ISSN 2278-3091
Volume 3, No.3, May – June 2014
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://warse.org/pdfs/2014/ijatcse02332014.pdf

A Survey of Improving Computer Program Readability to Aid Modification

Goodubaigari Amrulla [1], Murlidher Mourya [2], Abdul Ahad Afroz [3] and Syed Kashif Ali [4]
[1] Assistant Professor Deportment of CSE,
Vardhaman College of Engineering, JNTU Hyderabad, A.P, India
amrushafi12@gmail.com
[2] Assistant Professor Deportment of CSE,
Vardhaman College of Engineering, JNTU Hyderabad, A.P, India
murli_cool9@yahoo.com
[3] Assistant Professor Deportment of IT,
Green Fort Engineering College, JNTU Hyderabad, A.P, India
abdulahadafroz@gmail.com
[4] Assistant Professor Deportment of IT,
Green Fort College of Engineering, JNTU Hyderabad, A.P, India
Kashif556@gmail.com

ABSTRACT

A consensus exists that readability is an essential determining characteristic of code quality, but not about which factors contribute to human notions of software readability the most. We define readability as a human judgment of how easy a text is to understand. The readability of a program is related to its maintainability, and is thus a key factor in overall software quality. Typically, maintenance will consume over 70 percent of the total life-cycle cost of a software product. While software complexity metrics typically take into account the size of classes and methods and the extent of their interactions, the readability of code is based primarily on local, line-by-line factors. Our notion of readability arises directly from the judgments of actual human annotators who do not have context for the code they are judging. We present a descriptive model of software readability based on simple features that can be extracted automatically from programs. This model of software readability correlates strongly with available) notions of software quality, such as defect detectors and software changes.

Key words: Find Bugs, modifiability, software Quality, readability, program Style, Software maintenance.

1. INTRODUCTION

We define readability as a human judgment of how easy a text is to understand. The readability of a program is related to its maintainability, and is thus a key factor in overall software quality. Typically, maintenance will consume over 70 percent of the total life-cycle cost of a software product .claim that source code readability and documentation readability are both critical to the maintainability of a project.

The topic of source code readability has paramount importance in software engineering. Literature exists on how to write readable code; how to create analytical models and automatically predict readability; and how readability influences software cost and eventually the economy. In this article we follow a different path; we explore the question of why and how unreadable code gets written.

Other researchers have noted that the act of reading code is the most time-consuming component of all maintenance activities. Readability is so significant, in fact, that, after recognizing that many commercial programs were much more difficult to read than necessary, proposed adding a development phase in which the program is made more readable. Knight and Myers suggested that one phase of software inspection should be a check of the source code for readability to ensure maintainability, portability, and reusability of the code. proposed adding a dedicated readability and documentation group to the development team, observing that, “without established and consistent guidelines for readability, individual reviewers may not be able to help much” .We hypothesize that programmers have some intuitive notion of this concept, and that program features, such as indentation (e.g., as in Python), choice of identifier names, and comments, are likely to play a part. Dijkstra, for example, claimed that the readability of a program depends largely upon the simplicity of its sequencing control (e.g., he conjectured that go to unnecessarily complicates program understanding), and employed that notion to help motivate his top-down approach to system design. We present a descriptive model of software readability based on simple features that can be extracted automatically from programs. This model of software readability correlates strongly with human annotators and also with external (widely available) notions of software quality, such as defect detectors and software changes.

To understand why an empirical and objective model of software readability is useful, consider the use of readability metrics in natural languages. The Flesch-Kincaid Grade Level, the Gunning-Fog Index, the SMOG Index, and the Automated Readability Index are just a few examples of readability metrics for ordinary text. These metrics are all based on simple factors, such as average syllables per word and average sentence length. Despite this simplicity, they have each been shown to be quite useful in practice. Flesch-Kincaid, which has been in use for over 50 years, has not only been integrated into popular text editors including Microsoft Word, but has also become a United States governmental standard. Agencies, including the Department of Defense, require many documents and forms, internal and external, to meet have a readability grade of 10 or below). Defense contractors also are often required to use it when they write technical manuals.

