

STATUS OF THESIS

Title of thesis: IDENTIFICATION AND QUANTIFICATION OF VARIABILITY MEASURES AFFECTING CODE REUSABILITY IN OPEN SOURCE ENVIRONMENT

I, FAZAL-E-AMIN

hereby allow my thesis to be placed at the Information Resource Center (IRC) of Universiti Teknologi PETRONAS (UTP) with the following conditions:

- 1. The thesis becomes the property of UTP
2. The IRC of UTP may make copies of the thesis for academic purposes only.
3. This thesis is classified as

Confidential
Non-confidential

If this thesis is confidential, please state the reason:

Blank lines for stating reasons for confidentiality.

The contents of the thesis will remain confidential for years.

Remarks on disclosure:

Blank lines for remarks on disclosure.

Signature of Author

Permanent address:

128-Khayaban Garden, Sargodha Road

Faisalabad, Pakistan

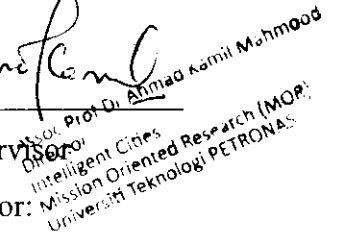
Date: 13/3/2012

Endorsed by Signature of Supervisor Assoc. Prof. Dr. Ahmad Kamil Mahmood

Name of Supervisor: Assoc. Prof. Dr. Ahmad Kamil

Bin Mahmood

Date: 13/3/2012



UNIVERSITI TEKNOLOGI PETRONAS

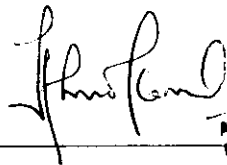
IDENTIFICATION AND QUANTIFICATION OF VARIABILITY MEASURES
AFFECTING CODE REUSABILITY IN OPEN SOURCE ENVIRONMENT

By

FAZAL-E-AMIN

The undersigned certify that they have read, and recommend to the Postgraduate Studies Programme for acceptance this thesis for the fulfilment of the requirements for the degree stated.

Signature:



Assoc Prof Dr Ahmad Kamil Mahmood
Director
Intelligent-Cities
Mission Oriented Research (MOR)
Universiti Teknologi PETRONAS

Main Supervisor:

ASSOC. PROF. DR. AHMAD KAMIL BIN MAHMOOD

Signature:

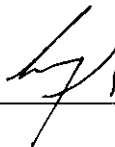


Dr Alan Oxley
Professor
Computer & Information Sciences Department
Universiti Teknologi PETRONAS

Co-Supervisor:

PROF. DR. ALAN OXLEY

Signature:



Assoc Prof Dr Mohd Fadzil Bin Hassan
Department of Computer & Information Sciences
Universiti Teknologi PETRONAS

Head of Department:

ASSOC. PROF. DR. MOHD FADZIL BIN HASSAN

Date:

8/3/2012

IDENTIFICATION AND QUANTIFICATION OF VARIABILITY
MEASURES AFFECTING CODE REUSABILITY IN OPEN SOURCE
ENVIRONMENT

by

FAZAL-E-AMIN

A Thesis

Submitted to the Postgraduate Studies Programme

As a Requirement for the Degree of

DOCTOR OF PHILOSOPHY
INFORMATION TECHNOLOGY
UNIVERSITI TEKNOLOGI PETRONAS
BANDAR SERI ISKANDAR,
PERAK

March 2012

DECLARATION OF THESIS

Title of thesis

IDENTIFICATION AND QUANTIFICATION OF
VARIABILITY MEASURES AFFECTING CODE
REUSABILITY IN OPEN SOURCE ENVIRONMENT

I,

FAZAL-E-AMIN

hereby declare that the thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at UTP or other institutions.

Witnessed by

Signature of Author

Permanent address:

128-Khayaban Garden, Sargodha Road

Faisalabad, Pakistan

Date : 13/3/2012

Signature of Supervisor

Name of Supervisor:

Assoc. Prof. Dr. Ahmad Kamil

Bin Mahmood

Date : 13/3/2012

Assoc Prof Dr Ahmad Kamil Mahmood
Director
Intelligent Cities
Mission Oriented Research (MOR)
Universiti Teknikal Malaysia TRONAS

DEDICATION

I dedicate this thesis to my beloved parents, fiancée, siblings, and friends who have supported me throughout the time of this degree and life. I also dedicate this thesis to my teachers who have made me able to reach this level.

ACKNOWLEDGEMENT

I thank Almighty Allah for giving me the strength to complete this endeavor. I would like to thank all those who have directly or indirectly helped me to complete this thesis. I would like to thank my supervisor Assoc. Prof. Dr. Ahmad Kamil Mahmood for his support and advices which has enabled me to shape up this work. I express my gratitude to my co-supervisor Prof. Dr. Alan Oxley for his kind support and valuable time to review the research work.

I would like to thank all my friends, fellow researchers and colleagues for making my time memorable. It would be a bit difficult without their support to settle here. Last, but not the least, I would like to thank my family, especially my father, Hamid Amin Soofi for guiding me and giving me a vision and mission for life.

ABSTRACT

Open source software (OSS) is one of the emerging areas in software engineering, and is gaining the interest of the software development community. OSS was started as a movement, and for many years software developers contributed to it as their hobby (non commercial purpose). Now, OSS components are being reused in CBSD (commercial purpose). However, recently, the use of OSS in SPL is envisioned recently by software engineering researchers, thus bringing it into a new arena. Being an emerging research area, it demands exploratory study to explore the dimensions of this phenomenon. Furthermore, there is a need to assess the reusability of OSS which is the focal point of these disciplines (CBSE, SPL, and OSS). In this research, a mixed method based approach is employed which is specifically ‘partially mixed sequential dominant study’. It involves both qualitative (interviews) and quantitative phases (survey and experiment). During the qualitative phase seven respondents were involved, sample size of survey was 396, and three experiments were conducted. The main contribution of this study is results of exploration of the phenomenon ‘reuse of OSS in reuse intensive software development’. The findings include 7 categories and 39 dimensions. One of the dimension factors affecting reusability was carried to the quantitative phase (survey and experiment). On basis of the findings, proposal for reusability attribute model was presented at class and package level. Variability is one of the newly identified attribute of reusability. A comprehensive theoretical analysis of variability implementation mechanisms is conducted to propose metrics for its assessment. The reusability attribute model is validated by statistical analysis of 103 classes and 77 packages. An evolutionary reusability analysis of two open source software was conducted, where different versions of software are analyzed for their reusability. The results show a positive correlation between variability and reusability at package level and validate the other identified attributes. The results would be helpful to conduct further studies in this area.

ABSTRAK

Perisian sumber terbuka (OSS) ialah salah satu bidang yang baru muncul dalam kejuruteraan perisian dan sedang mendapat perhatian di kalangan komuniti pembangunan perisian. OSS dimulakan sebagai satu gerakan dan untuk beberapa tahun para pembangun perisian menyumbang kepadanya sebagai satu hobi (tujuan bukan komersil). Kini, komponen-komponen OSS diguna semula dalam CBSD (tujuan komersil). Bagaimanapun, sejak akhir-akhir ini, penggunaan OSS di dalam SPL telah dibayangkan oleh para penyelidik kejuruteraan perisian, sekaligus membawanya ke satu arena baru. Sebagai satu bidang penyelidikan baru, memerlukan kajian eksploratori untuk mencari dimensi-dimensi fenomena ini. Tambahan pula, terdapat keperluan untuk menilai penggunaan semula OSS yang mana merupakan titik fokus disiplin-disiplin ini (CBSE, SPL dan OSS). Dalam kajian ini, pendekatan berasaskan kaedah campuran pekerjaan yang khusus 'sebahagian campuran kajian berurutan dominan'. Ia melibatkan kedua-dua kualitatif (temubual) dan fasa kuantitatif (kajian dan eksperimen). Semasa fasa kualitatif tujuh orang responden terlibat, saiz sampel kajian adalah 396, dan tiga eksperimen telah dijalankan. Sumbangan utama kajian ini adalah hasil penerokaan 'guna semula OSS dalam pembangunan perisian penggunaan semula intensif' fenomena. Penemuan merangkumi 7 kategori dan 39 dimensi. Salah satu dari faktor-faktor yang mempengaruhi penggunaan semula dibawa ke fasa kuantitatif (kaji selidik dan eksperimen). Atas dasar penemuan, cadangan untuk model atribut boleh gunapakai telah dibentangkan di peringkat kelas dan pakej. Kebolehubahan adalah salah satu sifat yang baru yang dikenal pasti boleh gunapakai. Sebuah analisis teoretikal yang komprehensif ke atas mekanisma pelaksanaan kepelbagaian fungsi ini dijalankan bagi mencadangkan metrik untuk penilaiannya. Sebuah model ciri penggunaan semula telah dicadangkan dan disahkan dengan analisis statistik ke atas 103 kelas dan 77 pakej. Satu analisis penggunaan semula secara evolusi ke atas dua perisian sumber terbuka telah dijalankan, di mana perisian pelbagai versi telah dianalisa untuk menentukan kebolegunaan semulanya.

Hasil kajian menunjukkan hubungan yang positif antara kebolehubahan dan gunapakai semula di peringkat pakej dan mengesahkan sifat-sifat lain yang dikenal pasti. Keputusan akan membantu anda untuk menjalankan kajian selanjutnya dalam bidang ini.

In compliance with the terms of the Copyright Act 1987 and the IP Policy of the university, the copyright of this thesis has been reassigned by the author to the legal entity of the university,

Institute of Technology PETRONAS Sdn Bhd.

Due acknowledgement shall always be made of the use of any material contained in, or derived from, this thesis.

© Fazal-e-Amin, 2011

Institute of Technology PETRONAS Sdn. Bhd.

All rights reserved.

TABLE OF CONTENTS

ACKNOWLEDGEMENT	vi
ABSTRACT.....	vii
ABSTRAK.....	viii
TABLE OF CONTENTS.....	xi
CHAPTER 1	1
1. INTRODUCTION	1
1.1 Overview	1
1.2 Software and Engineering of Software	1
1.3 Software Reuse.....	2
1.3.1 Objectives of Reuse	4
1.4 Background Research.....	4
1.4.1 Open Source Software and Reuse-Intensive Software Development.....	4
1.4.2 Assessment of Reusability	5
1.4.3 Selection of Components.....	7
1.4.4 Software Variability and Implementation Mechanisms	7
1.5 Problem Statement	8
1.6 Research Questions	9
1.7 Research Objectives	9
1.8 Research Activities.....	9
1.9 Contribution	11
1.10 Thesis Structure.....	11
2. LITERATURE REVIEW	13
2.1 Overview	13
2.2 Open Source Software.....	13

2.2.1	Use of OSS in CBSE and SPL.....	14
2.2.2	OSS Definition.....	14
2.2.3	Advantages of OSS.....	16
2.2.4	Drawbacks of OSS.....	17
2.3	Related Works on Using OSS.....	18
2.4	Reusable Software Assets.....	20
2.5	Reuse-intensive Software Development.....	21
2.5.1	Software Product Line.....	22
2.5.2	Component Based Software Development.....	23
2.5.3	CBSE Development Generic Activities.....	23
2.6	Open Source Components Based Product Lines.....	23
2.7	Literature Review Process.....	24
2.8	Classification of the Approaches.....	27
2.8.1	Types.....	27
2.8.2	Applicability.....	30
2.8.3	Validation Types.....	33
2.8.4	Synthesis of Literature Review.....	37
2.9	Literature Review on Variability.....	37
2.9.1	Variability Types (with respect to effect).....	37
2.9.2	Variability Types (with respect to functionality).....	39
2.9.3	Variability Scope.....	39
2.9.4	Binding Time of Variability.....	40
2.10	Software Engineering Measurements and Metrics.....	41
2.10.1	Reuse Metrics and Models.....	42
2.11	Identification of Need to Conduct This Study.....	42
2.12	Summary.....	43
CHAPTER 3.....		45
3.	RESEARCH METHODOLOGY.....	45
3.1	Overview.....	45

3.2	Philosophical Basis of Research	45
3.3	Basis of Mixed Method Research	47
3.3.1	Purpose of Mixed Methods.....	48
3.3.2	Types of Mixed Method Studies.....	48
3.3.3	Type of Mixed Method Chosen for Study	49
3.4	Research in Software Engineering	50
3.4.1	Mixed Methods in Software Engineering	51
3.5	Purpose Based Classification of Research Studies	51
3.5.1	Exploratory Research Studies.....	52
3.5.2	Descriptive Research Studies.....	52
3.5.3	Explanatory Research Studies.....	52
3.5.4	Emancipatory Research Studies.....	52
3.5.5	Interpretive Research Studies	53
3.6	Research Design.....	53
3.7	Qualitative Methods	54
3.7.1	Interview	55
3.7.2	Types of Interview	55
3.7.3	Question Formulation Process for Interview	57
3.7.4	Respondents' Profiles	58
3.7.5	Interview Guide	59
3.8	Qualitative Analysis (Content Analysis).....	61
3.8.1	Word Cloud.....	63
3.8.2	atlas.ti.....	63
3.9	Quantitative Methods	65
3.9.1	Survey	65
3.9.2	Quantitative Analysis.....	69
3.9.3	Statistical Techniques	70
3.9.4	Experiments	71
3.9.5	Statistical Analysis Tool	73
3.10	OSS Selection.....	73

3.10.1	Jasmin	73
3.10.2	pBeans.....	74
3.11	Metrics Calculation Tool.....	74
3.12	Goal Question Metric Approach	74
3.13	Validity of Research Results.....	75
3.13.1	Validity of Qualitative Results.....	76
3.13.2	Validity of Quantitative Results.....	78
3.13.3	Validity of Mixed Method Results	78
3.14	Validation of Findings in Context of This Study	81
3.14.1	Validation of Qualitative Findings.....	81
3.14.2	Validation of Quantitative Findings.....	82
3.14.3	Validation of Mixed Method Findings	83
3.15	Summary	85
CHAPTER 4	87
4.	IDENTIFICATION OF CATEGORIES & DIMENSIONS IN REUSING OSS..	87
4.1	Overview	87
4.2	Categories & Dimensions of Reusing OSS in Reuse Intensive Environment	87
4.2.1	Challenges in OSS	88
4.2.2	Current Reuse Practices	94
4.2.3	Using OSS in SPL.....	96
4.2.4	Role of OSS in promoting reuse	98
4.2.5	Factors Affecting Reusability	100
4.2.6	Desirable Characteristics of OSS.....	104
4.2.7	Suggestions	108
4.2.8	Other Considerations	110
4.3	Attribute Ranking Survey.....	111
4.3.1	Theoretical Analysis of Variability Implementation Mechanism.....	121
4.4	Summary	128
CHAPTER 5	129
5.	CORRELATION STUDY OF FACTORS AFFECTING REUSABILITY	129

5.1	Overview	129
5.2	Reusability Assessment Conceptual Model	129
5.3	Proposed Class Level Reusability Attribute Model	130
5.3.1	Reusability GQM Model.....	131
5.3.2	Attributes.....	132
5.3.3	Sub Attributes	138
5.3.4	Class Level Metrics.....	139
5.4	Metrics Threshold Values and Equations.....	141
5.5	Proposed Package Level Reusability Attribute Model	143
5.5.1	Package Level Metrics	146
5.6	Reusability Assessment at Class Level	149
5.6.1	Metrics and Attributes Analysis.....	149
5.6.2	Attribute Analysis	158
5.7	Reusability Assessment at Package Level	163
5.7.1	Attribute Analysis	164
5.8	Evolutionary Reusability Analysis at Package Level	168
5.8.1	Reusability Analysis of Jasmin.....	168
5.8.2	Reusability Analysis of pBeans	181
5.9	Summary	196
CHAPTER 6		199
6.	DISCUSSIONS	199
6.1	Overview	199
6.2	Key Findings of Research	199
6.2.1	Review of Reusability Assessment Approaches.....	199
6.2.2	Analysis of Variability Implementation Mechanisms	200
6.2.3	Identification of Challenges in OSS	201
6.2.4	Identification of Current Reuse Practices	202
6.2.5	Using OSS in an SPL.....	203
6.2.6	Role of OSS in Promoting Reuse.....	204

6.2.7	Identification of Factors Affecting Reusability	204
6.2.8	Identification of Desirable Characteristics of OSS Components.....	206
6.2.9	Proposed Reusability Assessment Model	207
6.2.10	Statistical Results.....	208
6.2.11	Evolutionary Analysis of Reusability	209
6.3	Synthesis of the Qualitative and Quantitative Results	209
6.3.1	Flexibility.....	210
6.3.2	Maintainability.....	210
6.3.3	Portability.....	211
6.3.4	Scope Coverage	211
6.3.5	Stability	211
6.3.6	Understandability.....	212
6.3.7	Usage History.....	212
6.3.8	Variability	212
6.3.9	Documentation.....	213
6.4	Key Findings and Implications	213
6.4.1	Review of Literature	213
6.4.2	Methodological Contribution.....	215
6.4.3	Practical Contribution	216
6.5	Summary	221
CHAPTER 7		223
7.	CONCLUSIONS	223
7.1	Research Summary.....	223
7.2	Achievement of Research Objectives	224
7.3	Contributions.....	228
7.4	Limitations	228
7.5	Future Directions.....	229
REFERENCES		230
List of Publications		244
APPENDICES		246

A. Interview guide	246
B. Code of Software used to calculate attribute values	248
C. Pseudo Code to calculate package level attribute values	252
D. Detailed Component Specifications.....	254
E. Detailed Package Specifications.....	260
F. Reusability Attribute Values.....	264
G. Questionnaire	274
H. List of Sources	278
I. Package Analysis (jasmin and pBeans).....	279
J. Critical Values of the Pearson Correlation Coefficient r	304

