Vol. XII No. 2 June 2016

The IUP Journal of INFORMATION TECHNOLOGY

IUP Publications
(A Division of The ICFAI Society)
www.iupindia.in

The IUP Journal of
INFORMATION TECHNOLOGY

Vol. XII No. 2 June 2016

Contents

IJIT

Vol. XII No. 2 June 2016

ISSN 0973-2896

Subscriptions

For subscriptions and related enquiries write to:
The Executive, Subscriptions
IUP Publications
(A Division of The ICFAI Society)
52, Nagarjuna Hills
Panjagutta, Hyderabad 500082
Telangana, India
Tel: +91 (40) 39933351/52; +918498843633
E-mail: jms@iupindia.in; info@iupindia.in
Website: www.iupindia.in

The IUP Journal of Information Technology is a 'peer-reviewed' journal published four times a year in March, June, September and December.

Send your feedback to
The Managing Editor
IUP Publications
(A Division of The ICFAI Society)
52, Nagarjuna Hills
Panjagutta, Hyderabad 500082
Telangana, India
E-mail: info@iupindia.in

Subscription Rates			
Period	Hard Copy		
	India ₹	Overseas ($)	
		By Post	By Courier
1 year	1000	50	55
2 years	1800	70	90
3 years	2600	110	130
4 years	3300	140	180
5 years	4000	160	200

Payment to be made by crossed Demand Draft drawn in favor of "ICFAI A/c IUP", Hyderabad. Online remittances can be made to HDFC Bank, Banjara Hills Branch, A/c. No. 50200000962639, "ICFAI A/c IUP", IFSC Code for NEFT: HDFC0000521.

Published by E N Murthy on behalf of IUP Publications (A Division of The ICFAI Society), # 52, Nagarjuna Hills, Panjagutta, Hyderabad 500082, Telangana, India, and printed at M/s. Mahathi Print Creations, 2-3-720/A/11, Ganganagar, Amberpet, Hyderabad 500013.

Focus

Today, we see computing outside the realm of personal computers. It has penetrated into everyday products, enabling useful and interesting applications. It has been estimated that around 90% of the computing devices are in embedded systems rather than personal computers. The growth rate is more than 10% per annum. According to industry forecast, there will be over 30 billion smart devices worldwide by 2020.

With an explosive growth of smart devices and ubiquitous computing, and a desire to be in a connected society, the Internet of Things (IoT) has become a reality gradually. It is an ever-growing network of computing devices, machines, vehicles, buildings, people, animals and other physical objects with the ability to exchange data over the Internet.

The goal of the IoT is to create smart environments that make energy, transport, agriculture, healthcare, tourism, buildings, homes, cities and many other areas more intelligent. Radio Frequency Identification (RFID) and Wireless Sensor Network (WSN) technologies being the critical hardware infrastructure, the IoT system provides remote tracking and monitoring of connected objects and thereby offers numerous applications so as to improve the quality of human life.

While the promises of IoT are exciting, the challenges it offers are manifold. These challenges can be both technical and social in nature and must be overcome in order to ensure its rapid adoption and diffusion. Some of them can be interoperable hardware and software, energy-efficient devices, efficient sensing and identification techniques, communication protocols, quality of service, data analytics and visualization algorithms, security and privacy issues.

Against this backdrop, the paper, "Android-Based City Bus Tracking System", by Snehal A Demapure and S V Kulkarni, describes an intelligent traffic information system using the notion of IoT. The authors claim that the proposed application is efficient, useful and cost-effective.

The second paper, "Mobile Learning and Evaluation: A New Paradigm of Teaching and Learning", by Kirti Panwar and Raj Kamal, presents a mobile application that enables a user to learn a subject and test his knowledge while on the move. The system is developed using Microsoft Technologies and claimed to be user-friendly.

The third paper, "E-Sign Detector: Image Steganography-Based Employee Identification System", by V Senthooran, M T Chathuranga and T Kartheeswaran, describes a system that identifies and tracks employees in an office who try to enter some restricted area. The authors claim that the proposed system will overcome several problems of manual identification and tracking.

The next paper, "A Comparison of Code Maintainability in Agile Environment", by Mary Adebola Ajiboye, Matthew Sunday Abolarin and Johnson Adegbenga Ajiboye, presents a research study on time to maintain code in an agile software development

environment. It reveals that random pair programmers spend more time per bug on the average, while individual experts spend less time to do so.

The last paper, "The Effect of ERP System on Organizational Performance: A Comparative Study", by Bharti Motwani and R K Sharma, is an impact study. It reveals that Enterprise Resource Planning (ERP) systems have positive effects on many factors related to organizational process.

