

# Positive Affects Inducer on Software Quality

Sharifah Lailee Syed-Abdullah

Faculty of Computer Science and Mathematics, Universiti Teknologi MARA

02600, Arau, Perlis. Malaysia

Tel: 60-4-987-5076 E-mail: shlailee@perlis.uitm.edu.my

Mazni Omar

Faculty of Computer Science and Mathematics, Universiti Teknologi MARA

02600, Arau, Perlis. Malaysia

Tel: 60-4-928-4735 E-mail: mazni@isiswa.uitm.edu.my

Mohd Nasir Abdul Hamid

Centre for Islamic Thought and Understanding (CITU), Universiti Teknologi MARA

02600, Arau, Perlis. Malaysia

Tel: 60-4-987-4702 E-mail: mdnasir@perlis.uitm.edu.my

Che Latifah bt Ismail

Centre for Islamic Thought and Understanding (CITU), Universiti Teknologi MARA

Perlis 02600, Arau, Perlis. Malaysia

Tel: 60-4-987-5050 E-mail: chelatifah@perlis.uitm.edu.my

Kamaruzaman Jusoff (Corresponding author)

Faculty of Forestry, Universiti Putra Malaysia

43400, Serdang, Selangor. Malaysia

Tel: 60-3-8946-7176 E-mail: kamaruz@putra.upm.edu.my

# Abstract

This paper presents an early empirical study on an agile methodology (Extreme Programming) using Positive Affect metric. The question of interest is whether an agile methodology has any distinct outcome on the positive affectivity of the software developers. And whether these affects will contribute to the quality of software produced. Quantitative methods were utilized, including participative observation and simple statistical tests such as Spearman Correlation and Mann-Whitney test. The results showed that Extreme Programming has positive affectivity which leads to the increase in software quality. This study suggests that when people experience joy and mild contentment, they are more likely to be more creative over wider range of problems, become more resilient over time and are more likely to develop long-term plans and goals.

Keywords: Agile methodology, Empirical study, XP, Positive affect, Software quality

# 1. Introduction

The traditional methodologies imposed a disciplined process upon software development, with the aim of making software development more efficient in order to produce better quality systems. The detailed process places a strong emphasis on planning and was inspired by other engineering disciplines. The most frequent criticism of these methodologies is that they are bureaucratic thus slowing the development process. The second problem with these methodologies is that the requirements specifications are not flexible. In reality, it is difficult to get the software customer to identify their requirements. Even if the requirements can be identified, the business world is forever changing. As a reaction to these problems, agile methodologies evolved. Agile methodologies welcome change and unpredictability because they are more adaptive than predictive, and more people-oriented than process-oriented. Adaptive approaches are better when the requirements are uncertain or dynamic as in the new type of software application being developed nowadays. When faced with unpredictable user requirements and changes that must be

accommodated during software-in-progress development, developers often experienced stressful emotions such as anxiety and depression (Sharifah Lailee, Holcombe, & Gheorge, 2006a, 2006b).

# 1.1 Agile Methodology

Due to rapid demand of technological changes, agile methodologies have emerged to alleviate the uncertainty of business requirements. The need to deliver quality software at economical cost is the main issue in software industry. Therefore, the Agile Alliance (2001) has expressed the values in Agile Manifesto as:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

## Responding to change over following a plan

The major agile methodologies consist of Scrum, Dynamic Systems Development Method (DSDM), Crystal Methods, Feature Driven Development (FDD), Lean Development and Extreme Programming (XP). Based on the agile manifesto, people and communication are the key ingredients towards producing quality software. Agile employs a lightweight process, which communication plays an important role over comprehensive documentation. This method focuses more on people-oriented approach, which relies on tacit or interpersonal knowledge whilst developing software. Study on agility level of these methodologies (Qumer & Henderson-Sellers, 2008) discovered that Crystal methodology has the highest degree of agility followed by XP, Scrum, and DSDM. Agility characterization were based on flexibility, speed, leanness, learning and responsive features that were calculated quantitatively.