List of Figures

Figure 1.1 Dimensions of software reuse	3
Figure 1.2 Generic research activities.....	10
Figure 2.1 Literature review process- adapted from (Brereton et al., 2007)	26
Figure 2.2 Year wise search results	26
Figure 2.3 Types of Approaches	27
Figure 2.4 Applicability of approaches - paradigms.....	31
Figure 2.5 Applicability of approaches - programming languages.....	32
Figure 2.6 Applicability of approaches - level.....	32
Figure 2.7 Types of validation	35
Figure 2.8 Relationship of Variability types and scope.....	40
Figure 3.1 Epistemology, paradigm, methodology and methods [91].....	46
Figure 3.2 Typologies of mixed method studies [92].....	49
Figure 3.3 Research design	54
Figure 3.4 Abstraction process	62
Figure 3.5 Content analysis process.....	62
Figure 3.6 Word cloud of interview transcription	64
Figure 3.7 Experience of population in years	67
Figure 4.1 Frequency distribution of the scales assigned to flexibility	112
Figure 4.2 Frequency distribution of the scales assigned to scope coverage.....	113
Figure 4.3 Frequency distribution of the scales assigned to portability	114
Figure 4.4 Frequency distribution of the scales assigned to maintainability.....	115
Figure 4.5 Frequency distribution of the scales assigned to variability.....	116
Figure 4.6 Frequency distribution of the scales assigned to understandability	117
Figure 4.7 Frequency distribution of the scales assigned to documentation	118
Figure 4.8 Frequency distribution of the scales assigned to usage history	119
Figure 4.9 Frequency distribution of the scales assigned to stability	120
Figure 5.1 Conceptual model of reusability assessment.....	130
Figure 5.2 Relationship of variability with metrics	137
Figure 5.3 Reusability attribute model (class level)	141
Figure 5.4 Reusability attribute model (package level).....	148

Figure 5.5 Scatter plot of MI vs. Maintainability	150
Figure 5.6 Scatter plot of CC vs. Maintainability	151
Figure 5.7 Scatter plot of CBO vs. Flexibility	152
Figure 5.8 Scatter plot of LCOM vs. Flexibility.....	153
Figure 5.9 Scatter plot of CBO vs. Understandability	154
Figure 5.10 Scatter plot of LCOM vs. Understandability.....	155
Figure 5.11 Scatter plot of Comments vs. Understandability	156
Figure 5.12 Scatter plot of LOC vs. Understandability	156
Figure 5.13 Scatter plot of NOM vs. Understandability	157
Figure 5.14 Scatter plot of flexibility vs. reusability	158
Figure 5.15 Scatter plot of Understandability vs. Reusability	159
Figure 5.16 Scatter plot of scope-coverage vs. reusability	160
Figure 5.17 Scatter plot of variability vs. reusability.....	161
Figure 5.18 Scatter plot of maintainability vs. reusability.....	162
Figure 5.19 Scatter plot of portability vs. reusability	162
Figure 5.20 Scatter plot of flexibility vs. reusability	164
Figure 5.21 Scatter plot of variability vs. reusability.....	165
Figure 5.22 Scatter plot of portability vs. reusability	166
Figure 5.23 Scatter plot of maintainability vs. reusability.....	166
Figure 5.24 Scatter plot of understandability vs. reusability	167
Figure 5.25 Reusability and its attribute values for packege-1.....	170
Figure 5.26 Graph plot of values of LOC and understandability package-1	171
Figure 5.27 Graph plot of values of NOM and understandability package-1.....	171
Figure 5.28 Graph plot of values of comments and understandability package-1.....	172
Figure 5.29 Graph plot of values of MI and maintainability package-1	173
Figure 5.30 Graph plot of values of CC and maintainability package-1	173
Figure 5.31 Reusability and its attribute values for packege-2.....	175
Figure 5.32 Graph plot of values of MI and maintainability package-2.....	176
Figure 5.33 Graph plot of values of CC and maintainability package-2	176
Figure 5.34 Reusability and its attribute values for packege-3.....	177
Figure 5.35 Graph plot of values of LOC and understandability package-3	178
Figure 5.36 Graph plot of values of NOM and understandability package-3.....	179

Figure 5.37 Graph plot of values of comments and understandability package-3.....	179
Figure 5.38 Graph plot of values of comments and understandability package-3.....	180
Figure 5.39 Graph plot of values of CC and maintainability package-3	181
Figure 5.40 Reusability and its attribute values for package-2.....	182
Figure 5.41 Graph plot of values of LOC and understandability package-2.....	184
Figure 5.42 Graph plot of values of NOM and understandability package-2.....	184
Figure 5.43 Graph plot of values of comments and understandability package-2.....	185
Figure 5.44 Graph plot of values of MI and maintainability package-2.....	186
Figure 5.45 Graph plot of values of CC and maintainability package-3	186
Figure 5.46 Reusability and its attribute values for package-3.....	187
Figure 5.47 Graph plot of values of LOC and understandability package-3	189
Figure 5.48 Graph plot of values of NOM and understandability package-3.....	189
Figure 5.49 Graph plot of values of comments and understandability package-3.....	190
Figure 5.50 Graph plot of values of MI and maintainability package-3.....	191
Figure 5.51 Graph plot of values of CC and maintainability package-4	191
Figure 5.52 Reusability and its attribute values for packege-4.....	192
Figure 5.53 Graph plot of values of LOC and understandability package-4.....	193
Figure 5.54 Graph plot of values of NOM and understandability package-4.....	194
Figure 5.55 Graph plot of values of comments and understandability package-4.....	194
Figure 5.56 Graph plot of values of MI and maintainability package-4.....	195
Figure 5.57 Graph plot of values of CC and maintainability package-5	196
Figure 7.1 Summary of the research work.....	225
Figure 7.2 Summary of findings	227

List of Tables

Table 2.1 Advantages of OSS	17
Table 2.2 Drawbacks of OSS.....	18
Table 2.3 Review Protocol.....	26
Table 2.4 Applicability of approaches	33
Table 2.5 Types of validation	36
Table 3.1 Information about the respondents	60
Table 3.2 Means used to conduct interviews.....	60
Table 3.3 Survey sample size and related figures.....	69
Table 3.4 Details of packages Jasmin and pBeans	74
Table 3.5 Experiment sample comparison-1	82
Table 3.6 Experiment sample comparison-2	82
Table 3.7 Experiment sample comparison-3	83
Table 4.1 Categories and their description	88
Table 4.2 Sub categories of challenges in OSS	92
Table 4.3 Sub categories of current reuse practices.....	95
Table 4.4 Sub categories of using OSS in an SPL.....	97
Table 4.5 Sub categories of role of OSS in promoting reuse.....	99
Table 4.6 Identified factors and representative quotes	102
Table 4.7 Desirable characteristics of OSS and representative quotes.....	106
Table 4.8 Suggestions and representative quotes	109
Table 4.9 Other considerations and representative quotes.....	110
Table 4.10 Flexibility rankings	112
Table 4.11 Scope Coverage rankings.....	113
Table 4.12 Portability rankings.....	114
Table 4.13 Maintainability rankings	115
Table 4.14 Variability rankings	116
Table 4.15 Understandability rankings	117
Table 4.16 Documentation rankings	118
Table 4.17 Usage History rankings.....	119

Table 4.18 Stability rankings	120
Table 4.19 Characteristics of aggregation	123
Table 4.20 Characteristics of inheritance.....	124
Table 4.21 Characteristics of parameterization/ generalization.....	125
Table 4.22 Characteristics of overloading	126
Table 4.23 Characteristics of AOP	128
Table 5.1 GQM Model Class Reusability.....	131
Table 5.2 GQM for class variability	137
Table 5.3 Reusability attributes sub-attributes and metrics	143
Table 5.4 Package level attributes, sub attributes and metrics	144
Table 5.5 GQM Model Package Reusability	145
Table 5.6 Pearson's correlation values MI, CC and maintainability.....	151
Table 5.7 Pearson's correlation values CBO, LCOM and Understandability.....	154
Table 5.8 Pearson's correlation values Understandability and its attributes.....	157
Table 5.9 Pearson's correlation values of Reusability and its attributes (class level)	163
Table 5.10 Pearson's correlation values of reusability & attributes (package level) .	168
Table 5.11 Version wise values of reusability and its attributes (package-1)	169
Table 5.12 Version wise values of understandability and its attributes (package-1).	170
Table 5.13 Version wise values of maintainability and its attributes (package-1)	172
Table 5.14 Version wise values of reusability and its attributes (package-2)	174
Table 5.15 Version wise values of maintainability and its attributes (package-2)	175
Table 5.16 Version wise values of reusability and its attributes (package-3)	177
Table 5.17 Version wise values of understandability and its attributes (package-3).	178
Table 5.18 Version wise values of maintainability and its attributes (package-3)	180
Table 5.19 Version wise values of reusability and its attributes (package-2)	182
Table 5.20 Version wise values of understandability and its attributes (package-2).	183
Table 5.21 Version wise values of maintainability and its attributes (package-2)	185
Table 5.22 Version wise values of reusability and its attributes (package-3)	187
Table 5.23 Version wise values of understandability and its attributes (package-3).	188
Table 5.24 Version wise values of maintainability and its attributes (package-3)	190
Table 5.25 Version wise values of reusability and its attributes (package-4)	192
Table 5.26 Version wise values of understandability and its attributes (package-4).	193

Table 5.27 Version wise values of maintainability and its attributes (package-4)	195
Table 6.1 Comparison of reviewed studies.....	200
Table 6.2 Comparison of findinds-1	201
Table 6.3 Comparison of findings-2	202
Table 6.4 Comparison of findinds-3	202
Table 6.5 Comparison of findings-4	203
Table 6.6 Comparison of findings-5	204
Table 6.7 Comparison of findings-6	205
Table 6.8 Comparison of findings -7	205
Table 6.9 Comparison of findings-8	206
Table 6.10 Comparison of findings-9	206

List of Abbreviations

Serial No.	Acronym	Meaning
1	A	Abstractness
2	CBO	Coupling Between Objects
3	CBSD	Component Based Software Development
4	CBSE	Component Based Software Engineering
5	CC	Cyclomatic Complexity
6	DIT	Depth of Inheritance Tree
7	GQM	Goal Question Metric
8	I	Instability
9	LCOM	Lack of Cohesion Metric
10	LOC	Lines of Code
11	MI	Maintainability Index
12	NOC	Number of Child Classes
13	NOM	Number of Methods
14	OSI	Open Source Initiative
15	OSS	Open Source Software
16	PL	Product Line
17	SE	Software Engineering
18	SPL	Software Product Line
19	SPLE	Software Product Line Engineering

CHAPTER 1

INTRODUCTION

Every program has (at least) two purposes: the one for which it was written, and the other for which it wasn't. (Alan J. Perlis, 1922-1990)

1.1 Overview

In this chapter a brief account of software and software engineering is presented. The chapter contains a detailed overview of software reuse and its dimensions. Reuse intensive software development and the inclusion of open source software during this development are highlighted. The background of problem, research question, objectives, research approach, and contributions are presented in this chapter.

1.2 Software and Engineering of Software

In 9th century a Muslim mathematician Muhammad Al-Khwarizmi presented the concept of algorithm [1]. An 'algorithm' is a sequence of steps to solve a problem. The concept of algorithm is implemented using computers in the form of computer program in 1950's. A computer program consists of a sequence of instructions given to hardware to perform a specific task. During this evolution, the term 'software' was used for the collection of computer programs and related documentation.

The term 'software engineering' was coined in 1968 during the first NATO software engineering conference [2]. Afterwards in the following decades it became more popular and researchers have come to realize its complexity and the difficulties due to technical, social and hardware related issues.

Software engineering is defined as “the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software” [3].

The primary aim of software engineering knowledge is the development of quality software within the resource and budgetary limits. Software reuse, being the process of creating new software by using existing software artifacts [4], is one of the ways to achieve the above mentioned goals.

1.3 Software Reuse

Software reuse has different facets. These facets can be categorized according to the substance, scope, mode, technique, intention and product [5].

The substance refers to the core of the reuse; that what is being reused. These substances include ideas, artifacts, and procedures [5]. Artifacts are not limited to but may include a piece of code, an object oriented class or a module. The idea of reuse is the case of reusing generic entity such as an algorithm. Procedures are the process elements that are reused in different stages of software development, such as the procedure to carry out inspections.

The scope or domain scope [6] of reuse is either vertical or horizontal. The reuse of generic components across different domains is horizontal reuse. The vertical reuse refers to the reuse of software artifacts within the same domain.

The approach [7] or mode [5] or management [6] of reusing software may include planned or opportunistic reuse. In planned reuse a software artifact is developed while keeping in mind that this artifact would be reused in future.

The reuse techniques [7] include compositional and generative based reuse. In compositional reuse new systems are composed using existing components. The generative reuse technique is more like knowledge reuse such as the application generators. Specifications of the new system are written in domain specific language, while generators translate the specifications into codes [8].

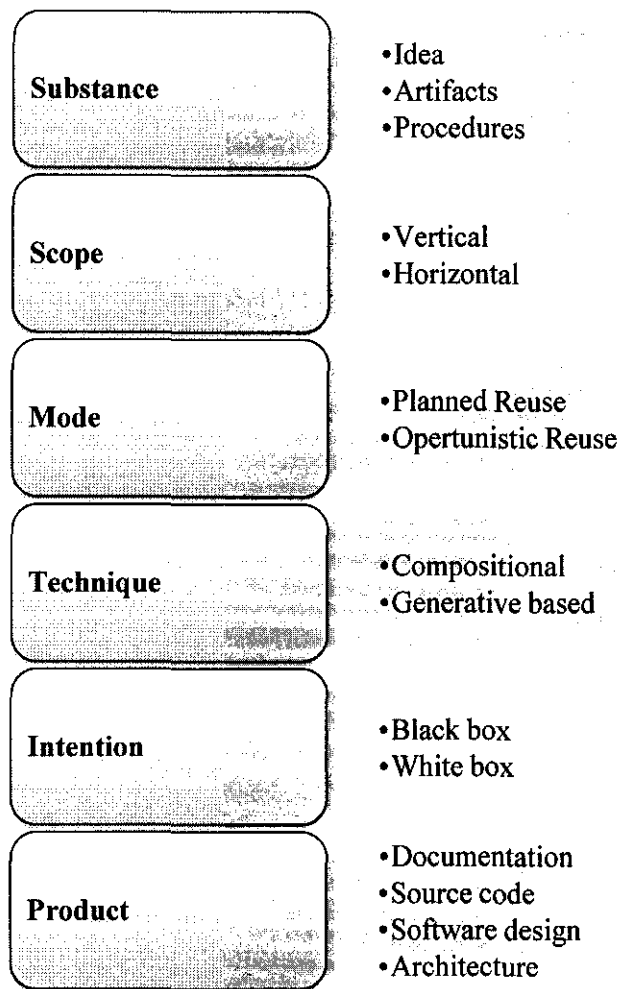


Figure 1.1 Dimensions of software reuse

The intention of reuse [5] can be categorized as black box reuse and white box reuse. Black box reuse is the 'as is' form of reuse. The components cannot be modified because the implementation of black box components is not visible from outside. The reuse is made possible by interacting with the specified interfaces. In white box reuse the implementation is visible from outside and components can be modified prior to their reuse.

The products [7] to be reused are the artifacts of software which include almost all products/outcomes that resulted from different phases of the software development such as documentation, source code, software design, and architecture.

1.3.1 Objectives of Reuse

Software reuse is not an objective in itself. Software is reused to gain benefits in terms of cost, effort and time. On the other hand, software reuse results in better quality, productivity and resource efficiency. An empirical study was conducted by [9] to provide the evidence of improved quality by employing reuse. The relationship between amount of reuse, quality, and productivity was explored in their study. It was concluded, on the basis of four sets of industrial data gathered, that more reuse results in better quality.

A review of industrial studies was conducted by [10] to explore the effects of reuse on productivity and quality. It states that reuse significantly reduces correction efforts and thus increases productivity.

1.4 Background Research

The background of this research is presented in the next sections. This research has been conducted by considering the literature on different issues. Each of the issue is discussed in the following sections.

1.4.1 Open Source Software and Reuse-Intensive Software Development

Open Source Software (OSS) is gaining the interest of the software engineering community due to its numerous benefits. These benefits fall into different dimensions. One dimension is financial benefits, e.g. the reduction in maintenance cost [11] and the escape from vendor lock-in [11-12]. Another dimension is technical benefits including; having a large number of developers and testers [13-14] and less maintenance risk [15]. Other dimensions include user support from the community [14], encouraging innovation [16-17] and increased collaboration [18]. As we can see, these are multifaceted advantages to the use of OSS. The benefits may relate to social aspects or to financial ones. The factors which contribute to the popularity of OSS may also include increased bandwidth, improved search facilities, and the existence of

code conjurers [19]. The growth of the Internet is also one of the factors which has a huge impact on the way that software is developed, marketed, and supported [20].

OSS was started as movement and for many years software developers contributed to it as their hobby (non commercial purpose). However, now the use of OSS in Component Based Software Engineering is already a norm in the industry [21]. Recently, researchers have envisioned the use of OSS in SPL development [22-23]. So, OSS has entered into a new arena. Being an emerging research area it demands an exploratory study to identify the issues. Currently available knowledge in this research area is limited. This lack of knowledge is also recognized in [24]: “there has been no systematic synthesis of the OSS challenges reported in the literature.” Therefore, it is obvious that there is a need to explore the use of OSS in reuse-intensive software development, especially in Software Product Lines (SPLs). In this study, not only the challenges, but also other dimensions to using OSS are explored, such as the current reuse practices, the desirable characteristics of OSS and the use of OSS in SPLs.

1.4.2 Assessment of Reusability

The disciplines of OSS, CBSE, and SPL share a common theme i.e. ‘reusability’. A broader definition of reusability states it as “reapplication of various kinds of knowledge about one system to another system in order to reduce the effort of developing and maintaining that system” [25]. OSS is developed to be reused and contributed by numerous software engineers.

