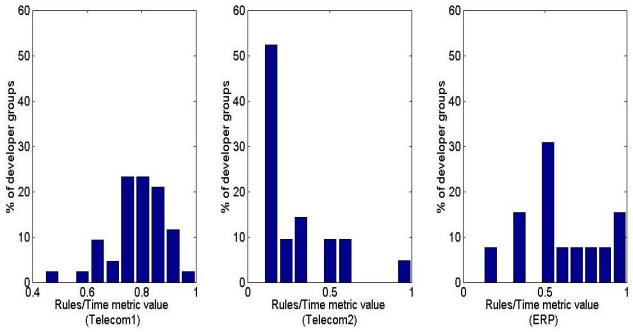

Fig. 3. Percentage distribution of developer groups for the $Rules/Time$ metric

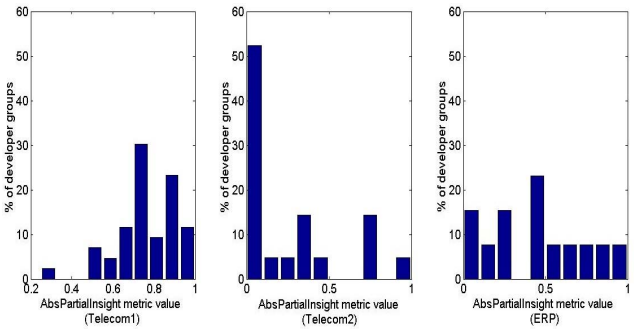

Fig. 4. Percentage distribution of developer groups for the $Abs_{PartialInsight}$ metric

TABLE VIII. SUMMARY STATISTICS FOR THE $Abs_{PartialInsight}$ METRIC

Project (Dataset)	Metric	$\text{Mean}_{[0,1]}$	Mean	Median	Max
Telecom1	$Abs_{PartialInsight}$	0.77	0.77	0.75	1.00
Telecom2	$Abs_{PartialInsight}$	0.25	0.25	0.00	1.00
ERP	$Abs_{PartialInsight}$	0.45	0.45	0.50	1.00

2) Theoretical Basis: In their information processing model, Johnson-Laird and Wason [28] classified the participants performance on Wason's selection task as no insight, partial insight and complete insight based on the kinds of systematic errors made by the participants. In their studies, Mataraso-Roth [29] and Evans and Lynch [30] discovered that participants performing at the level of no insight focus on cards mentioned in the rule whose validity is tested. The selection of cards by a participant with no insight might be due to the participants tendency to verify the rule, or (s)he might just match the symbols or words on the cards with those mentioned in the rule. On the other hand, participants performing at the level of partial insight or complete insight consider what symbols or words occur on the back of each card, such participants perform a systematic combinatorial analysis of the cards. The difference between these two performance levels is that the participants having partial insight select all cards that could either verify or falsify the rule, whereas the participants with complete insight select only the cards that have the potential to falsify the rule. Depending on whether the selection task in the written question set is abstract, or thematic, performance of a participant may vary [27]. Participants usually perform poorly on abstract questions [27], [28], since logical reasoning skills are required to solve abstract questions correctly, while in thematic questions memory cueing and real life experience can be helpful while answering those type of questions [27]. As a result of our analyses, partial insight regarding abstract questions turned out to be the most discriminative measure. Therefore, our metric set only included the metric $Abs_{PartialInsight}$.

3) Empirical Data: Percentage distribution of developer groups with respect to $Abs_{PartialInsight}$ metric values and summary statistics are shown in Figure 4 and Table VIII for all three projects Telecom1, Telecom2 and ERP.