A C Ojha
Consulting Editor

A Comparison of Code Maintainability in Agile Environment

Mary Adebola Ajiboye, Matthew Sunday Abolarin** and Johnson Adegbenga Ajiboye****

The demand for quick delivery of quality software is becoming high among software clients due to the fast changing technology in the dynamic world. Agile software development meets this demand and has gained appropriate and wide acceptance among software practitioners. However, the quality of such software is greatly impacted by its maintainability. Unfortunately, existing works focused only on the flexibility aspect of maintainability without paying attention to timely delivery. In this work, maintainability as a function of time to correct codes was examined among various categories of software developers. Deliberate errors, ranging from two to nine, were introduced into sets of agile codes written in python programming language and given to 100 programmers, each in the groups of individual junior, individual expert, random, expert pairs, junior pairs and junior expert pairs. The results revealed that random pair programmers spent the highest time of 21.88 min/bug on the average, while individual experts spent the least time of 16.26 min/bug.

Keywords: XP, PP, ASD, DMRT, Model Metrics, Agile

Introduction

The issue of how software development should be organized with a view to delivering faster, better and cheaper solutions has been discussed in software engineering circles. There have been lots of suggestions for improvement. This varies from standardization and measurement of the software process to a multitude of concrete tools, techniques and practices (Kaushal and Anju, 2013). Most of the suggestions for improvement have come from experienced software professionals who have individually developed methods and practices to respond to the expected change.

In order to tackle the challenges faced in the software industry, a group of 17 software experts met in Utah in February 2001 to discuss and came up with the agile manifesto. A collection of the different techniques and practices that share the same values and basic principles is called the agile methods. These methods for the agile

* Research Scholar, Department of Computer Engineering, Federal University of Technology, Minna, Nigeria. E-mail: ajiboyemary@gmail.com

** Professor, Department of Mechanical Engineering, Federal University of Technology, Minna, Nigeria. E-mail: abolarinmatthew@yahoo.com

*** Lecturer, Department of Electrical and Electronics Engineering, Federal University of Technology, Minna, Nigeria; and is the corresponding author. E-mail: ajiboye2003@yahoo.com

development are actually a response to the traditional plan-based method, which focuses mainly on a rationalized, streamlined and efficient engineering-based approach (Nerur *et al.*, 2005).

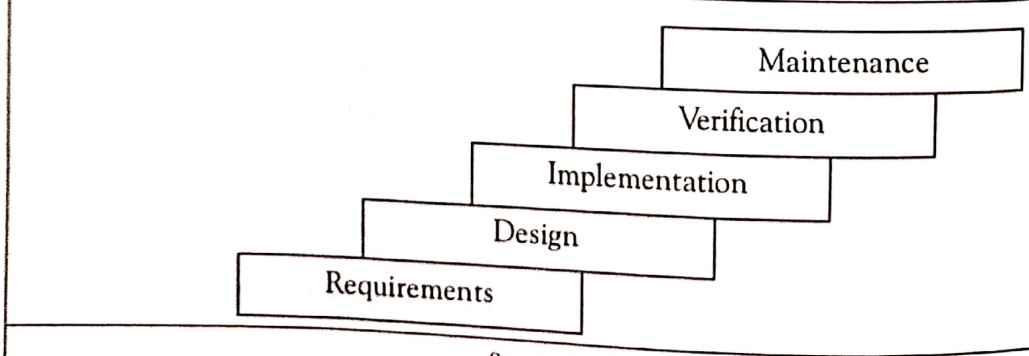
In the traditional approach to software development, problems can be fully specified. Optimum and predictable solutions are proffered to every problem which involves rigorous planning, codified processes, very thorough and meticulous reuse of codes, thereby making the development activity efficient. On the contrary, the agile software development processes tackle the challenge of an unpredictable world by focusing on people and their creativity rather than on processes (Dyba, 2000; and Rao *et al.*, 2011).

The traditional method starts with elicitation and documentation of the client's requirements and then the architectural and high-level design, development and inspection is done (Cohen *et al.*, 2004). However, in the mid-1990s, some software experts realized that these initial software development phases are very wearisome and almost impossible (Highsmith, 2002). The software industry and technology is not rigid, so there is continuous change in requirements making the traditional methods unfit (Highsmith *et al.*, 2000). As a result of this, software clients are not able to really set their requirements at once, yet they expect more from their software. In reaction to this, lots of software experts have individually developed methods and practices to respond to the expected change. The agile methods are a collection of different techniques (or practices) that share the same values and basic principles. Many are, for example, based on iterative enhancement, a technique that was introduced in the year 1975 (Munindar, 2006).

Since 2001, a majority of organizations have adopted and implemented agile software development because the traditional approach to software development is grossly inefficient and does not allow for change (Erika and Scott, 2013). In the traditional approach, customer's requirements are fixed at the beginning of the project. There is therefore a need to have newer approaches to software development that will be dynamic enough to be able to handle the changing requirements (Highsmith and Cockburn, 2001).