The reuse-intensive software development methodologies such as CBSD and SPL reuse software artifacts to develop new products. In CBSD, software components are composed to develop software systems. In SPL, software assets are ‘developed to reuse’ and ‘reused for development’ [26]. The concept of reusability is the central tenant in these areas. So, in the context of OSS based development of CBSE / SPL it is important to assess the reusability of an asset. The reusability assessment is recognized as a research area in software engineering [8].

In [8], it is pointed out that reuse assessment is necessary to make software reuse scientific and engineering approach. Several attempts, such as [27], [28], [29], [30] and [31], have been made to assess the reusability of software assets. The collection and synthesis of these works is also the need of time. It will provide the insight into the current state of the art and helps to identify the short comings which will pave the way forward. Once the factors affecting reusability of software are identified, a reusability assessment model can be formed to assess the reusability of code asset. It is a model which associates the reusability with its attributes. The model will help to understand the software. The importance of understanding and analyzing code cannot be neglected. As the code base increases the importance of its analysis, manipulation is also increased [32].

As stated earlier, the nature of this study is exploratory. A review of current approaches reveals that none of the proposed approaches have considered the emerging situation which arises due to the combination of OSS and SPLs. Reusability can be viewed as usability from the perspective of developer [33]. Usability is a subjective phenomenon. Interview can be used to have an insight into the opinions of the informants. In this study, these concerns are addressed and the interview is used as a tool to explore the phenomenon of reusability of OSS in reuse intensive software development. The potential benefits of reusability assessment include the facilitation in decision making at different levels (such as, managerial level, or at individual software developers' level).

In the context of OSS, reusability is more of a concern due to the contributions from numerous developers. The reusability assessment of OSS prior to its reuse will give an idea about the ease of using it. The work presented in this thesis i.e. reusability assessment extends the view of [8] to make software reuse scientific and engineering approach rather than basing it on the perception and experience of software developer [34].

1.4.3 Selection of Components

Software reuse based development has become a standard in business and commercial software development [21]. Software reuse is commonly employed in two ways i.e. by the use of component libraries, as in component based software development (CBSD), and in a systematic way as in software product line development. Software artifacts are developed, from existing artifacts, with the intention of being reused. The product line development concept revolves around the terms ‘commonality’ (the requirements which are common to family members) and ‘variability’ (the distinguishing requirements). All products of a product line share the commonalities and are distinguished on the basis of variability.

The three generic activities for using a component in CBSD are identification, selection and adaptation of component (if necessary). In the past, identification process was involved in decision making, either developing a component or buying a component [35]. Now with the emergence of OSS, it is the third option. During the selection of components, different criteria are used. These may include the legal aspects such as license type or maintenance support for the component. In the context of this thesis, a particular aspect i.e. reusability of component is assessed to facilitate the decision process. Reusability is the central concept in OSS (are being developed to reuse), CBSE and SPLs (products are developed by reusing software artifacts). So, reusability or ease of reusing software is one of the key factors while selecting software. This research that has been conducted will help to assess the software reusability which will facilitate the decision making process to select OSS.

1.4.4 Software Variability and Implementation Mechanisms

Variability is one of the central concepts in reuse-intensive software development environments. ‘Variability’ is defined as “the degree to which something exists in multiple variants, each having the appropriate capabilities” [36]. Variability is one of the current areas of interest in software engineering research community, its dimensions and issues are explored in [37] and [38]. Variability of software has many facets due to the complex nature of this phenomenon. Several researchers have

identified the variability implementation mechanisms [39-40], types of variability [41-42], scope of variability [43], level of applicability, and binding time. Although, a categorization of variability mechanisms with reference to their characteristics is provided in [39], however, to update the body of knowledge it is necessary to revisit the analysis of variability mechanisms to include recent works in this area. In this thesis the types of variability identified by [41] and the scope of variability presented in [43] are also included, which update the earlier works like [39]. This synthesis of the literature is a need of time. It will serve the knowledge to the software development community which may help to identify and select appropriate variability mechanism.

1.5 Problem Statement

The problem statement of this study is based on followings:

- The reuse of OSS in CBSD is an established phenomenon. However, the use of OSS in SPL is envisioned recently. A few studies are available in literature which discuss different dimensions of using OSS but these studies lack the context of SPL. So, the emergence of using OSS in systematic reuse environment i.e. SPL demands an exploratory study to revisit the dimensions of this research area.
- Reuse is a common theme in CBSD, SPL and OSS. OSS is developed to reuse and the reuse intensive software development is based on the reuse of existing software artifacts (as in CBSD and SPL). Therefore, the importance of reusability assessment is more signified in reuse intensive software development.
- A large number of OSS is accessible today through internet. It has changed the traditional decision process of software selection by adding OSS to existing choices which were either to develop software or to buy. The selection of OSS depends on many criteria; reusability is one of them which needs to be addressed.

1.6 Research Questions

Following research questions are raised on the basis of the background knowledge on this area and problem statements; described in previous sections.

RQ1 – How reuse of open source software has been practiced in reuse intensive software development (SPLE and CBSE)?

RQ2 – What are the factors affecting reusability of open source software in a reuse intensive software development?

RQ3 – How to measure the factors affecting the reusability?

RQ4 – What is the significance and nature of relationship between reusability and its identified attributes?

1.7 Research Objectives

The objectives of this research include:

- To explore the use of OSS in reuse intensive software development.
 - To identify the challenges in OSS.
 - To identify the current reuse practices.
 - To identify the practices of using OSS in SPL.
 - To explore the role of OSS in promoting reuse.
 - To identify the desirable characteristics of components.
- To identify factors affecting the reusability of software component.
- To identify metrics to measure the attributes of reusability.
- To analyze and validate the identified factors.

1.8 Research Activities

In this study, the research activities (Figure 1.1) have been undertaken in three phases. Phase one is comprised of three activities which are literature review, problem identification and research planning. The review of the literature led to problem

definition and research planning. The research plan includes the selection of research methodology i.e. a framework of overall research activities, and research methods i.e. specific data collection techniques. The methodology is based on mix methods. The research activities include interviews, survey, experiment and statistical analysis.

The second and third phase can be viewed as the execution phases. In phase two, interviews were conducted followed by the survey. The data collected using interviews were qualitatively analyzed. On the basis of the interviews, a questionnaire was designed, and a survey was conducted about the reusability attributes. The code level software metrics were searched to assess the identified reusability attributes. These metrics fall in two categories i.e. class level and package level. The outcome of phase two is a reusability attribute model.

During phase three the proposed model was applied by computing the metrics values of open source software, at class level and package level. Statistical analysis was conducted to have a deep understanding of the relationship among the attributes. The second activity in this phase is the evolutionary reusability analysis. In this activity, the reusability of two open source software was assessed. The selected software has different versions. Comparisons of reusability and attribute values were made for each version.

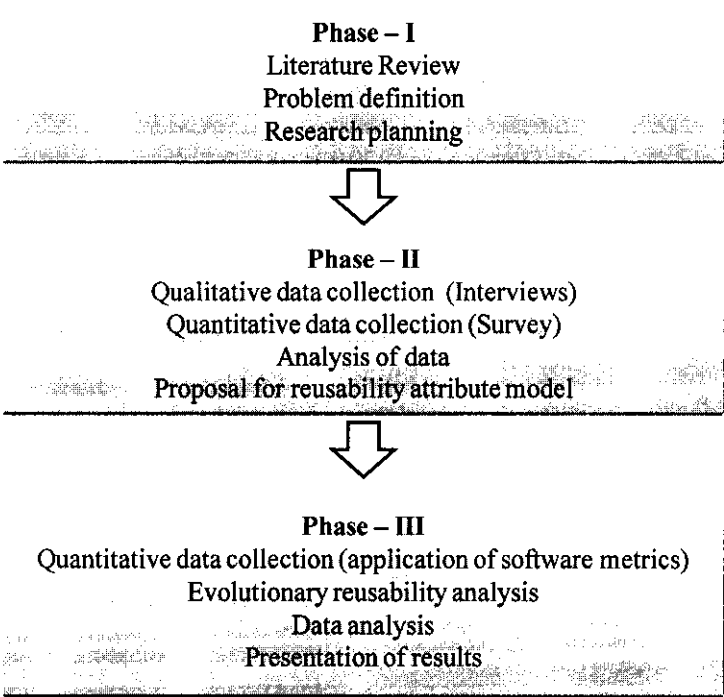


Figure 1.2 Generic research activities

1.9 Contribution

The main contribution of this study is the exploration of the phenomenon of reusing OSS in reuse intensive software development and proposal for reusability attribute model. The model is outcome of two empirical methods (interview and survey). The findings of the qualitative study based on the interviews (chapter 4) include the challenges in OSS, current reuse practices, issues related to OSS when using in SPL, the role of OSS in promoting reuse in software industry, desirable characteristics of OSS and factors affecting the reusability of OSS in SPL scenario.

The proposed reusability attribute model is presented in chapter 5 at class level and package level. The factors of reusability are quantified using well established software metrics. However, metrics for two attributes namely variability and scope coverage are newly defined in this study due to their non existence in literature. The phenomenon of variability is extensively analyzed from the view point of implementation mechanisms.

The proposed reusability attribute model is applied at the level of class and package in chapter 5. The results obtained by applying metrics are statistically analyzed to have a deep understanding about the relationship of attributes and reusability. Multiple versions of two open source software are analyzed to assess and observe their reusability during evolution. The results of these analyses are discussed under the light of earlier qualitative study and the studies available in literature. Other contributions of this thesis also include review of literature and methodological contributions discussed in chapter 2 and 3.

1.10 Thesis Structure

The thesis consists of seven chapters. In the second chapter a review of the literature is presented. The chapter is organized into two main sections; the first section contains the categorization of the available reusability assessment approaches, and in the second section the development of open source based SPL is discussed.

The research methodology of this study is presented in chapter three. The qualitative and quantitative methods used in this research are elaborated in this chapter.

Chapters four and five are based on the results of this research. The results obtained using the qualitative and quantitative methods are presented in chapter four. In the second section of chapter four, the theoretical analysis of variability implementation mechanisms is presented. The proposed reusability attribute model, description of attributes and metrics are presented in chapter five along with the results acquired using the quantitative methods.

The discussions about the results and implications of this study are presented in chapter six. The results are compared with the contemporary studies in this field. The thesis is concluded with limitations and future directions in chapter seven.

CHAPTER 2

LITERATURE REVIEW

Research is to see what everybody else has seen, and to think what nobody else has thought. (Albert Szent-Györgyi, 1893-1986)

2.1 Overview

This chapter presents the review of the relevant literatures. The chapter is organized into two main sections; and in the first section open source software is discussed along with its merits and demerits. The literatures regarding the proposals of OSS inclusion in SPL are reviewed. The second section contains the categorization of the available reusability assessment approaches and literature on software metrics.

2.2 Open Source Software

Open source software (OSS) development can be viewed as an alternate way of developing software. In traditional software development, including the proprietary or in-house development, the activities and the human resources are limited within a geographical location or conceptual jurisdictions of the company. In contrast to it, OSS is a community driven software development process.

2.2.1 Use of OSS in CBSE and SPL

Open source components can be used either as a black box (using binary files) or a white box (source code of component can be modified). OSS provides flexibility in deciding the type of reuse to be employed i.e. white box or black box. OSS can be modified to serve the purpose of an organization at any stage due to the availability of the source code.

2.2.2 OSS Definition

The definition of OSS [44] provided by the open source initiative (OSI) is based on the following 10 points.

1. Free Redistribution

The OSS license does not restrict any party from selling or giving away the software as component of an aggregate software distribution comprising of programs from different sources. There is no royalty or any other fee associated with the license.

2. Source Code

The distinguishing characteristic of OSS is that the inclusion of source code is mandatory with the compiled forms. In case, if the source code is not circulated with the software, it must be available from a well known public resource such as downloading through internet. The available source code must be in a form which is modifiable. The code must not be made unclear intentionally. Any intermediate form including the pre-processor or translators output is not allowed.

3. Derived Works

The license does not restrict the modifications and production of derived works. The derived work, when distributed must follow the conditions mentioned in the license of the original software.

4. Integrity of the Authors' Source Code

The user of OSS should uphold the integrity of author's code.

5. No Discrimination Against Persons or Groups

There is no discrimination in the license, against any person or group. The OSS can be used under the specified licenses. The producer of the OSS cannot impose restriction to discriminate the users.

6. No Discrimination against Fields of Endeavors

The license does place any restriction on the usage of software in a specific field or Endeavour. The producer of the OSS cannot impose restriction to discriminate the users from specific field or project.

7. Distribution of License

The rights of the program apply to all the parties having the redistributions of software. There is no need to execute any additional license by the parties.

8. License Must Not Be Specific to a Product

The rights of the program do not depend on the condition that the program is part of a particular distribution. In a situation, when the program is extracted from its original distribution, and used or distributed separately, it remains those rights assigned to it as a combination in the original distribution.

9. License Must Not Restrict Other Software

The license does not restrict other software which is distributed with the licensed software to be open source software.

10. License Must Be Technology-Neutral

The license of software contains no provisions regarding any specific technology or interface style.

The above mentioned points lay the foundation of the OSS. Furthermore, these criteria are more relevant to the rights and restrictions associated with the OSS. In the context of this thesis, the technical and social factors are more of the interest.

The advantages and drawbacks associated with the OSS are discussed in the next sections. After reviewing the literature, the advantages and drawbacks are categorized according to three aspects: financial, technical, quality. Some of the advantages and drawbacks fall in multiple categories such as the ‘encouraging innovation’. It may account for the financial aspect as well as the technical aspect.

The financial aspect looks at the advantages and drawbacks from the view point of business. The category of quality contains the issues related to the quality attributes of the OSS. The issues such as risks and collaborations are placed under the technical aspect.

2.2.3 Advantages of OSS

The financial advantage of OSS includes escape from vendor lock-in [11-12]. The OSS provides more freedom to the user as compared to proprietary software, where user always seeks to the vendor for maintenance support and other issues. Unlike the proprietary software, there is no increase in maintenance cost due to forced upgrades [11].

The technical aspects include culmination of software; the user or the organizations using OSS may customize it according to their own needs [45]. This can be done without any external support. The user/ developer’s contribution to the OSS results in lesser long-term maintenance risk [15]. The OSS has a large developer and tester base [13-14]. The users (software engineers) also act as testers.

The user support from the community [14], encouraging innovation [16-17] and increased collaboration [18] are multifaceted advantages of the OSS. These may be seen from the social aspect or from the financial aspect. However, for the purpose of this thesis these benefits are put under the technical factor. The reason is simply because they cannot be fit under the financial or quality aspect. A more

comprehensive classification is a subject of future research. These advantages are summarized in Table 2.1 along with the corresponding references.

Table 2.1 Advantages of OSS

	Advantages	Reference
Financial	Escaping vendor lock-in	[11-12]
Aspect	No increase in maintenance cost due to forced upgrades	[11]
Technical	Customization of software	[45]
Aspect	Less risk of long-term maintenance	[15]
	Large developer and tester base	[13-14]
	User support from the community	[14]
	Encourages innovation	[16-17]
	Increase collaboration	[18]
Quality	Fewer defects per line of code	[46]
Aspect	Reliability	[47-48]
	Security	[47]
	Performance	[47]
	Quality	[14, 47]
	Flexibility of use	[14, 48]

2.2.4 Drawbacks of OSS

The financial drawbacks of OSS include insufficient marketing of OSS [14]. The issue of insufficient marketing is associated with another drawback which is lack of ownership [13]. The OSS has a shared ownership, which explains why it is not the responsibility of any particular organization to market it. The second main reason is there is no budget allocated for the marketing of the OSS. The OSS needs a higher training investments [49], as compared to its counterpart.

The drawbacks from the technical aspect include lack of expertise [14] of OSS and version proliferation [14]; different builds of the same software are available to users. So, it is difficult for the potential user to decide which one is suitable for her. The OSS has a low level of compatibility [50] in some situations. Also, there is a security risk [49] associated with OSS, which may be the results of enormous contributions to OSS.

The OSS is considered less user friendly [13]. These drawbacks of OSS are summarized in Table 2.2.

Table 2.2 Drawbacks of OSS

	Drawbacks	Reference
Financial Aspect	Insufficient marketing	[14]
	Lack of ownership	[13]
	Higher training investment in OSS	[49]
Technical Aspect	Lack of expertise	[14]
	Version proliferation	[14]
	Compatibility	[50]
	Installation problems	[50]
	Security risks	[49]
Quality Aspect	Less user friendliness	[13]

2.3 Related Works on Using OSS

There are as many as hundreds of thousands open source software components are available today. The component repositories are just one click away from the software developers. Spinellis mentioned in an experience report [51], that the key to get great benefits from these software components is effective decision of choosing and using the component. The criteria presented in [51] is based on three aspects; (i) legal status

(ii) fitness and (iii) quality of software. The decision about the legal status of software matters in case when there is an intention to distribute the software. The other important decision is regarding the black or white box reuse of component. White box reuse of component also requires the acquisition of tools and infrastructure support for software development. Apart from these technical criteria to finalize the selection one may seek the opinion of colleagues. The popularity of software can be estimated by looking at the number of downloads and Google's search results. The availability of projects documentation also plays an important role during the selection. The interest of the community of developers of a specific open source is indicated by its release history, frequent releases represent an active community. Presence of a strong developer's community assures the line of support of software. The final check is the code inspection of software. During the code inspection one should look for the appropriate comments in the code, consistency of style and ease of understanding the code.

Code reuse is one of the forms of reuse in OSS based development. In [52] the motivating factors and hurdles in code reuse at the individual software developer level are assessed using a quantitative method (web based survey). One of the finding of this study is that code reuse plays an important role in OSS development. It reports that up to 30 percent of the functionality in software is added by reusing the code. The other results include that reuse of existing code is based on their developer's perception of effectiveness, efficiency and quality benefits of reuse.