4) Interpretation of Data: As it is shown in Figure 4, the difference in the percentage distribution of the developer groups with respect to the $Abs_{PartialInsight}$ metric values for all three projects (i.e.,Telecom1, Telecom2 and ERP) is statistically significant ($\chi^2(2, N = 77) = 29.3, p = 1E - 6$). The results obtained are inline with the results that we obtained in our previous study. Developer groups belonging to the projects Telecom2 and ERP performed better than the developer groups belonging to the project Telecom1 in terms of the metric $Abs_{PartialInsight}$. The values for the metric $Abs_{PartialInsight}$ are in the range $[0, 0.1]$ for the 52.38% of the developer groups belonging to the project Telecom2. Moreover, the values of this metric are in the range $[0, 0.5]$ for 80.94% and 61.55% of the developer groups, which belong to the projects Telecom2 and ERP, respectively. On the other hand, the metric values are in the range $[0.5, 1]$ for the developer groups, which belong to the project team Telecom1.

E. Metric 5: Score in Thematic Questions (S_{Th})

1) Definition: S_{Th} measures the portion of the correctly answered thematic questions in the written part of the confirmation bias test.

2) Theoretical Basis: A participant who has a high S_{Th} metric value makes use of the thematic facilitation effects, such as daily life experience or memory queuing in addition to his/her logical reasoning skills. During unit testing, it is likely that In addition to his/her logical reasoning skills, a developer's expertise about software development will facilitate his/her unit testing activities.

3) Empirical Data: Percentage distribution of developer groups with respect to S_{Th} metric values and summary statistics are shown in Figure 5 and Table IX for all three projects Telecom1, Telecom2 and ERP.

4) Interpretation of Data: As it is shown in Figure 5, developer groups that belong to projects Telecom2 and ERP performed better in the thematic part of the written test compared to Telecom1 developer groups. S_{Th} metric values for the 42.68% of the Telecom2 developer groups are in the range $[0.6, 0.7]$, whereas the metric values for the 69.23% of the developer groups of the project ERP are in the range $[0.9, 1.0]$. On the other hand, the peak values of the metric S_{Th} for the project Telecom1 are at the ranges $[0.4, 0.5]$ and $[0.6, 0.7]$, each corresponding to 23.26% of all developer groups belonging to that project. Moreover, the $[0.5, 1]$ value range of the S_{Th} metric corresponds to the 81.40%, 66.67% and 92.30% of the developer groups, which belong to the projects Telecom1, Telecom2 and ERP, respectively. However, the differences among the distributions for the developer

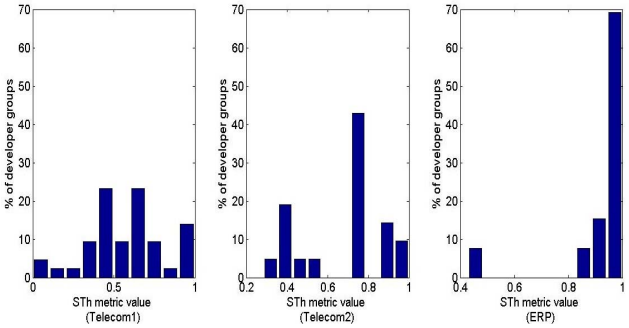

Fig. 5. Percentage distribution of developer groups for the S_{Th} metric

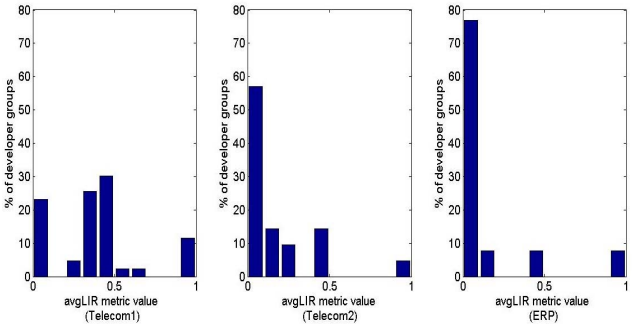

Fig. 6. Percentage distribution of developer groups for the $avgL_{IR}$ metric

TABLE IX. SUMMARY STATISTICS FOR THE S_{Th} METRIC

Project (Dataset)	Metric	Mean$_{[0,1]}$	Mean	Median	Max
Telecom1	S_{Th}	0.59	0.59	0.57	1.00
Telecom2	S_{Th}	0.67	0.67	0.71	1.00
ERP	S_{Th}	0.92	0.92	0.96	1.00

