MODELING HUMAN ASPECTS TO ENHANCE SOFTWARE QUALITY MANAGEMENT

Research-in-Progress

Gul Calikli
Ted Rogers School of ITM
Ryerson University,
M5B 2K3, Toronto, Canada
gcalikli@ryerson.ca

Ayse Bener
Ted Rogers School of ITM
Ryerson University,
M5B 2K3, Toronto, Canada
ayse.bener@ryerson.ca

Bora Caglayan
Dept. of Computer Eng.
Bogazici University,
34342, Istanbul, Turkey
bora.caglayan@boun.edu.tr

Ayse Tosun
Dept. of Information Processing
Science, Oulu University,
Oulu, Finland
ayse.tosun@oulu.fi

Abstract

The aim of the research is to explore the impact of cognitive biases and social networks in testing and developing software. The research will aim to address two critical areas: i) to predict defective parts of the software, ii) to determine the right person to test the defective parts of the software. Every phase in software development requires analytical problem solving skills. Moreover, using everyday life heuristics instead of laws of logic and mathematics may affect quality of the software product in an undesirable manner. The proposed research aims to understand how mind works in solving problems. People also work in teams in software development that their social interactions in solving a problem may affect the quality of the product. The proposed research also aims to model the social network structure of testers and developers to understand their impact on software quality and defect prediction performance.

Keywords: Cognitive bias, social network, software quality, defect prediction

Introduction

The demand for complex Information Systems (IS) has increased more rapidly than the ability to design, implement, test and maintain them (Lyu 1996). Today, modern business organizations become more dependent on their information systems to deal with rapidly changing environment and increasing complexity (Ward and Peppard 2002). Software development is an integral part of IS development. As the requirements for software systems increase, failures also increase due to lack of sufficient advancement in productivity, quality, cost and performance issues. Various historical software failure stories, such as the loss of Mars Climate Orbiter (Liu 2000) or radiation therapy miscalculation at National Cancer Institute (Germain 2004), showed that failures occurred in the field caused many million dollars of budget loss as well as significant health problems and loss of human lives.

As the complexity of software systems and the interactions between increasing number of developers have grown, the need for an engineering discipline has emerged to solve common problems of the domain: completing projects on time, within budget and with minimum errors. Software projects are inherently difficult to control. Managers struggle to make many decisions under a lot of uncertainty. These concerns have drawn much attention to software measurement, software quality, and software cost/ effort estimation. Quality of software is often measured by the number of defects in the final product. Minimizing the number of defects -maximizing software quality- requires a thorough testing of the software in question. On the other hand, testing phase requires approximately 50% of the whole project schedule (Song et al. 2006). This means testing is the most expensive, time and resource consuming phase of the software development lifecycle. An effective test strategy should consider minimizing the number of defects while using resources efficiently. Therefore, effective testing leads to a significant decrease in project costs and schedules. In this sense, defect prediction models are helpful tools for guiding software testing. The aim of defect prediction is to give an idea about the testing priorities, so that either exhaustive testing is prevented or larger number of defects is detected in shorter times.

In many real world problems, there are lots of random factors affecting the outcomes of a decision making process. Under such uncertainty, Artificial Intelligence (AI) methods such as data mining machine learning techniques are helpful tools for making generalizations of past experiences in order to produce solutions for the previously unseen instances of the problem.

Software engineering (SE) is a domain with many random factors and remarkably effective predictors for software products have been generated using data mining methods (Menzies et al. 2007). However, all defect prediction models hit a performance ceiling effect when they cannot find additional information that better relates software metrics with defect occurrence, or effort intervals (Menzies et al. 2008). Software engineering aims to formalize the software development process to better control and manage the risks. However, this formalization does not take into account cognitive and social aspects of software development process. The field of IS suggests that we should consider both formal and informal aspects of systems development (Avgerou 1987).

To overcome these limits, researchers use combinations of metric features from different artifacts of software, which we call information sources, in order to enrich the information content in the search space (Menzies et al. 2007; Misirli et al. 2011; Munson and Khoshgoftaar 1992; Nagappan et al. 2006; Ostrand et al. 2005). However, there has been little evidence on how *people* affects software quality even though *people* is the most important and fundamental ingredient of SE discipline, but very difficult to model (Shull et al. 2007). It is inevitable that we should move to a model that considers Product, Process, and People (3Ps).

Reduction of software development and maintenance costs and decreases in the probability of software errors contribute to the cost-effectiveness of an IS as a whole (Avgerou 1987). However, one of the problems stemming from the application of SE in the development of IS is that SE mostly gives emphasis to technical issues (i.e. *Product* and *Process*) neglecting social and cognitive aspects of *People* issues. As stated by Nygaard, advances of SE will be best utilized for the development of IS, if SE recognizes the importance of aspects regarding social activities and find ways to account for them (Nygaard 1986). We believe that, defect prediction models, which take into account cognitive and social aspects of people offers significant potential for the improvement of IS.