The creators of Agile Alliance agreed that detailed project tactic should be continuously innovated in order to discover a larger set of agile software practices (Cockburn, 2007). Therefore, it is not surprising to discover different set of practices that is similar and complementary to each other. In this paper, four agile methodologies, namely XP, Scrum, Crystal and DSDM are compared according to its principles, methods and tools.

According to Strode (2005), all of the agile methodologies highlight iterative, incremental, and rapid software delivery to deal with flexible requirements in addition to active participation between the team and stakeholders. Except for Scrum and DSDM methodologies that that cater for large projects, the other methodologies are more suitable for small and medium scale project.

Agile modeling (AM) is a practice-based methodology for effective software modeling. AM is not a prescriptive process but focuses more on a portion of an overall software process needed by other agile methodologies. Most agile methods implicitly defined its model to be used, thereby indicating that working software is imperative compared to documentation. However, it is also possible to use object-oriented modeling such as Unified Modeling Language (UML) and software tools to speed up development process. These tools are needed to support the main activities such as requirements management, coding, configuration and testing activities. Each agile methodology serves different purposes and has its own principles and methods, even though they share common characteristics of agile values. Table 1 show the comparison made according to principles, methods, and tools.

For the purpose of this study, a comparison between an agile methodology (XP) and formal methodology (RUP) was conducted. The study focus on the agility of each software methodology to induce positive affect on each member of SE teams and how these affects contribute to the level of software quality.

# 1.2 Extreme Programming

Beck (2000) introduces XP as a solution to the problems encountered by the formal methods. The XP methodology was created to address requirement changes and project risk. XP begins with 4 values; Communication, Simplicity, Feedback and Courage. It then builds up to practices that XP projects should follow. In XP, features that provide the most business value to the customer must be developed first because the real goal of this approach is to deliver the software that is needed when it is needed. Requirements are written as user stories, which are chunks of functionality that are valuable to the customers. Chunking is a technique in cognitive learning strategy that allows information to be broken down into smaller and meaningful collection of knowledge. Through the use of story cards, it is easier for developers to group different stories according to main functions. Chunking assist developers because human has limited memory capacity and often have difficulty to memorize a large amount of functions or information (Mazni, Sharifah Lailee, & Holcombe, 2009). XP encourages communication by having developers collectively own all codes and work in pairs. Collective code ownership considers code as belonging to the team and not to the individual developers, thereby encourages every developer to contribute new ideas to all segments of the project. Pair programming requires two developers to sit side by side in front of a computer. One person types and thinks tactically about the methods being created, while the other thinks strategically about how the methods fit into the class. Pair programming changes the environment from criticism and competition to learning and cooperation thus improving group cohesiveness and communication. Complex requirements must be simplified to enhance understanding between team members. XP simple design evolves through constant refactoring, which is guided by suitable metaphor and implemented in accordance to common coding standards. Fowler, Beck, Brant, Opdyke, & Roberts (1999) defined refactoring as the removal of redundant or unused functionality and the restructuring of obsolete designs in order to improve smelly codes. Tong (2004) considered too many comments as useless and further suggested refactoring of these comments into codes. System metaphor is a narrative that everyone (customer, programmer and managers) can associate with when discussing new functionality. The reason for using a metaphor is to achieve a common vision and shared vocabulary. On-site customer is a practice which requires the customer to sit with the development team on a full-time basis. Holcombe (2002) is more realistic by suggesting regular visits and meetings at both the development site and the business site. The humanistic aspect of the communication and the simplicity aspect promote good teamwork because it is an important towards developing quality software.

## 1.3 Rational Unified Process

Rational Unified Process (RUP) is a software development methodology originated by Rational Software, IBM (Dennis, Wixom, & Tegarden, 2005; Runeson & Greberg, 2004). RUP methodology claimed to be a heavyweight methodology due to its demands to produce extensive documentation. This approach is a use case driven and requires tool to support the software development activities. Unlike agile (XP), RUP emphasizes on specific roles that tailored for a particular project. Examples of these roles are team leader, programmer, user interface designer, software tester and quality assurance. RUP has four distinct phases, which are inception, elaboration, construction and transition phase (Bennet, McRobb, & Farmer, 2006). Although RUP has formal phases, it allows software development teams to design and develop software in iterative and incremental manner.