A study based on review of literature is conducted in [24] to report the challenges in using OSS in software product development. The challenges are categorized in six main categories. These categories include; product selection, documentation, community support and maintenance, integration and architecture, migration and usage, and legal and business challenges.

A focus group study is conducted in [35] to answer the questions (i)'how should open source components for inclusion in products be selected?' and (ii) 'To what extent is code given back to the open source community, and what are the reasons behind doing so?'. A total of 10 persons including the author of the study participated

in the study. The findings of the study are presented in four categories which include; identification, selection, modification and giving back.

The studies referred in this section show the interest of software engineering research community in the topic discussed in this thesis. In these studies different research methods such survey, focus group and literature review are used. In this study mixed methods are used to explore the phenomenon of using OSS in reuse intensive software development. A comparison of results with the above mentioned studies is made in the discussion chapter.

2.4 Reusable Software Assets

The term ‘software assets’ include the artifacts developed during software development process. The reusable artifacts range from the highest level artifact i.e. software architecture to the lowest level ‘objects’. At the source code level, these artifacts include objects, methods, classes, interfaces and packages.

It is stated in [53] that, a higher rate of productivity can be achieved with reusable code. Some leading companies like Toshiba, Microsoft, IBM and Symantec are capable of developing new applications by reusing 85 percent of the code [53]. In this thesis, the focus is primarily on the artifacts of code assets. The OSS provides flexibility in ways of reusing code assets by providing multiple choices. The choices include using classes, files, pre built libraries, copying a few lines, or running a complete system as separate entity [51].

2.4.1.1 Class

A class can be viewed as a collection of similar objects and related operations. The mechanism of class in java is used to implement the encapsulation principle of object oriented design [54]. Java classes uphold the encapsulation principle by hiding the implementation and providing access to data using public methods. In open source java projects, class is a reusable unit [55].

2.4.1.2 Interface

In java programming there are two types of classes which are concrete classes and abstract classes [56]. A concrete class can be said a complete class i.e. having the bodies of methods, while an abstract class contains one or more undefined body (ies) of methods. The concept of abstract method is extended in java by using interfaces [57]. An interface is an abstract class; all of its methods are abstract. However, interfaces differ from the abstract class in a way that they cannot have implementation of methods unlike abstract classes. Interfaces provide a solution to multiple inheritance problems in java. A class can implement multiple interfaces.

2.4.1.3 Method

Methods are basic building blocks of java programming [54]. Method provides the mechanism to interact with objects of a class. The data can be accessed and manipulated through methods of a class. In other words, method performs the required operation of a class [58].

2.4.1.4 Package

In java programming language the notion of package is similar to a folder in operating systems. A java package is a collection of related classes and interfaces [59]. A java package provides a structuring mechanism for large size programs [54].

2.5 Reuse-intensive Software Development

The reuse-intensive software development refers to the methodology, in which commonality and variability is exploited to develop software products. In the context of this thesis, reuse-intensive software development points towards Software Product Line (SPL) and Component Based Software Development (CBSD).

2.5.1 Software Product Line

Software product lines (SPL) are defined as “a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission, and that are developed from a common set of core assets in a prescribed way” [60]. SPL is a systematic way of reusing core assets. In SPL development, each software artifact is considered as core asset.

Another definition of SPL is “development for the reuse and development with reuse” [26]. This definition views SPL development as a two process development. The development for reuse is domain engineering and development with reuse is application engineering [61].

The domain engineering is defined as the “process of software product line engineering in which the commonality and the variability of the product line are defined and realized” [61].

The application engineering is defined as the “process of software product line engineering in which the applications of the product line are built by reusing domain artifacts and exploiting the product line variability” [61].

2.5.1.1 Adding Components in a Product Line

The software product line framework version 5.0 [62], states that a software component enters in an organization in the following three ways.

- A component can be developed in-house (built in-house).
- A component can be purchased; this purchase of component includes the COTS component, OSS or web services.
- A component can be commissioned from third party.

2.5.2 Component Based Software Development

Component based software development (CBSD) is one of the major paradigms in software engineering which reuses the software components. The motivation underlying component based development of software is rapid development of software by making use of existing components [63].

The activities of CBSD are governed by software process model. Software process refers to the activities performed to develop software [21]; whereas a software process model is an abstract representation of the software process [21].

2.5.3 CBSE Development Generic Activities

There are different software process models available. The common activities of almost all software processes include software specification, software design and implementation, software validation and software evolution [21].

2.5.3.1 Selection of Components

As the development of software using components is the primary concern of the CBSD, the CBSD process guides the development of component based system as well as the process to develop software components. In this context, developer of the component based system is the consumer of component, and developer of component is the producer of component. This distinction of roles between the consumer and producer of software component results in a third process i.e. finding and evaluating the software components.

2.6 Open Source Components Based Product Lines

The underlying philosophy of SPL is the intra organizational reuse of components. The successful implementation of this philosophy results in benefits like improved productivity, better quality and reduced cost [60]. The use of OSS as core asset in

SPL is envisioned by Ågerfalk et al. in [22]. They have raised some issue and challenges, and paved the way for OSS based SPLs.

A conceptual model for OSS based SPL development is presented in [23]. It highlights the activities necessary to develop an OSS based SPL. This model is a higher level abstraction of the process based on the commonly reported practices in literature.

In [64], ‘Y-model’, an approach to develop SPL using COTS is presented. The necessary phases and activities to develop COTS based SPL are elaborated.

2.7 Literature Review Process

A review of the literature is conducted following the guidelines presented in [65]. The process of systematic literature review is adapted in this work as depicted in Figure 2.1.

The review process has three phases and ten sub activities. In the first phase of the review, the following questions were posed:

Question-1: What approaches have been introduced to assess software component reusability?

Question-2: What is the applicability of these approaches?

Question-3: What is the procedure adopted for validating the approach?

A review protocol was developed after specifying the questions. The contents of the review protocol are: the ‘source’ used in the review, ‘time period’, and ‘search criteria’. The term source refers to the scientific data bases; time period is the bracket of time to limit the publications within specific time period. The search criteria was a paper is to be included if it contains either a model, or a framework, or metrics for reusability assessment, and there is also a demonstration of the applicability of the proposed solution with results.

The protocol was reviewed by the researchers and a few changes were made to it before its execution.

The accepted review protocol is presented in Table 2.3 Review Protocol. The literature search has been focused on the research papers published during the years 2000 to 2010. The sources included are: Scopus, Google scholar, IEEE Xplore, ScienceDirect and the ACM portal digital library.

In the second phase, the search was performed using different queries. The key words used in the queries are presented in Table 2.3. The initial collection of the research papers was refined by looking at the ‘keywords’ in the papers and studying the abstracts. The criteria lay down for this review is based on the following condition: The selected paper should include either a model, or a framework, or metrics for reusability assessment. The selected paper should also demonstrate the applicability of the proposed solution with results.

A paper was included only if it fulfilled the criteria. The required data which is necessary to answer the questions is extracted from the papers. A table was formed to extract data from the selected papers. The table contains the following columns: Year, Complete Reference, Proposal, Applicability, Application/Demonstration, and Validation. The construction of this table has helped to organize the extracted information/data with the aim of providing a clear picture of the works in this area. Furthermore, the table serves to give ‘the bare bones’ of the review of each paper. The information about the problem being highlighted and solved in the paper was not included in the table because the papers were initially screened, leaving only those that are concerned with the reusability assessment of components.

Additionally, to make this review more thorough another step was performed by searching the related work section of the collected papers. This step helps to enhance the strength of the review by ensuring that no valuable reference is missed during the search process.

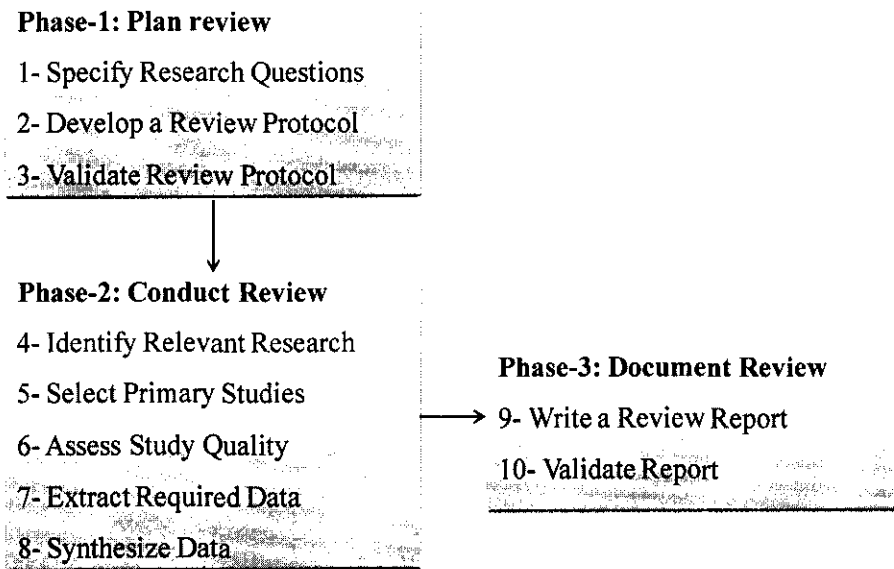


Figure 2.1 Literature review process- adapted from (Brereton et al., 2007)

Table 2.3 Review Protocol

Year	Sources	Keywords
2000-2010	Scopus, Google scholar, IEEE Xplore, ScienceDirect, ACM portal digital library	software reusability evaluation/ assessment/ measurement, software component reusability metrics/ model/ framework

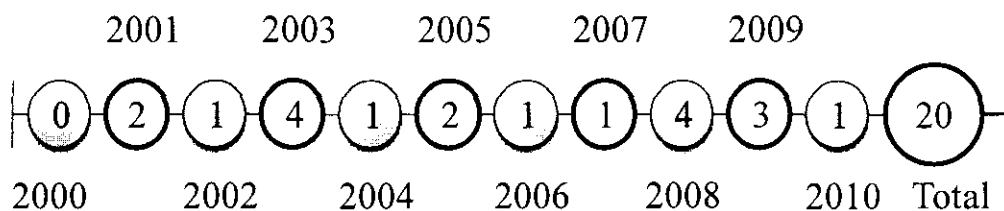


Figure 2.2 Year wise search results

A similar kind of study was performed by [66]. However, the focus of the study was to systematically review the software engineering measurements. It also mentioned that a few studies had attempted to measure the software reusability. In our work, we have managed to locate more studies that had attempted to measure reusability. This may be due to the fact that we used two resources that Gómez *et al.* did not, namely the Scopus repository and the Google Scholar search engine.

2.8 Classification of the Approaches

The results of the questions which are stated earlier in this chapter provides the results in terms of type, applicability and validation of the proposed solution.

2.8.1 Types

The review of the literature revealed the following types (Figure 2.3) of the proposed reusability assessment approaches. The breakdown is presented in Figure 2.3.

- Hierarchical model and metrics
- Quality Model Including Reusability As a Quality Factor
- Metrics
- Process
- Guidelines
- Framework
- Neural network-based approach

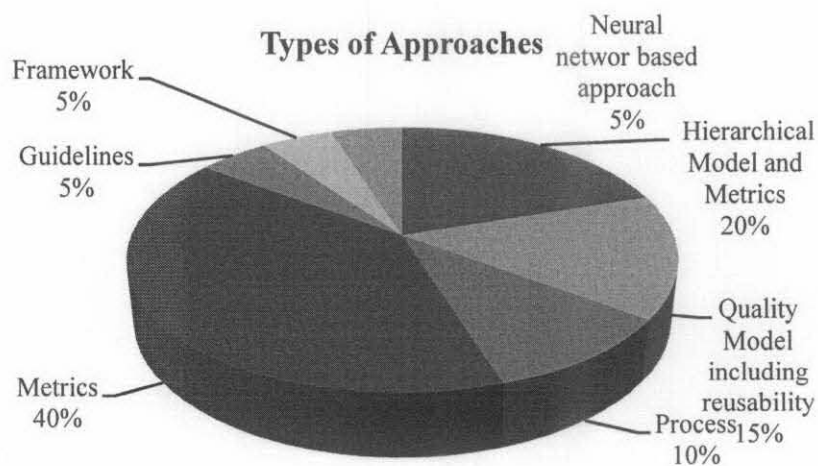


Figure 2.3 Types of Approaches

2.8.1.1 Hierarchical Model and Metrics

The hierarchical models and metrics represent 21% of the selected studies. These models associate the factors and sub factors of reusability, and metrics are defined to measure the lower level factors to assess reusability.

In [67], three views of reusability are defined which are: reusability in class, reusability in a hierarchy/ subsystem, and reusability in the original system. Factors, sub factors and metrics were proposed to measure reusability. The proposed solution was applied to graphical user interface packages to generate the results.

In [68], adaptability, completeness, maintainability, and understand-ability are considered as factors affecting reusability. These factors were measured by the metrics, which were applied to the components of a scientific application in order to evaluate the approach.

A metric suite to measure reusability is presented in [69]. Metrics were applied on components available on the Web.

In [70], a framework is presented that contains a reusability and maintainability model, and metrics for aspect oriented software. The proposed approach was applied on the aspect oriented implementation of GoF design patterns.

2.8.1.2 Quality Model Including Reusability as a Quality Factor

Some of the results (15%) of the review contain software quality models including reusability as a quality factor. These include [71], in which the quality characteristics of COTS components are identified in order to build a quality model. Commonality, customizability, modularity, and comprehensiveness are defined as measures of reusability. In [30], a component quality model is presented which includes reusability as quality factor. The model was applied to applications in the digital TV domain. A quality model is presented in [72], with reusability as a quality factor. The model was accompanied by metrics; it was applied on two software projects to generate the results.

2.8.1.3 Metrics

The results show that the most common proposed approach is to define metrics (40%) to assess reusability. In [73], metrics to measure complexity, customizability and reusability are proposed. The degree of features reused in developing an application was used to measure reusability. Two types of metrics were proposed, one is the metrics to be used at the design phase and the second is the metrics used after coding - the number of lines of code; the proportion of overall functionality that each component has. The application of the proposed approach was demonstrated by applying it to components in the banking domain.

A set of metrics to measure understand-ability and reusability of software components is presented in [27] and was applied to the measurement of twelve components. [74] discussed the adaptability and complexity of software components. Compos- ability and adaptability were considered as the main factors influencing reusability, and metrics were presented to measure these factors.

Two metrics are proposed in [75] to measure the amount of generic code. The proposed metrics were applied to ten projects.

In [76], measures for cohesion are defined to assess the reusability of a component; the specific purpose is the prediction of effort required to reuse a component in a larger system. The proposed metrics were applied to a HTML parser, a lexical tokenizer, and a bar code application, all of which are in the form of components. In [28] coupling metrics are proposed to rank the reusability of components. Metrics were applied to three types of component to generate the results. The metrics to measure coupling [28] and cohesion [76] are combined in [77] to measure the reusability of software components. In [29], coupling and cohesion metrics are also proposed to evaluate the reusability of components.

2.8.1.4 Process

Results of the literature review show that 10% of the studies used measurement 'processes'. [78] proposed a collaborative scoping approach for an organization to

migrate existing products to a product line. This approach makes use of metrics to assess reusability in one of its tasks. These metrics were applied at class, method and ‘lines of code’ levels. A reusability measurement process is proposed in [79]; it uses McCabe and Halstead methods to measure reusability.

2.8.1.5 Guidelines, Framework, Neural Network-Based Approach

Guidelines, neural network-based, and framework approaches each represent 5% of the results. In [80], guidelines are provided for reusability of software components, which can also be used to measure reusability. An artificial neural network-based approach to assess reusability is presented in [31]. The network was trained with forty examples of Java and tested afterwards with twelve examples. A framework to measure and evaluate program source code is proposed in [81]; it contains a quality model and metrics. The model includes reusability as a quality factor. The implementation of the framework was demonstrated for use with software written in the C language and is restricted to this language.

2.8.2 Applicability

The applicability of the proposed approaches is categorized in three ways. First, the approaches are differentiated according to whether they are object oriented, aspect oriented, etc. (Figure 2.4). Second, the approaches are categorized according to the language used to apply the approach - either C/C++ or Java (Figure 2.5). Third, the approaches are categorized according to whether the metrics used to assess reusability did so without reference to the source code of components, or by optionally referring to the source code, or by compulsorily referring to the source code (Figure 2.6). The results are presented in Table 2.4.

2.8.2.1 Programming Paradigm Based Categorization

70% of the proposed approaches are object oriented; they make use of the constructs of object orientation to measure reusability. In 5% of the studies, aspect-oriented

implementations are targeted while component based software development is considered in 20 % of the selected studies.

