https://doi.org/10.48009/2_iis_2007_273-279

METACOGNITION AND SOFTWARE DEVELOPER COMPETENCY: CONSTRUCT DEVELOPMENT AND EMPIRICAL VALIDATION

Paul J. Ambrose, University of Wisconsin - Whitewater, ambrosep@uww.edu

ABSTRACT

Software developer competence is essential for developing quality systems. Typically past experience, education and training, academic and professional references, tests, and interviews are used to assess developer competence. In this paper we propose that to obtain a holistic assessment of competence, it is essential to evaluate developer perceptions and beliefs on what they can achieve since these beliefs can impact their performance, independent of the skills possessed. Using social cognitive theory, we propose and develop a measure of developer self-efficacy, a metacognitive factor, to assess a critical facet of developer competence. We also empirically validate our self-efficacy measure through an experiment, and discuss the results of the findings.

Keywords: Self-efficacy, Self-awareness, Social Cognitive Theory, Programmer Competency, Partial Least Squares.

INTRODUCTION

Software development is a complex socio-technical undertaking and its success depends on four key factors – the development process, the technologies used, the people involved in its development and use, and realistic time and cost estimates [18, 21]. Software is an essential component of information systems (IS) and IS success is hence dependent on quality software [11, 12]. Research, systematic implementation of the best of the breed development practices, adoption of total quality management principles, and sophisticated technology have helped improve the quality of software developed over the past two decades [20]. However, IS continue to fail both pre- and post- implementation on account of software quality issues [1].

Software quality is two dimensional consisting of software process and product qualities, with process quality being the antecedent of product quality [20]. Substantial effort on enhancing process quality has been expended on improving requirements gathering, and analysis. From a research standpoint the implementation phase of the development process has received less attention, particularly from a behavioral perspective. Software is coded and tested during the implementation phase, and the quality of the product to emerge from the implementation phase is dependent on the quality of the code written and tested. However, even with sophisticated CASE tools, coding and testing are programming intensive activities, and hence software quality will depend on the programmers' competencies[i] [18].

Traditional measures of programmers' competencies include experience, and professional references for experienced programmers; training, transcripts, and academic references for novice programmers; professional certifications; and written, oral, and other demonstrative assessments during job interviews [2, 23]. While these measures serve as important indicators of an individual's competency, they do not however provide a holistic view of the required competency. To obtain a more complete depiction of competency, it is imperative to include an individual's metacognitive factors such as his/her thoughts, understanding, emotions, and intentions. Metacognitive factors influence behavior, and even among individuals with the same skill set, these factors can produce diverse behavioral patterns with differential performance outcomes. IS research however, has not addressed the measurement of metacognitive factors of a programmer that can influence behavior and performance.

This research addresses the measurement of programmers' cognitive competencies. Specifically, this research focuses on self-efficacy, a key metacognitive factor shown to impact behavior in other literature streams. While the unit of analysis is the programmer as the primary software developer, the concepts developed can be extended to other members of the development team as well. To provide focus to the research, the following explicit research questions are considered in this paper.

1. How can we evaluate a programmer's competency from a metacognitve perspective? Using social cognitive theory (SCT), we present self-efficacy as a measure of competency.

2. How can we establish the validity of the metacognitve measure of competency? We

establish a theoretical framework using SCT to test the validity of the competency construct.

3. What is the empirical evidence for the validity of this measure of competency? We undertake an experiment to establish empirical evidence for the construct.

The rest of the paper proceeds as follows. The next section theoretically develops and operationalizes programmers' self-efficacy (PSE) as a measure of programmers' competency. This section also provides the necessary theoretical background to establish the validity of the PSE construct. The section following presents the empirical study and its results. The paper concludes with a discussion, and directions for future research.

THEORETICAL DEVELOPMENT OF PSE

Social Cognitive Theory and IS Research

Bandura [3, 4] proposed social cognitive theory (SCT), whose essential premise is the existence of a dynamic interplay among an individual's cognitive (i.e., thoughts), affective (i.e., emotions), conative (i.e., intentions) factors, behavior or actions, and environment as shown in Figure 1. The social cognitive theory, originating in the field of social psychology, is well established as a popular theory to explicate human action, particularly to investigate how an individual's cognitive framework can guide behavior [26].