1.1 Objective

- To improve the quality of maintainability**.**
- Enhance the chances to reusability.
- To make portability easier and flexible**.**
- Mechanically predict human readability judgments.
- Determine code features that are predictive of readability.

1.2 Existing system

The system provided the unbeliever to calculate the quality of software by the automation. The tester who's calculate the coding of application to be quality so who's verified the bulk of coding which's dominated by them. While it can be used to predict human readability judgments for existing software, it can be directly interpreted to prescribe changes that will improve readability.

1.3 Disadvantages

- The tester should know the language whish's developed by.
- Time consuming is high so we can't deliver the product in time.
- The result of quality should not be accurate.

1.4 Proposed System

We proposed that this metric correlates strongly with three measures of software quality: code changes, automated defect reports, and defect log messages. These metrics can help organizations gain some confidence that their documents meet goals for readability very cheaply, and have become ubiquitous for that reason. We believe that similar metrics, targeted specifically at source code and backed with empirical evidence for effectiveness, can serve an analogous purpose in the software domain. It is important to note that readability is not the same as complexity, for which some existing metrics have been empirically shown useful. Brooks claims that complexity is an "essential" property of software; it arises from system requirements, and cannot be abstracted away. In the Brooks model, readability is "accidental" because it is not determined by the problem statement.

1.5 Advantages

- We have to generate three measures of software quality together like as code changes, automated defect reports, and defect log messages.
- Time consuming is low.
- It may help developers to write more readable software by quickly identifying code that scores poorly.
- A technique for the construction of automatic software readability metric based on local code features.

2. MODULES DESCRIPTION

A. Authentication

B. Source code

C. Code readability

D. Defects log message

A. Authentication

This module is used to secure our application from the unauthorized persons so it wants to ask the user to submit those details into our database so only valid users can login into the application.

B. Source Code

Source code is the means most often used by programmers to specify the actions to be performed by a computer. The source code which constitutes a program is usually held in one or more text files sometimes stored in databases as stored procedures and may also appear as code snippets printed in books or other media. A computer program's source code is the collection of files needed to convert from human-readable form to some kind of computer-executable form. The source code may be converted into an executable file by a compiler, or executed on the fly from the human readable form with the aid of an interpreter.

C. Code Readability

We present a descriptive model of software readability based on simple features that can be extracted automatically from programs. This model of software readability correlates strongly with human annotators and also with external (widely available) notions of software quality, such as defect detectors and software changes. To understand why an empirical and objective model of software readability is useful, consider the use of readability metrics in natural languages.

D. Automated Defect Log Messages

We present a descriptive model of software readability based on simple features that can be extracted automatically from programs. This model of software readability correlates strongly with human annotators and also with external (widely available) notions of software quality, such as defect detectors and software changes.

3. Motivation

The software community is increasingly concerned about the code readability (Buse et al. 2010):
The readability of source code is related to its maintainability, and is thus a key factor in overall software quality. Typically, maintenance consumes over 70% of the total lifecycle cost of a software product (Boehm 2001). Aggarwal claims that source code readability and documentation readability are both critical to the maintainability of a project. Other researchers have noted that the act of reading code is the most time-consuming component of all maintenance activities. Readability is so significant, in fact, that Elshoff and Marcotty, after recognizing that many commercial programs were much more difficult to read than necessary, proposed adding a development phase in which the program is made more readable. In this article, we first explore the various importance of readability. Later, we explore how and why readability gets compromised.

3.1 Importance of Readability

The importance of code readability is clearly stated in the book Structure and Interpretation of Computer Programs (Abelson and Sussman 1996): Programs must be written for people to read, and only incidentally for machines to execute. Readability of source code is important for various reasons, as highlighted in this section.