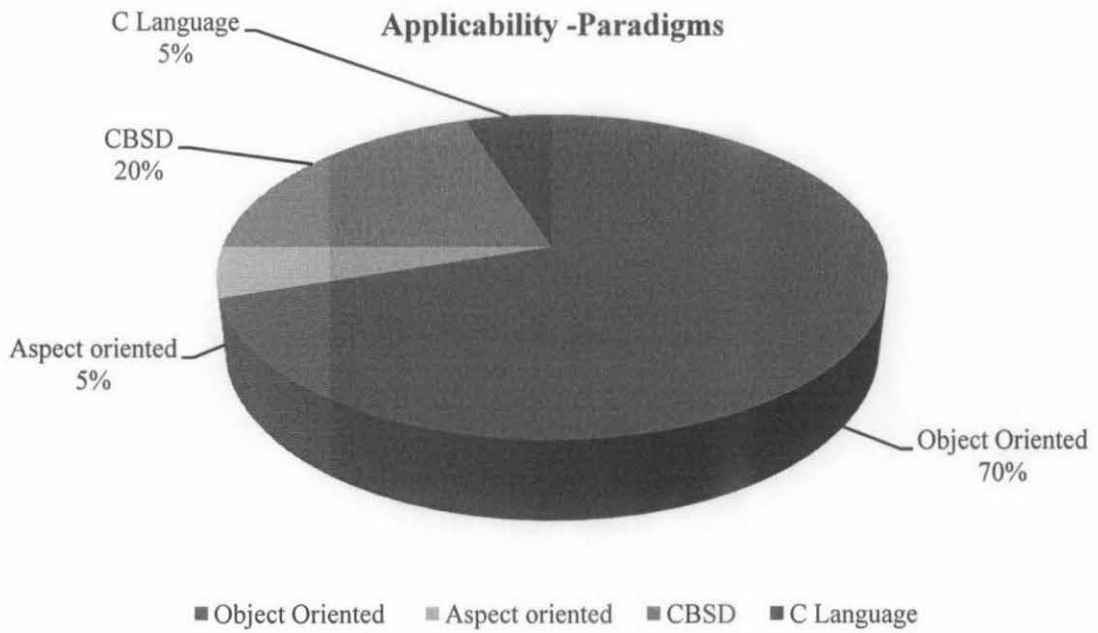


Figure 2.4 Applicability of approaches - paradigms

2.8.2.2 Programming Language Based Categorization

The results show that reusability assessment approaches applicable to Java are 71% while 29% of the approaches are applicable to C/C++.

Applicability - Programming Language

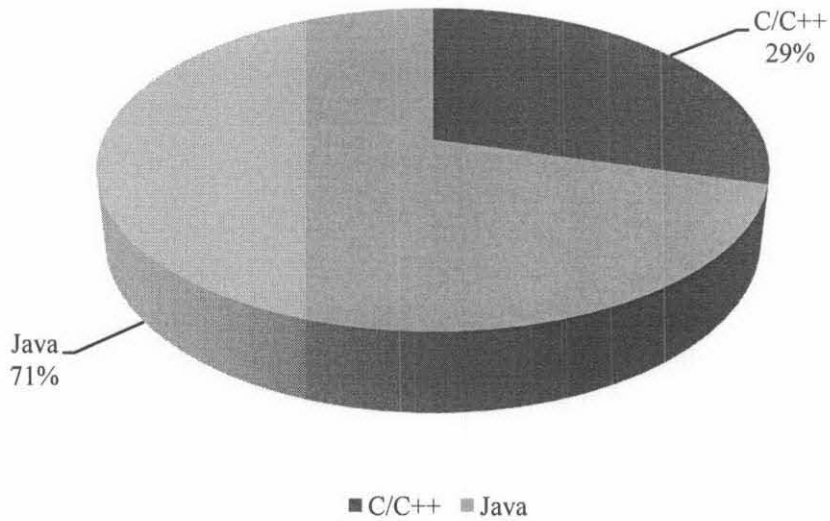


Figure 2.5 Applicability of approaches - programming languages

2.8.2.3 Level of Application

The results reveal that 60% of the proposed approaches are white box; i.e. the availability of source code is mandatory with these approaches, while 33% are black box, i.e. where source code is not referred to. The approach proposed in [67] can be used for both white and black box components..

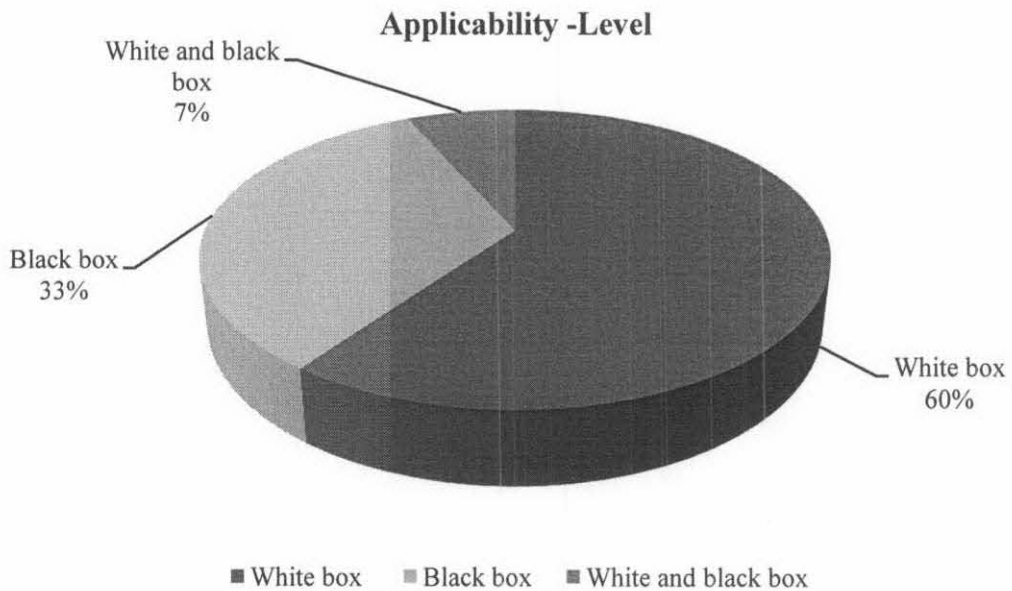


Figure 2.6 Applicability of approaches - level

Table 2.4 Applicability of approaches

Applicability of Approaches	
Category	No. of papers
Object Oriented	14
Aspect Oriented	1
CBSD	4
C Language	1
Total	20
C++	4
Java	10
C	1
Total	15*
White Box	9
Black Box	5
White and Black Box	1
Total	15*

* Some of the publications cannot be categorized according to this breakdown.

2.8.3 Validation Types

The type of validation of the approach (Figure 2.7) refers to the procedure adopted for validation. The break down in numbers is presented in Table 2.5. The proposed reusability assessment approaches available in the literature are categorized according to the following types:

- Humans Based Evaluation,
- Statistical Analysis,
- Using Weyuker's Properties,
- Experiment,
- Case Studies,
- Survey / Questionnaire,
- Using Test Data, and
- No Validation.

2.8.3.1 Humans Based Evaluation

These are the evaluations where the results are obtained by applying the proposed assessment approach, and the results are compared against the results gathered by the user/expert assessment of the software.

Human evaluation as the form of validation is employed in 10% of the selected studies. Here the qualitative validation refers to a comparison between the results collected by applying the proposed approach and human evaluation of the components; these evaluators may include experts and users/software engineers. In [67], expert opinion about the reusability of the components were collected and then compared with results generated through the application of the proposed approach; regression analysis was used to interpret the results. [69] used the ratings of an evaluation committee of a website, from where the components were selected, to assess their reusability.

2.8.3.2 Statistical Analysis

In some of the papers results were analyzed by using some statistical technique to validate the approach. Statistical analyses of the results are provided in 25% of the studies, which include [27]. They used linear regression and the mean to perform the statistical analysis. In [76], linear regression was used to evaluate the performance of the proposed measures of reusability. Rank correlation and linear regression were used to assess the performance of the proposed metrics in [28], [29] and [77].

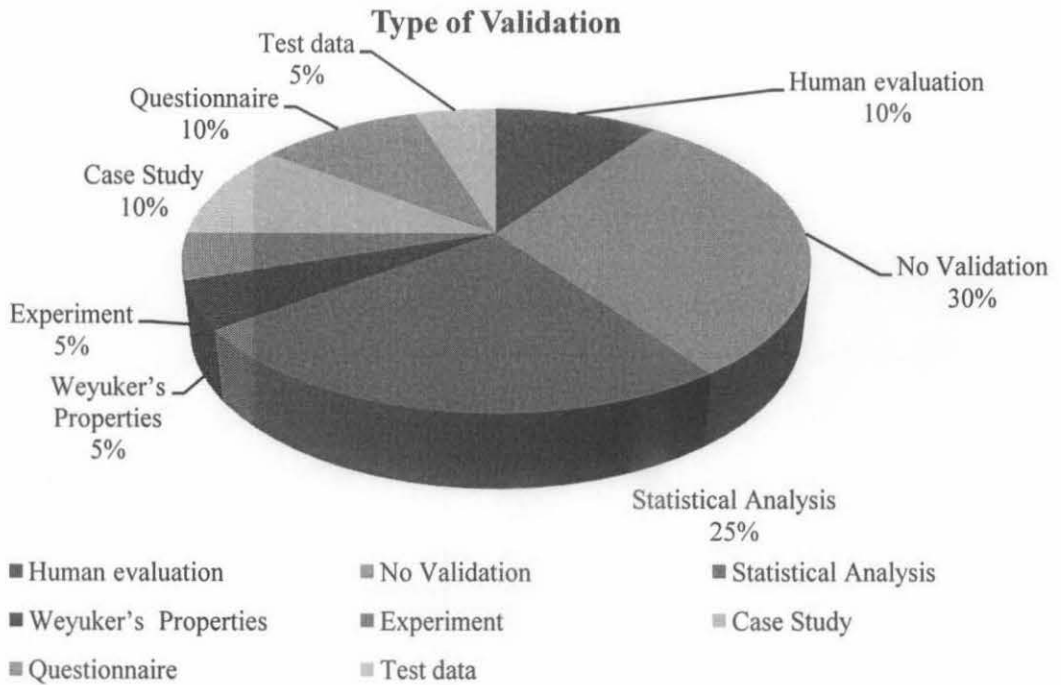


Figure 2.7 Types of validation

2.8.3.3 Using Weyuker's Properties

In [75], Weyuker's properties were used to evaluate the metrics. Weyuker's properties are a set of nine properties for the evaluation of complexity metrics.

2.8.3.4 Experiment

In one of the study available in literature, experiment was carried out to validate the results of proposed approach, such as [82], who performed a semi-controlled experiment with human subjects and implemented two versions of a web portal development system.

2.8.3.5 Case Studies

Case studies are used to validate proposed approach in 10% of the selected papers, which include [30] where the case study of a digital TV application and related

platform application was used. In [78] the initial validation of the proposed approach is demonstrated through two open source projects.

2.8.3.6 Survey / Questionnaire

Questionnaire is used to assess the component, and then the results of the approach are compared against the results collected from the questionnaire. Questionnaires are used in 10% of the studies. In [68] the correlation coefficient between results of direct measure, using the proposed approach results, and measures collected via the survey instrument, are presented to validate the results. A questionnaire was used to validate the framework using quantitative evaluation in [81].

2.8.3.7 Using Test Data

In [31] a neural network-based approach is proposed to assess the reusability and it was validated by using test data.

Table 2.5 Types of validation

Types of Validation	
Category	No. of papers
Human Evaluation	2
No Validation	6
Statistical Analysis	5
Weyuker's Properties	1
Experiment	1
Case Study	2
Questionnaire	2
Test Data	1
Total	20

2.8.3.8 *No Validation*

In some of the selected papers (30%) approaches to assess reusability were proposed, but no validation of the proposed approaches is provided.

2.8.4 **Synthesis of Literature Review**

The results of the review show that the majority of the approaches are based on metrics (70%). The applicability of majority of the approaches is object-oriented paradigm (70%). The implementation language which is targeted in most of the approaches is java (71%). The intention of (60%) of the approaches is white box measurement. These figures show that the software development community is more interested in object-oriented paradigm and java based implementations.

One of the issue rose after the literature review, which pointed the lack of validation of the proposed approaches in most of the previous works. The results show that (30%) of the proposed approaches lacks the validation of results. The software research community needs to give attention to validation as it is necessary to validate results in order to gain the confidence of software practitioners.

2.9 **Literature Review on Variability**

In this section a literature review on variability is presented. The review includes the synthesis of literature containing the variability types, scope and binding time. A variability map is generated on the basis of the review which helps to present a complete picture of the variability at implementation level. Variability map also relates the type of variability to its scope and binding time. The types, scope and binding time of variability is discussed in detail in next sections.

2.9.1 **Variability Types (with respect to effect)**

A software product line provides an infrastructure for developing different products. These products are distinguished from each other on the basis of their variant features.

In this regard, a product line provides support to the variant features of products within its scope.

This section contains types of variability; the types which follow are defined in the context of product lines, and members products are distinguished on the basis of the variability exhibited by them.

- Attribute Variability
- Logic variability
- Persistency variability
- Work flow variability

2.9.1.1 Attribute Variability

In [41], attribute variability types are defined, where an attribute is supposed to be a placeholder for values to be stored – such as constants, variables or data structures. Furthermore, three cases of attribute variability are presented. First is when the number of attributes varies between products of a product line. Second is the variation in the data type of the values assigned to the attributes, and the third case represents the variation of the value assigned to the attribute that is persistent.

2.9.1.2 Logic Variability

Logic variability is the variation of the algorithm or logical procedure. There are several cases of logic variability, each case dependent upon the entity that varies, be it the procedural flow, the post condition, the exception handling, or the side effects between products of a product line.

2.9.1.3 Work Flow Variability

Work flow variability is variation in the order, type and number of methods invoked by family members when carrying out a common task.

2.9.1.4 *Persistency, Interface Variability*

Persistency variability refers to the variation on the values of attributes that are stored in secondary storage. Interface variability is the variation in the signature of the interface method, i.e. to implement the same requirement, different members of a family implement their methods in different ways. These are distinguished by the name, return type, and order and type of parameters.

2.9.2 **Variability Types (with respect to functionality)**

Another classification of variability types are presented in [42]. It is based on the functionality.

- Positive
- Negative
- Optional
- Alternative
- Function
- Platform/Environment

The variability is said to be positive - when some functionality is added; negative - when there is a withdrawal of functionality; optional - when code is added; alternative - when code is removed; function - when functionality is changed; platform/environment -when the platform or environment is changed.

2.9.3 **Variability Scope**

A variation point can be open or closed [43]. A variation point is said to be open if new variants can be added and older ones removed. Conversely, a variation point is closed when new variants cannot be added. The open variation point can be described further using the term 'scope of variability' [41]. Scope is classified into binary, selection, and open. It is binary when there can only be two variants at a variation

point. The scope is termed ‘selection’ when three or more already known variants are available at a variation point. The scope is termed ‘open’ when there can be any number of known and unknown variants at a variation point. In Figure 2.8, the relationship between variability types and scope is depicted.

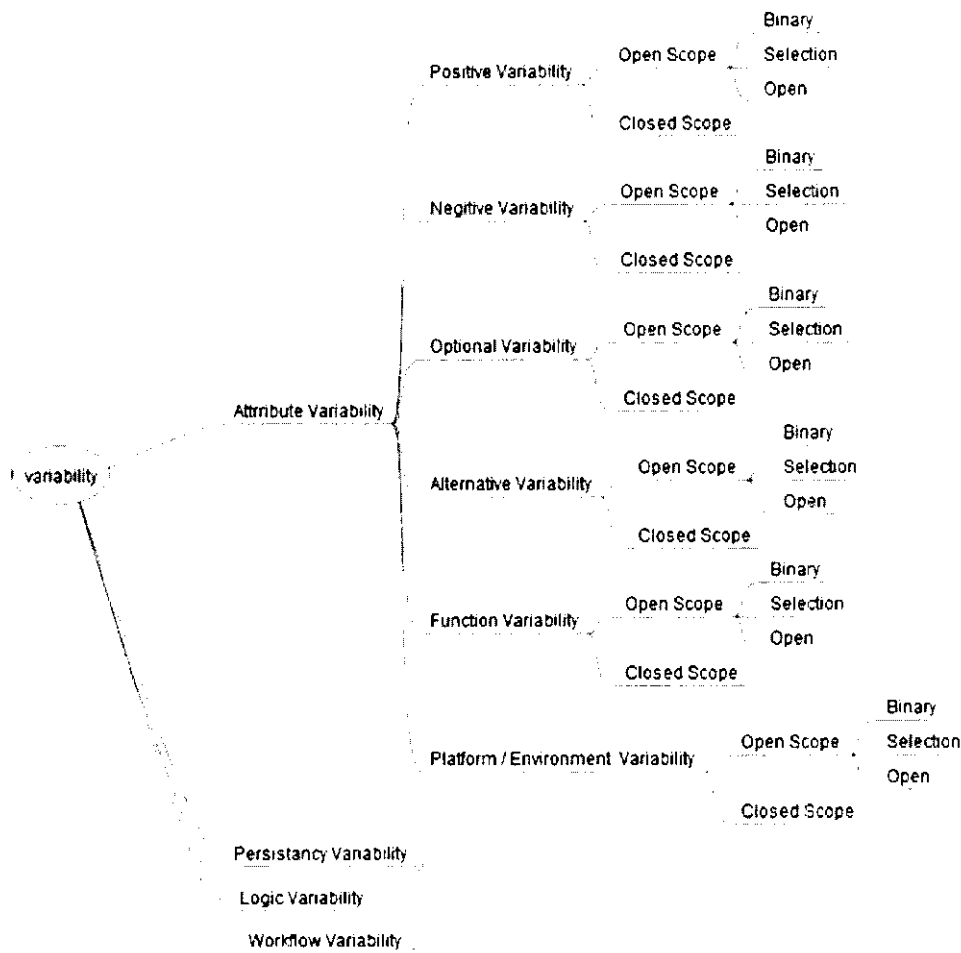


Figure 2.8 Relationship of Variability types and scope

2.9.4 Binding Time of Variability

The binding time refers to either the time at which a variant was assigned to a variation point [43], or the latest time during the development when a variation can be bound to a variation point [83]. Different variability realization mechanisms are used to implement variability. The binding time of variability depends on the mechanism used to implement variability [83]. In [39], the following binding times are defined with respect to the ‘implementation level’: Compile time (variability is resolved

before the compilation); Link time (variability is resolved during module or library linking); Runtime; (variability is resolved during execution of the program); Update/Post Run time (variability is resolved after execution or when a program is being updated).

2.10 Software Engineering Measurements and Metrics

The importance of measurement cannot be ignored in any engineering discipline. Similar is the case with software engineering. Software engineering metrics are of different types. The metric types can be categorized on the basis of the entity and attribute (property of software) need to be measured. Further classification can be on the basis of nature of metric and type of data used by the metric. These dimensions of software metrics are elaborated further in this section.