TABLE X. SUMMARY STATISTICS FOR THE $avgL_{IR}$ METRIC

Project (Dataset)	Metric	Mean$_{[0,1]}$	Mean	Median	Max
Telecom1	$avgL_{IR}$	0.40	0.40	0.40	1.00
Telecom2	$avgL_{IR}$	0.47	0.47	0.00	1.00
ERP	$avgL_{IR}$	0.76	0.76	0.25	1.00

groups of Telecom1, Telecom2 and ERP projects did not turn out to be statistically significant at the 0.05 significance level ($\chi^2(2, N = 77) = 5.21, p = 0.074$).

F. Metric 6: Length of Immediate Rule Announcement ($avgL_{IR}$)

1) Definition: $avgL_{IR}$ is the average length of the immediate rule announcements made by a participant during the interactive part of the confirmation bias test. Immediate rule announcements are made consecutively one after another without giving instance(s) in between any two consecutively announced rules. If a participant fails to discover the correct rule after announcing consecutive rules, according to the testing protocol (s)he may start over the whole procedure, which may also end up with a series of consecutively announced rules. This procedure may go on iteratively until the participant announces the correct rule or the interactive session is terminated. Therefore, administration of the interactive test may result in one or more series of consecutively announced rules. $avgL_{IR}$ estimates the average number of consecutively announced rules in all these series.

2) Theoretical Basis: Immediate rule announcements are an indication of a participants inadequate hypotheses testing strategies. As a result of such announcements, participants cannot come up with a single rule at the end by eliminating alternative hypotheses in their minds. A developer who makes immediate rule announcements during the interactive part of the test, is likely to exhibit poor unit testing performance. For instance, during functional unit testing equivalence partitioning technique may be referred by the developer. Equivalence partitioning is a non-exhaustive functional testing technique that is applied to each functional unit mostly together with boundary testing. In equivalence partitioning, a set of dimensions of input data are identified for each functional unit, and a set of equivalence classes are identified for each dimension. A developer who makes immediate rule announcements when solving the interactive question is very likely to fail to identify all dimensions of input data to be tested in the functional unit testing. Moreover, (s)he will probably fail to properly determine equivalence classes for each dimension.

3) Empirical Data: Percentage distribution of developer groups with respect to S_{Th} metric values and summary statistics are shown in Figure 6 and Table X for all three projects Telecom1, Telecom2 and ERP.

4) Interpretation of Data: Majority of the Telecom2 and ERP developer groups made seldom consecutive rule announcements during the interactive part of the test. The values of the metric $avgL_{IR}$ are in the range $[0, 0.1]$ for 58% and 85.71% of Telecom2 and ERP developer groups, respectively. On the other hand, $avgL_{IR}$ metric values are in the range $[0, 0.1]$ only for 23.35% of Telecom1 developer groups. There is a statistically significant difference among the percentage distribution of developer groups with respect to $avgL_{IR}$ metric values for Telecom1, Telecom2 and ERP developer groups ($\chi^2(2, N = 77) = 13.7, p = 0.001$).

G. Discussions

Values of $Ind_{Elim/Enum}$ and S_{Th} metrics for Telecom1 developer groups are in general lower than those of Telecom2 and ERP developer groups. As mentioned previously, low $Ind_{Elim/Enum}$ and S_{Th} metrics values are among the indications of high confirmation bias. Moreover, high values for the $Inst_{Compatible}$, $Rules/Time$, $Abs_{PartialInsight}$ and $avgL_{IR}$ metrics were observed for Telecom1 developer groups. High values of the metrics $Inst_{Compatible}$, $Rules/Time$, $Abs_{PartialInsight}$ and $avgL_{IR}$ are among the indications of high confirmation bias. When we shared our findings with developers and the manager of Telecom1 project, they stated that one rationale behind high amount of post-release defects in their software product is their confirmatory behaviour during testing activities.