#### 1.4 Positive affect

Past research has shown that a positive affect induction leads to a greater cognitive flexibility and facilitates creative problem solving across a broad range of settings. Research works by Carnevale & Isen (1986), Aspinwall (1998), Ashby, Isen, & Turken (1999) and Isen (2001), suggest that positive affect increases a person's ability to organize ideas in multiple ways, to access alternative perspectives and also to improve performance in several tasks that are typically used as indicators of creativity or innovative problem solving. In a study of the role of affect on human life, Norman and colleagues (Norman, Ortony, & Russell, 2003) show that affect makes humans smart because affect is always passing judgments and presenting them with immediate information about the world. The affective signals work through neurochemicals, bathing the relevant brain centres and changing the way humans perceive, decide, and react. These neurochemicals change the parameters of thought, adjusting such things as whether reason is primarily 'depth first' (focused, not easily distracted) or 'breadth first' (creative, out of the box thinking, but easily distractible).

It is the intention of this paper to discuss Extreme Programming (XP) as a positive affect inducer and to discuss the findings of the possible impact of the selected XP practices on the software quality. To achieve this, a comparison study was conducted on the software engineering teams consisted of third year students at University Utara Malaysia (UUM). Findings revealed that the XP methodology does have an impact on the positive affectivity and level of software quality.

## 2. Method

A replicated study (Sharifah Lailee et al., 2006b) were carried out in UUM, to determine empirically, whether teams using XP methodology would experience higher positive affectivity than the teams using the Rational Unified Process (RUP) approach. To measure the developers' state of the positive affect, the positive affect scale of the Positive and Negative Affect Schedule (PANAS) was used. Positive affect was induced by introducing and requiring the XP methodology to be used by half of the development teams. The studies do not include the negative affect because previous research has shown that positive affect can operated as a single construct, indicating that the fluctuation of the positive affectivity, has no effect on the negative affectivity of a person (Anderson & Thompson, 2004).

The validity and reliability of PANAS scale has been demonstrated by other studies (Watson & Clerk, 1997; Watson, Pennebaker, & Folger, 1987; Watson & Tellegen, 1985). The Positive Affect scale showed a satisfactory internal consistency coefficient, Cronbach alpha = 0.78 during the first reading (Week 2), Cronbach alpha = 0.89 during the second reading (Week 6) and Cronbach alpha = 0.87 during the third reading (Week 15). At the beginning of the study, Independent sample t-test was used to compare the total mean score for Positive Affect variables and the result showed no significant difference between Formal teams [N=28, Mean Score (M) = 34.12, Standard Deviation (SD)=4.77] and Agile (XP) teams [N=30, Mean Score (M) = 34.53, Standard Deviation (SD)=4.41].

#### 3. Results and discussions

Statistical test Mixed between-within ANOVA was conducted. The analysis indicated that there is no significant difference between the three intervals; Reading 1 (Week 2), Reading 2 (Week 6) and Reading 3 (Week 15) for both methodologies; Formal (N=28,  $M_1$ = 34.12 SD<sub>1</sub> = 4.77; N=28  $M_2$  =33.61, SD<sub>2</sub> = 5.07; N=28  $M_3$  =33.86, SD<sub>3</sub> = 4.57) and Agile (N=30,  $M_1$ =34.53, SD<sub>1</sub> = 4.41; N=30  $M_2$  = 34.17, SD<sub>2</sub> =5.90; N=30  $M_3$  =35.37, SD<sub>3</sub> = 6.37) (see Figure 1and

Table 2). This may due to the small effect size (eta squared =0.010). Besides, the results may be moderated by others factors such as partial adoption of Agile (XP) practices during this study. This finding supported earlier finding on the positive effect of Extreme Programming on SE teams (Sharifah Lailee, Holcombe, Karn, Cowling, & Gheorge, 2005).

At the end of Week 15, each information system projects were graded by teams of evaluators which consisted of project client and a lecturer. The mean scores awarded by both evaluators were assessed. The following graph shows grades achieved by both teams according to the projects. The Mann-Whitney non parametric statistical test was used to compare the mean scores and the results showed significant differences in the mean scores for the Formal teams [M=21.09, SD=2.91] and the Agile (XP) teams [M=23.96, SD=1.31]. The graph indicates that teams using Agile (XP) approach were awarded higher score than Formal (RUP) teams (see Figure 2).