In software measurement, three kinds of entities are measurable - processes, products, and resources [84]. A product can be defined as any artifact developed as a result of process activity. These entities may have attributes which are of two kinds - internal and external. An external attribute is one that cannot be measured directly. In contrast, internal attributes can be measured directly. If we can measure something directly then this means that we can measure it independently. Relevant metrics are termed 'direct metrics' [85]. For example, the size of a program can be measured directly in several ways: by counting the number of lines of code; by counting the number of 'methods'; etc. In software engineering measurement terminology, a metric is a quantitative indicator of a software attribute; a metrics model specifies relationships between metrics and the attributes being measured by these metrics. Another dimension in this field is the definition of metric as being elementary, in that it requires only one attribute, or composite, in that it needs more than one attribute [86].

In context of this thesis, the metrics proposed and used can be categorized as product metrics. The metrics are applied on software artifact i.e. code. The internal attributes of software of code are considered during the measurement. So, direct metrics are used to measure the entities. Both the composite and elementary metrics

are used such as maintainability index (MI) which is a composite metric and lines of code LOC, which is an elementary metric.

2.10.1 Reuse Metrics and Models

The reuse metrics and models fall into six categories [6], these include the followings:

- Reuse cost benefit analysis
- Reuse maturity model
- Amount of reuse metrics
- Failure mode analysis
- Reusability Assessment
- Reuse library metrics

Reuse cost and benefit analysis involves the estimation of investment of time and cost to develop a systematic reuse environment, and the benefits of it. The reuse maturity model helps the organizations to understand their reuse programs. An organization can review their past, current and future goals of reuse. Amount of reuse metrics are concerned with the percentage of reuse achieved in the organization. Software reuse failure mode model deals with the evaluation of systematic reuse program. These models help to improve the reuse strategy of an organization.

2.11 Identification of Need to Conduct This Study

In this chapter, the literature on related recent works on using OSS is discussed. The review of literature reveals that there is a lack in the available studies that the dimensions regarding inclusion of OSS in SPL is not discussed. The proposals to include OSS in SPL are proposed in literature which is also discussed. On the basis of the review, it is evident that there is a need to conduct an exploratory study to explore the phenomenon of using OSS in reuse intensive software development.

The literature review on the reusability assessment approaches revealed that none of the approach has considered one of the key factors i.e. variability in relation to

reusability. Furthermore, the lack of validation is also evident in literature. In this study a reusability attribute model is proposed and variability is also brought into picture.

Variability is one of the key concepts in reuse intensive software development. The literature on variability is reviewed from the perspective of implementation mechanisms. The literature review on variability exposed that different dimension are highlighted in different studies. This scattered knowledge demands the synthesis of literature to create a broader picture of variability at implementation level. Furthermore, the analysis of available variability implementation mechanisms is also required to facilitate the software engineers in choosing appropriate mechanism.

2.12 Summary

Different approaches to assess reusability are available in the literature. These approaches can be categorized according to their types such as metrics, hierarchical models, and processes. The second classification of approaches is based on their programming paradigm such as the approaches for object oriented or aspect-oriented. Another classification is based on the target programming language such as java or C/C++.

The benefits / drawbacks of OSS and recent works related to the use of OSS are present in this chapter. The use of OSS in systematic reuse environment is also envisioned by the software engineering researchers. This reuse will help to improve the reuse of components at larger level. Software metrics and specifically metrics and models related to software reuse are also elaborated in this chapter.

CHAPTER 3

RESEARCH METHODOLOGY

Every discourse, even a poetic or oracular sentence, carries with it a system of rules for producing analogous things and thus an outline of methodology.

(Jacques Derrida, 1930-2004)

3.1 Overview

This chapter presents an overview of the research philosophies and their underlying assumptions. A brief account on qualitative and quantitative research methods is presented. The qualitative and quantitative methods used in this research are elaborated in this chapter.

3.2 Philosophical Basis of Research

The ultimate goal of research is the quest of knowledge. In Greek, the word 'epistêmê' is used for knowledge, and the word 'epistemology' originated from it. Epistemology is the philosophy of knowledge. It is related to: 'what is knowledge?' and 'how to obtain it?' [87]. During the process of obtaining knowledge the researcher has a specific belief system or worldview which guides the researcher. This worldview or belief system is termed as 'paradigm' [88]. Paradigms can be seen as the theoretical perspectives of research [89]. These theoretical perspectives may include the positivism, constructivism, feminism etc. Different paradigms are based on different assumptions.

Positivism is the dominant epistemological paradigm of twentieth century [87]. It is based on the assumptions that reality is constituted by the evidences available to the senses (sense of seeing, smell and touch etc.). Scientific observations should be the basis of inquiries. The logical and methodological principles of natural and human sciences are same i.e. both deal with facts not with values.

Constructivism is the paradigm which is associated with qualitative approaches. The worldview of the constructivists is based on the understandings and meanings of the subjective views of subjects and participants [90]. It is a bottom up approach, where the investigation starts at the level of individuals, and patterns are identified to understand the phenomenon.

The theoretical perspectives are related to the different ‘research methodologies’ i.e. a framework of overall research activities. The research methodologies are in turn associated to ‘research methods’ i.e. specific data collection techniques. A pictorial depiction of this relationship is presented in Figure 3.1.

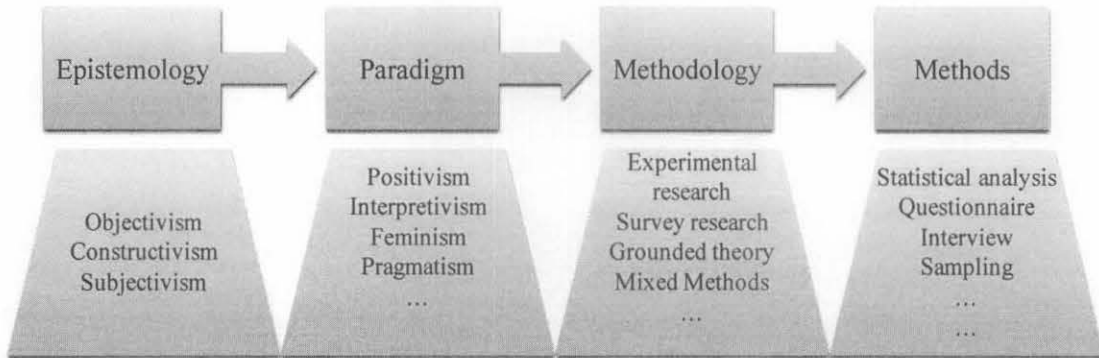


Figure 3.1 Epistemology, paradigm, methodology and methods [91]

Qualitative and quantitative research methods are distinguished on the bases of the nature of the data and the processes followed to collect the data during the research. Qualitative data are in the form of text and pictures and are collected through ethnographies, case studies and interviews. Quantitative data are in the form of numbers and are collected using surveys, experiments and quasi experiments.

In [90], mixed methods are classified as methodology. The authors explained, although mixed methods come under the methods, however, there is an involvement

of strategy for conducting the research. Therefore, mixed methods can be fit at the methodology level.

3.3 Basis of Mixed Method Research

Mixed method research is emerging as a third choice to the researcher other than qualitative and quantitative research [92]. Mixed method studies combine the qualitative and quantitative approaches. In [93] mixed method studies are defined as “These are studies that are the products of pragmatist paradigm and that combine the qualitative and quantitative approaches within different phases of the research process”.

The paradigm or theoretical perspective of mixed methods study is ‘pragmatism’, suggested by [94]. The pragmatism and mixed methods are linked by [94] on the basis of the following arguments:

- Both qualitative and quantitative methods may be employed in a single study.
- The research question is more important than the method or the philosophical worldview of method.
- There should be no forced choice of being either positivist or constructivist.
- The use of metaphysical concept like ‘truth’ and ‘reality’ should be left behind.
- The choice of methods should be based on the practical and applied research philosophy.

In light of the above points, and as the endorsement given by [95], it can be stated that pragmatism involves in using multiple methods, different worldviews, collection of different forms of data and employing multiple analysis tools in mixed method study.

3.3.1 Purpose of Mixed Methods

[96] have highlighted five purposes of mixed method studies, the first being triangulation i.e. convergence of results, rationale for triangulation is to increase the validity of research by using different methods [97]. Three possible outcomes are expected as a result of triangulation which are (1) converging results, (2) contradictory results, and (3) partially consistent results [97].

The second purpose is complementarity i.e. “seeks elaboration, enhancement, illustration, clarification of the results from one method with the results from the other method” [96]. Third is development, which uses the results of one method to develop or inform other method. Next is initiation which involves recasting of questions or results from one method with the questions or results from the other, and the fifth is expansion where different inquiry components are inquired by different methods [97].

3.3.2 Types of Mixed Method Studies

The types of mixed method studies are classified in [98] and [92]. In the earlier classification by Mingers, five types of mixed methods studies were identified, while in a later study Leech and Onwuegbuzie identified eighth types of mixed method studies.

Mingers classification includes; sequential studies i.e. using the methods in a sequence such that the results of one method are used by the other. Parallel studies i.e. methods are applied in parallel with results provided to each other. Dominant [98] that states one method is used as a main method with a share of others. Multimethodology [98]; it combines methods from different paradigms specifically to serve the purpose of the study. Multilevel [98] in which different methods are applied simultaneously at different levels of an organization.

In [92] the types are based on three dimensions. These dimensions are identified based on the content analysis of the available mixed method studies. These dimensions include time orientation, emphasis of the approach, and level of mixing. The time orientation dimension refers to the time continuum of the study. Studies

where the qualitative and quantitative phases occur at the same time are concurrent mixed method studies. A study in which one phase is preceded by others is sequential type of mixed method study. The dimension of emphasis is associated with weight assigned to the qualitative or quantitative phase. On the basis of these dimensions, eight types of mixed method studies are represented in Figure 3.2 which is redrawn from [92]. In this research study, partially mixed sequential dominant type is used.

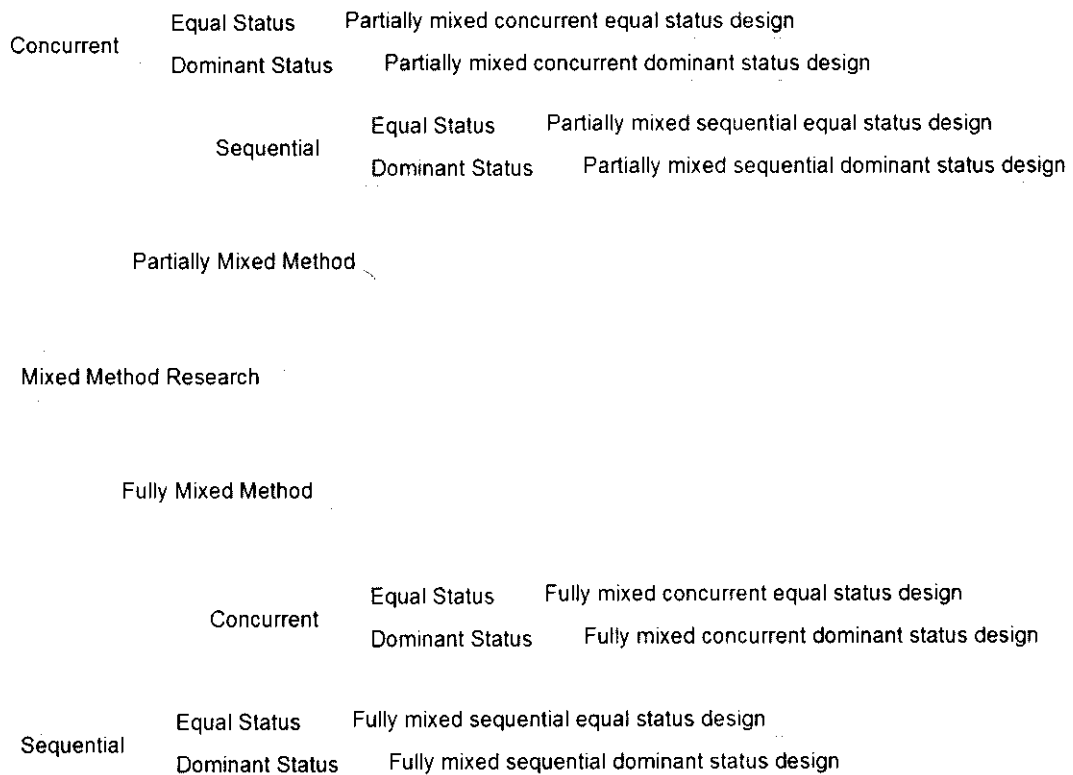


Figure 3.2 Typologies of mixed method studies [92]

3.3.3 Type of Mixed Method Chosen for Study

In this study, partially mixed sequential dominant approach is employed. Qualitative phase of study is followed by the quantitative phase. Interview is used as qualitative method. Survey and experiments are used as quantitative methods. The methods are employed sequentially. The results of qualitative method i.e. interview is used to develop survey instrument (questionnaire). On the basis of the results from interview and survey a reusability attribute model is formed. Experiments are conducted by using the proposed reusability attribute model to validate the relationship indicated in

the model. The qualitative phase is dominant during the study. The study is based on the findings of the qualitative phase. The mixing of methods is partial as there is no quantization or qualitzation of data is performed.

3.4 Research in Software Engineering

Software can be viewed as making machines accessible to humans. Research in software engineering is directly related to the social and behavioral sciences as software is an entity which interacts with humans. The end users, who are the ultimate beneficiaries of software, are humans. If we take as an example the software for a mobile phone system or the software for a hotel reservation system, then the human involvement is quite clear.

An alternative view of software engineering is that, like other engineering disciplines, it helps humans to solve their problems [99]. This view of software engineering pushes it into the realm of a traditional engineering discipline and their associated research methodologies become relevant.

The importance of software and research in software engineering is highlighted by many. This emphasis is not only given by the research community, but is also reflected at the highest levels of government. A 1998 report by the US President's Information Technology Advisory Committee (PITAC) states: "Increases in research on software should be given the highest priority".

There is a lack of guidance regarding the research approaches; what research approach is appropriate to answer a particular research question in software engineering [100].

Software engineering is a multidisciplinary field. It deals with social and technological issues. A software engineering activity is not only based on the processes and tools, but it also depends on the social and cognitive process around it [101]. Therefore, study of human activities is necessary to understand a problem and its solution in software engineering domain. The aforementioned importance of human activities in software engineering field compels to use the research methods of

fields related to the study of human behaviour. For instance, in a situation when the investigation is desired at individual level research methods from psychology are appropriate. In a situation when the problem under consideration is concerned with teams and organizations, research methods of sociology become more relevant [101].

3.4.1 Mixed Methods in Software Engineering

In the software engineering field, opinions exist that suggest there should be a combination of both qualitative and quantitative research methods in software engineering research [102]. The authors also state that using a combination of qualitative and quantitative methods may be beneficial in that it provides information from a number of perspectives. The combination of qualitative and quantitative approaches in a single study is referred as mixed method studies.

An example of mixed method studies in software engineering can be found in [103], where an investigation of object oriented is made using survey questionnaire, structured interview and controlled laboratory experiments.

Mixed method software engineering research studies provides possibly more generalized and reliable results. Mixed method approach may help to limit the experimental validity threats in software engineering [103].

According to [104], the empirical evidences have a psychological effect on the researchers and practitioners which helps to convince them that the results are useful and correct. Mixed method research in software engineering helps to improve the impact of results by increasing the confidence in the findings. Mixed method studies increase the level of rigor.

3.5 Purpose Based Classification of Research Studies

In the previous sections of this chapter different types of research have been discussed. These types are distinguished on the basis of the methodology. There is another classification of research studies provided in [105], which is based on the

purpose of inquiry. The research studies can be classified as Exploratory, Descriptive, Explanatory, Emancipatory [105] and interpretive [106].

3.5.1 Exploratory Research Studies

Exploratory research studies are conducted in a situation when there is little understanding of the situation within the research community [105]. These studies are motivated to seek new insight into the situation. The studies are intended to assess the phenomenon from a new perspective. Exploratory research generates ideas and hypothesis for further research [105]. Exploratory studies can be conducted by searching the literature, talking to experts or by conducting a focus group interview [107].

3.5.2 Descriptive Research Studies

Descriptive research is focused to depict an accurate profile of persons, events, or situations [105]. The pre-requisite of descriptive studies includes an extensive previous knowledge of the situation. The background knowledge provides information about the aspects on which the information is to be collected.

3.5.3 Explanatory Research Studies

Explanatory studies are intended to explain a situation or problem. It is not necessary in explanatory studies that the causal relationships are identified [105]. These studies may involve in identifying patterns related to the phenomenon under consideration.

3.5.4 Emancipatory Research Studies

These are the studies conducted to create opportunities and a will to engage in a social action [105]. It seeks to challenge and change the inequalities. The objective of emancipatory research is social change. It changes research into a political activity.

3.5.5 Interpretive Research Studies

Interpretive research studies are conducted when there is a need to seek the experience of people and their views or perspective of experience [91]. Interpretive researchers attempt to understand phenomena through accessing the meanings, participants assigned to them [108]. In these studies qualitative data collection and analysis process are used.

In this section, purpose based classifications or research is presented. In context of this thesis the purpose of enquiry is the exploration of the emerging phenomenon of using OSS in reuse-intensive software development, especially in software product lines. Therefore, this research can be categorized as exploratory research. An exploratory research design is followed to conduct this research which is presented in the next section.

3.6 Research Design

The research design of this study is depicted in Figure 3.3. The activities of research design are divided into three phases. Phase-I was started with a survey and review of the literature. The topic of research was selected after the initial survey of literature. The topic was further searched and a problem statement was formulated. After the formulation of problem statement, a thorough literature review was conducted to know and report the current state of the art in this area. The research methodology was selected by considering the relevant literature and nature of the problem.