V. PRACTICAL IMPLICATIONS

In our previous studies, we defined a methodology to quantify/measure confirmation bias. In this study, we formed a metric set that can be used by software practitioners to improve their software management process and hence quality of the software product. Based on our findings, we make the following recommendations to software practitioners:

A. Foreseeing a Major Project Crisis:

The metrics in the proposed metric suite can be used to foresee a crisis regarding quality of the software product so that required precautions can be taken on time. For instance, Telecom1 project group is currently having crisis with its customers due to high amount of post-release defects. During the interviews, Telecom1 project group developers and the project manager stated that the problems are due to inadequate testing strategies and the origins of the problem go back to 2-3 years. We administered confirmation bias tests to Telecom1 developers in January 2010. During the period 2010-2011, we administered confirmation bias tests to more project groups. During the analysis of the test outcomes, we observed that confirmation bias levels of Telecom1 developers were significantly, higher compared to developers of other project groups [15], [16], [17], [18]. We shared our analysis results with developers and the project manager. Developers and manager of project group Telecom 1 indicated that if necessary precautions had been taken earlier based on our findings, they would not be currently dealing with a software crisis.

B. Implementation and Rework

Project managers should take into account confirmation bias metrics values besides other factors in order to decide which developer(s) should fix bugs in the most critical parts of the software. Moreover, project managers can use confirmation bias metrics values to prevent contribution of developers having high confirmation biases (i.e., confirmation boas metrics values that are close to worst-case values that are given in Table III) to same set of files that are supposed to be created for the upcoming releases. Information about defect rates introduced by some developer groups during previous releases gives clues about the future performance of such developer groups. However, new members may have been included to the project team and/or some combinations of existing developers may have contributed to same set of files during previous releases (i.e., such developer groups may not have been encountered during previous releases). In such cases, confirmation bias metrics values can guide project managers to decide whether a group of developers should contribute to the same set of files or not.

C. Forming a Balanced Project Organization

It would not be wise to expect any organization to be created solely with members of low confirmation bias levels (i.e., members with confirmation bias metrics values that are close to the corresponding ideal metric values). It also applies to project teams as dynamic subgroups of the organization under consideration. The project teams (i.e., subgroups) should be balanced with respect to confirmation bias level. Using these metrics as one of the inputs for recruitment is also another option.

VI. Threats to Validity

In order to avoid mono-method bias that is one of the threats to construct validity, we used more than a single version of a confirmation bias measure. In this current study, we defined metric set to be used by practitioners. To form our confirmation bias metric set, we conducted an extensive survey of the cognitive psychology literature [19], [20], [27], [28], [29].

Another threat to construct validity is the interaction of different treatments. Before the administration of confirmation bias tests to participant groups, we ensured that none of the participants were simultaneously involved in several other experiments designed to have similar effects.

Evaluation apprehension is a social threat to construct validity. Many people are anxious about being evaluated. Participants may perform poorly as a result of feeling psychologically pressured. In order to avoid such problems, before the tests we informed the participants that the questions they are about to solve do not aim to measure IQ or any related capability. Participants were also told that the results would not be used in their performance evaluations and their identity would be kept anonymous. Moreover, participants were told that there was no time constraint for completing the questions.

Another social threat to construct validity is the expectancies of the researcher. Since, there are many ways a researcher may bias the results of a study. Hence, the outcomes of both written and interactive parts of the test were independently evaluated by two researchers, one of whom was not actively involved in the study. The said researcher was given a tutorial about how to evaluate the confirmation bias metrics from the outcomes of the written question set and the interactive question. However, in order not to induce a bias, she was not told about what the desired answers to the questions were. The inter-rater reliability was found to be high for the evaluation of each confirmation bias metric. The average value for Cohen's kappa was 0.89.