The waterfall model, shown in Figure 1, therefore is obsolete (Spence, 2005). Hence the need for a more flexible model that permits software engineers to change

Figure 1: The Waterfall Software Development Process



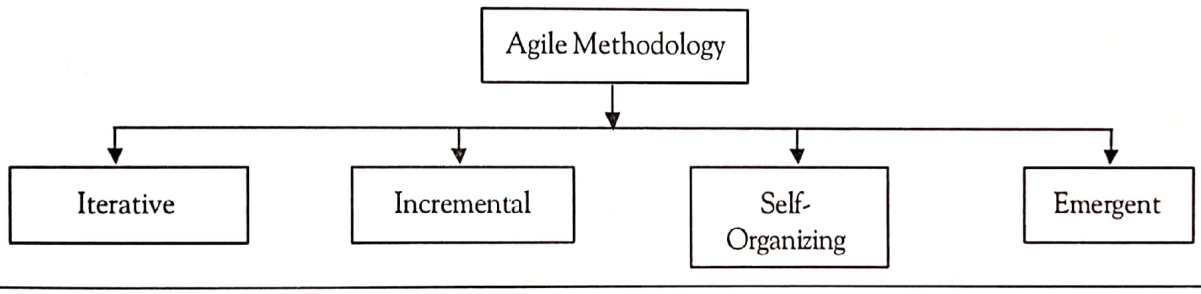
Source: Cockburn, 2002

their requirements much later to reflect the changing software business world (Patel *et al.*, 2012).

Therefore, an approach for adopting agile methods is proposed to accommodate the flexibility that is not found in the waterfall model. The software development industry has experienced changes lately (Petersen, 2010). This has made agile software development popular and widely accepted. Flexibility in the development of software is highly required due to the unpredictable nature of the current software industry. The traditional method of software development cannot suffice due to the changing nature of the client's requirements.

According to Lindvall *et al.* (2002), agile methodologies are a group of software development processes that are iterative, incremental, self-organizing and emergent, as shown in Figure 2. The agile software development is an iterative development process which gives room for change in the requirements throughout the development cycle (Mnkandla and Dwolatzky, 2007). It also encourages and emphasizes close coordination and relationship between the developer and software client. A software development method that delivers faster, better and cheaper solutions is the goal of every organization (Sohaib and Khan, 2010). The core issue in agile is the interaction and frequent communication between the team of developers and stakeholders and the delivery of functional products promptly.

Figure 2: Agile Methodology



Source: Mnkandla and Dwolatzky, 2007

The major shortcoming of agile software development is in the area of quantifiable measurement. Traditional measurement does not consider agile culture and can result in an unwanted effect. There is, therefore, the need for a measurement approach to ensure and strengthen the agile principles. Besides the performance measurement which is usually done on the completed project, there is the need for a technique that continuously assesses the improvement of a process. The study will guide managers to make right decisions when pairing programmers, thereby improving upon the quality of maintainability service.

The aim of this research is to perform an experiment and compare the amount of time spent in correcting python code errors in an agile software development environment and thereby improving upon the maintainability quality factor of software codes. Python programming language is very useful especially for software development

for the embedded systems. Python is a programming language that allows programmers to work very fast to integrate systems effectively (Lutz, 2006). It runs on all major operating systems and can be used for web development. Python provides features of most programming languages. It presents most features of object-oriented language in a simple and powerful way.

2. Materials and Methods

Software maintainability is one of the most important factors that impacts greatly on the quality of any kind of software and is determined by using software metrics which gives important and useful feedback to software designers with a tendency of influencing the decisions that are made during design, coding, architecture and specification phases. Here, the methods and procedures used to acquire the data for this research are described.

An experiment was carried out with codes written in python language. Deliberate errors were introduced into these sets of codes and given to software developers in the agile environment to debug in order to study and evaluate maintainability issues. The expertise of the professionals was put into consideration based on their number of years of experience and all are knowledgeable in python programming. The choice of python is based on the fact that it is an agile, robust, fully object-oriented and scalable programming language. Those above five years of experience were categorized as experts, while juniors are those with five years of experience or below. They were further grouped to work as pairs, where two programmers work together on a task and individuals. The pairing was based on their knowledge of pair programming and their years of experience. The grouping is shown in Table 1.

Table 1: Grouping of Software Programmers

Grouping	Remark
Random Pairs	Irrespective of their years of experience in pair programming
Expert Pairs	Both have more than five years of experience in pair programming
Expert-Junior	Combination of programmers, of which one has more than five years of experience and the other less in pair programming
Junior-Junior	Both have less than five years of experience in pair programming
Individual Expert	More than five years of experience in agile programming
Individual Junior	Less than five years of experience in agile programming

2.1 Samples of Codes with Errors and Outputs

Figure 3 shows a sample of python codes with seven errors. The output of the codes is shown in Figure 4.

