Journal of Information Technology Management

ISSN #1042-1319

A Publication of the Association of Management

ORGANIZING SOFTWARE TESTING FOR IMPROVED QUALITY AND SATISFACTION

XIHUI ZHANG
UNIVERSITY OF NORTH ALABAMA
xzhang6@una.edu

JASBIR DHALIWAL
UNIVERSITY OF MEMPHIS
jdhaliwl@memphis.edu

MARK L. GILLENSON
UNIVERSITY OF MEMPHIS
mgillnsn@memphis.edu

ABSTRACT

The way testing is organized for software development has not been adequately addressed in both the practitioner and academic research literature. In practice, a diverse set of methods is being used to organize testing. Some organizations emphasize one-to-one matching between developers and testers while others do not. Additionally, some organizations have a distinct testing unit for their testing professionals while others have them in the same unit as developers. Such practices are also influenced by the development methodologies of the organization such as the lifecycle and agile approaches. This paper attempts to shed light on whether these governance choices matter. It considers the influence of the development methods, the existence of one-to-one matching between developers and testers, and the existence of a distinct corporate testing unit on software quality and job satisfaction. The results of this study suggest that development methods do not significantly influence software quality or job satisfaction. However, one-to-one matching of developers and testers has a positive influence on both software quality and job satisfaction. The existence of a dedicated organizational unit for software testing also has a positive influence on the quality of software developed. These results suggest that organizations must emphasize one-to-one matching and a distinct testing unit for improved software quality and job satisfaction.

Keywords: Software development, lifecycle models, agile methods, software testing, software quality, job satisfaction.

INTRODUCTION

In the last twenty years, there has been steady interest in the use of adaptive, flexible methods for developing software. A key reason for this is the perception that methodologies based on the traditional lifecycle methods (e.g., waterfall models) are too slow, often characterized by "paralysis of analysis," and the over-specialization promoted by having different groups undertake separate sequential stages for each activity. In the early days, the focus on adaptive, flexible methods took the form of prototyping, or rapid and joint application development (RAD/JAD). More recently, it has shifted to

the notion of agile methods [13] such as extreme programming and scrum [4][18]. While most major software development organizations that develop large, complex, integrated, and often real-time systems generally subscribe to established methodologies based on the lifecycle methods, there is an increasing tendency to also allow exceptions focused on experimentation with more agile methodologies [6][19]. Other large information technology (IT) organizations are also increasingly starting to struggle with the question of how to organize and implement agile methodologies on a large scale [11][18]. Clearly, the trend toward agile methods is increasing.

The choice of software development methodology has significant implications for software testing, which is a significant component of software development [12][14]. The fact that the software development lifecycle identifies a distinct stage when code is to be tested has led to the birth of a distinct profession called "software testers." This fact also prescribes defined "hand-offs" and structured interactions between developers and testers groups. For example, developers hand code to a testing group for integration testing only after unit testing has been performed by the developers themselves. The testing group then communicates back to the developers with specified defect reports and change requests. This environment generally is not characterized by close one-to-one individual interactions between developers and testers who can build a stable working relationship over time. Instead of one-to-one interactions, the focus is on structured exchanges between a "faceless" development group in the overall organization and a different testing group. Agile methods, it is argued, promote more one-to-one matching that emphasizes personal interactions between individual developers and testers who are generally expected to work together as part of a small team over longer time periods, such as at Microsoft [24]. In such an environment, developers and testers generally belong to the same organizational unit.

From a software testing viewpoint, this research tries to provide empirical clarification as to the influence of using lifecycle versus agile methods for software development. The focus is on answering the following two questions, as suggested in Zhang et al. [35]: First, does the use of one methodology over another lead to better quality of software? Second, does the use of a particular methodology lead to better job satisfaction? Software quality and job satisfaction are chosen because they are closely related to the context of software development and testing. Software with poor quality leads to unhappy end users, and low job satisfaction leads to employee turnover. Therefore, these critical questions must be answered for any IT organization to make critical decisions about how to do development and testing. Given that the application of the lifecycle methodology generally means that testers report to a distinct organizational sub-unit within the larger corporate IT unit, it is also important to understand the influence of this governance choice.

Further, this study examines the influence of the existence of one-to-one matching between developers and testers, and the existence of a distinct corporate testing unit on two dependent variables that represent impacts: software quality and job satisfaction. The paper proceeds as follows. First, we describe the research model that was used to drive investigation of the research questions discussed above. Next, we provide detailed methodological aspects about the empirical study that was undertaken to collect data. Then, we describe measurement issues pertaining to the constructs and data analysis as well as the findings of the study. Finally, we conclude the paper with a discussion of critical implications of the findings.