In the next phase, interviews were conducted and a qualitative data analysis was performed. The results of qualitative analysis include attributes of reusability along with the other findings (chapter 4). A survey was conducted to rank the attributes of reusability in order to know their importance. The results obtained from the analysis of interviews and surveys were used to propose a reusability attribute model. After proposing the reusability attribute model, code level software metrics were selected for the application of model. The selected metrics fall in two categories i.e. class level metrics and package level metrics. The metrics were applied to the open source software at class level and package level.

The third phase is concerned with the validation and analysis of the obtained results. This validation and analysis was performed using statistical analysis. The results obtained from the application of metrics were statistically analyzed. An evolutionary reusability analysis was conducted on two open source software. It demonstrated the potential applicability of the proposed model and provided a deeper understanding. All of the results obtained during this study have been written in the form of research papers and presented in different journals, conferences and in relevant software engineering publications. The list of publications is provided at the end of thesis.

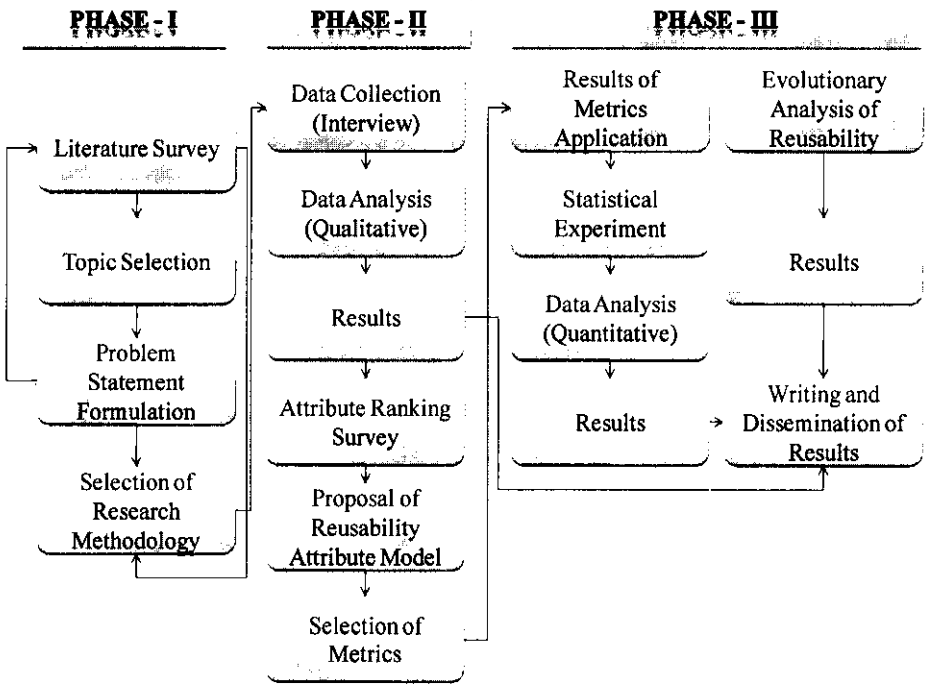


Figure 3.3 Research design

3.7 Qualitative Methods

The qualitative methods are used to collect and analyze qualitative data; data in form of words, text and pictures. These methods include document analysis, content analysis, focus groups and interviews etc. In this research, interview is used as a qualitative data collection mean. The analysis of qualitative data is performed on the

basis of content analysis principles. A brief account on interview and coding process is provided in the next section.

3.7.1 Interview

Qualitative evaluation and validation approaches seek to collect data in the form of text and pictures. The interview is one of the forms of qualitative data collection. In [109], a detailed discussion about the use of participant observation and interviewing is provided.

The interview is a means of collecting primary data; it is a conversation between two persons, one of which is a researcher. Interviews can be used for data collection where the nature of the study is exploratory. Interviews are helpful when the data to be gathered is about a person's knowledge, preferences, attitude or values [91]. Interviews are useful in situations when the logical order of the questions is not clear or predetermined [91]. Interviews may help to gather impressions and opinions about something [109]. Interviews enable one to get personalized data, provide an opportunity to probe, establish technical terms that can be understood by the interviewee, and facilitate mutual understanding. The interview provides an in-depth view for exploring the perspective of informants [91]. Interviews enable the researcher to understand experiences of others. Several types of interviews are reported in the literature [110]. In this study semi-structured type of interview is used.

3.7.2 Types of Interview

The interview is a costly and time consuming activity as compared to other data collection techniques. Therefore, it requires a deep understanding of the phenomenon under consideration. This understanding helps to select appropriate type of interview. The following types of interview are available in the literature [91].

3.7.2.1 Structured Interviews

The data collected through structured interviews can be analyzed quantitatively. A set of same pre planned question are prepared. The same questions are asked to all the respondents. The response is recorded by the researcher in a standardized manner. In some cases, structured interviews provide a basis for further open ended interviews. The role of researcher is confined to the questions and the respondent give responses to the questions. The tone, sequence and wording of the questions are kept same for all the respondents. A highly structured interview may have fixed scales for each question such as agree/ disagree.

3.7.2.2 Semi-structured Interviews

The semi structured type of interview differs from the structured interview in the way that instead of pre planned set of questions it makes use of a list of issues. These issues are discussed during the semi structured interview. There is no restriction on the sequence of questions to be asked. It may be possible that during the interview new questions are asked, viewing the flow of conversation. The data collected through semi structured interviews can be analyzed qualitatively.

3.7.2.3 Non-directive/ Un-structured Interviews

The questions asked in non-directive form of interviews are not pre-planned. The respondents freely talk about the subject. The researcher questions subject, keeping in view the research objectives. Non directive interviews are used to collect qualitative data. Unlike, the structured type of interviews, in un-structured interviews both questions and answers rest on the part of respondent. Usually, the researcher begins by asking the respondent's views of the topic being studied.

3.7.2.4 Focused Interviews

Focused interviews are useful in a scenario when the response of respondent is required on a specific situation. The researcher knows the situation prior to conducting the interview. During the interview, if the conversation diverts from the required course the researcher intervenes and takes the conversation back on track.

3.7.2.5 Informal conversational Interviews

The most open ended form of interviews is informal conversational interviews. It is flexible in a way that the course of discussion cannot be pre decided. The data collected through conversational interviews is hard to analyze. This is because of the variation of questions asked to different people. During the conversation, interviewee may influence the course of discussion. Emerging patterns are identified by the researcher to interpret the meanings.

3.7.2.6 Reason for Choosing Semi-structured Interviews

Unstructured interviews are costly in terms of time and resources as they require a lot of time to conduct the interviews and to analyze the data. On the other hand, structured interviews are efficient, requiring less time and resources. However, structured interviews follow a set pattern that does not allow for a detailed exploration of the issues. Semi-structured interviews offer a compromise, making use of both open-ended and specific questions. This combination allows the researcher to explore the issues by collecting expected information using specific questions, and unforeseen information from open-ended questions. The semi-structured type of interview is used in this study.

3.7.3 Question Formulation Process for Interview

The following process was used to formulate questions to be included in interview guide.

- General research area
- Specific research questions
- Interview topic
- Formulate Interview questions
- Review revised interview questions
- Pilot guide
- Identify novel issues
- Revise interview questions
- Finalize guide

3.7.4 Respondents' Profiles

The research issues investigated in this study are of a specialized nature. Not everybody working in industry or academia is able to answer these questions. The respondents chosen for this study is based on their expertise. It should be noted that the respondents have up to date information regarding the research in this area and industrial practices.

The interviews were audio recorded and transcribed prior to performing the analysis. The first respondent is a software engineering researcher and developer. He has an experience related to human computer interaction application development.

The second respondent is a researcher having a doctorate degree in software engineering in the area of software product lines. He is an author of many publications, some of which are book chapters. His publications include those specifically targeting software product lines and related issues.

The third respondent is an expert in software reuse research, and has been authoring research papers on software reuse since the 1980's. He actively participates in research activities and currently is the editor of a publication in software engineering published by a prestigious body. He is currently serving as the principle software architect in a well known organization.

The fourth respondent started his career as a software engineer and had been promoted to software project manager during his career. He has managed several projects in the domains of accounts, student information service, examination systems and a few others to automate the small industries and NGOs.

The fifth respondent has worked in the domains of micro finance system, accounts systems, medical laboratory systems, visa system, and billing systems.

The sixth respondent is working in a multinational software development company. He has an experience of working in the education and health sector domains. Currently, he is serving as software quality assurance engineer.

The seventh respondent is also associated with software industry, working in a well reputed and nationally certified software company. He has been involved in developing software related to project management domain.

The profiles of the respondents are diverse, which influenced the design of the interview guide and meant that not all of the questions were posed to all of the respondents. Table 3.1 summarizes the profiles of the respondents. Different means (Table 3.2) are used for conducting the interviews due to the location of the respondents. The interviews are conducted between November 2010 and December 2010.

3.7.5 Interview Guide

An interview guide helps the researcher in organizing the interview. The contents of an interview guide include the list of open-ended questions to be asked during the interview and notes to direct the interviewer to the desired direction. Like field notes, an interview guide is again confidential, i.e. it is not shown to the respondent. For novice interviewers it is usually difficult to conduct the interview and write notes at the same time. An audio recording of the interview provides a solution to this problem. The permission to audio tape the interview is essential; it is ethical binding on the researcher to inform the respondent that the conversation is being taped.

The interview guide contains open ended questions, or in other words the issues to be discussed. The conversation starts with a brief introduction by the interviewer. In our case, the introduction of the topics is not desirable because these were already known to the researchers. In fact some of the respondents are experts who are well known in the research community. The respondents answered the questions differently due to their varying knowledge and level of experience. They used examples (citing names of software) and referred to their talks with other researchers. The transcribed interviews are not presented here, neither are the names, the places or the events. The crux of the conversation and results are presented in the results section.

Table 3.1 Information about the respondents

Respondent ID	Experience	Experience Type	Current Affiliation	OSS Experience
Rsp-A	5 years	Academic, Software Industry	Academia	4 years
Rsp-B	10 years	Academic, Software Industry	Industry	7 years
Rsp-C	22 years	Academic, Software Industry	Industry	15 years
Rsp-D	8 years	Academic, Software Industry	Academia	5 years
Rsp-E	10 years	Academic, Software Industry	Academia	6 years
Rsp-F	3 years	Software Industry	Industry	2 years
Rsp-G	4 years	Software Industry	Industry	2 years

Table 3.2 Means used to conduct interviews

Means used	Skype	Face to face	Telephone	Total
Number of Interviews	3	3	1	7

3.8 Qualitative Analysis (Content Analysis)

The content analysis approach is used in this study for the analysis of qualitative data. Content analysis is a scientific tool which helps to understand the phenomenon. The content analysis is a “research technique for making replicable and valid inferences from the text (or other meaningful matter) to the contexts of their use” [111].

In [112] three approaches to content analysis are presented. These include conventional, direct and summative approaches. In this study, conventional approach to the content analysis is employed. Conventional approach is used when the available knowledge about the phenomenon is limited [112]. It helps to gain direct knowledge from participants because preconceived categories are avoided during analysis. Conventional content analysis results in model building or concept development.

Content analysis can be employed to serve either inductive or deductive research [113]. In this study the inductive content analysis is conducted due to limited knowledge availability about the phenomenon.

The analysis is conducted following the approach presented in [113] and [112]. The analysis process (Figure 3.5) is started by generating the transcriptions from the audio recorded interviews. These transcripts are read carefully to extract the open codes [113]. The open coding process results in a list of codes. The open coding process is performed by using atlas.ti software (explained in section 3.8.2). In addition to it, a word cloud is generated. This step is taken to make sure that none of the recurring words are missed. After the analysis of all the transcriptions, we have all of the key words related to the concepts. Similar ones are grouped into generic categories. The sub-categories are created through the abstraction process. The abstraction process (Figure 3.4) is continued to reach a reasonable and possible level [113]. The categories are named such that they provide meaningful insight into them. These names also emerged from the transcripts. Definitions for each category and sub-category are developed which are presented in chapter 4 along with their representative quote from the transcript.

3.8.1 Word Cloud

A word cloud is a technique used to represent the frequencies of words in textual data. On the World Wide Web, word clouds are also referred to as tag clouds. Word clouds are used to depict the relative importance, frequency, and popularity of a word [114]. The examples of using word cloud can be seen on web / search engines, news sites. However, to our knowledge word cloud is used for the first time in academic / qualitative analysis process in this thesis.

In this research, a word cloud is used in addition to ‘open coding’ to make sure that none of the recurring words are missed. The interview transcripts contain 6,483 words. A word cloud of these words is generated by an online word cloud service (www.tagxedo.com), and is shown in Figure 3.6. The cloud includes the 300 most frequently recurring words in the transcripts. The word cloud helped to ensure that the concepts related to these words are included in the code list.

3.8.2 atlas.ti

‘atlas.ti’ [115], is a specially designed software to assist in qualitative analysis. It has features such as linking, searching and sorting of data. It manages the primary documents (audio, video, text files, pictures etc.). It helps to create the ‘open codes’, and link them to corresponding quotation in the primary document. It helps to link the codes to the memos. The categories can also be defined and codes can be linked to the categories. In this study the audio recorded interviews are transcribed using this software. The ‘open codes’ are identified and multiple quotations in the audios are linked. The memos related to the codes are created and linked to the codes and quotations.

3.9 Quantitative Methods

The quantitative methods are used to collect and analyze quantitative data; data in the form of numbers. These methods include surveys and experiments. Statistical analysis may be used to interpret the results. In this research, survey is used as a qualitative data collection mean. The analysis of the data is performed using the pie charts, scatter diagrams and pearsons' correlation analysis. A brief account on the survey and experiment is provided in the next section.

3.9.1 Survey

As with other engineering disciplines, software engineering is intended to help humans in solving their problems [99]. Software engineering, being a multidisciplinary field of research, involves issues raised by technology and society (humans). Software engineering activities depend on tools and processes. However, due to the involvement of humans, social and cognitive processes should also be considered [101]. Validation of new tools and processes is a necessary part of the advancement of software engineering [83].

The involvement of humans in software engineering demands the usage of research methodologies from the social sciences. Therefore, to rank the relative importance of attributes a survey is conducted. Survey can be defined as a comprehensive system for collecting data using a standardized questionnaire [116-117].

The information collected from a survey is used to “describe, compare or explain knowledge, attitudes and behavior” [116].

Survey research is common in the software engineering discipline. Due to the effectiveness of surveys in software engineering, researchers have laid down a process to conduct surveys.

3.9.1.1 Survey Process

In [117], a comprehensive seven step process for conducting a survey is explained. In this study this process is used to design and conduct the survey. The specific steps taken to conduct this variability assessment survey were:

- Identification of aim
- Identification of target audience
- Design of sampling plan
- Questionnaire formulation
- Pilot test of questionnaire
- Questionnaire distribution
- Analysis of the results

A sampling plan was designed to decide the kind of statistical test used to interpret the results. The questionnaire was formulated and reviewed by the authors. The questionnaire was pilot tested and revised. The survey was conducted using web based survey system. The results of the survey were analyzed using statistical software.

3.9.1.2 Questionnaire Formulation

The survey instrument / questionnaire are formulated on the basis of the findings of the interview. As explained in previous section (Types of mixed method), that this study is sequential mixed method study. So, the methods are applied sequentially and findings of one method i.e. interview is used to develop the following method i.e. survey. One of the subcategory of qualitative findings i.e. factors affection software reusability is used to develop the questionnaire. The standard definitions of identified factors are studied to understand them and to formulate them into the questionnaire statements.

3.9.1.3 Survey Population

The objective of software engineering research is to provide results which are useful for the software industry. The selection of a population for a survey is one of the critical decisions. In this study the target population consists of the individuals related to the software development in Malaysia.

3.9.1.4 Sampling Technique

In this research, sample is collected using convenience sampling. The driving force to make this decision is time and resource constraints. It's a non probability sampling method. On contrary, the probability sampling method also requires the identification of every individual in the population which is quite difficult in this case.

There are two common strategies to minimize the sample biasness in convenience sampling. These include a clear description of sample collection process and participants, and to ensure that sample is reasonably representative without any bias [118]. Both of these remedies are applied in this research. The description of sample collection process is described in the next sections. It is tried as much as possible during the data collection that data should be collected from the representative population. The experience of population is depicted in Figure 3.7.

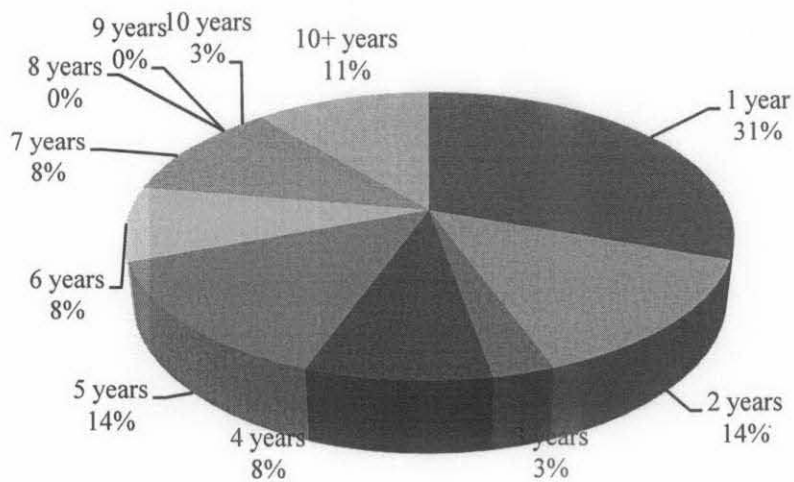


Figure 3.7 Experience of population in years

3.9.1.5 Means Used for Survey

In this research a large number of software engineers are approached through internet, Email, and personal contact. Recently the use of internet to conduct surveys is increased. The internet surveys help to access large population in less time and cost [119]. The internet surveys can be categorized as e-mail based surveys and web-based / online / internet surveys these terms are used interchangeably. In this study web based survey is used. An online survey service (SurveyMonkey) is used for this purpose. The link of survey was posted to the target population using emails, different Facebook groups and pages related to software engineering, java programming, software reuse and open source software.