3.2 Understanding

A portion of code written by a programmer (author) must be understandable by current stakeholders, e.g. the author's immediate team members (and even the author at a future time). But that is not all; the code must be understandable by future stakeholders, e.g. rest of the programmers in the project or organization, especially programmers who might be hired in future. A program might be an application program, a library, a framework, or any other software. Understanding code has many perspectives, the two main ones are: application-level understanding and programming language-level understanding

3.3 Interfacing

New modules interface with existing modules. At that time, the author(s) of a new module has to understand the interfaces exposed by the existing module. In theory, understanding only the interface is enough. But, in reality, the inner details of the existing modules need to be understood, at least at a high level.

3.4 Extending/Enhancing

When a module is extended or enhanced, the existing model and concepts need to be understood so that the extension aligns with the existing code

3.5 Fixing

This is perhaps the most cited reason for readability. Many times, a programmer is responsible for fixing an unfamiliar module. The existing code, the classes, the methods, the variables, their names, 3 Origins of Poor Code Readability and the control flow, must be understandable enough so that such a newcomer to the module can easily identify the place to fix and the nature of the fix.

3.6 System Architecture

It was decided to use only programs written in high level programming languages. These are widely used and they are machine independent. Two kinds of programs can be used, namely artificially constructed programs and real-world programs. Weismann [5] used the former kind. In investigating the mnemonicity of variable names he used three programs for solving the eight queens' problem with three levels of mnemonicity: fully mnemonic, shortened mnemonic, and meaningless, respectively. Real-world programs are not written with artificially varied levels of programming style characteristics, however. Therefore we decided to use real-world programs.

The subjects determining the readability and modifiability of a set of programs should evidently be familiar with the programming language. They should also be familiar with the task of the programs and the applied algorithms so as to make their performances as far as possible independent of these irrelevant matters. Readability can determine the ease in which computer program code can be read by humans, such as through embedded documentation

- User first need to login with his ID and password to see the defect log report.
- If user not created ID then user can create newly.
- For old users the ID and password check from the database for authentication where they are stored in the creation time through the software application.
- After login through the software program user can give the source code as a input.
- It will be go through the software quality process and do the readability.
- Then it will store the defect or error report in to the database.

Finally from the database information it will fetch and will generate the defect report.