THEORETICAL DEVELOPMENT

Guided by theory and past research, a research model in Figure 1 is proposed which asserts that the use of development methods such as agile methods, the existence of one-to-one matching between developers and testers, and the existence of a distinct corporate testing unit will have significant influence on both software quality and job satisfaction. We provide theoretical support for the hypothesized relationships in the following sections.

Agile versus Lifecycle Methods

In the early 1950's, the software development process had only two steps: an analysis step followed by a coding step [28]. This analysis-coding model, however, is practically ineffective and inefficient for the development of large and complex programs [28]. Lifecycle models were then developed, with the intent to bring control and order into software development. Lifecycle methods are plan-based approaches which divide the software development process into clear-cut phases, typically including phases such as analysis, design, coding, testing, and implementation [21]. While bringing structure and order to software development, lifecycle methods often have difficulty in meeting the changing needs of end users.

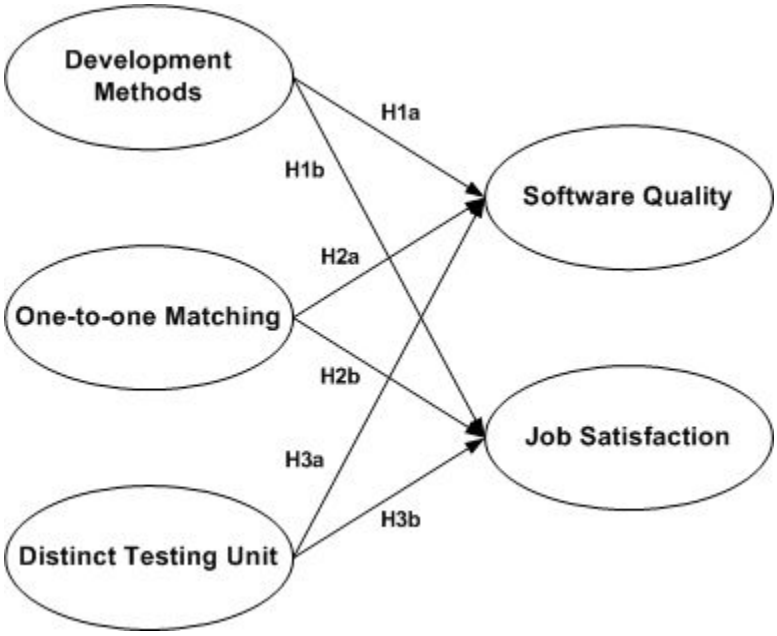

Figure 1: Research Model

Agile software development is basically an iterative approach that focuses on incremental specification, design, and implementation [29], while requiring full integration of testing and development [31]. Because of its adaptability, an agile method can easily adapt to different project contexts [2]. In the agile environment, testing is a frequent activity as small amounts of code are tested immediately upon being written [9]. This is a feature called "testing throughout the lifecycle" [2]. According to the *Manifesto for Agile Software Development* (http://agilemanifesto.org), agile methods value: (1) individuals and interactions over processes and tools, (2) working software over comprehensive documentation, (3) customer collaboration over contract negotiation, and (4) responding to change over following a plan. The intent is to produce high quality software in a cost effective and timely manner, and in the meantime, meet the changing needs of end users.

There are many agile software development methods, and eXtreme Programming (XP), which is dependent on highly skilled programmers [14], is the most prominent one. After a short planning stage, XP goes through analysis, design, and implementation stages quickly [8]. A timebox, usually spanning one to four weeks, is used to ensure that new, enhanced software is ready to be delivered for each iteration. Agile methods require developers and testers to interact more and often [13]. Beck and Andres [4] note that XP encourages communication, simplicity, feedback, courage, and respect, which may improve both software quality and job satisfaction. With quantitative and qualitative data, Talby et al. [31] demonstrate that agile software development dramatically improves development quality and productivity. Errors discovered late in the software development process can be more readily handled in an agile development environment than in a lifecycle environment through the use of automated test suites [32]. According to Martens and Gat [20], "agile is a systemic change. It drives cost down, quality up and service levels higher by making the entire process leaner, the entire staff more responsible, and the customer more involved" (p. 27). Thus, we hypothesize:

H1a: The use of development methods such as agile methods will positively influence the quality of the software.

H1b: The use of development methods such as agile methods will positively influence the level of job satisfaction.

One-to-one Matching of Developers and Testers

Many software development organizations such as Microsoft [24] match developers and testers on a one-

to-one basis in teams when implementing agile methods. Note that although not as often as in organizations using agile methods, one-to-one matching between developers and testers does exist in organizations using lifecycle methods. For the same reasons mentioned above, this one-to-one matching between developers and testers may increase their job satisfaction and improve software quality. One-to-one matching of developers and testers facilitates good communication and free flow of information between them. Both developers and testers can get help and feedback quickly, improving their work effectiveness and efficiency. This has the potential to improve the quality of the software they collaborate on. Furthermore, in a one-to-one matching environment, developers and testers spend more time together, getting to know each other better, which increases the likelihood of enjoying their working with each other. Thus, we hypothesize:

H2a: The one-to-one matching of developers and testers will positively influence the quality of software.