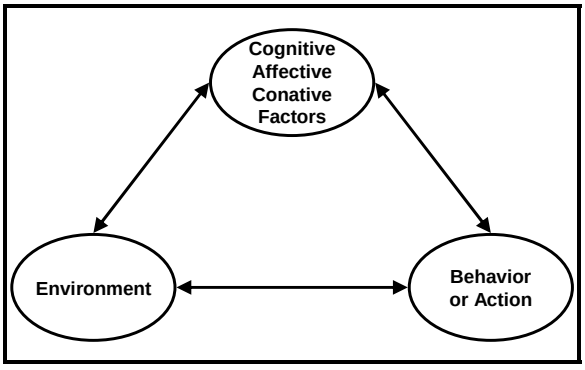

Figure 1: Bandura's SCT Theory

Specifically, SCT posits that an individual's belief structures, such as self-efficacy can guide behavior independent of the actual skills an individual possesses [4, 8]. Self-efficacy is defined by Bandura [4] as *"people's judgments of their capabilities to organize and execute courses of action required to attain designated types of performance. It is concerned not with the skills one has but with judgments of what one can do with whatever skills one possesses"*. Individuals who display low self-efficacy have been shown to exhibit behavior that inhibits high levels of performance that they are capable of given the skills they possess [8, 9]. On the other hand, individuals with high levels of self-efficacy regarding a task or activity generally are able to produce high levels of performance [8, 9]. The impact of self-efficacy on behavior and consequently on performance has been empirically proven in many diverse knowledge domains [16].

Seminal work on self-efficacy in the field of IS can be traced to Compeau and Higgins [7], who adapted the self-efficacy construct to measure individuals' judgments on their ability to use computers to assist their job related tasks. Compeau and Higgins [7] termed their adapted self-efficacy construct as computer self-efficacy (CSE). CSE is defined as individuals' beliefs about their abilities to competently use computers across multiple domains. CSE was hence conceptualized per Bandura's original definition, but was adapted to the IS context. CSE was operationalized, measured, and validated in a computer usage context by extending the measures for self-efficacy developed in social cognitive theory [7, 9].

Since Compeau and Higgins' [7, 9] adaptation of self-efficacy as CSE, it has been widely employed in IS research and has been used to evaluate user competence to use technology [e.g. see 15, 16, 19, 24, 25 for studies and detailed literature review on CSE research]. CSE is now firmly entrenched in IS research as a measure of end-user competency regarding computer usage [25, 26]. IS usage models and even the overarching IS success models now incorporate CSE as a construct of interest [26]. Further, other allied constructs from the SCT that are proximate determinants or consequents of CSE such as past performance, social persuasion, vicarious learning, outcome expectations, affect, and anxiety have been used in conjunction with CSE to estimate and validate user competency with respect to Information Systems/Information Technology (IS/IT), and even IS usage and success [16, 25]. However, self-efficacy has been limited to examining CSE in an end-user context and has not been widely used to investigate other behavioral interests that exist in the field of IS.

SCT continues to evolve in the field of social psychology, particularly into the realm of metacognition, which refers to an individual's process of thinking about thinking [13]. Zimmerman and Schunk [28], expanded Bandura's social

cognitive theory to identify eight dimensions of metacognition: self-efficacy, self-awareness, self-monitoring, self-motivation, goal, resourcefulness, choice, setting, and attribution. Zimmerman and Schunk [28] posit that self-efficacy can positively influence self-awareness (self-assessment of knowledge), and emphasize the importance of this linkage in understanding the impact of individual self-efficacy beliefs on behavior and performance. Strong self-efficacy beliefs can motivate individuals to participate in training and learning experiences, which consequently increase their awareness of what they know and do not know. High levels of self-efficacy can thus lead to an accurate estimation of one's self-awareness [13].