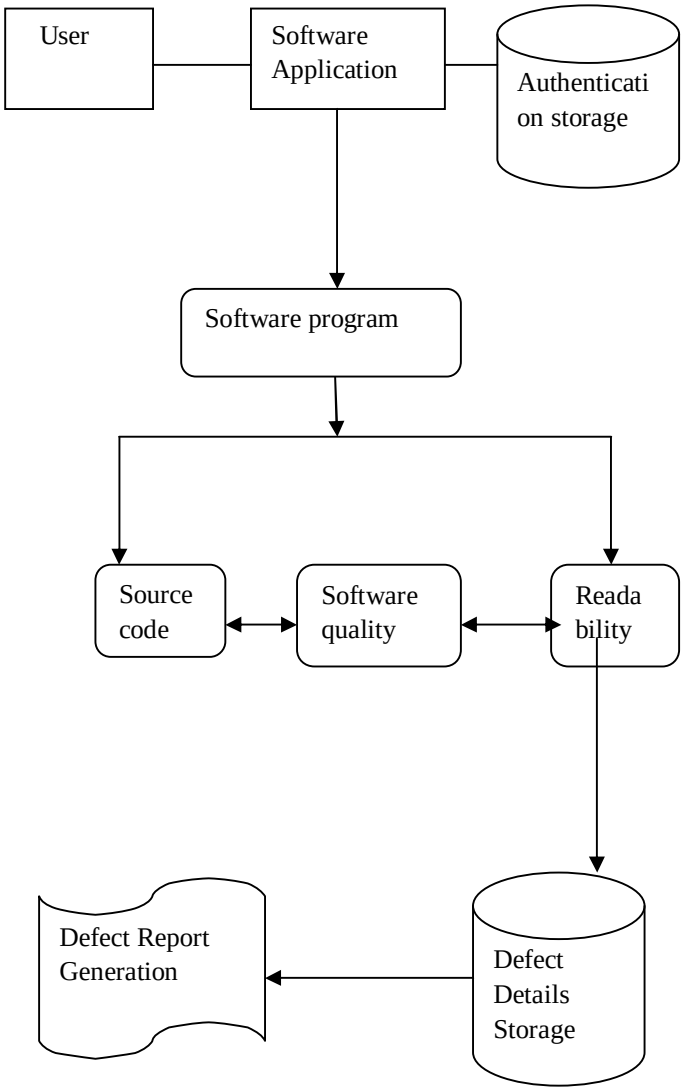

Figure: 1 System Architecture

3.7 Test Case

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner.

Table 1: Test cases

Test S. No	Input	Expected Behavior	Observed Behavior	Status P = Passed F = Failed
1	Login as user with correct login details	The window will open from which .we can send the file to the destination.	-do-	P
2	Login as a user with wrong login details	Error message will be displayed	-do-	P
3	Signup a new User.	It should add a new record in the database with new and unique secret key	-do-	P
4	Choosing a file and load.	We can upload the file For check readability	-do-	P
5	Check source code readability	It checks against Different metrics	-do-	P
6	Generating defects log message	For every readability check it generates defect log and send it to destination	-do-	P
7	Report generation	For every readability check any content which is not up to metrics it generate error no related description	-do-	P

3.8 Project flow

- User first need to login with his ID and password to see the defect log report.
- If user not created ID then user can create newly.
- For old users the ID and password check from the database for authentication where they are stored in the creation time through the software application.
- After login through the software program user can give the source code as a input.
- It will be go through the software quality process and do the readability.
- Then it will store the defect or error report in to the database.

Finally from the database information it will fetch and will generate the defect report.

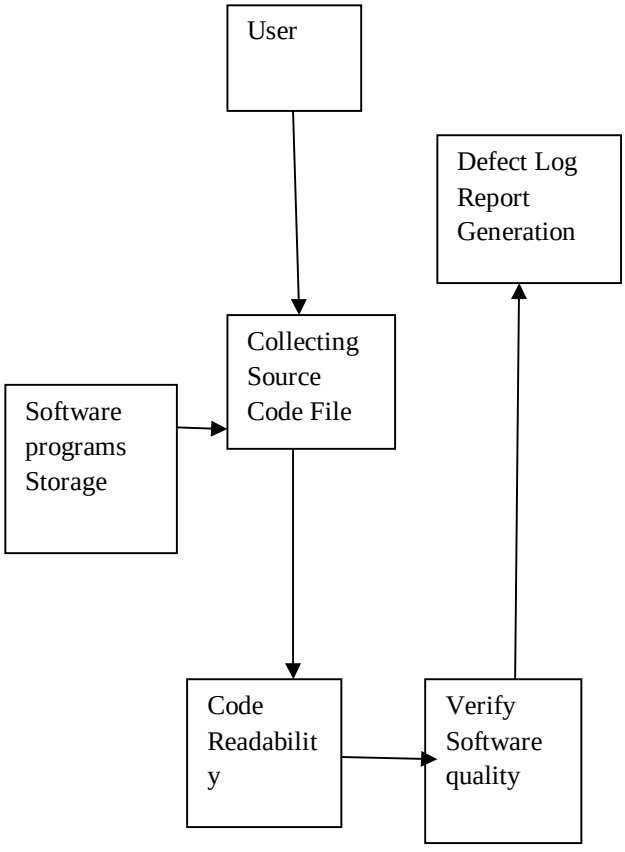

Figure: 2 Shows a Project Flow Diagrams

4. IMPLEMENTATION

Implementation notes helps to improve readability as it discusses difficult or subtle algorithms and data structures. It includes graphs, drawing, charts and other representations difficult to reproduce in source code library. It also comprises photocopies of portions of books or articles relevant to the design or implementation. All these documentation enhance readability of program. The more readable a module the faster and more accurately a rouser can obtain information about it. Here readability can be gauged by the average number of right answers to a series of questions about the program in a given length of time. Comments could indirectly rescue a not so modular program and make it as readable as modular program by increasing its readability.