H2b: The one-to-one matching of developers and testers will positively influence their job satisfaction.

A Distinct Organizational Unit for Testing

Largely because the software development lifecycle identifies testing as a dedicated sequential stage (after coding) towards the end of the lifecycle, organizations using lifecycle methods tend to have a distinct organizational unit comprising testers who are then assigned to development projects rather than being matched to identified developers. Note that although not as often as in organizations using lifecycle methods, a distinct organizational unit for testers does exist in organizations using agile methods. The use of agile methods necessitates iterative "spirals" of interactions between individual developers and testers who are working as an integrated team. In most organizations that use this approach (e.g., Microsoft), there is no distinct organizational unit for software testing and both the testers and developers report to the same organizational unit. That is, organizations using lifecycle development methods tend to use a distinct organizational unit for software testing. Therefore, there is theoretical value to investigating its influence given the increasing concerns pertaining to the governance of agile methodologies [31][30].

Having a distinct organizational unit for software testing has several advantages for improving software quality. First, testers in a distinct testing unit will focus on testing but nothing else, and they will become experts in what they are doing in the long run. Second, testers in a distinct testing unit will feel less pressure to ship in spite of quality issues than their counterparts who are more integrated with development or operations. Third, a distinct testing unit, whose manager is organizationally equal to the development manager, will provide "an objective look at the software being tested" [5, p. 297]. Myers [22] argues that "a programming organization should not test its own programs" (p. 16) because "a programming organization … is largely measured on the ability to produce a program by a given date for a certain cost" (p. 17). The "meeting the schedule and the cost objectives" are considerably different from the testing objective, which is to deliver the end product with the best quality.

Unfortunately, having a distinct testing unit has some disadvantages too. "The test group can become squeezed between development and operations groups" [15, p. 68], competing for time, personnel, and other scarce project resources. Oftentimes, testers get frustrated because they cannot begin testing early enough to get all the necessary tests done before "the shipping day." To make things worse, testers often get the blame. When communication fails and the flow of information becomes choked, conflict will develop in the project team, especially between developers and testers [34]. All of these may have the potential to negatively influence the job satisfaction of software developers and testers. Thus, we propose:

H3a: The existence of a distinct organizational unit for software testing will positively influence the quality of software.

H3b: The existence of a distinct organizational unit for software testing will negatively influence job satisfaction.

RESEARCH METHODOLOGY

Measurement Items

The use of development methods such as agile methods was measured by one item: *Software code was developed using agile methods as opposed to the systems development lifecycle*. The existence of one-to-one matching between developers and testers was measured by one item: *Testers were largely assigned to support particular developers*. The existence of a distinct corporate testing unit was measured by one item: *Software testing represented an identifiable and distinct organizational unit*. Respondents were asked to score these three measurement items on 7-point Likert-type scales anchored at (1) = strongly agree, (4) = neutral, and (7) strongly disagree.

Software quality was measured using a six-item scale (see Table 1) adapted from scales developed and validated by Barki and Hartwick [3], measuring six dimensions of the construct: functionality, reliability, usability, efficiency, maintainability, and portability. This adapted six-item scale is in accordance with ISO 9126 Standard "Software Product Evaluation – Quality Characteristics and Guidelines" [7], as well as the software quality measurement scales recommended by Issac et al. [16] and Ortega et al. [23]. Respondents were asked to score the measurement items on 7-point Likert-type scales anchored at (1) = not at all, (4) = neutral, and (7) = definitely.

Job satisfaction was measured using a five-item scale (see Table 1) adapted from scales developed and validated by Wright and Cropanzano [33], measuring five dimensions of the construct: degree of satisfaction with the work itself, degree of satisfaction with co-workers, degree of satisfaction with the way one is supervised, degree of satisfaction with opportunities for promotion, and degree of satisfaction with pay and benefits. Respondents were asked to score the measurement items on 7-point Likert-type scales anchored at (1) = strongly agree, (4) = neutral, and (7) = strongly disagree.

Table 1: Measurement Items

Construct	Measurement Item
Development Methods	Software code was developed using agile methods as opposed to the systems development lifecycle.
One-to-one Matching	Testers were largely assigned to support particular developers.
Distinct Testing Unit	Software testing represented an identifiable and distinct organizational unit.
Software Quality	The software developed is reliable (it is always up and running, runs without errors, and does what it is supposed to do).
	It is easy to tell whether the software is functioning correctly.
	The software can easily be modified to meet changing user requirements.
	The software is easy to maintain.
	The software is easy to use.
	The software performs its functions quickly.
Job Satisfaction	I am satisfied with the work that I do in my job.
	I am satisfied with my co-workers.
	I am satisfied with the way I am supervised.
	I am satisfied with opportunities for promotion in my job.
	I am satisfied with my pay and benefits.