Figure 3: Python Codes with Seven Errors Introduced

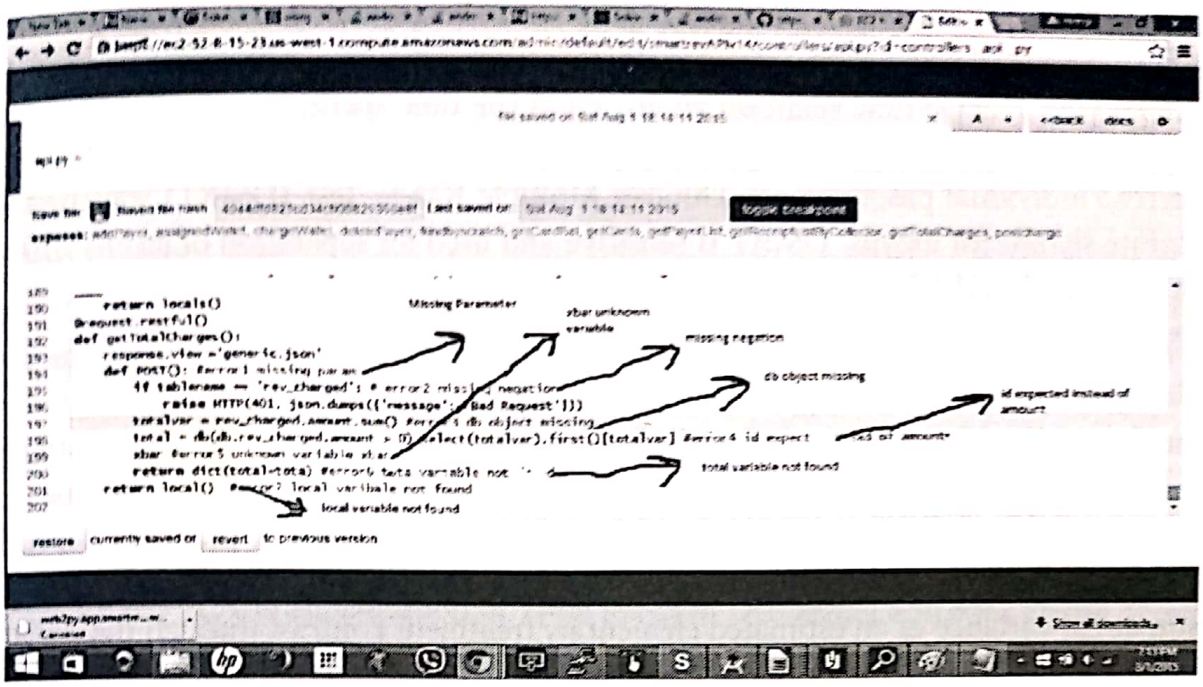
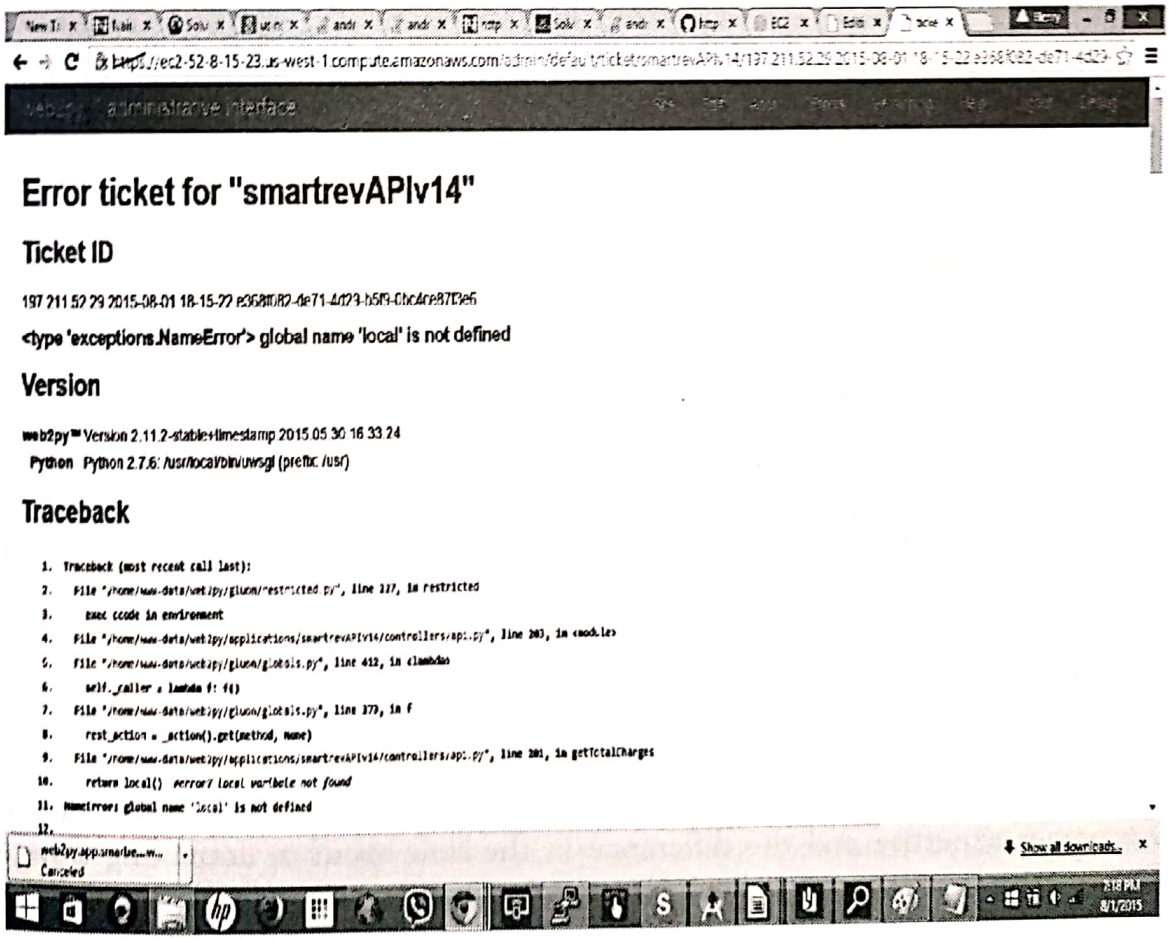