- Create the ID to login, if already created.
- Do authentication from database.
- If match allow user to login else through error message user is not valid.
- If user allows from step3 provide source code to software programmer to do software quality check through readability process.
- If any differences or unmatched write the error in to data base else continue until the end.

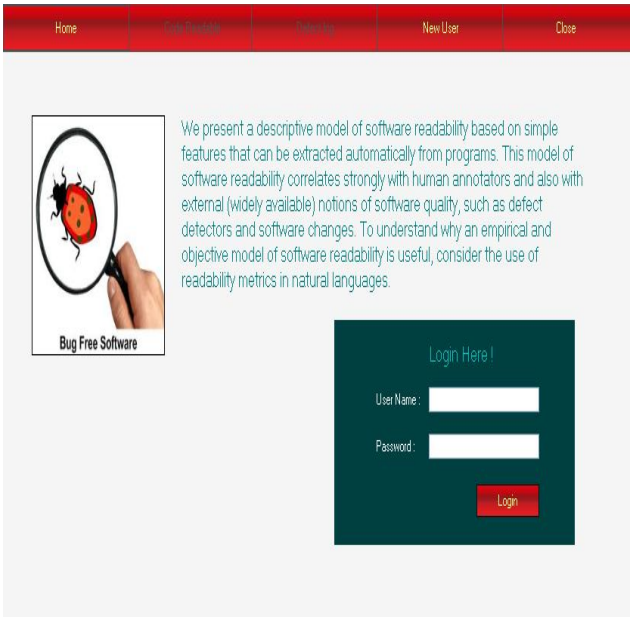

Figure: 3 Shows a User Login home page

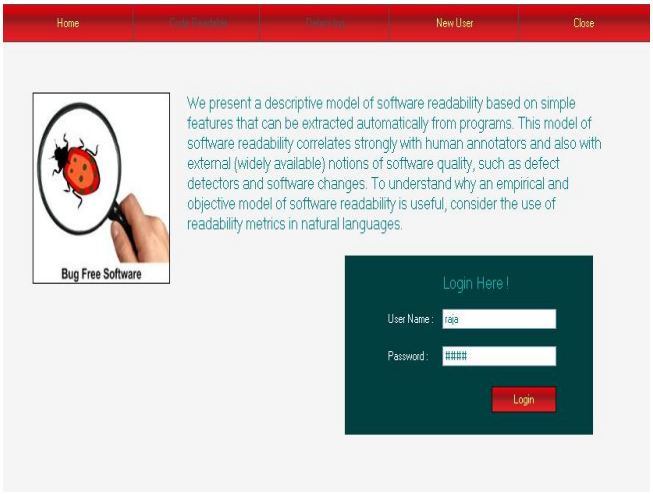

Figure: 4 Shows a User Login Process

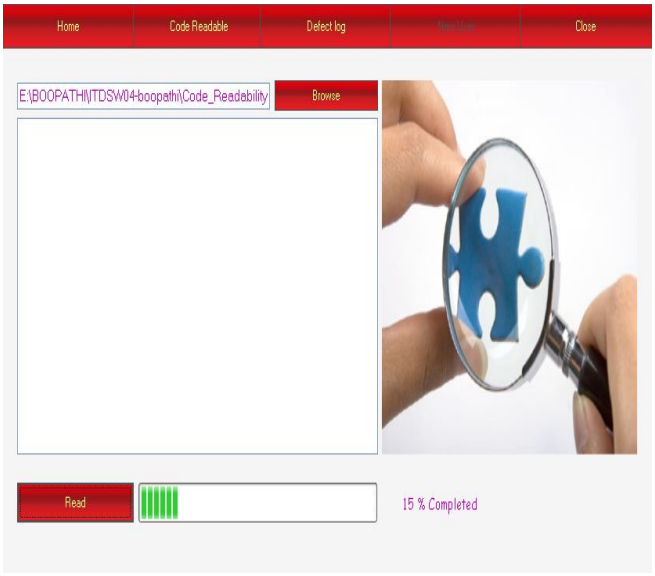

Figure: 5 Shows a Code Files Loading

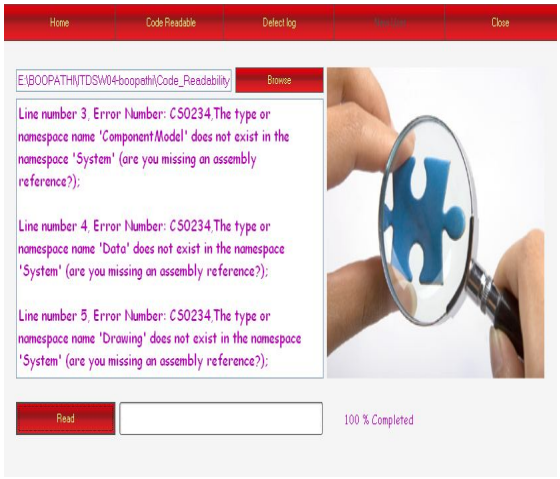

Figure: 6 Shows a Error Identification

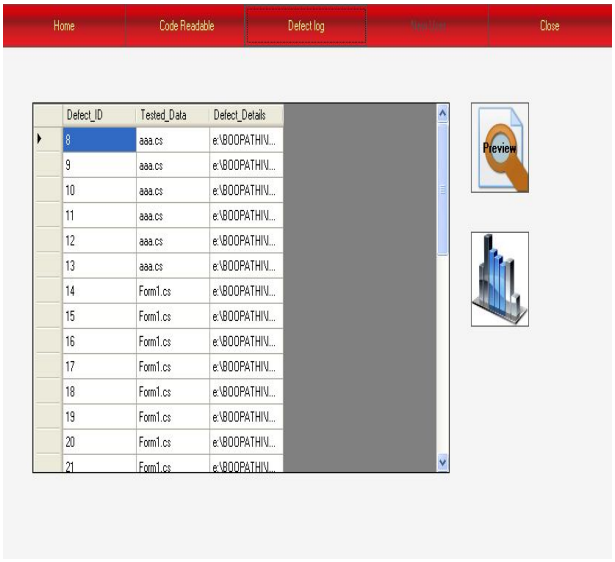

Figure: 7 Shows a Errors List in Defect log

Figure: 8 Shows a Graphical view of Errors

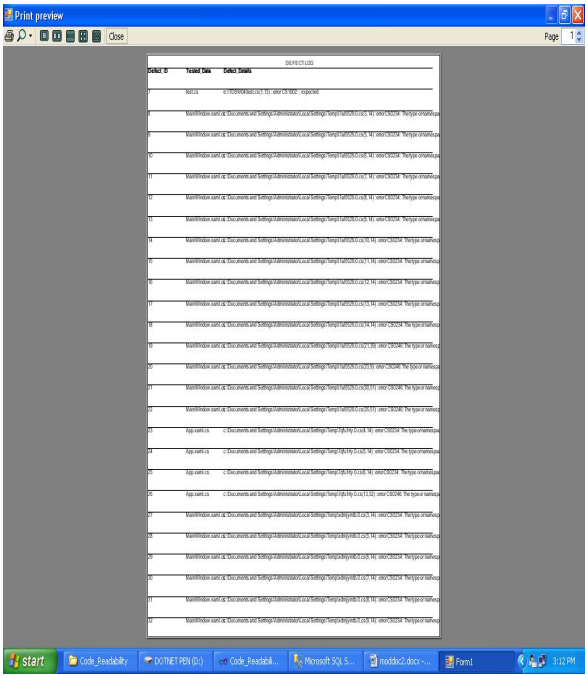

Figure: 9 Shows a Crystal Report Generations

4.1 Future Scope

The techniques presented in this paper should provide an excellent platform for conducting future readability experiments, especially with respect to unifying even a very large number of judgments into an accurate model of readability. While we have shown that there is significant agreement between our annotators on the factors that contribute to code readability, we would expect each annotator to have personal preferences that lead to a somewhat different weighting of the relevant factors. It would be interesting to investigate whether a personalized or organization-level model, adapted over time, would be effective in characterizing code readability. Furthermore, readability factors may also vary significantly based on application domain. Additional research is needed to determine the extent of this variability, and whether specialized models would be useful. Another possibility for improvement would be an extension of our notion of local code readability to