Data Collection

An online survey instrument including all the aforementioned constructs and their associated measurement items was developed, and the survey link was distributed to professional software developers and testers by individual emails. We used "Request for Research Assistance" for the subject line of the soliciting emails. In the body of the email, we provided information about the purpose of our study. We also assured the recipients that their responses would be kept completely confidential and that there would not be a way for us to link their responses back to them or to their organizations. A second email, serving as a reminder, was sent three weeks after the first one.

We obtained a total of 1836 unique names and their corresponding email addresses from three major sources: a database provided by a software testing research center, an online directory of software testers and consultants, and SourceForge.net. All in all, 196 people (10.68%) responded to the online survey. Among them, 46.4% identified themselves as developers, another 42.3% identified themselves as testers, and the remaining 11.2% identified themselves as other software development professionals. Responses were removed from the final data set if (1) they were not from developers or testers, or (2) they contained over 60% missing values. As a result, a total of 159 responses were included in our data analysis: 80 were from developers, and 79 were from testers.

Demographics of the Respondents

Demographics of the respondents assessed included: gender, education, years of job related work experience, years with current software development and testing organization, and gross annual income (see Table 2). The ratio of male respondents (67.92%) to female respondents (32.08%) was roughly 2:1. More than 80% of the respondents had a bachelor's degree (43.40%) or a master's degree (38.99%) as their highest degree. More than half of them (54.09%) had over 10 years work experience related to their current job, and more than half of them (51.57%) had spent 1 to 5 years with their current software development and testing organization. 79.25% of the respondents had a gross annual income in the range of $50,000 - $100,000.

Table 2: Demographics of the Respondents (N = 159)

Category	Value	Frequency	Percentage
Gender	Male	108	67.92%
	Female	51	32.08%
Education	HS diploma	10	6.29%
	Associate's degree	12	7.55%
	Bachelor's degree	69	43.40%
	Master's degree	62	38.99%
	Doctoral degree	6	3.77%
Years of job related work experience	Less than 1 year	2	1.26%
	1 to 3 years	15	9.43%
	3 to 5 years	11	6.92%
	5 to 7 years	18	11.32%
	7 to 10 years	27	16.98%
	Over 10 years	86	54.09%
Years with current software development and testing organization	Less than 1 year	8	5.03%
	1 to 3 years	39	24.53%
	3 to 5 years	43	27.04%
	5 to 7 years	21	13.21%
	7 to 10 years	22	13.84%
	Over 10 years	26	16.35%
Gross annual income	Under $25,000	9	5.66%
	$25,000 to $50,000	10	6.29%
	$50,000 to $75,000	57	35.85%
	$75,000 to $100,000	69	43.40%
	$100,000 to $125,000	11	6.92%
	Over $125,000	3	1.88%

DATA ANALYSIS AND RESULTS

Before analyzing the data, we transformed all the reverse-worded data items for the construct of job satisfaction. Specifically, we created a new item for each original data item of the construct, assigning a value to it by using the formula (8 – the original value of the data item). As such, a higher value of the newly created data item will represent a higher level of job satisfaction.

Data analysis was conducted using the partial least square (PLS) estimation technique [10] as applied in the software package WarpPLS 1.0. The primary goal was to determine whether each of the following three factors influences software quality and job satisfaction: the use of development methods such as agile methods, the existence of one-to-one matching between developers and testers, and the existence of a distinct corporate testing unit.

Measurement Model Assessment

Collinearity Diagnostics. The existence of multicollinearity among measurement items is a threat to con-

struct reliability. Data analysis results indicated that the maximum value of the variance inflation factor (VIF) was 1.076, which is below the recommended cut-off value of 3.3 for identifying suspect items that can cause multi-collinearity issues [26]. Therefore, we concluded that there was no threat of multicollinearity within the items.

Convergent and Discriminant Validity. All but four of the individual item loadings for the software quality and job satisfaction constructs were above 0.70. The composite reliability coefficients were 0.880 for software quality and 0.818 for job satisfaction (see Table 3). The Cronbach alpha coefficients were 0.835 for software quality and 0.721 for job satisfaction (see Table 3). Thus, no items were dropped because both the aforementioned coefficients were greater than the suggested value of 0.7. Also in Table 3, the diagonal elements (shaded) are the square root of the average variances extracted (AVE's), i.e., the variance shared between the constructs and their measures. The off-diagonal elements are the correlations among constructs. For discriminant validity, diagonal elements should be larger than off-diagonal elements. Table 3 confirms that all constructs were more strongly correlated with their own measures than they were with any of the other constructs; thus, discriminant validity was observed.