While there exists some work on modeling people as individuals through their cognitive aspects (Calikli and Bener 2012) and as groups in social network studies (Bicer et al. 2011), the focus of this paper is integration of human aspects into software defect prediction models in order to complete three pillars 3Ps ın software development and improve software quality. Therefore, our research question is:

RQ: How can we model software engineers' thought processes and their social interactions to improve performance of defect prediction models?

In the following sections, we will review related literature on software defect prediction, explain background on cognitive biases and social network analysis, and present our methodology and provide concluding remarks.

Related Work

Validation and Testing (VV&T) activities must be included in every stage of software development lifecycle to ensure overall software quality. Different VV&T strategies have been proposed so far to optimize the time and effort utilized during the testing phase: code reviews (Adrian et al. 1982; Shull et al. 2002), inspections (Fagan 1976; Wohlin et al. 2002) and automated prediction tools (Menzies et al. 2007; Nagappan et al. 2006; Ostrand et al. 2005). Defect predictors improve the efficiency of the testing phase in addition to helping developers assess the quality and defect-proneness of their software product (Fenton and Neil 1999). They also help managers in allocating resources.

Most defect prediction models combine well known methodologies and algorithms such as statistical techniques (Nagappan et al. 2006; Ostrand et al. 2005; Zimmermann 2004) and machine learning techniques (Fenton and Neil 1999; Lessmann et al. 2008; Moser et al. 2008; Munson and Khoshgoftaar 1992). They require historical data in terms of software metrics and actual defect rates, and combine these metrics and defect information as training data to learn which modules seem to be defect-prone.

Recent research on software defect prediction shows that defect predictors which employ AI techniques can detect 70% of all defects in a software system on average (Menzies et al. 2007), while manual code reviews can detect between 35 to 60% of defects (Shull et al. 2002), and inspections can detect 30% of defects at the most (Fagan 1976). Furthermore, code reviews are labor-intensive since depending on the review procedure, they require 8 to 20 Line of Code (LOC)/minutes for each person in the software team to inspect the source code (Menzies et al. 2007). Therefore, AI-based models are popularly used by various organizations in order to predict pre-release defects prior to testing phase (Menzies et al. 2007; Tosun et al. 2010) and post-release defects after the release (Nagappan et al. 2006; Nagappan et al. 2008; Ostrand et al. 2005).

Tosun et al. (2010) presented a useful insight into the real challenges associated with every aspect of defect prediction, but particularly on the difficulties of collecting reliable metrics and fault data. Goals of their study were to build a measurement repository, defect tracing program and a defect prediction model. Furthermore, authors presented how management objectives were aligned with these goals in order to show tangible benefits to stakeholders. Table 1 shows a mapping between steps of defect prediction and the objectives of stakeholders. In their proceeding work (Misirli et al. 2011b), they summarized the quantitative benefits of defect predictors in a software organization in terms of a decrease in testing effort by 11% and decrease in post-release defects by 44%.

According to a recent research, defect predictors are upper bounded due to limited information content of code metrics such that different algorithmic approaches could not go beyond the performance achieved so far (Menzies et al. 2008). For this reason, recent research in defect prediction has focused on increasing the information content of the model by adding metrics from other aspects of software development process. So far, *Process* aspect is characterized by churn metrics (Misirli et al. 2011a; Nagappan et al. 2006), *Product* aspect is characterized by network dependencies (Tosun et al. 2009; Turhan et al. 2008), and *People* aspect is characterized by organizational network of developers (Nagappan et al. 2008) to enhance the performance of prediction models.

Table 1. Goals of a Defect Prediction Project Aligned with Stakeholders' Objectives		
	Goals of the Project	Management Objectives
Goal/Objective 1	Code measurement & analysis of the software system	Improve code quality
Goal/Objective 2	Storing a version history and defect data	Measure/control the time to repair the defects
Goal/Objective 3	Construction of a defect prediction model to predict defect prone modules	Decrease lifecycle costs such as testing and maintenance. Decrease defect rates.

Background

We can examine *human as an individual* as well as a member of a social group taking into account *social interactions among humans*. As individual aspects, thought processes of software professionals are a fundamental concern. Thought processes cover a wide range of human aspects, however, we focus on cognitive biases that are believed to affect software development life cycle (SDLC) (Stacy and MacMillan 1995).

