# EFFECTIVE IMPLEMENTATION OF AGILE PRACTICES – OBJECT ORIENTED METRICS TOOL TO IMPROVE SOFTWARE QUALITY

Veerapaneni Esther Jyothi<sup>1</sup>, Kaitepalli Srikanth<sup>2</sup> and K. Nageswara Rao<sup>3</sup>

<sup>1</sup>Assistant Professor, Department of Computer Applications, V.R.Siddhartha Engineering College, Vijayawada – 7, Andhra Pradesh, India nejyothi@hotmail.com
<sup>2</sup>Assistant System Engineer, Tata Consultancy Services, Hyderabad, India srikanthmca2012@gmail.com
<sup>3</sup>Professor & Head, Department of Computer Science and Engineering, P.V.P.Siddhartha Institute of Technology, Vijayawada – 7, Andhra Pradesh, India drknrao@ieee.org

#### **ABSTRACT**

Maintaining the quality of the software is the major challenge in the process of software development. Software inspections which use the methods like structured walkthroughs and formal code reviews involve careful examination of each and every aspect/stage of software development. In Agile software development, refactoring helps to improve software quality. This refactoring is a technique to improve software internal structure without changing its behaviour. After much study regarding the ways to improve software quality, our research proposes an object oriented software metric tool called "MetricAnalyzer". This tool is tested on different codebases and is proven to be much useful.

## **KEYWORDS**

Agile, Refactoring, Metrics, Interpretation, Advice

## **1. INTRODUCTION**

Based on the developments in the business Software frequently undergoes changes. These changes in the software generally made under the demand of time and budget. Obviously the developer have to compromise on quality if he does any changes not taking into account the impact of the changes on the software. Though a solitary change may not have a negative impact, multiple changes with their interplay might affect severe troubles in the long run. Lehman's law states some important characteristics of software system: Law I: continuous change is necessary to preserve satisfaction and Law II: says that the complexity of the system increase with the number of changes if it is not reduced with additional work [1].

Eventually we are left with a software design, which is complex and entangled. To improve the quality of the software the design problems must be identified and eradicated otherwise they may cause the system to display low maintainability, low reuse, high complexity and flawed behaviour. Software inspection involves a vigilant assessment of the code, the design, and the certification of the software and checking for the features that are notorious to be potentially challenging based on the past occurrence. It is normally acknowledged that the cost of fixing a

bug is much lower when that bug is found earlier in the development cycle. Refactoring is a technique to improve software internal structure without changing its behaviour [2].

We tried to detect the code smells [3] in the codebase of a java project by using our tool called "MetricAnalyzer". This tool examined the code and has come up with an advice to refactor in order code to improve the software quality. This process involves different phases.

This research paper is organised into different phases systematically as follows:

Section 2: Decide upon Metrics [IDENTIFYING PHASE] Section 3: Apply metrics on codebase [EVALUATION PHASE] Section 4: Result analysis [INTERPRETATION AND ADVISING PHASE] Section 5: Refactoring the codebase [APPLICATION PHASE]

# 2. IDENTIFYING PHASE

Object-oriented design and development is becoming very popular in today's software development environment. It entails not only a special approach to design and implementation; it requires different software metrics. Since object oriented technology exploits objects and not algorithms as its elemental building blocks, the approach to software metrics for object oriented programs must be different from the standard metrics set. Some metrics, such as lines of code and cyclomatic complexity, have become accepted as "standard" for traditional functional/ procedural programs, but for object-oriented, there are many proposed object oriented metrics in the literature.

Object oriented metrics was to focus on the primary, critical constructs of object orienteddesign and to select metrics that apply to those areas. The suggested metrics are supported by most literature and some object oriented tools. The metrics evaluate the object oriented concepts: methods, classes, complexity and inheritance. The metrics focus internal object structure that reflects the complexity of each individual entity andon external complexity that measures the interactions among entities. The metricsmeasure computational complexity that affects the efficiency of an algorithm and the use of machine resources, as well as psychological complexity factors that affect the ability of a programmer to create, comprehend, modify, and maintain software.

Table 1 presents an overview of the metrics applied he "MetricAnalyzer" tool for object oriented systems.