Table 3: Reliability and Discriminant Validity Coefficients

Construct	CRC	CAC	DM	OM	DTU	SQ	JS
DM	1.000	1.000	1.000				
OM	1.000	1.000	0.068	1.000			
DTU	1.000	1.000	-0.200	0.136	1.000		
SQ	0.880	0.836	-0.062	-0.164	-0.191	0.742	
JS	0.818	0.721	0.023	-0.202	-0.063	0.416	0.689

Notes: CRC = Composite Reliability Coefficient; CAC = Crobach Alpha Coefficient; DM = Development Methods; OM = One-to-one Matching; DTU = Distinct Testing Unit; SQ = Software Quality; JS = Job Satisfaction.

Analysis of Common Method Bias. To assess common method bias in our survey data, we performed a Harman's single-factor test, following the procedure outlined by Podsakoff and Organ [27]. In this test, all of the items of the principal constructs in the research model were entered into a principal components factor analysis (PCA) using SPSS v. 17. A substantial amount of common method bias exists "when a single factor emerges from the analysis or when one general factor accounts for the majority of the covariance in the independent and dependent variables" [25, p. 388]. The PCA results showed that there existed five factors with eigen values greater than 1 in the data and no single factor emerged as a dominant factor accounting for most of the variance (the factor with the greatest eigen value accounts for 30.251% of the variance), indicating no substantial common method bias in our survey data.

Analysis of Nonresponse Bias. Nonresponse bias was assessed by comparing early and late respondents, as recommended by Armstrong and Overton [1]. Since the second email, serving as a reminder, was sent three weeks after the first one, respondents of the first three weeks were classified as early respondents, while those responding later than the first three weeks were classified as late respondents. A t-test of all the demographic and research variables showed that there were no statistically significant differences between early respondents and late ones, indicating nonresponse bias was not a major concern.

Structural Model Assessment

The structural model was assessed based on the significance of the path coefficients and the associated p values between the constructs and the values of R^2 obtained for the dependent variables. Path coefficients and associated p-values (obtained by running WarpPLS 1.0 with a bootstrapping procedure with 100 iterations) are presented in Table 4. All but one of the six links are negative, and three of them are significant. Together, the use of development methods such as agile methods, the existence of one-to-one matching, and the existence of a distinct testing unit explained 7.0% of the variance in software quality and 6.0% of the variance in job satisfaction.

Table 4: Effects of Development Methods, One-to-one Matching, and Distinct Testing Unit on Software Quality and Job Satisfaction

IV	SQ		JS		SQ R^2	JS R^2
	β	p	β	p		
DM	-0.085	0.195	0.068	0.383	0.07	0.06
OM	-0.161*	0.026	-0.213**	0.001		
DTU	-0.152*	0.011	-0.074	0.229		

*Note: *p < 0.05. **p < 0.01. ***p < 0.001.*
β = Path Coefficient;
IV = Independent Variable; DM = Development Methods;
OM = One-to-one Matching; DTU = Distinct Testing Unit;
SQ = Software Quality; JS = Job Satisfaction.

Hypothesis Tests and Results

The hypotheses were assessed by examining path coefficients and the associated p-values as well as their significance levels. The hypothesis test results are illustrated in Figure 2 and also summarized in Table 5. H2a, H2b, and H3a were supported at $p < 0.05$, and the remaining hypotheses (H1a, H1b, and H3b) were not supported.

Specifically, neither Hypothesis H1a nor H1b was supported. The use of development methods such as agile methods did not improve the quality of software developed or the level of job satisfaction of developers and testers. Both Hypotheses H2a and H2b were supported. The existence of one-to-one matching between developers and testers was shown to exert a significant positive influence on both software quality and job satisfaction. Hypothesis H3a was supported, but Hypothesis H3b was not supported. The existence of a distinct organizational unit for testing was shown to exert a significant positive influence on software quality, but no significant influence on job satisfaction.