Conceptualizing PSE

This research is guided by the tenets of SCT. We propose that assessing a programmer's job related competencies from merely observing behavior, or through skills assessment as discussed in the introduction will not produce a complete profile of the individual's competency for a programming job. An individual's cognitive, affective, and conative factors can deter or enhance a required job related behavior, and this can be independent of the skill set the individual possesses. Empirical tests of SCT across diverse domains have shown that self-efficacy measurements can indicate an individual's competency [16]. Further, an understanding of an individual's self-efficacy beliefs can provide intervention opportunities such as encouragement, training, or even persuasion to change if the belief levels are low. It can also help identify high achievers and channel their competencies to enhance productivity.

Hence we propose the measurement of the self-efficacy of a programmer with respect to his/her programming task as another key measure of competency. Similar to the CSE research stream in IS, we adapt self-efficacy from SCT to introduce programmer's self-efficacy, and define programmer's self-efficacy or PSE as *the extent of individual programmers' beliefs about their abilities to competently use a programming technique across multiple problem domains*.

Establishing the Validity of PSE

The development and validation of the PSE construct were done along the lines suggested by Churchill, and Sethi and King [6, 22]. The first step of construct development involves specifying its domain and definition. This was done in the earlier section where we conceptualized and defined PSE. Second, the construct needs to be operationalized by developing its measures. The recommended practice is to survey existing literature for items that have been previously used for the construct and adopt these measures after adjusting for context. Self-efficacy measures are well established, and we adapted the 9 commonly used measures of CSE after adjusting the measure for the programming domain. The PSE measures are shown in Table 1. The third requirement is the establishment of the face validity of the measures, and this was done by validating the measures with two experts. Fourth, the validity of the construct needs to be ascertained, to ensure it measures what it is supposed to measure. We chose to examine validity by testing PSE empirically for its predictive validity.

Table 1: PSE and Self-awareness measures

Programmers' Self-Efficacy Measures *10-point scale anchored at 'Not at all Confident' and 'Totally Confident'*
I could complete a programming task using object-oriented programming technique : 1. if there was no one around to tell me what to do as I program. 2. if I had only programming texts for reference. 3. if I had seen someone else do it before trying it myself. 4. if I could call someone for help if I got stuck. 5. if someone else helped me get started. 6. if I had a lot of time to complete the job. 7. if I had just the Visual Studio built-in help facility for assistance. 8. if someone showed me how to do it first. 9. if I had just the Internet for assistance.
Self-Awareness Measure *7-point scale anchored at 'Very limited knowledge' and 'Complete Knowledge'*
1. How thorough is your current knowledge of the object-oriented programming technique?

Testing for predictive validity first requires a theoretically identified dependent construct for PSE. Then if it can be empirically established that PSE is a valid predictor of this dependent construct, then the predictive validity of PSE is established. We use *self-awareness*, or *self-assessment of knowledge* identified earlier as the dependent construct for our validation. Self-awareness is defined as *the extent of knowledge possessed by individual programmers on a programming technique*, self reported by the

individuals. Again this construct was adapted to the programming domain, and its single item measure modified from the CSE literature stream. The relationship between self-efficacy and self-awareness is shown in Figure 2. The empirical validation of PSE is discussed in the next section.

EMPIRICAL VALIDATION OF PSE

Research Design and Data Collection

The empirical validation was done through an experiment. 60 seniors enrolled in an undergraduate MIS course on objected oriented systems development using C++ voluntarily participated in the study. All students had completed courses in C++ and Visual Basic prior to enrolling in this class. This experiment was conducted towards the end of the semester when the students had a greater depth of understanding of object-oriented programming using C++. The mean age of the participants was 25.9 years, and 22 were female and 38 were male. The participants were graduating that semester, or the following, and hence closely resembled entry level or novice programmers in organizations.

The student programmers were required to write a game of chance involving a deck of card. The program should generate, shuffle, and deal a deck of cards. The game is played by placing a bet that a player guessed card will be among the top 10 cards of a shuffled deck. The game ends if the player chooses to walk away with money earned, or if all money is lost. The appropriateness of the programming task and the program requirements specification sheet were validated with the two experts who had validated our measurement items. The measurement instrument containing PSE and self-awareness measures were administered upon completion of the programming task.