Human as an Individual: Cognitive Biases

Deviation of the human mind from the laws of logic and mathematics results in *cognitive biases*. In their influential work, Tversky and Kahneman initially introduced the term *cognitive bias*, as a byproduct of processing limitations (Kahneman et al. 1982). The biological limitations on the information processing power of human mind lead to "bounded rationality" so that people would make decisions based on the information and the time they have (Hilbert 2012; Simon 1955). This view suggests that people are only partly rational and they make irrational decisions. In order to reduce cognitive effort, humans employ heuristics, which are the shortcuts in their information processing (Gigerenzer and Goldstein 1996; Kahneman et al. 1982; Shah and Oppenheimer 2008). Therefore people make sufficing and satisfying decisions due to limited time information and processing capacity (Simon 1956). In his recent work Kahneman defines 'fast thinking' (System 1), and 'slow thinking' (System 2) mind (Kahneman 2011). He argues that System 1 is there, it is hard to train, it may be useful, but it is error prone. In other words, people use intuitive heuristics in making decisions regardless of their experience, and these intuitions make them over confident that they do not realize that they make mistakes. He suggests that System 2 should slow down System 1 to block the errors of System 1. Software testing is a task that fast thinking heuristics would be problematic since software tester is expected to try to break the code instead of trying the confirm that the code satisfies a corresponding requirement. Therefore, software tester needs to have System 2 alert to avoid errors. In this research we would like to understand the impact of biases (fast thinking) as it was defined by Kahneman and Tversky, and Wason. There may be many evolutionary reasons for the existence of cognitive biases (Cosmides and Tooby 1994). There may be costs in evolutionary terms, because the development of certain brain circuits may remove potential energetic allocation away from the development of other mechanisms (Arkes 1991). Moreover, there may be costs in real time, because using complex algorithms will take longer time or require more additional resources than decisions using simpler alternatives (Arkes 1991). Since heuristics mostly lead us to correct solutions, they are preferred in daily life situations. However, since cognitive biases do not always result in correct solutions, they negatively affect the quality of software products as well as the quality of other IS components (Calikli et al. 2012; Stacy and MacMillian 1995; Parsons an Saunders, 2004). Stacy and MacMillian discussed various scenarios where cognitive biases could negatively affect the software development process (Stacy and MacMillian 1995). In our previous work, we also found a statistically significant correlation between software defect density and confirmation bias, which is a specific cognitive bias type (Calikli et al. 2012). Moreover, Parsons and Saunders empirically showed that another type of cognitive bias called anchoring and adjustment, leads to the propagation of errors in reused software code and artifacts (Parsons and Saunders 2004). Therefore, we focus on the negative effects of cognitive biases on software development.

Over the last six decades, many cognitive bias types have been introduced and the list is still evolving. In addition to information processing shortcuts (heuristics) (Kahneman et al. 1982) mental noise and mind's limited information processing capacity (Simon 1955), some cognitive bias types, which are due to social influence (Boehm and Phister 2008), and emotional and moral motivations (Wang et al. 2001), have been added to the list. In the literature, there are discussions and some empirical evidence about the effect of the following cognitive biases on software development: *representativeness*, *availability*, *anchoring and adjustment*, and *confirmation bias* (Parsons and Saunders 2004; Stacy and MacMillian 1995; Teasley et al. 1993).

Representativeness

While making predictions and judgments under uncertainty, people do not appear to follow the statistical theory (Tversky and Kahneman 1971). Due to *representativeness*, people expect to obtain results, which they consider to be representative or typical, without considering the size of the sample being used. In fact, smaller samples are more likely to yield atypical (non-representative) results. *Representativeness* can lead to problems during testing phase of a software product. Assume that a developer has to decide whether a set of test case *y* should be eliminated or not. Such a decision depends on the similarity of y to *X* (i.e. whether *y* is representative of *X*) where *X* belongs to the population of test cases that produce errors. As a result of representativeness, test cases, which produce errors may be overlooked and this in turn leads to an increase in software defect density.

Availability

Due to *availability*, humans judge the frequency of instances by the ease that they come to mind. *Availability* might reveal itself during software unit testing phase. In addition to formal tests, good programmers also use "error guessing" techniques which means creating test cases based upon guesses about where the program might have errors (McConnell 2004). Such guesses might be based on past experience of the developer about the type and the location of the errors he has made most frequently. In this case, developer can misidentify the most frequently occurring error types and locations based on the ease that they come to his/her mind. For instance, developer might remember the parts of the code, where he/she spent more time, more easily. Hence, she/he might identify those parts of the code as the most error-prone parts.

Anchoring and Adjustment

Our mental search process, which is based on a first thought, is limited. Therefore, the solution of a problem is biased (i.e. adjusted) towards the initially considered values (i.e. anchors). The effects of *anchoring and adjustment* can be observed during software artifact reuse leading to propagation of errors to next release of the software product. Moreover, due to *anchoring and adjustment,* developers may fail to include an artifact that is part of software requirements. They may also include an artifact that is not part of software requirements (Parsons and Saunders 2004).

Confirmation Bias

The tendency of people to seek for evidence that could verify their theories rather than seeking for evidence that could falsify them is called *confirmation bias*. The term, "confirmation bias" was first used by Peter Wason in his rule discovery experiment (Wason 1960) and later in his selection experiment (Wason 1968). Empirical evidence shows that software testers are more likely to choose positive tests rather than negative tests (Teasley et al. 1993; Teasley et al. 1994). However, during all levels of software testing, the attempt should be to *fail the code* to reduce software defect density. In our previous research, we found empirical evidence supporting the claims about the negative effects of confirmation bias on software development (Calikli and Bener 2010a; Calikli and Bener 2010b; Calikli and Bener 2012).
We claim that understanding the cognitive aspects of individuals is equally, if not more, important than understanding their social interactions. and process and product characteristics. The performance of defect prediction models, which we built by using only confirmation bias metrics, turned out to be