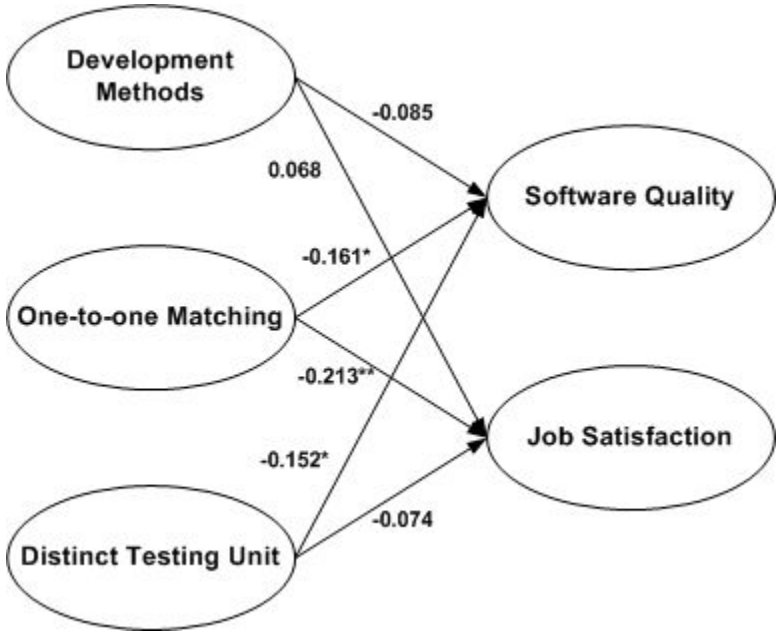

Figure 2: Structural Model Assessment Results

Table 5: Summary of Hypothesis Tests

Hypothesis	Path Coefficient	P-Value	Support for Hypothesis
H1a: The use of development methods such as agile methods will positively influence the quality of the software.	-0.085	0.195	Not Supported
H1b: The use of development methods such as agile methods will positively influence the level of job satisfaction.	0.068	0.383	Not Supported
H2a: The one-to-one matching of developers and testers will positively influence the quality of software.	-0.161*	0.026	Supported
H2b: The one-to-one matching of developers and testers will positively influence their job satisfaction.	-0.213**	0.001	Supported
H3a: The existence of a distinct organizational unit for software testing will positively influence the quality of software.	-0.152*	0.011	Supported
H3b: The existence of a distinct organizational unit for software testing will negatively influence job satisfaction.	-0.074	0.229	Not Supported
*Note: *p < 0.05. **p < 0.01. ***p < 0.001.*			

DISCUSSION

Implications of Findings

Our findings suggest that development methods do not influence the development variables studied in this research. Specifically, our findings suggest that the use of development methods such as agile methods for software development provide no advantages over established traditional lifecycle methods with regard to our dependent variables. The quality of software developed does not get better nor does the level of job satisfaction of developers and testers increase. Typically, more unstructured and iterative interactions between developers and testers take place in agile environments. From a technology management perspective, the increased levels of interaction could imply that more managerial time and attention will be needed in agile environments. However, in our case, the increased managerial time and attention are not manifesting into higher quality of code being produced or more satisfied developers and testers. In other words, environments in which lifecycle methods are used are just as good but may require fewer managerial efforts. Readers may want to consider this result in the context of other research that focuses on particular agile methods such as test-driven development [17].

Another key result of our empirical analysis is that the one-to-one matching of developers and testers does provide significant advantages. Not only does the quality of software produced get better but those involved also perceive greater job satisfaction working together. Given that we did not find any positive impacts on software quality and job satisfaction arising from the use of agile methodologies (see above), these findings showing positive influence from one-to-one matching could be interpreted as suggesting that the current manner in which agile methodologies are being implemented may not be working. While more research into the underlying reasons for this conclusion is warranted (as this study did not deconstruct "agile methods" using a more refined breakdown), the key lesson for practitioners and theorists is that there is value to designing software development and testing methods that promote one-to-one matching between developers and testers. In our view, this finding applies to the case of both agile, lifecycle, and other software development methodologies. Irrespective of the methodology used for software development, there is value to ensuring that testers know particular developers with whom they are working.

A final finding is that there is significant improvement in the quality of software developed resulting from the existence of a distinct unit for software testing within the software development organization. Our reading of the software development industry suggests that when agile development methodologies are used, there is seldom a distinct software testing unit in place. The specialization that is generally fostered by lifecycle development methodologies, however, does often result in the setting up of distinct software testing organizational units. Our study shows that this is prudent from the perspective of improving the quality of software produced.

Limitations and Suggestions for Future Research

This study has several limitations. First, all the three independent variables were measured by single items. This can be troublesome in survey research such as ours. Future research is encouraged to develop and validate multi-item scales for these constructs. For example, further refinement of the one-to-one matching construct between developers and testers is a viable area for new and follow-up studies. Second, software quality was measured by a survey of software developers and testers. It is understandable that there may be a difference between software quality perceived by developers and testers and that perceived by end users. Future research can focus on end users to measure the perceptions of software quality. Third, as mentioned before, this study did not deconstruct "agile methods" using a more refined breakdown. Future research needs to compare and contrast the influence of different agile methods (e.g., XP vs. Scrum). Furthermore, contrary to the majority of the literature, this study did not find that the use of agile methods provided any advantages over lifecycle methods in terms of software quality and job satisfaction. One possible explanation of this might be that agile methods are in the introduction stage and many developers and testers might lack training in agile methods. Future research needs to identify how well the participants know agile methods and how often they use it on the job. Future research may also want to focus on whether the positive benefits of one-to-one matching between developers and testers extends to methodologies used in situations where either the developer or tester works for an outsourcing partner.