include broader features. While most of our features are calculated as average or maximum value per line, it may be useful to consider the size of compound statements, such as the number of simple statements within an if block. For this study, we intentionally avoided such features to help ensure that we were capturing readability rather than complexity. However, in practice, achieving this separation of concern is likely to be less compelling.

5. CONCLUSION

In this paper, we have presented a technique for modeling code readability based on the judgments of human annotators. In a study involving 120 computer science students, we have shown that it is possible to create a metric that agrees with these annotators as much as they agree with each other by only considering a relatively simple set of low-level code features. In addition, we have seen that readability, as described by this metric, exhibits a significant level of correlation with more conventional metrics of software quality, such as defects, code churn, and self reported stability. Furthermore, we have discussed how considering the factors that influence readability has potential for improving the programming language design and engineering practice with respect to this important dimension of software quality. Finally, it is important to note that the metric described in this paper is not intended as the final or universal model of readability.

REFERENCES

1. Raymond P. L. Buse, Westley R. Weimer, “Learning a Metric for Code Readability,” Transactions on Software Engineering, vol. 36, no. 4, pp. 546-558, July/August 2010.
2. K. Aggarwal, Y. Singh, and J. K. Chhabra, “An integrated measure of software maintainability,” Reliability and Maintainability Sympo-sium, pp. 235–241, Sep. 2002.
3. B. Boehm and V. R. Basili, “Software defect reduction top 10 list,”Computer, vol. 34, no. 1, pp. 135–137, 2001.