Data Analysis and Results

Data collected were examined for missing, implausible, and coding errors, and for departures from normality prior to formal analysis. No discrepancies were found. Next, the multi-item PSE construct was tested for reliability and unidimensionality. Cronbach's alpha measure of reliability of PSE was 0.74, and above the minimum requirement of 0.70 [10]. A principle components analysis (PCA) of PSE measures indicated a single factor structure with high factor loadings, indicating the unidimensionality of PSE [14].

Further, convergent validity, which establishes that a single construct underlies a set of measurement items, was established by analyzing the measurement model using EQS 6.1 as follows. First, the degree of association between PSE and its measurement items were ascertained to be significant. Second, composite reliability [27] was computed from standardized factor loadings and was found to be greater than 0.70. Finally, average variance extracted by PSE computed using standardized factor loadings was greater than 0.50 indicating that variance captured by PSE was greater than that attributable to error. The results are summarized in Table 2.

Table 2: Measurement Model Properties

Unidimensionality & Reliability			
Factor Loadings (Range)	**Eigen Values**	**Variance Extracted**	**Cronbach's Alpha**
0.736-0.918	6.31	70.5%	0.74
Convergent Validity			
Factor Loadings t Values	**Composite Reliabilities**		**Average Variance Extracted**
4.66-6.51	0.94		65.6%

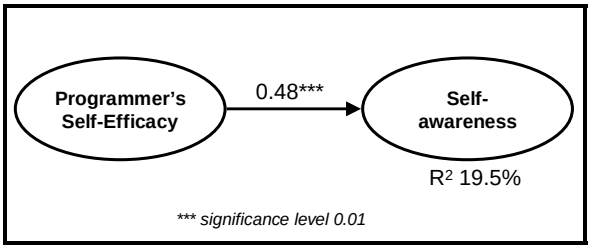

Figure 2: Research Model for PSE Validation

The structural model was analyzed using partial least squares (PLS), a second generation variance based structural equation modeling technique typically used for theory building [5]. The bootstrapping technique with 30 bootstrap samples was used to estimate the significance of all model parameters. PSE was modeled as a reflective construct as specified by theory. PLS analysis indicated a strong significant relationship between PSE and self-awareness in the theoretically proposed direction. In addition, the R^2, which indicates the variance of self-awareness explained by PSE, was 19.5%. This is acceptable given that there can be other predictors of self-awareness too. The results are summarized in Figure 2. The PLS analysis indicates the predictive validity of the PSE construct.

DISCUSSION AND CONCLUSION

We argued for a holistic measure for assessing a programmer's competency by including self-efficacy, a metacognitive measure of competency. Using social cognitive theory developed in social psychology research, and its extensions in IS literature, we conceptually developed the PSE construct. We operationalized the PSE construct, and also empirically validated the construct using a theoretically substantiated predictive validity framework. We provide additional insights into the use of this construct.

Marcolin et at [17] argue that a portfolio of measures be developed for CSE or end-user computer competency. Their portfolio calls for measures to be developed from 1) different measurement perspectives (e.g. self-reported, pen-and-paper), 2) across multiple contexts within the problem domain, and 3) across cognitive, affective, and skills factors. We had two measurement perspectives, self-reported covering self-efficacy and self-awareness, and a variation of pen-and-paper, which is the actual work produced by the participants of the experiment. However, we did not include the pen-and-paper version of the measurement in our research as this research is focused specifically on developing a metacognitive measure of programmer's competence. Also, for scope reasons we did not test our PSE construct across multiple contexts, but defer that exercise to future research. With self-efficacy beliefs consistently emerging as a strong predictor of behavior we restrict our cognitive competency measure to this factor.

So how can we use the knowledge of an individual's PSE beliefs to enhance software quality? The immediate response may be to hire only programmers with high PSE beliefs. However, we contend that this may not be the correct approach, but instead this should be use in conjunction with the actual skills possessed by a programmer. Individuals with good programming skills are not easily available, but if their self-efficacy levels are low, then the quality of software developed by these programmers can be compromised. But if the self-efficacy beliefs can be enhanced particularly for skilled programmers, then their job related actions and behaviors can enhance the quality of software produced. Social cognitive theory provides key self-efficacy antecedents, and we discuss these now, and suggest that they be managed to enhance self-efficacy beliefs.