CONCLUSIONS

This paper attempts to assess the influence of development methods, the existence of one-to-one matching between developers and testers, and the existence of a distinct corporate testing unit on software quality and job satisfaction. Our results suggest that development methods do not influence software quality or job satisfaction. The existence of one-to-one matching of developers and testers has a positive influence on both software quality and job satisfaction. The existence of a dedicated organizational unit for software testing also has a positive influence on the quality of software developed, but exerts no influence on job satisfaction.

Currently, lifecycle models are still widely used in large software development organizations; agile methods, however, are on the rise. Lifecycle models and agile methods are not mutually exclusive [29]. In practice, organizations often use a hybrid methodology, especially for the development of large, complex, integrated, and real-time systems. The results of this research deliver a clear message to software development organizations shifting their gears from lifecycle models to agile methods: development methodology does not matter, but the way you organize testers does.

REFERENCES

[1] Armstrong, J. S. and Overton, T. S. "Estimating nonresponse bias in mail surveys," *Journal of Marketing Research* (14:3), 1977, pp. 396-402.

[2] Aydin, M. N., Harmsen, F., van Slooten, K., and Stagwee, R. A. "On the adaptation of an agile information systems development method," *Journal of Database Management* (16:4), 2005, pp. 24-40.

[3] Barki, H. and Hartwick, J. "Interpersonal conflict and its management in information system development," *MIS Quarterly* (25:2), 2001, pp. 195-228.

[4] Beck, K. and Andres, C. *Extreme programming explained: Embrace change (2nd ed.)*, Addison-Wesley, Boston, Massachusetts, 2005.

[5] Craig, R. D. and Jaskiel, S. P. *Systematic software testing*, Artech House Publishers, Norwood, Massachusetts, 2002.

[6] Crispin, L. and Gregory, J. *Agile testing: A practical guide for testers and agile teams*, Addison-Wesley, Boston, Massachusetts, 2009.

[7] Copeland, L. *A practitioner's guide to software testing design*, Artech House Publishers, Norwood, Massachusetts, 2004.

[8] Dennis, A., Wixom, B. H., and Roth, R. M. *Systems analysis and design (3rd ed.)*, John Wiley & Sons, New York, New York, 2005.

[9] Erickson, J., Lyytinen K., and Siau, K. "Agile modeling, agile software development, and extreme programming: The state of research," *Journal of Database Management* (16:4), 2005, pp. 88-100.

[10] Gefen, D., Straub, D. W., and Boudreau, M. "Structural equation modeling techniques and regression: Guidelines for research practice," *Communications of the Association for Information Systems* (4:7), 2000, pp. 1-78.

[11] Grewal, H. and Maurer, F. "Scaling agile methodologies for developing a production accounting system for the oil & gas industry," *Proceedings of AGILE 2007*, Washington, D.C.: August 13-17, 2007, pp. 309-315.

[12] Henderson-Sellers, B. and Serour, M. K. "Creating a dual-agility method: The value of method engi-

neering," *Journal of Database Management* (16:4), 2005, pp. 1-23.

[13] Highsmith, J. and Cockburn, A. "Agile software development: The business of innovation," *IEEE Computer* (34:9), 2001, pp. 120-122.

[14] Hilkka, M.-R., Tuure, T., and Matti, R. "Is extreme programming just old wine in new bottles: A comparison of two cases," *Journal of Database Management* (16:4), 2005, pp. 41-61.

[15] Hutcheson, M. L. *Software testing fundamentals: Methods and metrics*, Wiley Publishing Inc., Indianapolis, Indiana, 2003.

[16] Issac, G., Rajendran, C., and Anantharaman, R. N. "Determinants of software quality: Customer's perspective," *TQM & Business Excellence* (14:9), 2003, pp.1053-1070.

[17] Janzen, D. and Saiedian, H. "Does test-driven development really improve software design quality?" *IEEE Software* (25:2), 2008, pp. 77-84.

[18] Larman, C. and Vodde, B. *Scaling lean & agile development: Thinking and organizational tools for large-scale scrum*, Addison-Wesley, Boston, Massachusetts, 2008.

[19] Lee, E. C. "Forming to performing: Transitioning large-scale project into agile," *Proceedings of AGILE 2008*, Toronto, Ontario, Canada: August 4-8, 2008, pp. 106-111.