Figure 4: Output of Python Codes with Seven Errors Introduced



2.2 Statistical Tools

The Analysis of Variance (Anova) provides different types of variance analysis and performs a simple analysis of variance on parametric data which is drawn from a known population for three or more samples. In this work, one-way ANOVA was used to compare the average time spent on an error and the time spent on each of the project containing different numbers of bugs between the different pair programmers and the different individual programmers. Duncan Multiple Range Test (DMRT) was used to separate significant means. DMRT is sensitive and used for separation of means within the range of 3 and 10 samples.

A widely used procedure for comparing all pairs of means is the multiple range test developed by Duncan. The application of DMRT involves the computation of numerical boundaries that allow for the classification of the difference between any two treatment means as significant or non-significant. DMRT requires computation of a series of values, each corresponding to a specific set of pair comparisons. It primarily depends on the standard error of the mean difference. This can easily be worked out using the estimate of variance of an estimated elementary treatment contrast through the design. Application of DMRT ranks all the treatment means in decreasing or increasing order based on the preference of the character under study.

2.3 Correlation Coefficient

The correlation coefficient is a measure of the extent to which two measurements, say, X and Y vary together. Correlation analysis examines each pair of measurements to determine whether the two tend to move together. The correlation coefficient can assume any value between –1 and +1. Positive correlation is obtained when large values of one variable tend to be associated with large values of the other. Negative correlation results when small values of one variable tend to be associated with large values of the other and a correlation near 0 (zero) is obtained when values of both variables tend to be unrelated. Bivariate correlation given in Equation (1) was used to check the relationships between the number of bugs in projects and the time spent to correct the errors. Bivariate shows relationship between two variables.

$$Correl(X,Y) = \frac{\sum (x-\bar{x})(y-\bar{y})}{\sqrt{\sum (x-\bar{x})^2 \sum (y-\bar{y})^2}} \quad \ldots(1)$$