Metric	Object Oriented Concept	Measurement Method
TNOA Number of Attributes	Class	Both Static and Instance Variables
WMPC Weighted Methods per Class	Class	Both Static and Instance Methods
NOCP Number of Classes	Package	All the classes in a Package
MIT Maximum Inheritance Tree	Class	Maximum length from class to Root node
CCIM Cyclomatic Complexity	Method	Number of linearly independent paths in the Method
SIZE Lines of Code	Class	Non Comment and Non Blank Lines

Table 1. Metrics applies on code base

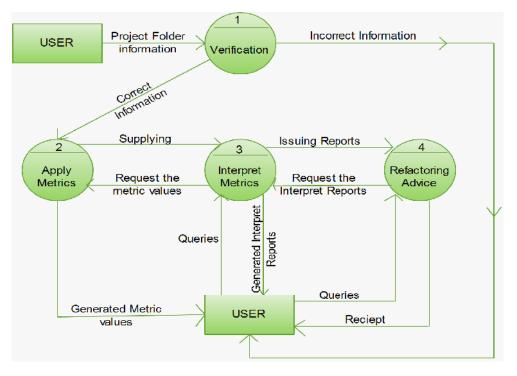


Figure 1. DFD of the MetricsAnalyzer tool

# **3. EVALUATION PHASE**

The Metrics part of the MetricAnalyzer tool applies and extracts information from the development environment. As Figure 2 illustrates, this information may deal with artifacts in the repository (R) or with the Projects themselves (P1, P2, P3).

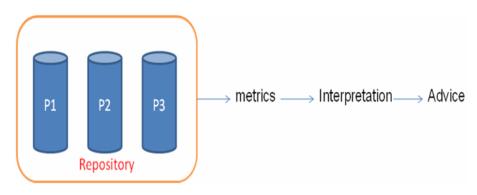


Figure 2. Process involved in MetricsAnalyzer tool

The following are the code snippets for four metrics TNOA (Total Number of Attributes), WMPC (Weighted Methods per Class), MIT (Max Inheritance Tree), NOCP (Number of classes Per Package) written as a part of "MetricAnalyzer" tool development which are provided for examination.

## **TNOA** (Total Number of Attributes)

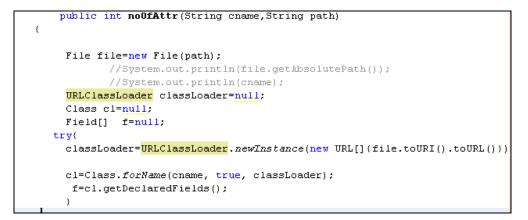


Figure 3. Code snippet for the metric TNOA

## WMPC (Weighted Methods per Class)

```
public int noOfMethods(String cname,String path)
{
    File file=null;
    URLClassLoader classLoader=null;
    Class cl=null;
    Method[] m=null;
    try{
        file=new File(path);
        classLoader=URLClassLoader.newInstance(new URL[]{file.toURI().toURL()})
        cl=Class.forName(cname, true, classLoader);
        m=cl.getDeclaredMethods();
    }
}
```

#### Figure 4. Code snippet for the metric WMPC

#### **MIT (Max Inheritance Tree)**

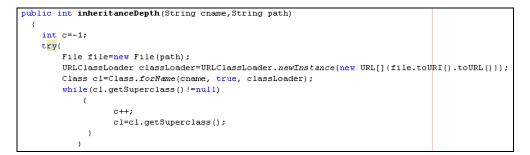


Figure 5. Code snippet for the metric MIT

#### **NOCP** (Number of classes Per Package)

```
public List<String> no0fClasses(File directory, String packageName)
       List<String> classes = new ArrayList<String>();
       int cnt=0;
       File[] files = directory.listFiles();
        for (File file : files)
        {
         if (file.isDirectory())
         {
         assert !file.getName().contains(".");
         classes.addAll(noOfClasses(file, packageName + "." + file.getName()));
         3
       else if (file.getName().endsWith(".class"))
       {
                       cnt++;
          }
       }
   <mark>classes</mark>.add(packageName);
   classes.add(Integer.toString(cnt));
   return classes;
```

Figure 6. Code snippet for the metric NOCP

# 4. INTERPRETATION AND ADVISING PHASE

As we can see in Figure 1 and Figure 2, the interpretation part of the MetricAnalyzer takes the extracted metric values in Metric Evaluation Phase from the source code and compare these values with the threshold values of the corresponding metric values. If the extracted metric values lies below the corresponding metric threshold value means that no occurrence of the issue that is being observed and extracted metric values lies above the corresponding metric threshold value means maximum occurrence of the observed issue.

The crucial step in defining the interpretation for the extracted metrics lies in defining the thresholds for metrics. This should preferably be done based on historical data and experience from earlier projects [4]. Industry bench- marks can also be used for this purpose. The thresholds should be monitored and updated as the team learns and its process evolves. The Metrics and the threshold values of these metrics are shown in Table 2.