In this study, XP methodology was chosen as a positive inducer because of the existence of several XP practices that warrant feedback to the developers. Positive feedback about one's performance has been known as a positive affect inducer (Estrada, Isen, & Young, 1997; Sharifah Lailee et al., 2006a). The XP practices associated with feedback seeking are simple design, pair programming, continuous testing, continuous integration and frequent review (release).

Studies by Aspinwall (1998) and Muraven, Tice, & Baumeister (1998) noted that people must have a surplus of resources such as time, energy and attention to engage in a proactive behaviour. Using XP approach, the developers experienced a surplus of time during coding because less time was engaged in the designing phase. Using simplified design XP developers are actually releasing the stressful task of creation, thus liberating the mind to be more creative and innovative. By reducing the technical aspect of design, the mind was able to approach the problem solving task through a breadth first approach. Design is only an early manifestation of ideas, whereas the coding process allows the developers to realize their idea in a more concrete way. This approach is considered as a positive affect inducer because it allows feedback on the design through the programming code. The ability to see the results and identify the flaws in the design allows the developers to be more creative in the next part of the system. This is the reason why simple design can accommodate flexible requirements because the process of creating part of the system in this manner allows developers to be more innovative in the problem solving process. It was observed that the practice of pair programming started with initial socializing amongst the pair thus creating a positive mood before any formal programming commenced. The ability to discuss the advantages and disadvantages of certain coding ideas enables the pair to seek improvements and avoid specific weaknesses. Studies on pair programming have provided the evidence about the benefits of pair programming (Coman, Sillitti, & Succi, 2008; Succi, Marchesi, Pedrycz, & Williams, 2002; Williams, 2002). With pair programming practice, positive affect is induced through early socializing, more attention and immediate feedback amongst the pair.

Continuous testing allows feedback on the developed code. In the normal software testing domain, testing is usually left at the end of the development cycle thus leaving a very mood which is experienced and the attention of the two developers allow the pair to engage in a more short time for complete testing. In this situation, often the developers were faced with products that have too many defects, as the bugs were discovered too late. The benefit of testing as the software is developed is that the developers are always certain that the software developed is always test compliant. Continuous testing is a practice that is structured so that different levels of testing can be conducted as the solution is being built. In the study by Trope & Pomerantz (1998), participants in whom positive affect has been induced showed greater interest in the part of the test they had failed than did neutral mood participants. The emphasis of the continuous testing enables the developers to feel more confident about the correctness of the code and therefore bolster their confidence and self-esteem.

Continuous integration is another feedback seeking practice, which allows the developers to address performance problems earlier in the development process. The more frequently the developers were able to test the integrated system, the more often they were able to check the functional integrity of the application as some problems do not manifest themselves until they are in the integration environment, such as when a database application is finally tested in a genuine load. The ability to address the performance problems early and to continuously improve the system allows the developers to enjoy a level of self regard or positive affect. Developers using this practice had the advantage of improving their self-esteem continuously, as they worked to perfect the functionality of the integrated system. Frequent release (review) is another practice that commands feedback. Feedback from the client be it positive or negative, is also a positive affect inducer. Accumulating evidences suggest that positive affect can create an increased interest in information about one's liabilities. A study by Trope & Neter (1994) has shown that prior positive experience subsequently increased the interest in feedback of high rather than low self-relevance, even when the feedback was expected to diagnose weaknesses rather than strengths.

## 4. Conclusion

In this study, even though there was no significant difference in positive affect between the methodologies, it is interesting to observe the impact of the XP on the positive affectivity of the developers which result in higher score being awarded to the agile teams. The teams using a more flexible approach, such as the XP methodology, were able to

incorporate the constant changes made by the clients and thus able increase their positive mood. When a person experiences a positive affect, they show a greater preference for a larger variety of actions and are able to see and think of more possibilities and options to solve whatever problem is faced. People with a positive affect are more likely to take action because they are proactive. This study suggests that when people experience joy and mild contentment, they are more likely to think of a wider range of actions, become more resilient over time and are more likely to develop long-term plans and goals.