comparable with the performance of the defect prediction models that use product and process metrics (Calikli and Bener 2012). Although there is immense amount of cognitive bias types and the list is still evolving, any comprehensive theory of what creates these biases has not emerged yet. Therefore, the possible relationships among cognitive biases have not been clarified yet, either. However, based on their definitions, relationship between some of the cognitive bias types is obvious (e.g. *backfire effect* and *confirmation bias*, *focalism* and *anchoring, illusion of validity and representativeness, availability and ego-centric biases)* (Ross and Sicoly 1979; Tversky and Kahneman 1974). Employing highly correlated cognitive bias metrics will not lead to a significant increase in defect prediction performance. The relation between the four cognitive biases (e.g. confirmation bias, representativeness, availability, and anchoring and adjustment) is not tight. Therefore, these metrics can be taken into account individually in order to improve the prediction performance of our models.

Human as a Member of a Social Group: Social Interactions

While extremely valuable, modeling individuals in isolation does not give the overall picture of the human aspects in SE. The missing part in such a model is the interaction between developers and testers. Software development is a social activity (Kautz and Nielsen 2004; Russo and Stolterman 2000; Sawyer et al. 2010). In every nontrivial software project, there is collaboration between different developer teams and stakeholders. Interest in social network of developers and using that information for prediction or explanation of various phenomena started after wide availability of this collaboration data. One reason behind this surge of interest may be the emergence of *global software development*. If distributed software teams cannot reduce interdependence of tasks between different sites overall efficiency of the software teams may be reduced as well as negatively affecting quality of the software product (Herbsleb and Mockus 2003).

Network models have been used by researchers for than fifty years in order to model this activity idea (Easley and Kleinberg 2010). The theoretical basis of the validity of social network model in estimating human interaction comes from the work of social anthropologist J. A. Barnes. He claims that a proper model for social networks cannot be formed and the basis for social networks will remain a basic idea (Barnes et. al 1972). Therefore, we use "Networks can be used to model human interaction" as an assumption in our research. The theoretical basis for the proposed social networks measures comes from the work of Friedkin (Friedkin 1991). Network measures such as density and betweenness measures, aims to calculate a key property of the social network deterministically. Earlier research in sociology focuses on modeling the derivation of centrality measures from basic human interaction processes idea. These metrics are based on estimating a network property by using several assumptions. Friedkin built a theory for the derivation of the centrality measures in an influence network from a fundamental influence relation among people (Friedkin 1991). However, to the best of our knowledge, a general theory for the collaboration network measures is not proposed in the literature. We believe that the validity of proposed measures can only be proven empirically. Graph theory is the study of graphs in mathematics, and proposes many methods to estimate the structure of a network such as density. We will use graph theory to propose candidate metrics. Our purpose for using social network measures to model software quality is that software quality may change due to the structure of the collaboration (Herbsleb and Mockus 2003). Previous research also empirically validated the merits of social network metrics in defect prediction (Bicer et al. 2011; Meneely et al. 2008).

Interaction of the developers during software development can be modeled as a collaboration network and the structure of such a network may be an important aspect of software quality. Network is a useful abstraction in computer science and it is used widely in a range of application areas (Easley and Kleinberg 2010). Software collaboration network consists of every developer who collaborated on a software artifact *(nodes)* and links among the developers for every contribution *(edges)*. The collaboration network may be constructed from version control system of the software project. A different collaboration network is formed for each software module within a software project. The collaboration network metrics makes up a new metric set which can be used in defect prediction (Bicer et al. 2010). Also collaborative network in issue manager tools may be used to estimate number of bugs in the future. Organizational changes may be suggested by analyzing the network, in order to avoid or change possibly problematic network structures. For example, networks with large average path lengths may make information sharing more indirect causing communication problems within the project.

Methodology

This study will be conducted on 200 software engineers. The conceptual research model is shown in Figure 1. Based on the grounded theory in cognitive psychology literature, we will prepare a cognitive bias test. The cognitive bias test would be an extension of the confirmation bias test that we prepared and modified during our previous research (Calikli and Bener 2012). Test administration, evaluation of the test results and extracting cognitive bias metric values will all be done automatically, once we complete the implementation of the related modules in our web based tool (Caglayan et al. 2012). In order to externally validate our results, our study will cover four software development companies each of which are specialized in different domains. 50 software engineers from each company will take the cognitive test. In order to validate our results internally, all software engineers within each company will take the test in a computer laboratory that is isolated from distracting factors such as noise, under the supervision of a researcher. In order to avoid construct validity, we will define a metric set for each of the remaining three cognitive bias types (representativeness, availability, and adjustment and anchoring). In this way, we aim to prevent mono-method bias. We will modify cognitive bias and social network metrics, if necessary. Correlation analysis will be performed and feature selection algorithms will be executed to form our final metric set. In order to disprove/verify our hypotheses, we will examine the correlation of these metrics with software defect density. After having formed the final metric set, we will build a defect prediction model using people related metrics and then evaluate performance of our model.