In order to avoid internal threats to validity, we set the test dates for all project groups for a time when the workload of the developers was not intense. No event took place in between the confirmation bias tests that could have influenced the performance of the subjects in any of the groups. For internal validity, we also interviewed with the developers and project managers in order to interpret our empirical results and used the information we obtained through interviews in order to cross-validate the obtained empirical results.

To avoid external threats to validity, we collected data from two different companies specialized in two different software development domains. We also selected two different projects within one of these companies.

For statistical validity, we used Chi Square test while interpreting the results of our empirical study.

VII. Conclusions and Future Work

In this study, we proposed a confirmation bias metric set to guide practitioners while making decisions about software development activities. Although, "confirmation bias" is only a single human aspect, the results we have obtained so far are quite promising [27]. Therefore, we strongly recommend that people's thought processes and other cognitive aspects should be further investigated within the context of software engineering. Existing software metrics deal with the measurement of the software *product* and *process* by which it is developed. We believe that *people* metrics that focus on human thought

processes, problem solving skills and other cognitive aspects need to be defined.

As future work, we intend to extend our dataset to include 200 more software engineers from SMEs and large scale companies in Canada and Turkey. We also would like to refine our tests to decrease the time and effort required to solve the tests so that they can also be used by practitioners besides serving research-related purposes.

In Section II-A, we mentioned the theoretical link between confirmation bias and defect rates through testing activities. As a result of our previous research, we found a correlation between developers' confirmation biases and defect rates. In this study, besides empirical analyses, we also conducted interviews with developers and project managers in order to find out about actual unit testing performed, the employed testing strategies and how much testing is done by developers. As future work, we aim to conduct observational studies and think aloud protocols in order to further investigate developers' testing behavior.

In the long run, our previous and current research may lead to the design of a training and monitoring program which aims to lower confirmation biases of developers. In cognitive psychology literature, there are de-biasing strategies that are based on recent results in meta-cognition research. Meta-cognitive skills can be taught, however they are not necessarily transferred from one context to another [31]. Therefore, cognitive psychologists and computer scientists must collaborate to design context-specific materials for the training of software professionals. Such a training program can be combined with project performance monitoring in order to be conducted to target groups. In addition to monitoring project performance, improvement in developers, confirmation bias levels can be monitored by using the metrics in our proposed metric scheme.

ACKNOWLEDGMENT

The authors would like to thank Turkcell A. S. and Ayhan Inal from Logo Business Solutions for their support in sharing data.

REFERENCES

[1] E. E. Mills, *Software metrics*, SEI curriculum Module SEI-CM-12-1.1 Carnegie Mellon University, Software Engineering Institute, 1988.

[2] M. Halstead, *Elements of software science*, New York: Elsevier, 1977

[3] T. McCabe, *A Complexity measure*, IEEE Transactions on Software Engineering, vol. 2, pp. 308-320, 1976.

[4] T. McCabe, *Measuring application development productivity*, Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium, October 1417, IBM Corporation, pp. 83-92, 1979.

[5] E. Weyuker, *Evaluating software momplexity measures*, IEEE Transactions on Software Engineering, vol. 14, pp. 1357-1365, 1988.

[6] S. R. Chidamber and C. F. Kemerer, *A metrics suite for object oriented design*, IEEE Transactions on Software Engineering, (20)6, pp. 476–493, 1994.

[7] N. Nagappan and T. Ball, *Using software dependencies and churn metrics to predict field failures: An empirical case study*, Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement, pp. 364–373, 2007.

[8] N. Nagappan, B. Murphy and V. R. Basili, *The influence of organizational structure on software quality: An empirical case study*, Proceedings of the 30th International Conference on Software Engineering, pp. 421–530, 2008.

[9] E. J. Weyuker, T. J. Ostrand and , R. M. Bell, *Do too many cooks spoil the broth? Using the number of developers to enhance defect prediction models*, Journal of Empirical Software Engineering, vol. 13, pp. 539-559, 2008.