	TNOA		WM	PC	NOCP		
	Avg	Max	Avg	Max	Avg		Max
Version1	35	95	11.56	15	3	3	
	MIT		CCIM		TLO		
	Avg	Max	Avg	Max	Total	Avg	Max
Version1	2.57	6	3.36	56	2500	500	1800

Table 2.	Metric	values o	f Version1

The following are the screenshots of the "MetricAnalyzer" tool for all the six metrics TNOA (Total Number of Attributes), WMPC (Weighted Methods per Class), MIT (Max Inheritance Tree), NOCP (Number of classes Per Package), CCIM (Cyclomatic Complexity), SIZE (Total Lines of Code). The red highlighter shows that the value exceeded the metric threshold value.

😹 Matric Analyzar		- Ið 🔀
Select the Java F	E Upon	
( Project View ) <mark>'An</mark>	Downloads	
	Felder name: O 12 det wents ond Sectire bible80/y Documente2Libra y System in Felder of juppe: All False Toppan Cancel	

Figure 7. Screen shot for selection of project in the "MetricAnalyzer" tool

#### **4.1. TNOA (Total Number of Attributes)**

- It Count the total number of attributes for a class.
- The attributes of a class include the number of static variables and number of instance variables.

Threshold value: A high number of attributes (> 10) probably indicate poor design. Problem: A class with too many attributes may indicate the presence of coincidental cohesion.

Solution: The class requires ecomposition, in order to better manage the complexity of the model.

			Prestory Documented Ibra	y System	Browse
1.05	Analyze Metrics	na sautan Pres	-		
1 View	ATHRY?E METRICS TIME	when viernice [1130			
oc 👘					
24	Class Hame	Yetric Value	Threshold Range	Refactoring Acoice	Dre Caus i gi¥ez
PC	frm#ddEditBarrowed	42	2-5	Decompose the class	C/Documents and Sel
T	trmäddEditBarrowers	26	2.5	Decompose the class	EPUocuments and Sel
3	frmåddEditBooks	34	2.5	Decompose the class	C/Documents and Sel
-	frmAddEditCategory	21	2.5	Decompose the class	C/Documents and Sel
M	frmAddEditCourse	10	2 5	Decompose the class	C/Documents and Sel
	frmAddEditSections	22	2 5	Decompose the class	C/Documents and Sel
	frmAddEditUser	10	2 5	Decompose the class	C/Documents and Sel
	frmAddEdilYear	15	2-5	Decompose the class	C/Documents and Set
		15	2.5	Decompose the class	C/Documents and Set
	frmAuther				
	frmAuthor frmBarrowed	26	2-5	Decompose the class	C/Documents and Set
			2-5 2-5	Decompose the class Decompose the class	C/Documents and Set C/Documents and Set
	frmBarrowed	26		Decompose the class	
	frmBarrowed frmBarrowers	26 26	2.5	Decompose the class Decompose the class	C:Documents and Set C:Documents and Set
	frmBarrowed frmBarrowers frmBooks	26 26 26	2-5 2-5	Decompose the class Decompose the class Decompose the class	C:Documents and Set

Figure 8. Screen shot for TNOA metric in the "MetricAnalyzer" tool

# 4.2. WMPC (Weighted Methods Per Class):-

- It counts all class operations (methods) per class.
- The methods of a class include the number of static methods and number of instance methods.

Threshold value: Between 3 and 7. This indicate that a class has operations, but not too many. Problem: Lack of primitiveness in class operations (inhibiting re-use). Solution: Decompose the class having more methods.