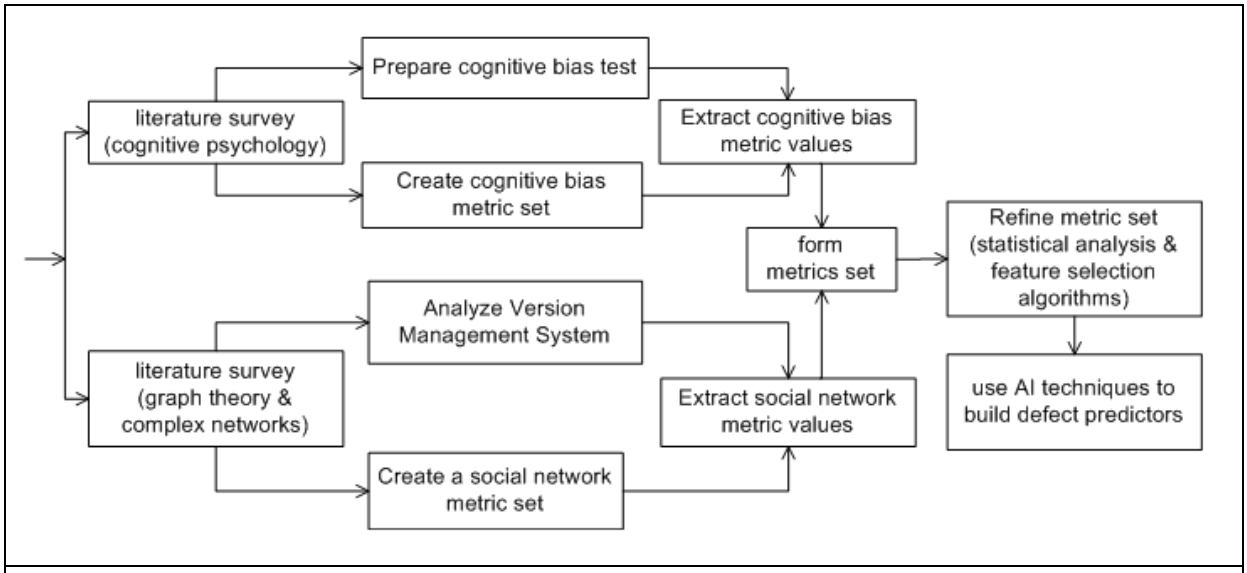

Figure 1. Conceptual Research Model

Confirmation bias metrics are based on grounded theory in cognitive psychology (Johnson-Laird and Wason 1970; Matarasso-Roth 1979; Reich and Ruth 1982; Wason 1960). Metrics for representativeness, availability, and adjustment and anchoring are also based on grounded theory in cognitive psychology literature (Tversky and Kahneman 1974). The list below includes initial versions of the metrics for representativeness, availability, and adjustment and anchoring as well as some of the basic confirmation bias metrics. A detailed version of the confirmation bias metrics is given in our previous work (Calikli and Bener 2012).

In order to quantify the various aspects of the collaboration network, we adapted various metrics from the complex network literature (Easley and Kleinberg 2010). As we collect data social network metric set will be modified and finalized. The initial metric set, which has been formed to quantify, social interactions is explained as follows:

- *Betweenness Centrality:* The betweenness centrality of a node *i* is the number of shortest paths between pairs of other nodes that run through *i* (Freeman 1977). It is a measure of the influence of a node over the flow of information between other nodes, especially in cases Try to avoid long or complex sentence structures.
- *Closeness Centrality:* The closeness centrality of a node *i* is the number of steps required to access every other node from a given node (Freeman 1979). This metric is also called as "distance centrality". Higher value for a node means it is easier for the node to spread information through the network
- *Closeness Centrality:* The closeness centrality of a node *i* is the number of steps required to access every other node from a given node (Freeman 1979). This metric is also called as "distance centrality". Higher value for a node means it is easier for the node to spread information through the network
- *Degree Centrality:* The degree centrality of a node *i* is the number of neighbors it has. Higher degree centrality for a node means it is more popular in network.
- *Group Degree Centrality Index:* A measure of how central a network is (Freeman 1979). e.g. Group degree centrality index of a network will have its highest value with star topology. Higher values of group degree centrality index indicate there are more hubs which control information flow through the network.
- *Density:* Proportion of edges in a network relative to total number of possible edges. Indicates how nodes in a network are tightly bounded together.
- *Diameter:* Considering any two nodes in a network, the biggest value of the shortest possible path between them is diameter of the network. Higher values of diameter indicate that diffusion of a piece of information to entire network would take longer time.
- *Clustering Coefficient:* Measures the cliquishness of a typical neighborhood (a local property) (Watts and Strogatz 1998). Calculated as described in (Watts and Strogatz 1998). Its interpretation is same as density.
- *Bridge:* A bridge in a network is an edge, which disconnects its endpoints completely when it's removed from the network. Bridge is an important kind of edge, bridges enables individuals in a social network to reach novel information. This metric can be thought as inverse of density and clustering coefficient.
- *Characteristic Path Length:* Characteristic Path Length is defined as the number of edges in the shortest path between two nodes, averaged over all pairs of nodes (Watts and Strogatz 1998). A small value of characteristic path length indicates presence of short-cut connections between some pairs of nodes. This means information can spread through the network easily.