### Acknowledgement

The authors would like to express our appreciation to Software Engineering Project 1 course lecturers, from Universiti Utara Malaysia, Dr Azman Yasin and Dr Haslina Mohamad who have given their support towards the completion of this research. In addition, we would like to thanks all clients, supervisors and students in this study.

# References

Alliance, A. (2001). *Manifesto for Agile Software Development*. Retrieved 31 March, 2009, from http://www.agilemanifesto.org

Anderson, C., & Thompson, L. L. (2004). Affect from top down: How powerful individuals' positive affect shapes negotiation. *Oranizational Behavior and Human Decision Processes*, 95, 125-139.

Ashby, F. G., Isen, A. M., & Turken, A. U. (1999). A neuropsychological theory of positive affect and its influence on cognition. *Psychological Review*, *106*, 529-550.

Aspinwall, L. G. (1998). Rethinking the Role of Positive Affect in Self-Regulation. *Motivation and Emotion*, 22(1), 1-32.

Beck, K. (2000). Extreme Programming Explained: Embrace Change. USA: Addison-Wesley.

Bennet, S., McRobb, S., & Farmer, R. (2006). *Object-oriented systems analysis and design using UML*. Berkshire: McGraw-Hill.

Carnevale, P. J. D., & Isen, A. M. (1986). The influence of positive affect and visual access on the discovery of integrative solutions in bilateral negotiation. *Organizational Behaviour and Human Decision Processes*, 37, 1-13.

Cockburn, A. (2007). Agile Software Development: Cooperative Game (2nd Ed.). USA: Addison-Wesley.

Coman, I. D., Sillitti, A., & Succi, G. (2008). Investigating the Usefulness of Pair-Programming in a Mature Agile Team. In P. Abrahamsson, R. Baskerville, K. Conboy, B. Fitzgerald, L. Morgan & X. Wang (Eds.), *Lecture Notes in Business Information Processing*. Berlin-Heidelberg: Springer-Verlag.

Dennis, A., Wixom, B. H., & Tegarden, D. (2005). Systems analysis and design with UML version 2.0: an object -oriented approach with UML (2nd Ed.). Hoboken, NJ: John Wiley and Sons.

Estrada, C. A., Isen, A. M., & Young, M. J. (1997). Positive Affects Facilitates Integration of Information and Decreases Anchoring in Reasoning among Physicians. *Organizational Behaviour and Human Decision Processes*, 72(1), 117-135.

Fowler, M., Beck, K., Brant, J., Opdyke, W., & Roberts, D. (1999). *Refactoring: Improving the Design of Existing Code*. USA: Addison-Wesley.

Holcombe, M. (2002). Extreme Programming for Real: A disciplined, agile approach to software engineering. Sheffield: University of Sheffield.

Isen, A. M. (2001). An influence of positive affect on decision making in complex situations: Theoritical Issues with practical implications. *Journal of Consumer Psychology*, *11*(2), 75-85.

Mazni, O., Sharifah Lailee, S.-A., & Holcombe, M. (2009, 3-4 February 2009). *Being Agile in Classroom: An Improvement to Learning Programming*. Paper presented at the National ICT in Education Seminar, Ipoh, Malaysia.

Muraven, M., Tice, D. M., & Baumeister, R. F. (1998). Self control as limited resource: Regulatory depletion patterns. *Journal of Personality and Social Psychology*, 74, 774-789.

Norman, D. A., Ortony, A., & Russell, D. M. (2003). Affect and machine design: Lessons for the development of autonomous machines. *IBM Systems Journal*, 42(1), 38-43.

Qumer, A., & Henderson-Sellers, B. (2008). An evaluation of the degree of agility in six agile methods and its applicability for method engineering. *Journal of Information and Software Technology*, *50*, 280-295.

Runeson, P., & Greberg, P. (2004). *Extreme Programming and Rational Unified Process-Contrast or Synonyms?* Paper presented at the Software Engineering Research and Practice (SERPS'04), Sweden.