Among several antecedents of self-efficacy identified in SCT, verbal persuasion (by a credible mentor/teacher), vicarious learning (social comparison by observing someone performing similar tasks), enactive mastery (prior success or failure), emotional arousal (e.g. fear, distress) degree/quality of feedback, and perceived effort can all enhance or decrease self-efficacy beliefs. [16].

Enactive mastery, where prior good/bad experience or performance can enhance/diminish self-efficacy beliefs of individuals is a key antecedent of self-efficacy [8]. In cases where a skilled programmer's self-efficacy beliefs are low on account of some past failure, it will be prudent to retrain or reequip the programmer with the necessary technical and behavioral skill set to increase self-efficacy. Also, constructive feedback can help the programmer regain lost self-efficacy beliefs, as the degree and quality of feedback are key antecedents of self-efficacy.

Further, belief and attitudinal changes can be brought about through social or verbal persuasions by peers or superiors whose opinions are well respected [8]. Such individuals can help programmers with low self-efficacy to change their belief structures. Interventions through such respected individuals or even social groups can hence have positive performance related outcomes through positive self-efficacy beliefs. Such verbal influences can also help surmount perceived effort barrier, particularly when the perceived effort is greater than the actual effort it takes to do the task.

Individuals are good mimics of their social peers. The thought that "I can do anything, she can do" is the underlying principle behind vicarious learning. As a result, if an individual sees a peer succeed at a particular task, his/her self efficacy will rise, and a peer's failure can reduce self-efficacy [8]. Programmers with low self-efficacy beliefs can be embedded with highly successful programmers to raise their perceptions of their self-efficacy. Also, it is useful to maintain work environments with individuals who are successful, so as to empower everyone with a belief that he/she can do the task as well those who are successful.

Finally, emotional arousal refers to psychological factors that can inhibit performance. For e.g., fear of speaking, or butterflies in the stomach can be a stumbling block even to a speaker who has great thoughts. Similar fear exists among individuals with respect to their task. Fear of learning a new tool or reorienting to a new programming environment can hinder programmers. Such emotional factors need to

be addressed through regular training and development programs.

In conclusion, we reiterate the importance of measuring attitudes and beliefs in addition to observable measures of programmer's competence. While we developed the PSE construct specifically in the object oriented programming context, we recommend that further research be undertaken in other programming contexts. Competence in other developmental activities, such as requirements gathering, analysis and modeling can also be similarly assessed. Further, such a measure can be extended to assessing competency even at the development methodology (e.g. agile, traditional) level. Empirical validation of the construct can also be carried out using real world developers with experience, moving beyond novice programmers.