4. B. Boehm and V. R. Basili, “Software defect reduction top 10 list,” Computer, vol. 34, no. 1, pp. 135–137, 2001.
5. L. Weissman: A methodology for studying the psychological complexity of computer programs. Technical Report CSRG-37, University of Toronto (1974).
6. Anker helms Jorgensen: A methodology for Measuring
The Readability and Modifiability Of computer programs received September 27, 1979. Revised august 18,bit 20 1980.
7. Arun Saha: Origins of poor code readability, Short Paper, Conjectural Fujitsu Network Communications 1250 E Arques Avenue, Sunnyvale, CA, 94085

About The Authors

Mr. GOODUBAIGARI AMRULLA is assistant professor at Vardhaman College of Engineering. He has 1.5 years of experience in teaching field. He has received B.Tech (Information Technology) degree from VVIT Chevella, JNTUH University in the year 2010 and M.Tech (Software Engineering) degree from NIET Deshmukhi, JNTUH University in the year 2013.

Mr. MURLIDHER MOURYA is assistant professor at Vardhaman College of Engineering. He has 5 years of experience in teaching field. He has received B.Tech (CSE)

degree from GRIET Bachupally, JNTUH University in the year 2007 and M.Tech (CSE) degree from TKRCET Medbowli, Meerpet, JNTUH University in the year 2010.

Mr. ABDUL AHAD AFROZ is assistant professor at Green Fort Engineering College. He has 4 years of experience in teaching field. He has received B.Tech (Information Technology) degree from Green Fort Engineering College Bandla Guda, Chandrayanagutta, JNTUH University in the year 2008 and M.Tech (Software Engineering) degree from NIET Deshmukhi, JNTUH University in the year 2013.

Mr. SYED KASHIF ALI is assistant professor at Green Fort Engineering College. He has 2 years of experience in teaching field. He has received B.Tech (CSE) degree from RITS Chevella, JNTUH University in the year 2010 and M.Tech (Software Engineering) degree from NIET Deshmukhi, JNTUH University in the year 2013.