Sharifah Lailee, S.-A., Holcombe, M., & Gheorge, M. (2006a). *Extreme Programming and its ability to improve the creativity and innovative of SE teams*. Paper presented at the National ICT Conference, Perlis, Malaysia.

Sharifah Lailee, S.-A., Holcombe, M., & Gheorge, M. (2006b). The Impact of an Agile Methodology on the Well Being of Development Teams. *Empirical Software Engineering*, 11(1), 143-167.

Sharifah Lailee, S.-A., Holcombe, M., Karn, J., Cowling, T., & Gheorge, M. (2005). The Positive Affect of the XP methodology. In *Lecture Notes in Computer Science* (Vol. LNCS 3556, pp. 218-221). Berlin-Heidelberg: Springer-Verlag.

Strode, D. E. (2005). *The agile methods : an analytical comparison of five agile methods and an investigation of their target environment*. Massey University, Palmerston North, New Zealand.

Succi, G., Marchesi, M., Pedrycz, W., & Williams, L. (2002, May, 26-30). *Preliminary Analysis of the Effect of Pair Programming on Job Satisfaction*. Paper presented at the 3rd International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2002), Sardinia, Italy.

Tong, K. L. (2004). Essentials Skills for Agile Development: Macau Productivity & Tech.

Trope, Y., & Neter, E. (1994). Reconciling competing motives in self-evaluation: The role of self-control in feedback seeking. *Journal of Personality and Social Psychology, 66*, 646-657.

Trope, Y., & Pomerantz, E. M. (1998). Resolving Conflicts Among Self-Evaluative Motives: Positive Experiences as a Resource for Overcoming Defensiveness. *Motivation and Emotion*, 22(1), 53-72.

Watson, D., & Clerk, L. A. (1997). Extraversion and its positive emotional core. In R. Hogan & J. A. Johnson (Eds.), *Handbook of Personality Psychology* (pp. 767-793). San Diego, Ca: Academic Press.

Watson, D., Pennebaker, J. W., & Folger, R. (1987). Beyond negative affectivity: Measuring stress and satisfaction in the workforce. *Journal of Organizational Behavior Management*, 8(2), 141-157.

Watson, D., & Tellegen, A. (1985). Towards a consensual structure of mood. Psychological Bulletin, 98, 219-235.

Williams, L. (2002). Pair Programming: Why Have Two Do the Work of One? In M. Marchesi, G. Succi, D. Wells & L. Willaims (Eds.), *Extreme Programming Perspectives* (pp. 23-33). Boston: Addison-Wesley.

Criteria	Extreme Programming (XP)	Scrum	Crystal	DSDM
Principles	Programmer-oriented Based on five values: communication, simplicity, feedback, courage, and respect Small and medium scale project	Project- management oriented Facilitates tracking activities Small and large scale project	Problem-solving for well-defined problem Allow adjusting to project size and criticality Small and medium scale project	Business value is imperative Project that are requirements are flexible Small and large scale project
Methods	Iterative and incremental development planning game, pair programming, refactoring, simple design, continuous integration, test-first programming, collective ownership, coding standards, short releases, metaphor, sustainable pace, on-site customer	Incremental phases adjustment Product backlog, sprint, sprint goal, sprint backlog, daily meeting, sprint review meeting	Iterative and incremental development Project interview, team workshop, reflection workshop	Iterative and incremental development Timeboxing, prioritise requirements, prototyping, regular meeting
Tools	Automated testing and configuration tool	Project management tool	Automated regression testing tool	Requirements analysis, system prototyping, testing and configuration management

Table 1. Comparison of four agile methodologies according to its principles, methods and tools

Table 2. Descriptive statistics of positive affectivity of both teams (UUM 2008)

Method	N	<b>M</b> <sub>1</sub>	SD <sub>1</sub>	<b>M</b> <sub>2</sub>	SD <sub>2</sub>	$M_3$	SD <sub>3</sub>
Formal	28	34.12	4.77	33.61	5.07	33.86	4.57
(RUP)							
Agile(XP)	30	34.53	4.41	34.17	5.90	35.37	6.37

Note: p< 0.05







Figure 2. Bar graph showing teams performance according to project