REFERENCES

1. Agrawal, M. & Chari, K. (2007). Software Effort, Quality, and Cycle Time: A Study of CMM Level 5 Projects. IEEE Transactions on Software Engineering, 33,(3), 145-156.
2. Arnold, D. & Niederman, F. (2001). IT The Global Work Force. Communications of the ACM, 44,(7), 30-33.
3. Bandura, A. (1977). Self Efficacy: Toward a unifying theory of behavioral change. Psychological Review, 84,(2), 191-215.
4. Bandura, A. (1986). Self Efficacy, Social foundations for Thought and Action: A Social Cognitive Theory, Englewood Cliffs, NJ: Prentice-Hall.
5. Chin, W. W., Marcolin, B. L., & Newsted, P. R. (2003). A Partial Least Squares Latent Variable Modeling Approach for Measuring Interaction Effects: Results from a Monte Carlo Simulation Study and an Electronic-Mail Emotion/Adoption Study. Information Systems Research, 14,(2), 189-217.
6. Churchill, G. A. (1979). A Paradigm for Developing Better Measures of Marketing Constructs. Journal of Marketing Research, 16,(February), 64-73.
7. Compeau, D. R. & Higgins, C. A. (1991). A Social Cognitive Theory Perspective on Individual Reactions to Computing Technology. Proceedings of 12th International Conference on Information Systems, New York, NY, ACM, 187-198.
8. Compeau, D. R. & Higgins, C. A. (1995). Application of Social Cognitive Theory to Training for Computer Skills. Information Systems Research, 6,(2), 118-143.
9. Compeau, D. R. & Higgins, C. A. (1995). Computer self-efficacy: Development of a measure and initial test. MIS Quarterly, 19,(2), 189.
10. Cronbach, L. J., "Test Validation," in R. L. Thorndike, ed., Educational Measurement, Washington, DC: American Council on Education, 1971.
11. DeLone, W. H. & McLean, E. R. (1992). Information Systems Success: The Quest for the Dependent Variable. Information Systems Research, 3,(1), 60-95.
12. DeLone, W. H. & McLean, E. R. (2003). The DeLone and McLean Model of Information Systems Success: A Ten-Year Update. Journal of Management Information Systems, 19,(4), 9-30.
13. Gravill, J. I., Compeau, D. R., & Marcolin, B. L. (2006). Experience effects on the accuracy of self-assessed user competence. Information & Management, 43,(3), 378-394.
14. Hair, J. F., Black, W. C., Babin, B., Anderson, R. E., & Tatham, R. L. (2005). Multivariate data analysis, 6th ed., Upper Saddle River, N.J.: Prentice Hall.
15. Hasan, B. (2006). Effectiveness of Computer Training: The Role of Multilevel Computer Self-Efficacy. Journal of Organizational & End User Computing, 18,(1), 50-68.
16. Marakas, G. M., Yi, M. Y., & Johnson, R. D. (1998). The Multilevel and Multifaceted Character of Computer Self-Efficacy: Toward Clarification of the Construct and an Integrative Framework for Research. Information Systems Research, 9,(2), 126-163.
17. Marcolin, B. L., Compeau, D. R., Munro, M. C., & Huff, S. L. (2000). Assessing User Competence: Conceptualization and Measurement. Information Systems Research, 11,(1), 37.
18. Nerur, S. & Balijepally, V. (2007). Theoretical Reflections on Agile Development Methodologies. Communications of the ACM, 50,(3), 79-83.
19. Piccoli, G., Ahmad, R., & Ives, B. (2001). Web-Based Virtual Learning Environments: A Research Framework and a Preliminary Assessment of Effectiveness in Basic IT Skills Training. MIS Quarterly, 25,(4), 401-426.
20. Ravichandran, T. & Rai, A. (2000). Quality Management in Systems Development: An Organizational System Perspective. MIS Quarterly, 24,(3), 381-415.
21. Ravichandran, T. & Rai, A. (2003). Structural Analysis of the Impact of Knowledge Creation and Knowledge Embedding on Software

Process Capability. IEEE Transactions on Engineering Management, 50,(3), 270-284.
22. Sethi, V. & King, W. R. (1991). Construct Measurement in Information Systems Research: An Illustration in Strategic Systems. Decision Sciences, 22,(3), 455-472.
23. Surakka, S. (2007). What Subjects and Skills are Important for Software Developers? Communications of the ACM, 50,(1), 73-78.
24. Thatcher, J. B. & Perrewe, P. L. (2002). An Empirical Examination of Individual Traits as Antecedents to Computer Anxiety and Computer Self-Efficacy. MIS Quarterly, 26,(4), 381-396.
25. Thompson, R., Compeau, D., & Higgins, C. (2006). Intentions to Use Information Technologies: An Integrative Model. Journal of Organizational & End User Computing, 18,(3), 25-46.
26. Venkatesh, V., Morris, M. G., Davis, G. B., & Davis, F. D. (2003). User Acceptance of Information Technology: Toward A Unified View. MIS Quarterly, 27,(3), 425-478.
27. Werts, C. E., Linn, R. L., & Jöreskog, K. G. (1974). Interclass Reliability Estimates: Testing Structural Assumptions. Educational Psychological Measurement, 34, 25-33.
28. Zimmerman, B. J. & Schunk, D. H. (2001). Self-Regulated Learning and Academic Achievement, Mahwah, New Jersey: Lawrence Erlbaum Associates.

[i] The rest of the paper focuses on programmer's competency. While the term developers can include programmers, it can also include individuals who perform a variety of other development activities. To keep our focus on the programming activity, we measure programmer's competence.