[20] Martens, R. and Gat, I. "Wrestling with scaling software agility," *Software Development Times* (236: December 15), 2009, p. 27.

[21] McKeen, J. D. "Successful development strategies for business application systems," *MIS Quarterly* (7:3), 1983, pp. 47-56.

[22] Myers, G. J. *The art of software testing (2nd ed.)*, revised and updated by Badgett, T., Thomas, T. M, and Sandler, C., John Wiley & Sons, Hoboken, New Jersey, 2004.

[23] Ortega, M., Pérez, M., and Rojas, T. "Construction of a systemic quality model for evaluating a software product," *Software Quality Journal* (11:3), 2003, pp. 219-242.

[24] Page, A., Johnston, K., and Rollison, B. J. *How we test software at Microsoft*, Microsoft Press, Redmond, Washington, 2008.

[25] Pavlou, P. A. and Gefen, D. "Psychological contract violation in online marketplaces: Antecedents, consequences, and moderating role," *Information Systems Research* (16:4), 2005, pp. 372-399.

[26] Petter, S., Straub, D., and Rai, A. "Specifying formative constructs in information systems research," *MIS Quarterly* (31:4), 2007, pp. 623-656.

[27] Podsakoff, P. M. and Organ, D. W. "Self-reports in organizational research: Problems and Prospects," *Journal of Management* (12:4), 1986, pp. 531-544.

[28] Royce, W. W. "Managing the development of large software systems," *Proceedings of IEEE WESCON 1970*, Los Angeles, California: August 25-28, 1970, pp. 1-9.

[29] Sommerville, I. *Software engineering (8th ed.)*, Addison-Wesley, Boston, Massachusetts, 2007.

[30] Talby, D. and Dubinsky, Y. "Governance of an agile software project," *Proceedings of the 2009 ICSE Workshop on Software Development Governance*, Vancouver, British Columbia, Canada: May 16-24, 2009, pp. 40-45.

[31] Talby, D., Keren, A., Hazzan, O., and Dubinsky, Y. "Agile software testing in a large-scale project," *IEEE Software* (23:4), 2006, pp. 30-37.

[32] Turk, D., France, R., and Rumpe, B. "Assumptions underlying agile software-development processes," *Journal of Database Management* (16:4), 2005, pp. 62-87.

[33] Wright, T.A. and Cropanzano, R. "Emotional exhaustion as a predictor of job performance and voluntary turnover," *Journal of Applied Psychology* (83:3), 1998, pp. 486-493.

[34] Zhang, X., Dhaliwal, J.S., Gillenson, M.L., and Moeller, G. "Sources of conflict between developers and testers in software development," *Proceedings of 14th Americas Conference on Information Systems*, Toronto, Ontario, Canada: August 14-17, 2008, pp. 1-12.

[35] Zhang, X., Hu, T., Dai, H., and Li, X. "Software development methodologies, trends and implications: A testing centric view," *Information Technology Journal* (9:8), 2010, pp. 1747-1753.

AUTHOR BIOGRAPHIES

Xihui Zhang is an Assistant Professor of Computer Information Systems in the College of Business at the University of North Alabama. He earned a Ph.D. in Business Administration with a concentration on Management Information Systems from the University of Memphis. His teaching and research interests include technical, behavioral, and managerial aspects of Information Systems. He has published numerous articles in refereed journals and conference proceedings. He serves on the Editorial Review Board for several academic IS journals such as *Journal of Computer Information Systems* and *Journal of Information Technology Education*. Additional information about him can be found at http://sites.google.com/site/xihuizhang/.

Jasbir Dhaliwal is Professor of Information Systems and Associate Dean for Research and Academic Programs of the Fogelman College of Business and Economics at the University of Memphis. He also directs the Systems Testing Excellence Program at the FedEx Institute of Technology whose mandate is to advance the science of testing and to provide a stronger theoretical basis for industry best practices. He has a Ph.D. from the University of British Columbia, Canada and has published over fifty research papers in journals such as *Information Systems Research*, *Information & Management*, *IEEE Transactions on Engineering Management*, *International Journal of Production Economics*, *European Journal of Information Systems*, and in numerous conference proceedings.

Mark L. Gillenson is Professor of Management Information Systems in the Fogelman College of Business and Economics of the University of Memphis. He received his B.S. degree from Rensselaer Polytechnic Institute and his M.S. and Ph.D. degrees in Computer and Information Science from the Ohio State University. Dr. Gillenson worked for the IBM Corp. for 15 years and has consulted for major corporations and government organizations. Dr. Gillenson's research has appeared in *MIS Quarterly*, *Communications of the ACM*, *Information & Management*, and other leading journals. His latest book is *Fundamentals of Database Management Systems*, 2005, John Wiley & Sons.