3. Results and Discussion

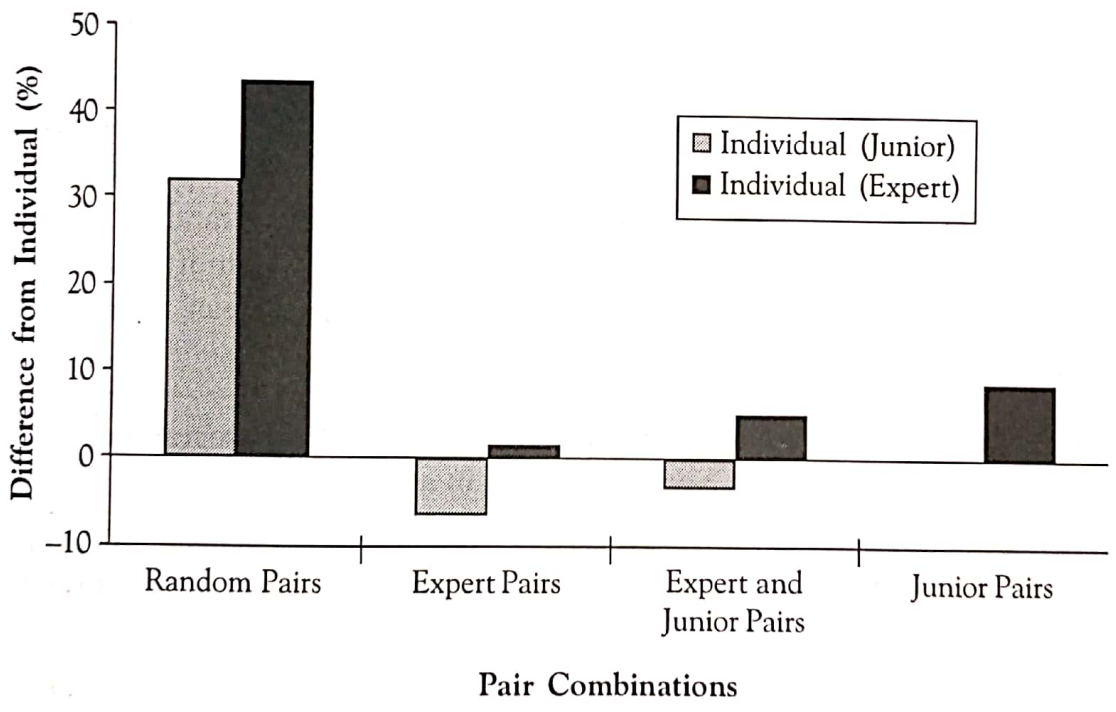
3.1 Average Time Spent on One Error

The average time spent in correcting an error in a system software by the different programmer expertise and the difference in the time spent in correcting a system software error from that of individual programmers (junior and expert) is shown in Table 2 and graphical presentation is shown in Figure 5.

Table 2: Average Time Spent on Debugging System Software Error			
Pair Group	Average Time Spent on Error (min/error)	Difference from Individual (%)	
		Individual (Junior)	Individual (Expert)
Random Pairs	21.88 ± 7.37[a]	32.05	43.38
Expert Pairs	15.5 ± 3.91[b]	–6.46	1.57
Expert and Junior Pairs	16.07 ± 4.39[b]	–3.02	5.31
Junior Pairs	16.59 ± 4.23[b]	0.12	8.72
Individual Expert	15.26 ±3.30[b]		
Individual Junior	16.57 ± 3.36[b]		

Note: ± Means standard deviation on the same column with different superscripts are significantly different ($p < 0.05$); Negative figure indicates that less time was spent than individual (Junior).

Figure 5: Average Duration of Time Spent in Debugging Software Errors by Different Level of Programmer Expertise



The average time spent by the programmers on an error showed some level of significant difference ($p < 0.05$). The randomly paired programmers spent the highest average time (21.88 min) on correcting an error which was significantly higher than other paired and individual programmers. The junior pair, individual junior, expert and junior, expert pair and individual expert programmers spent statistically comparable average time on correcting a system software error. The average time spent in decreasing order was random pair > junior pair > individual junior > expert and junior > expert

pair > individual expert, although differences in time between junior pair, individual junior, expert, and junior, expert pair and individual expert programmers were insignificant.

Comparing the average time spent on an error by the paired programmers with the two different individual programmers, it was found that all the paired programmers spent more time debugging an error compared to the individual expert programmer; random pair spent 43.38% more time, expert pair spent 1.57% more time, expert and junior pair spent 5.31% more time and junior pair spent 8.72% more time. Compared to individual programmer, random pair spent 32.05% more time on an error, while junior pair spent 0.12% more time; expert pair and expert – junior pair spent less time compared to individual junior programmer; expert pair spent 6.46% and expert – junior pair spent 3.02% less than the time individual junior programmer spent on an error.

3.2 Time Spent on Each Project

The time spent by the programmers on each project containing different numbers of bugs (errors) is shown in Table 3 and the graphical representation is shown in Figure 6. The difference in time spent on each project by pair programmers relative from the individual programmers (junior and expert) is shown in Table 4. The correlation coefficient showing the relationship between the number of bugs in a project and the time spent on the project is shown in Table 5.

There were no significant differences ($p > 0.05$) in the average time on system software with 2 and 8 errors between the different programmer expertise but by observation, the decreasing order in the time spent on 2 errors was random > junior pair = expert pair > junior – junior pair > individual junior > individual expert while that of 8 errors in a system software was random > junior pair = individual junior > expert and junior pair > individual expert > expert pair.

Table 3: Average Time Spent by Different Skill Level of Pair Programmer in Debugging Specific Number of Bugs in a System Software

No. of Bugs	Random	Expert-Expert	Expert-Junior	Junior-Junior	Individual-Junior	Individual-Expert
2	60^a	50^a	45^a	50^a	45^a	35^a
3	80^a	52^b	57^b	57^b	56^b	49^b
4	89^a	64^b	63^b	67^b	67^b	63^b
5	113^a	79^b	82^b	81^b	84^b	78^b
6	119^a	82^b	87^b	95^b	88^b	83^b
7	141^a	98^b	98^b	98^b	111^b	99^b
8	136^a	109^a	115^a	118^a	118^a	111^a
9	171^a	132^b	140^b	142^b	145^b	139^b

Note: Means on the same row with different superscripts are significantly different ($p < 0.05$).