	a thus in state a settion of the	1. 15 8. 1. 1. 5			
t une Jav	a Project Location Cibe	comenta ar o pennicette sitia o	ocon Entstabrary cystem	Bi owrse	_
t View	Analyze Metrics InterPr	ret Metrics   Exit			
c					
A	Class Name	Victor Value	It pohels Hange	Petastnur <u>s</u> Atkise	
c					
	frmSearch	0	3-7		
	frmSearch IrmSections	0	3-7 3-7		
• I		-			
P	trinSections	D	3-/		
r 2	frmSections frmSettings	D D	3-/		
r 2	frmSections frmSettlags frmSplash	0 0	3-/ 3.7 3-7		
r 2	frmSections frmSettings frmSplash frmUsers	0 0 0 0	3-/ 37 3-7 3-7		
r 2	frmSections frmSettings frmSplash frmUsers frmYsear	0 0 0 0 0	3-/ 37 3-7 3-7 3-/ 37		
r 2	FrmSections frmSertings frmSplash frmUsers frmYear MDL_FORM	0 0 0 0 0 0 0	3-/ 3.7 3.7 3.7 3.7 3.7 3.7		
r P	InnSections frmSertlags frmSplash InntBers frmYear MOL_FORM mdHunchons	0 0 0 0 0 0 0 0 0	3./ 37 3.7 3.7 3.7 3.7 3.7 3.7 3.7 3.7		
RC R M	IrmSections fmiSerings fmiSerash IrmBear MOL_FORM mdBeactors mdBeactors mdMethods	0 0 0 0 0 0 0 0 0 0 0	3./ 37 3.7 3.7 3.7 3.7 3.7 3.7 3.7 3.7 3.7		
r P	InnSections TrinSertings TrinSelfush InntSeas TimYear MDL_FORM mdBuections mdBuections mdBlethods mdBSQLStutements	0 0 0 0 0 0 0 0 0 0 0 0 0 0	3.7       3.7       3.7       3.7       3.7       3.7       3.7       3.7       3.7       3.7       3.7       3.7	Drax up the Class & deleg	
r P	InnSections frmSentings frmSelvish Innikers frmYear MDL_FORM nutlitunctions nutlitunctions nutlistic Sul Statements 2435406 Interime	0           0           0           0           0           0           0           0           0           0           0           0           0           0           0           0           0           0           0           0	3./ 3.7 3.7 3.7 3.7 3.7 3.7 3.7 3.7 3.7 3.7		
P	Finisections finisectings finisections finisections finisections mOLECORM millienctions millientiods millisOLSub interits arbitections millisOLSub interits arbitections	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	3./ 3.7 3.7 3.7 3.7 3.7 3.7 3.7 3.7	Drank up the Class & deleg	

Figure 9. Screen shot for WMPC metric in the "MetricAnalyzer" tool

## 4.3. NOCP (Number of classes Per Package):-

• It Count the Number of classes in a package p. More classes means more complex, because the number of potential interactions between objects is higher.

Threshold value: Maximum 12.

Problem: Reduces the comprehensibility of the system, which in turn makes it harder to test, debug and maintain.

Solution: Extract the classes that are co-related and create the new package.

Metric Analyzer					
Select the Java P	roject Location CID	cournents and Dellings', aslympt	ooun entatLibrary Svatem	De ovvise	
Project View And	lyze Metrics	net Metrics Exit			
TLOC	Package Name	xicht: Value	Turestrid Stage	Prefactoring Adviso	1
	l Ibrary System	67	Мак 12	Extract the Package	
MIT					
NOGP					
CCIM					

Figure 10. Screen shot for NOCP metric in the "MetricAnalyzer" tool

## 4.4. MIT (Max Inheritance Tree)

- It calculates the longest path from the class to the root of the inheritance tree.

Threshold value: The value of DIT must be between 0 and 4

Problem: Classes with high DIT indicate that It Compromise encapsulation and increase complexity

Solution: Decompose the class that having the high depth of inheritance tree.