[10] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, *Predicting fault incidence using software change history*, IEEE Transactions on Software Engineering, (26)7, pp, 653–661, 2000.

[11] A. Mockus and D. M. Weiss, *Predicting risk of software changes*, 5, Bell Labs Technical Journal, pp. 169–180, 2000.

[12] E. J. Weyuker, T. J. Ostrand and , R. M. Bell, *Using developer information as a factor for fault prediction*, Proceedings of the 1st International Workshop on Predictor Models in Software Engineering, vol. 13, pp. 539-559, 2008.

[13] A. Meneely, L. Williams, W. Snipes, and , J. Osborne, *Predicting failures with developer networks and social network analysis*, Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, pp. 13-23, 2008.

[14] N. Bettenburg, *Mining development depositories to study the impact of collaboration on software systems*, Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 376-379, 2011.

[15] G. Calikli, A. Bener and , B. Aslan, *An analysis of the effects of company culture, education and experience on confirmation bias levels of software developers and testers*, Proceedings of the 32nd International Conference on Software Engineering, 2010.

[16] G. Calikli, B. Aslan, and A. Bener, *Confirmation bias in software development and testing: An analysis of the effects of company size, experience and reasoning skills*, Proceedings of the 22nd Annual Psychology of Programming Interest Group Workshop, 2010.

[17] G. Calikli and A. Bener, *Empirical analyses factors affecting confirmation bias and the effects of confirmation bias on software developer/tester performance*, Proceedings of the 5th International Workshop on Predictor Models in Software Engineering, 2010.

[18] G. Calikli and A. Bener, *Influence of Confirmation Biases of Developers on Software Quality: An Empirical Study*, Software Quality Journal, (21)2, pp. 377–416, 2013.

[19] P. C. Wason, *On the failure to eliminate hypotheses in a conceptual task*, Quarterly Journal of Experimental Psychology, vol. 12, pp. 129–140, 1960.

[20] P. C. Wason, *Reasoning about a rule*, Quarterly Journal of Experimental Psychology, vol. 20, pp. 273–281, 1968.

[21] B. F. Teasley, L. . Leventhal, L. M. Mynatt, and D. S. Rohlman, *Why software testing is sometimes ineffective: Two applied studies of positive test strategy*, Journal of Applied Psychology, 79(1), pp. 142–155, 1994.

[22] W. Stacy and J. MacMillan, *Cognitive bias in software engineering*, Communications of the ACM, , 38(6), pp. 57–63, 1995.

[23] M. J. Allen and W. M. Yen, *Introduction to measurement theory*, Long Groove, IL, USA: Waveland Press Inc., 1979.

[24] C. L. Allen, *Principles of behavior*, New York, USA: Appleton-Century-Crofts , 1943.

[25] L. L. Thurstone, *Primary mental abilities*, Chicago, USA: University of Chicago Press , 1938.

[26] R. M. Warner, *Applied statistics: From bivariate through multivariate techniques*, 2nd Ed., USA: Sage Publications, 2013.

[27] J. St. B. T. Evans, S. E. Newsted and R. M. Byrne, *Human reasoning: the psychology of deduction*, East Sussex, UK: Lawrence Erlbaum Associates Ltd., 1993.

[28] P. N. Johnson and P. C. Wason, *A theoretical analysis of insight into a reasoning task*, Cognitive Psychology, vol. 1, pp. 134-148, 1970.

[29] E. Mataraso-Roth, *Facilitating insight in a reasoning task*, British Journal of Psychology, vol. 70, pp. 265-271, 1979.

[30] J. St. B. T. Evans and J. S. Lynch, *Matching bias in the selection task*, British Journal of Psychology, vol. 64, pp. 391–397, 1973.

[31] C. Mair and M. Shepperd, *Human judgment and software metrics: Vision for the future*, Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics, pp. 81–84, 2011.