Figure 6: Effect of Pair Programming on Duration of Correcting Different Number of Bugs in System Software by Different Level of Programmer Expertise

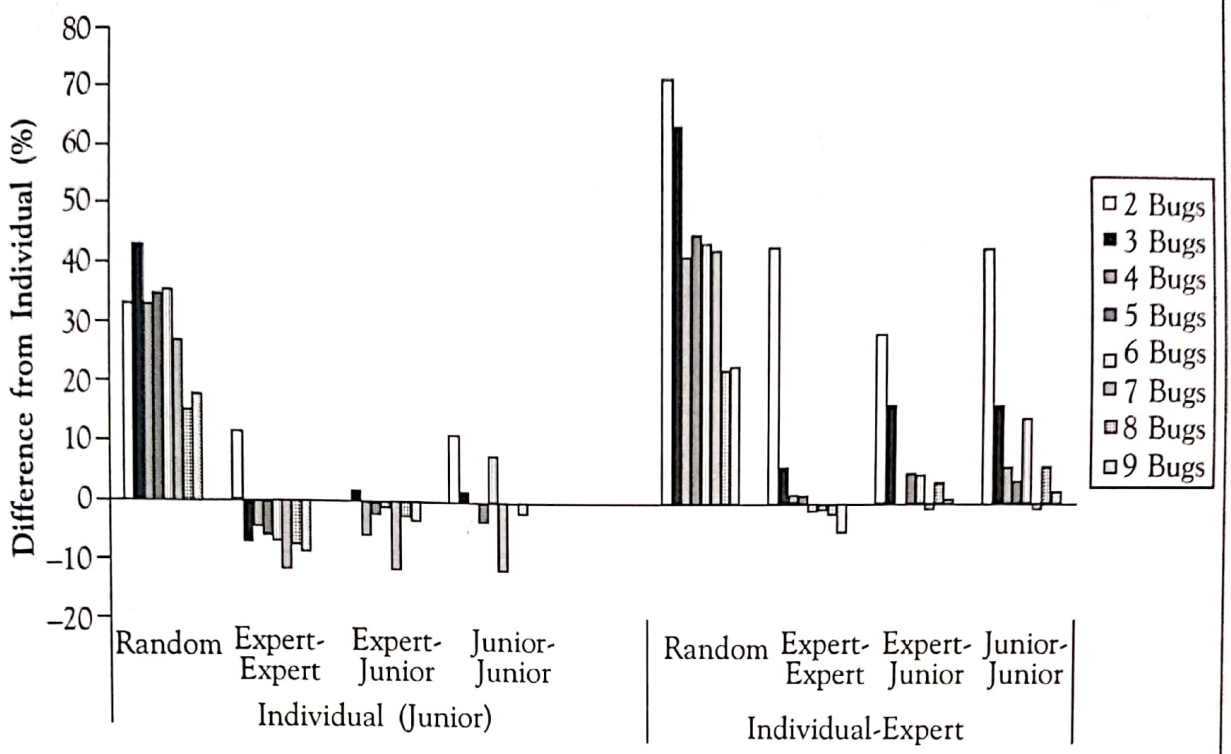


Table 4: Percentage Difference in the Time Spent by Pair Programmer and Individual Programmer on Different Number of Errors in a Project

Number of Bugs	Individual (Junior)				Individual (Expert)			
	Random	Expert-Expert	Expert-Junior	Junior-Junior	Random	Expert-Expert	Expert-Junior	Junior-Junior
2	33	11	0	11	71	43	29	43
3	43	–7	2	2	63	6	16	16
4	33	–4	–6	0	41	2	0	6
5	35	–6	–2	–4	45	1	5	4
6	35	–7	–1	8	43	–1	5	14
7	27	–12	–12	–12	42	–1	–1	–1
8	15	–8	–3	0	23	–2	4	6
9	18	–9	–3	–2	23	–5	1	2

Note: Negative figure indicates that less time was spent than individual (Junior) or individual (expert) as the case may be.

Table 5: Correlation Coefficients of the Number of Bugs in a Project Debugged and the Time Spent on Debugging the Errors

Pair Group	Number of Bugs Vs Time Spent to Debug Error (min)
Random	0.659**
Expert-Expert	0.802**
Expert-Junior	0.779**
Junior-Junior	0.799**
Individual (Junior)	0.870**
Individual (Expert)	0.850**

Note: ** Correlation is significant at 1% level.