lect the Jan	vs Project Location out	nour tembers of the felling - Version y t	Franciana – na koli abarsa ny Giyosi e na	lino	W30H
ject View	Analyze Metrics InterP	Poet Metrics   Exit			
	earing a searce and a				
FLOC					
NOA	Cass Nari -	X - m Value	U testo di Varige 0.4	Peladum 18 k os	_
VMPC	IrmCourse	1			-
MET	IrmBusikooks	1	0.4		
MIT NOCP	Irrolines	1	0.4		
	Irmines IrmKeya	1	04 04		
400	Irmines Irmies Irmies Irmies	1 1 1	04 04 04	Decompose the Glass	
400	Frmi mes FrmKeya FrmKeya FrmKeiturn	1 1 1 5	04 04 04 04	Discorreposes this Classic Discorreposes this Classic	
400	irmi mee irmkeys irmi ege irmikeium irmikeiumedilooks	4 4 5 5 6	04 04 04 04 04	Decorrigine the Class	
400	Irmines IrmReys IrmReturn IrmReturn IrmReturnetilooks IrmSearch		04 04 04 04 04 04 04		
400	Irmi mes Irmikeys Irmi ages Irmiketum Irmiketum Irmiketum Irmiketum	1	D 4 D 4 D 4 D 4 D 4 D 4 D 4 D 4 D 4	Decorrigine the Class	
400	Frinkeys Frinkeys Frinkelum Frinkelum Frinkeutionske Frinkeutionsk Frinkeutionsk Frinkeutionsk		0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4	Decorrigine the Class	-
400	Irmi maa Irmi aya Irmi aya Irmi aya Irmi kumadi looka Irmiserch Irmiserch Irmisel maa Irmisel maa Irmised maa	1	B 4 B 4 B 4 B 4 B 4 B 4 B 4 B 4 B 4 B 4	Decorrigine the Class	
400	Urril Unix Urrillays Urrillays Urrilla Urrilla Urrilla Urrilla Urrilla Urrilla Urrilla Urrilla Urrilla Urrilla Urrilla	1 0 1 1	04 04 04 04 04 04 04 04 04 04 04 04 04	Decorrigine the Class	
400	Irrificana Irrifikaya Irrifikaya Irrifikatar Irrifikatar Irrifikatar Irrifikatar Irrifikatar Irrifikatar Irrifikatar Irrifikatar Irrifikatar	4 10 4 4	04 04 04 04 04 04 04 04 04 04 04 04 04	Decorrigine the Class	
1009	Urril Unix Urrillays Urrillays Urrilla Urrilla Urrilla Urrilla Urrilla Urrilla Urrillays Urrillays Urrillays	1 0 1 1	04 04 04 04 04 04 04 04 04 04 04 04 04	Decorrigine the Class	-

Figure 11. Screen shot for MIT metric in the "MetricAnalyzer" tool

## 4.5. CCIM (Cyclomatic Complexity):-

- It measures the number of linearly independent paths through a program module. i.e, the amount of decision logic in a single software module. It is calculated per class.
- Counts the number of flows through a piece of code. Each time a branch occurs (if, for, while, do, case, catch and the ?: ternary operator, as well as the && and || conditional logic operators in expressions) this metric is incremented by one

Threshold value: The acceptable range must be between 1and10 Problem: A method with a CC value greater than 10 indicates the high code complexity. Solution: Extract the Class or Extract Method

Method Name	Nebic Value			
initiComboRoxes 3		Inteshold Ivange	Refectoring Advice	
	1	1-10		-
resetTield 1		1-10		_
updateField 7	,	1-10		
resetTieki 1		1-10		
upriatel reidi 8	1	1-10		
resett veld		1-10		
upitateheid 1	2	1-10	Extract the Class or Extract.	
resetineki 1	1	1 10		
upsiateiheid /	•	1 10		
reseltneki 1		1 10		
updateField G	•	1 10		
resetField 1		1 10		
updateField 6	•	1 10		-
resetField 1		1-10		

Figure 12. Screen shot for CCIM metric in the "MetricAnalyzer" tool

### 4.6. SIZE (Total Lines of Code):-

There are two variants of size metric they are function points (fp) and Total Lines of code (TLOC) metric. TLOC is considered in our paper.

- TLOC metric will count the non-blank and non-comment lines in a class.
- This is the simplest way to measure the size of the system.

Threshold value: The acceptable range is between Max 750

Problem: A class with TLOC value greater than 750 increases overall size of the system and reduces the comprehensibility.

Solution: Split up the class and delegate its responsibilities.

		"Executionand Bettings":	e study Documents (Library 8)	iste m	Browse.	
	warroject Location	iberer ferns er a seinne a.	Sole y Doctin Entrolling y 8,	ete n	provise.	
ct Vlew	🕺 Analyze Motrics 🕴 Ente	orProt Metrics Exit				
oc	Class Harra	Metric Valua	Threathr o Range	Defactoring Advin	c .	
O A	IrmDueBooks	83	< 750		-	
ec.	Implaces	83	< 750			
т	frmKeys	71	< 750			
0	frmLogin	93	< 750			
-M	ButtonHandler	11	< 750			
	moBeturn	183	< 750			
	frmBeturnedBooks	83	< 750			
	frmSearch	20	< 750			
	frmSections	126	< 750			
	frmSattlings	61	< 750			
	frmSplash	10	< 750			
	frmUsers	152	< 750			
	funition	101	= 750		-	
	MOL FORM	1453	× 750	Split the Close		
	mdlf uncliens	89	<b>= 750</b>			
	mdiMelliods	11	1 / 50			
	and 1000 bit 801 when an end of	47	- 440		*	

Figure 13. Screen shot for SIZE metric in the "MetricAnalyzer" tool

# **5. APPLICATION PHASE**

Software Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves the internal structure [6]. It improves the design of the software by eliminating redundancy and reducing complexity. The resulting software is easier to understand and maintain.