Conclusions and Possible Implications

Software defect prediction models have tackled the problem of which parts of the software are likely to be defective to help managers effectively allocate resources during the testing phase of the product. However, these models only take into consideration the product (e.g. lines of code, code complexity, etc.) and process- (e.g. change history of code) related attributes of SDLC. The overall aim of our research program is to explore the impact of cognitive biases and social interactions in the development and testing of software.

The objective of this research in the long run is to help software development managers make specific resource allocation decisions by considering the metrics related to the thought processes of people and their social interactions. The guidance of metrics related to the thought processes of people and social interactions among people may decrease the uncertainty in Human Resource (HR) related decisions to a significant extent. As predictive models mature, especially with an in-depth understanding of people, they can be used to define policies in software development organization

References

Adrian, R.W., Branstad, A. M., and Cherniavsky, C. J. 1982. "Validation, Verification and Testing of Computer Software", *ACM Computing Surveys* (14:22), pp. 159-192.

Arkes, H. R. 1991. "The Costs and Benefits of Judgment Errors: Implications for Debiasing," *Psychological Bulletin* (110:3), pp. 48-498.

Avgerou, C. 1987. "The Applicability of Software Engineering in Information Systems Development," *Information & Management* (13:3), pp. 147-156.

Barnes, J. A. 1972. *Social Networks*, New York, NY: Addison-Wesley Modular Publications.

Bicer, S., Caglayan, B., and Bener, A. 2011. "Defect Prediction Using Social Network Analysis on Issue Repositories," in *Proceedings of the 5th International Conference on Software and Systems Process*, D. Raffo, D. Pfahl, and L. Zhang, Waikiki, Honolulu (eds.), HI, pp. 63-71.

Boehm, B. 1984. "Software Engineering Economics," *IEEE Transactions on Software Engineering*, (10:1), pp. 4-21.

Boehm, G., and Phister, H. R. 2008. " Anticipated and experienced emotions in environmental risk perception," *Judgement and Decision Making* (3:3), pp. 73-86.

Caglayan, B., Misirli, T.A., Calikli, G., Bener, A., Turgay, A., and Turhan, B. 2012. " Dione: An Integrated Measurement and Defect Prediction Solution," to appear in *Proceedings of the 20th International Symposium on on the Foundations of Software Engineering*, Cary, North Carolina.

Calikli, G., and Bener, A. 2010a. "Empirical Analyses Factors Affecting Confirmation Bias and the Effects of Confirmation Bias on Software Developer/Tester Performance," in *Proceedings of the 6th International Conference on PredictiveModels in Software Engineering*, T. Menzies, and G. Koru (eds.), Timisoara, Romania, pp. 1-10.

Calikli, G., and Bener, A. 2010b. " Preliminary Analysis of Confirmation Bias on Software Defect Density," in *Proceedings of the 4th International Symposium on Empirical Software Engineering and Measurement*, G. Succi, M. Morisio, and N. Nagappan (eds.), Bolzano/Bozen, Italy, pp. 1-1.

Calikli, G., and Bener, A. 2012. " Influence of Confirmation Biases of People on Software Quality: An Empirical Study," Software Quality Journal, July 2012, pp. 1-40.

Cosmides, L., and Tooby, J. 1994. " Better than Rational: Evolutionary Psychology and the Invisible Hand," *American Economic Review* (84:2), pp. 327-332.

Easley, D. and Kleinberg, J. M. 2010. *Networks, Crowds, and Markets: Reasoning about a Highly Connected World*, New York, NY: Cambridge University Press.

Fagan, M. 1976. "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal* (15:3), pp. 575-607.

Fenton, N., and Neil, M. 1999. "*A* Critique of Software Defect Prediction Models," *IEEE Transactions on Software Engineering* (25:5), pp. 675-689.

Freeman, L. C. 1977. " A Set of Measures of Centrality Based on betweenness," *Sociometry,* (40:1), pp. 35-41.

Freeman, L. C. 1979. "Centrality in Social Networks: Conceptual Clarication", *Social Networks*, (1:3), pp. 215- 239.

Friedkin, N. 1991. "Theoretical Foundations for Centrality Measures," *The American Journal of Sociology*, (96:6), pp. 1478-1504.

Germain, M. J. 2004. "Can Software Kill you?", TechnewsWorld, Technology Special Report.

Gigerenzer, G., and Goldstein, D. G. 2003. " Reasoning Fast and Frugal Way: Models of Bouned Rationality," *Psychological Review* (103:4), pp. 650-669.

Girvan M., and Newman M. E. J. 2002. " Community Structure in Social and Biological Networks," in *Proceedings of the 14th National Academic Sciences of United States of America*, L. A. Shepp (ed.), New Brunswick, Picataway, NJ, pp. 7821-7826.

Herbsleb J. and Mockus A. 2003. "An Empirical Study of Speed and Communication in Globally Distributed Software Development," *IEEE Transactions on Software Engineering*, (29:6), pp. 481-494.