There were significant differences ($p < 0.05$) in the average time spent on system software with 3, 4, 5, 6, 7 and 9 errors. The average time random pair spent in debugging software with 3, 4, 5, 6, 7 and 9 bugs was significantly higher than those of expert pair, junior pair, expert – junior pair, individual expert and individual junior. The average time spent on system software with 4, 5, 6, 7 and 9 bugs by expert pair, junior pair, expert – junior pair, individual expert and individual junior was not significantly different from each other. The time spent on debugging the bugs in the decreasing order was random > junior pair = expert – junior pair > individual junior > expert pair > individual expert for system software with three bugs, Random > junior pair = expert – junior pair = individual junior > expert pair > individual expert for system software with four bugs, random > individual junior > expert – junior pair > junior pair > expert pair > individual expert for system software with five bugs, random > junior pair > individual junior > expert – junior pair > individual expert > expert pair for system software with 6 bugs, random > individual junior > junior pair = expert - junior pair = expert pair > individual expert for system software with seven bugs, random > individual junior > junior pair > expert – junior pair > individual expert > expert pair for system software with nine bugs.

The correlation coefficients showing the relationship between time spent on the projects and the number of bugs in the projects for all the programmers shown in Table 5 were highly significant (they were significant at 1% level), strong and positive which implied that the time spent on projects increased significantly as the number of bugs in the project increased for all the programmer expertise.

Conclusion

This research identified that aside from the issue of dynamism and flexibility that are required in software maintainability, delivery time is a great factor. This is also a function of time spent in correcting code errors. The results of several experiments on debugging different error complexities by different combinations of programmers with varying levels of expertise revealed that random pair spent the highest time of 21.88

min/bug on an average. This was significantly different compared to other categorization, while individual expert spent the least time of 15.26 min/bug and this is closely followed by expert-expert pair 15.5 min/bug. The correlation between the number of bugs and time of debugging was highly significant, strong and positive. This revealed that the time spent in correcting system software errors increased significantly as the number of bugs increased.

References

1. Cockburn A (2002), *Agile Software Development*, pp. 304, Addison Wesley Longman.
2. Cohen D, Lindvall M and Costa P (2004), "An Introduction to Agile Methods", *Advances in Computers*, Vol. 62, pp. 1-66.
3. Dyba T (2000), "Improvisation in Small Software Organizations", *IEEE Software*, Vol. 17, No. 5, pp. 82-87.
4. Erika S M and Scott J H (2013), "Scientific Software Process Improvement Decisions: A Proposed Research Strategy", 5th International Workshop on Software Engineering for Computational Science and Engineering (SE-CSE), San Francisco, May 18, 2013, San Francisco, CA, USA.
5. Highsmith J (2002), *Agile Software Development Ecosystems*, Addison-Wesley, Boston, MA.
6. Highsmith J and Cockburn A (2001), "Agile Software Development", *The Business of Innovation Computer*, Vol. 34, No. 9, pp. 120-127.
7. Highsmith J, Orr K and Cockburn A (2000), "Extreme Programming in E-Business Application Delivery", pp. 4-17, available at http://www.cutter.com/freestuff/ead0002.pdf
8. Kaushal P and Anju S (2013), "Review of Agile Software Development Methodologies", *International Journal of Advanced Research in Computer Science and Software Engineering*, Vol. 3, No. 2, Research Paper available at www.ijarcsse.com
9. Lindvall M, Basili V R, Boehm B (2002), "Empirical Findings in Agile Methods", Proceedings of Extreme Programming and Agile Methods - XP/agile Universe, pp. 197-207.
10. Lutz M (2006), *Programming Python*, 3rd Edition, Ebook, Safari Books Online, O'Reilly Media.
11. Mnkandla E and Dwolatzky B (2007), "Agile Methodologies Toolbox", Proceeding of the 2nd International Conference on Software Engineering Advances - ICSEA, Cap Esterel.

12. Munindar P S (2006), *The Practical Handbook of Internet Computing*, Chapman and Hall/CRC Computer and Information Science Series, Boca Raton London New York Washington DC.

13. Nerur S, Mahapatra R and Mangalaraj G (2005), "Challenges of Migrating to Agile Methodologies", *Communications of the ACM*, May, pp. 72-78.

14. Patel A, Seyfi A, Taghavi M *et al.* (2012), "A Comparative Study of Agile, Component-Based, Aspect-Oriented and Mashup Software Development Methods", *Technical Gazette*, Vol. 19, No. 1, pp. 175-189.

15. Petersen K (2010), "Is Lean Agile and Agile Lean? A Comparison between two development Paradigms", *Modern Software Engineering Concepts and Practices: Advanced Approaches*.

16. Rao K N, Naidu G K and Chakka P (2011), "A Study of the Agile Software Development Methods, Applicability and Implications in Industry", *Int. J. Softw. Eng. Appl.*, Vol. 5, No. 2, pp. 35-45.

17. Sohaib O and Khan K (2010), "The Role of Software Quality in Agile Software Development Methodologies", *Journal of Engineering and Sciences*, pp. 6-18.

18. Spence J W (2005), "There Has to Be a Better Way!", Agile Conference, 2005 Proceedings, July, pp. 272-278.

Reference # 35J-2016-06-04-01