Software refactoring is a technique to enhance the maintainability of software, improve reusability and understandability of the software. Our tool "MetricAnlyzer" applies the Object Oriented metrics on the code base and these metric values are then interpreted. Then various refactoring techniques were used to improve the code design and along with that we also studied the impact of refactoring on the software quality through various metrics [5]. The red highlighter showed the values which exceeded the metric threshold value and this made us to refactor those parts of the code base and improve the quality of software.

In the refactoring process the metric values are improved by moving close to the threshold values. Version1 values are the metric values before refactoring and Version2 and Version3 values are the metric values after refactoring which are given in the following tables for the six metrics taken.

	TNOA		WM	PC	NOCP		
	Avg	Max	Avg	Max	Avg	Max	
Version1	35	95	11.56	15	3	3	
Version2	16.1	39	5.89	15	9	9	
Version3	11.3	41	5.5	15	12	12	

Table 3. Comparison of Metric values of Version1, Version2 and Version3 for the first three metrics

	MIT		CC	CCIM		TLOC		
	Avg	Max	Avg	Max	Total	Avg	Max	
Version1	2.57	6	3.36	56	2500	500	1800	
Version2	4.11	6	2.53	26	2300	290	900	
Version3	4.5	6	2.43	13	1524	127	508	

Table 4. Comparison of Metric values of Version1, Version2 and Version3

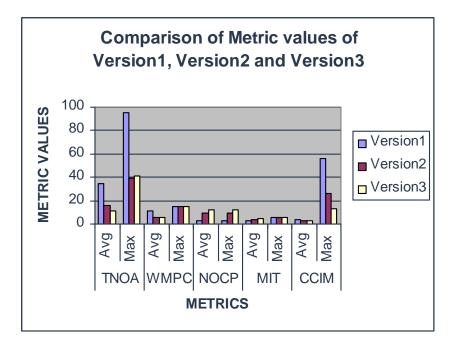


Figure 13. Graph depicting the Metric values of Version1, Version2 and Version3

## **6.** CONCLUSIONS

In Agile software development, refactoring helps to improve software quality. Our object oriented software metric tool "MetricAnalyzer" proposed in this paper showed us great results after refactoring the code compared to the original code base of different projects. As we have shown the graphical result, this tool was very much useful in eliminating code smells from the code and maintains threshold values of the considered object oriented metrics.

## REFERENCES

- [1] M.M Lehman, D.e Perry, J.F.Ramil, W.M Turksi, and P.Wernick, "Metrics and Laws of Software Evolution nineties view." In Proceedings of the 4th International Symposium on Software Metrics, Metrics, Albuquerque, New Mexico, 2000. IEEE
- [2] Markuz Pizka, "Straightening Spaghetti-Code with Refactoring"
- [3] Eva van Emden, Leon Moonden, "Java Quality Assurance by Detecting Code Smell"
- [4] R.S. Arnold, "An Introduction to Software Restructuring," Tutorial on Software Restructuring, R.S. Arnold, ed., 1986.
- [5] D.M. Coleman, D. Ash, B. Lowther, and P.W. Oman, "Using Metrics to Evaluate Software System Maintainability," Computer, vol. 27, no. 8, pp. 44-49, Aug. 1994.
- [6] N. Van Eetvelde and D. Janssens, "A Hierarchical Program Representation for Refactoring," Proc. UniGra'03 Workshop, 2003.

## Authors

Mrs. Veerapaneni Esther Jyothi is a Microsoft certified professional, currently pursuing Ph.D. She is working as an Asst. Professor in the department of Computer Applications, Velagapudi Siddhartha Engineering College since 2008 and also has Industrial experience. She has published papers in reputed international conferences recently. Her areas of interest include Software Engineering, Object Oriented Analysis and Design and DOT NET.

Dr. K. Nageswara Rao is currently working as Professor and Head in the department of Computer Science Engineering, Prasad V. Potluri Siddhartha Institute of Technology, Kanuru, Vijayawada-7. He has an excellent academic and research experience. He has contributed various research papers in the journals, conferences of International/national.

Mr. Kaitepalli Srikanth is Asst. System Engineer in TCS (Tata Consultancy Services). He was recruited as a part of campus placements from Velagapudi Ramakrishna Siddhartha Engineering College, Vijayawada and is going to complete his master's degree from the department of Computer Applications.