Hilbert, M. A. 2012. " Toward a Synthesis of Cognitive Biases: How Noisy Information Processing Can Bias Human Decision Making," *Psychology Bulletin* (138:2), pp. 211-237.

Johnson-Laird, P. N. and Wason, P. C. 1970. "A Theoretical Analysis of Insight into a Reasoning Task," *Cognitive Psychology* (1:2), pp. 134-148.

Kahneman, D., Slovic, P. and Tversky, A. 1982. *Judgement under Uncertainty: Heuristics and Biases*, Enew York, NY: Cambridge University Press.

Kautz, K., and Nielsen, P. A. 2004. "Understanding The Implementation of Software Process Improvement Innovations in Software Organisations," *Information Systems Journal* (14:1), pp. 3-22.

Lessmann, S., Baesens, B., Mues, C., and Pietsch, S. 2008. "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings" *IEEE Transactions on Software Engineering* (34:4), pp.1-12.

Lyu, M.R. 1996. "Chapter 1: Introduction" in *Handbook of Software Reliability Engineering*, M. R. Lyu (ed.), New York: IEEE Computer Society Press and McGraw-Hill, pp. 3-25.

Mataraso-Roth, E. 1979. " Facilitating Insight in a Reasoning Task," *British Journal of Psychology* (70:2), pp. 265-271.

McConnell, S. 2004. *Code Complete*, Redmond, WA: Microsoft Press.

Meneely, A., William, L., Snipes, W. and Osborne, J. 2008. "Predicting Failures with Developer Networks and Social Network Analysis," in *16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, M. J. Harrold, and G. C. Murphy (eds.), Atlanta, GA, pp. 13-23.

Menzies, T., Greenwald, J., and Frank, A. 2007. "Data Mining Static Code Attributes to Learn Defect Predictors," *IEEE Transactions on Software Engineering*, (33:1), pp. 2–13.

Menzies, T., Turhan, B., Bener, A., Gay, G., Cukic, B., and Jiang, Y. 2008. "Implications of Ceiling Effects in Defect Predictors," in *Proceedings of the 4th International Workshop on Predictor moels in Software Engineering*, B. Boetticher, and T. Ostrand (eds.), Leipzig, Germany, pp. 47-54.

Misirli, T. A., Bener, A. and Kale, R. 2011b. " AI-Based Software Defect Predictors: Applications and Benefits in a Case Study," *AI Magazine* (32:2), pp. 57-68.

Misirli, T.A., Caglayan, B., Miranskyy, A., Bener, A., and N. Ruffolo. 2011a. "Different Strokes for Different Folks: A Case Study on Software Metrics for Different Defect Categories", *in Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics (WETSOM),* G. Concas, E. Tempero, H. Zhang, and M. Di Penta (eds.), Honolulu, HI, *pp. 45-51.*

Moser, R., Pedrycz, W., and Succi, G. 2008. "A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction," in *Proceedings of the 30th International Conference on Software Engineering*, W. Schaefer, M. B. Dwyer, and V. Gruhn (eds.), Lepzig, Germany, pp. 181-190.

Munson, J.C., and Khoshgoftaar, T. M. 1992. " The Detection of Fault-Prone Programs," *IEEE Transactions on Software Engineering* (18:5), pp. 423-433.

Nagappan, N., Ball, T., and Murphy, B. 2006. "Using Historical In-Process and Product Metrics for Early Estimation of Software Failures", in *Proceedings of the 17th International Symposium on Software Reliability Engineering*, C. Schmidts, and A. Paradkar (eds.), Raleigh, NC, pp. 62-74.

Nagappan, N., Murphy, B., and Basili, V. 2008. "The Influence of Organizational Structure on Software Quality: An Empirical Case Study", in *Proceedings of 30th International Conference on Software Engineering*, W. Schaefer, M. B. Dwyer, and V. Gruhn (eds.), Lepzig, Germany, pp.521-530.

Nygaard, K. 1986. "Program Development as a Social Activity," in *Proceedings of the 10th World IFIP Congress*, H. J. Kugler (ed.), Dublin, Ireland, pp. 1-24.

Ostrand, T. J., Weyuker E. J., and Bell, R. M. 2005. " Predicting the Location and Number of Faults in Large Software Systems," *IEEE Transactions on Software Engineering* (31:4), pp.340-355.

Parsons, J. and Saunders, C. 2004. " Cognitive Heuristics in Software Engineering: Applying and Extending Anchoring and Adjustment to Artifact Reuse," *IEEE Transactions on Software Engineering* (30:12), pp. 873-888.

Reich, S. and Ruth, P. 1982. "Wason's Selection Task: Verification, Falsification and Matching," *British Journal of Psychology* (73:3), pp. 395-405.

Ross, M., and Sicoly, F. 1979. "Egocentric Biases in Availability and Attribution," *Journal of Personality and Social Psychology* (37:3), pp. 322-336.

Russo, N. L., and Stolterman, E. 2000. "Exploring the Assumptions Underlying Information Systems Methodologies: Their Impact on Past, Present and Future ISM Research," *Information Technology and People* (13:4), pp. 313-327.

Sawyer, S., Guinan, P. J., and Cooprider, J. G. 2010. "Social Interactions of Information Systems Development Teams: A Performance Perspective," *Information Systems Journal*, (20:1), pp. 81-107.

Shah, A. K., and Oppenheimer, D. M. 2008. "Heuristics Made Easy: An Effort-Reduction Framework," *Psychological Bulletin,* (134:2), pp. 207-222.

Shull, F., Boehm, V. B., Brown, A., Costa, P., Lindvall, M., Port, D., Rus, I., Tesoriero, R., and Zelkowitz, M. 2002. " What We Have Learned About Fighting Defects," in *Proceedings of the 8th International Software Metrics Symposium*, Ottawa, Canada, pp. 249-258.

Shull, F., Singer, J., and Sjoberg, D. I. K. 2007. *Guide to Advanced Empirical Software Engineering*, Seaucus, NJ: Springer-Verlag.

Simon, H. A. 1955. "A Behavioral Model of Rational Choice," *The Quarterly Journal of Economics*, (69:1), pp. 99-118.

Song, Y. D., Dong, T., and Li, Y. 2006. "Using Grid Computing for Distributed Software Testing"' in *Proceedings of the 2006 International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, NV, pp. 931-936.

Stacy, W., and MacMillian, J. 1995. " Cognitive Bias in Software Engineering," *Communications of the ACM*, (38:6), pp. 57–63.

Liu, J. W. S. 2000. *Real Time Systems*, New York, NY: Prentice-Hall.

Teasley, B., Leventhal, L. M., and Rohlman, S. 1993. "Positive Test Bias in Software Engineering Professionals: What is Right and What's Wrong," *in Proceedings of the 5th Workshop on Empirical Studies of Programmers,* C. R. Cook, J. C. Scholtz, and J. C. Spohrer (eds.), Palo Alto, CA, pp. 210-218.

Teasley, B. F., Leventhal, L. M., Mynatt, C. R., and Rohlman D. S. 1994. "Why Software Testing is Sometimes Ineffective: Two Applied Studies of Positive Test Strategy," *Journal of Applied Psychology* (79:1), pp. 142-155.

Tosun, A., Turhan, B., and Bener, A. 2010a. "Validation of Network Measures as Indicators of Defective Modules in Software Systems," in *Proceedings of the 5th International Conference on PredictiveModels in Software Engineering*, T. Ostrand (ed.), Vancauver, BC, Canada, pp. 5-5.

Tosun, A., Bener, A., Turhan, B., and Menzies, T. 2010. Practical Considerations in Deploying Statistical Methods for Defect Prediction: A Case Study within the Turkish Telecommunications Industry," *Information and Software Technology*, (52:11), pp.1242-1257.

Towfic, F., VanderPlas, S., Oliver, C. A., Couture, O., Tuggle, C. K., Greenlee, M. H. W., and Honavar, V. 2009. "Detection of Gene Orthology from Gene Co-Expression and Protein Interaction Networks," in *Proceedings of the 2009 IEEE International Conference on Bioinformatics and Biomedicine*, R. Bilof (ed.), Washington, D.C., pp. 48-53.

Turhan, B., Kocak, G. and Bener, A. 2008. "Software Defect Prediction Using Call Graph Based Ranking (CGBR) Framework," in *Proceedings of the 34th International EUROMICRO Software Engineering and Advanced Applications Conference*, Parma, Italy, pp. 191-198.

Tversky, A., and Kahneman, D. 1971. "Belief in Small Numbers," *Psychological Bulletin* (76:2), pp. 105-110.

Wang, X. T., Simons, F., and Bredart, S. 2001. "Social Cues and Verbal Framing in Risky Choice,"
Journal of Behavioral Decision Making, (14:1), pp. 1 –15.

Ward, J., and Peppard, J. 2002. *Strategic Planning for Information Systems*, New York, NY, Wiley.

Wason, P. C. 1960. "On the Failure to Eliminate Hypotheses in a Conceptual Task", *Quarterly Journal of Experimental Psychology*, (12:3), pp. 129-140.

Wason, P. C. 1968. "Reasoning about a Rule", *Quarterly Journal of Experimental Psychology*, (20:3), pp. 273-281.

Watts, D. J., and Strogatz, S. H. 1998. "Collective Dynamics of Small-World Networks", *Nature*, (393:6684), pp.440-442.

White, S., and Smyth, P. 2003. "Algorithms for Estimating Relative Importance In Networks," in *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, L. Getoor, T. Senator, P. Domingos, and C. Faloutsos (eds.), New York, NY, pp. 266-275.

Wohlin, C., Aurum, A., Petersson, H., Shull, F., and Ciolkowski, M. 2002. "Software Inspection Benchmarking: A Qualitative and Quantitative Comparative Opportunity," in *Proceedings of the 8th International Symposium on Software Metrics*, IEEE Computer Society, Washington, DC, USA, pp.118-127.

Zimmermann, T. 2004. "Mining Version Histories to Guide Software Changes," *IEEE Transactions on Software Engineering*, (31:6), pp. 429-445.