3.9.1.6 Confidence Interval

The confidence interval or margin of error provides the information that “to what extent is the response from the sample likely to reflect the population of the interest?” [120]. The confidence interval is written with ‘ \pm ’ with the quantity. It shows that the range is plus or minus up to the confidence interval.

3.9.1.7 Confidence Level

The confidence level is associated to the confidence interval. It is the estimation of success rate of the method used to create the interval [121]. In this survey a 95 % confidence level is used, which is a recommended level [122].

The following sample size and related figures are calculated using the website; (www.surveysystem.com/scalc.htm). The population size is 91410 according to MSC Malaysia supply - demand study of the ICT industry [123]. The confidence interval 95% is used. The required sample size is 383; however, 396 samples are collected using different means / sources, the list is provided in appendix H. A total of more than 2707 individuals are contacted. The details are presented in Table 3.3.

Table 3.3 Survey sample size and related figures

Survey Details	
Confidence level	95 %
Population	91410*
Sample size needed	383
Population Accessed	2707+
Sample size collected	396
Confidence interval	4.91
Percentage	50
Calculated using calculator	
(www.surveysystem.com/scalc.htm)	
*Population Estimation Source : (MSC Malaysia Supply-Demand Study of ICT Industry 2009)	

3.9.1.8 Scale used in Survey

Ordinal scale is used in this study to collect the response of the population. The ordinal scale is capable of describing the order. In ordinal measurement numbers are assigned to the objects. These numbers represents the order of ranking. Likert scale is one of the example of ordinal scale [117].

The response of users collected using Likert scale is in from numbers such as 1 to 5 - strongly disagree (1); disagree (2), neither agree nor disagree (3); agree (4); strongly agree (5).

3.9.2 Quantitative Analysis

In this research work quantitative analyses are conducted using statistical data. Statistics are used to describe the results of an experiment or investigations [124]. This section contains the descriptions of statistical techniques used in this research.

3.9.2.1 Measurement Scales

Quantitative values are associated with numbers in a way that all quantities are represented using numbers. Different scales such as Nominal, Ordinal, Interval, and Ratio are used to measure and express the quantities. In this research ordinal scale is used.

3.9.2.2 Ordinal Scale

In ordinal scale, entities are categorized to form a rank order [125]. Numbers are assigned to the categories; these numbers are representative of the rank or order of the category. The order of the numbers is of the interest in ordinal scale instead of the number itself [126]. Likert scale is one of the examples of ordinal scale, where numbers are assigned to represent the order of variable. Ordinal variables have discrete values. Discrete values have a gap between consecutive values.

3.9.3 Statistical Techniques

In this section those statistical techniques are elaborated which are used in this research study.

3.9.3.1 Hypothesis and Hypothesis Testing

One of the key activities performed using statistics is to draw inference between the populations. This inference is drawn on the basis of the sample taken from the population [127]. The term ‘hypothesis’ refers to a claim or statement about the property of a population [128]. In contrast to research questions, hypothesis can be tested and predictive [91]. The hypothesis states a relationship between two or more variables.

A hypothesis test usually derives from a prior research hypothesis [129]. Hypothesis test is a procedure to test the claim about a property of population [128]. Two statistical hypothesis are formed to test a hypothesis statistically. These

two hypotheses include 'Null' hypothesis and 'Alternative' hypothesis. Null hypothesis is represented by H_0 and alternate hypothesis is represented by H_1 . The hypotheses are generated on the basis of the findings of the interviews and survey. These hypotheses are presented and tested during the experiments (chapter 5).

3.9.3.2 Pearson Correlation Analysis

The Pearson correlation coefficient r is a numerical measure that assesses the strength of the linear relationship between two variables. The following assumptions of Pearson correlation coefficient r are presented in [127]:

1. r ranges from +1 to -1 i.e. $-1 \leq r \leq +1$. The value 1 shows a perfect positive linear correlation, while the value -1 shows a perfect negative linear correlation. The value 0 represents an absence of any linear correlation.
2. A positive value of r is an indication that y will increase with the increase in x . On the other hand, a negative value of r implies that the value of y will decrease when the value of x increases.
3. r is not affected by the order of x and y , i.e. r is the same for the pairs (x, y) and (y, x) .
4. r is not affected by a change in the units of the variables.

The correlation coefficient r is a measure of the strength of the association between two variables. However, it does not implicate about the cause and effect. In other words the two variables x, y having a strong correlation, and increasing or decreasing together does not mean that x is the cause of increase / decrease in y .

3.9.4 Experiments

In this research work three experiments are conducted. The details are presented in the next sections.

3.9.4.1 Experiment – 1 (Reusability Assessment at Class Level)

The experiment-1 is intended to test the hypotheses related to the class level reusability attribute model. The results are analyzed using SPSS by calculating pearsons' correlation coefficients. The results are presented in the form of scatter plots along with the correlation values between (i) attributes and metrics and (ii) reusability and its attributes. In this experiment 103 classes are analyzed. A comparison of sample size used in this research and some of the related studies is drawn in section 3.14.2. The details regarding the specification of classes are presented in appendix D. The selection process of these classes is described in section 3.10.

3.9.4.2 Experiment – 2 (Reusability Assessment at Package Level)

The second experiment is intended to test the hypotheses related to the package level reusability attribute model. The results are analyzed using SPSS by calculating pearsons' correlation coefficients. The results are presented in the form of scatter plots along with the correlation values between (i) attributes and metrics and (ii) reusability and its attributes. In this experiment 77 packages are analyzed. A comparison of sample size used in this research and some of the related studies is drawn in section 3.14.2. The details regarding the specification of classes are presented in appendix E. The selection process of these classes is described in section 3.10.

3.9.4.3 Experiment – 3 (Evolutionary Reusability Analysis)

The term evolution refers to the “process by which different kinds of living organism are believed to have developed from earlier forms, especially by natural selection” [130]. In the context of software engineering, evolution is the “process of progressive (e.g. beneficial) change in the attributes of the evolving entity or that of one or more of its constituent elements” [131]. Two software (Jasmin and pBeans) are analyzed to assess the reusability of their packages in different versions. The details on OSS selection are presented in section 3.10.

3.9.5 Statistical Analysis Tool

The statistical analyses in this research work are performed using Statistical Package for the Social Sciences (SPSS). It is helpful in descriptive, bivariate statistical analysis and prediction for numerical outcomes, and identification of groups.

3.10 OSS Selection

In this research three experiments are conducted first at class level, second at package level and third is evolutionary analysis of different versions of two open source software. The classes, packages and software were selected on the basis of the following criteria (i) the source code should be available, (ii) to ensure the consistency of comparisons and results; all of the selected classes/packages and software should be implemented in java. (iii) For the evolutionary analysis at package level, enough versions of software should be available.

A total of 103 classes were analyzed as part of the first experiment. These classes were drawn from 15 open source software. In the second experiment, 77 packages were assessed for their reusability. The software was downloaded from multiple sources which include Sourceforge (www.sourceforge.net), Merobase (www.merobase.com), and FreshMeat (www.freshmeat.net). More details of the components are provided in appendix D and E.

The following OSS were used in the evolutionary reusability analysis. The selection was made on the basis of criterion that the selected software exhibits a significant evolution. Secondly, the source codes of all versions should be available. The selected software have research value; they have already been used in evolutionary research studies of software such as [132].

3.10.1 Jasmin

Jasmin is an open source java assembler; it converts the ASCII descriptions of java classes to their respective binary java class files. The binary class files can be loaded into java virtual machine. Six versions of Jasmin software were analyzed which are

available at (www.sourceforge.net). Detailed specifications are provided in the appendix E.

3.10.2 pBeans

pBeans is open source software which facilitates automatic object/relational mapping of java objects to data base tables. Ten versions of pBeans were analyzed which are available at (www.sourceforge.net). Detailed specifications are provided in the appendix E.

Table 3.4 Details of packages Jasmin and pBeans

Software	Versions	Classes	Methods	LOC
Jasmin	6	99 to 118	618 to 792	8256 to 11467
pBeans	10	28 to 49	161 to 341	1497 to 1057

3.11 Metrics Calculation Tool

The calculation of metrics is a time consuming and extremely laborious work. Several tools are available which can be used for this purpose. In this research work, JHawk 5 [133] was used to calculate the metrics at different levels.

3.12 Goal Question Metric Approach

The software measurements were performed under the guidelines of measurement frameworks. These frameworks ensure the suitable definition of measures for a particular entity such as process, product or resource. The Goal Question Metric (GQM) Approach [134] is one of the measurement frameworks which is widely used in software engineering. In this thesis, GQM approach is employed to define the measures.

GQM approach helps to interpret the data by providing guidance to define goals and measures to achieve those goals. The basic premise of GQM is all measurements should be goal oriented. GQM does not impose specific measurement goals to be achieved by the measurement program. It provides a structure to define goals, refinement of goals to produce quantifiable questions, and suitable measures to answer the questions. The measurement goal is achieved by collecting appropriate data. GQM consist of the following three steps:

1. Specification of goals

The specification of goal is associated with an object. There could be many reasons for specifying a goal. These reasons include the points of view, environment and quality model. There are three objects of measurement in software engineering, which are product, process and resources.

2. Generating a set of quantifiable questions

The goals of measurement are translated into operational level statement/question. The questions are related to the object of measurement. The focus of these questions remains on the selected quality issue and view point.

3. Defining a set of measures to answer the questions

The definition of measures to answer the question is the quantitative level / metric level of GQM. A data set is collected to answer the question. The nature of data may vary (objective or subjective). In a case when data depend only on the object being measured, it is objective. On the other hand, when the data depend on the object being measured and the view point from where they are taken, then it is subjective.

3.13 Validity of Research Results

The validity or trustworthiness [135] of research refers to the soundness of the research study. The term validity also implies that “the research actually measures or describes the phenomenon it sets out to measure or describe” [136]. It is one of the component of good research that it uses procedures to validate the data, results and their interpretations [90].

The notion of validity in research studies is a complex phenomenon. Validity of research falls under several kinds. [136] states, it cannot be said for a research study that it has no threat to validity. One has to accept that there are standard errors in quantitative research and participants' subjectivity in qualitative research.

3.13.1 Validity of Qualitative Results

In qualitative research community the term validity or trustworthiness is used to express the quality difference of research studies [137]. The claims of qualitative studies cannot be generalized. On the other hand, results of the qualitative study cannot be considered invalid on the basis of this ground [136].

Qualitative studies are more inclined towards the exploratory purposes. Qualitative studies are more open-ended as compared to quantitative studies. The results are based on the interpretations of researcher. Hence, one of the potential threats to qualitative study is researchers' bias [137].

The other factor which influence the validity of qualitative research is the use of techniques, methods and the strategies used during the research process [135].

Six types of validity are considered important in context of qualitative research. These types include descriptive, interpretive, concurrency, internal, external and theoretical validity [137].

3.13.1.1 Descriptive Validity

The descriptive validity of a qualitative research study refers to the factual accuracy of the reported findings of the study [137]. The accuracy of descriptive information collected by the researcher during the qualitative data collection is more of concern to this validity. The crux of this validity is the correct reporting of events, behaviors, settings, people, places and times [138].

3.13.1.2 Interpretive Validity

In qualitative research studies interpretive validity has a key role. Its importance increases due to the fact that qualitative data is available in the forms of text, audio, video, and pictures. Qualitative data need interpretations. Interpretive validity is concerned with portraying the correct meanings given by the participant [138].

Interpretive validity of qualitative research findings is the level of correct understanding related to experiences, thoughts, view points, feelings and intentions [137].

3.13.1.3 Theoretical Validity

Theoretical validity of results answers the questions of ‘how’ and ‘why’ the phenomenon being study operates [138]. Theoretical validity of qualitative findings can be demonstrated by building such theatrical explanations which fits the data [138].

3.13.1.4 Internal, External and Concurrency Validity

Internal validity is related to the correct mapping of findings to the phenomenon in question [136]. In context of qualitative research internal validity exhibits the plausibility and credibility of results. Internal validity of qualitative research is ensured by providing evidence in favor of claims. The other factor that ensures the internal validity of qualitative results is the clearness of nature of claims i.e. description, definition, explanation or generation of theory [136].

External validity of results comes into the picture when there is a question of generalization of the results. The generalization of results is not the purpose of qualitative studies that’s why the external validity of such studies tend to be weak [137].

Concurrent validity of data is shown by the correlation of data collected using multiple sources [136]. Concurrent validity increases the confidence in the findings of the research study.

3.13.2 Validity of Quantitative Results

Quantitative results are based on statistics. The validity of statistical results refers to the validity of relationship of two variables and the strength of the relationship [137]. Two statistical inferences are made about the variables. First inference is about the existence of a relationship, and the second is about the magnitude of the relationship.

The validity of quantitative results or validity of conclusions based on such results is the extent that the conclusions about null hypothesis are reasonable or correct [139]. The conclusions made on the basis of statistical results are the outcome of hypothesis testing. Two types of errors are associated with hypothesis testing; which are Type I and Type II errors [140]. Type I error is the case when H_0 is rejected wrongly and H_1 is accepted. Type II error is the case when H_0 is accepted wrongly and H_1 is rejected.

Both of the errors (Type I and II) reduce the validity of quantitative results. The probability of error or statistical significance (p value) is set by the researcher.

3.13.3 Validity of Mixed Method Results

Mixed method research studies make use of both qualitative and quantitative methods in a single study. So, by keeping this in mind all types of qualitative and quantitative validity are relevant to the mixed method studies / results [137]. In [141], nine types of validity are identified which are associated with the mixed method research studies. The types include the followings:

3.13.3.1 Sample Integration Validity

Sample Integration validity is concerned with the integration of multiple samples. The sample integration validity threat arises in a situation when a focus group is conducted

as qualitative part, and a survey is conducted as quantitative part. In this situation, a careful treatment is needed to generalize the results. The underlying reason is that different groups have different beliefs.

3.13.3.2 Inside-outside Validity

Inside-outside validity concerns with appropriately employing and presenting the insiders' and observers' view to describe and explain the phenomenon. The threat of inside-outside validity can be minimized by peer reviewing. The researcher may get help of another researcher to review the interpretations and conclusions made from the data.

3.13.3.3 Weakness Minimization Validity

Mixing the research methods in a single study is the basic premise of the mixed method research. Weakness minimization validity refers to the extent to which shortcomings of one approach are overcome by using the other approach. The results of different methods complement each other and increase the quality of mixed method study.

3.13.3.4 Sequential Validity

Sequential validity addresses the issues related to the sequence of qualitative / quantitative methods employed in a mixed method study. The threats to the sequential validity of research study are minimized by analyzing the effect of changing the order of methods i.e. using qualitative method prior to quantitative or vice versa.

3.13.3.5 Conversion Validity

Conversion validity is associated with the transformation of data during the mixed method study e.g. quantizing the qualitative data or qualitzing the quantitative data.

The quantitizing of qualitative data may include counting the words and qualitizing may include converting numbers into themes. The conversion validity threats can be removed by accurate conversion of data.

3.13.3.6 Pragmatic Mixing Validity

Pragmatic mixing validity refers to the extent to which the philosophical beliefs that underlie qualitative and quantitative are combined. The pragmatic mixing validity can be achieved by using pure assumptions of both qualitative and quantitative parts. The conclusions are drawn on the basis of the two components of the research study.

3.13.3.7 Commensurability Validity

This type of validity is concerned with the worldview of mixed method researcher. Commensurability Validity can be achieved by the mixed method researcher by switching from qualitative to quantitative viewpoint. This switching of viewpoint provides the researcher with a more fully mixed worldview.

3.13.3.8 Multiple Validates

Multiple validates refers to the degree of successful resolving of components of a mixed method research i.e. qualitative and quantitative components. It concerns with the extent to which the relevant validity types of qualitative and quantitative methods employed in the study are handled.

3.13.3.9 Political Validity

Political validity of mixed method research refers to the level of representation of interests, values and standpoints of multiple stakeholders. The other causes of political validity threats include use of different researchers for qualitative and

quantitative phases. The threats to political validity can be minimized by understanding the key stakeholder groups, their issues and concerns.

3.14 Validation of Findings in Context of This Study

The validity of research findings refers to the trustworthiness of results. A brief account on the types of validity of qualitative, quantitative and mixed methods has been provided in previous sections. Here, in the next sections arguments are provided that, how the validity of results are upheld during the study.

3.14.1 Validation of Qualitative Findings

The validity of qualitative research is related to following types; descriptive, interpretive, concurrent and theoretical validity. The descriptive validity is related to the reporting of events, behaviours, settings, people, places and time is not more of the concern in this study. The interpretive validity is more of the concern regarding this research. Whenever, there is ambiguity the transcriptions are reviewed by the researcher to ensure the interpretive validity of results. Furthermore, the findings of the qualitative studies are provided to the respondents. This measure was taken to cater for possible apprehensions of respondents about the results. The respondents verified the interpretations.

The theoretical validity of results is maintained by comparing the findings of this research study with the contemporary studies. It can be safely said that the findings presented in this thesis are in line with the available theory.

Concurrent validity of results is exhibited by the fact that the qualitative data were collected using seven interviews. Similar patterns and trends are identified from the collected data. The only findings are reported which are concurrent and extracted from multiple respondents. The findings of two categories, namely other considerations and suggestions are not in subject to the concurrent validity.

3.14.2 Validation of Quantitative Findings

The validity of quantitative results is suffered by type I and type II errors. Both of these errors are related to either rejection or acceptance of the null hypothesis. In this research study, the lower value of probability is used i.e. 0.01. This lower level of p-value ensures minimum possibility of type I or type II error in the results.

Furthermore, the validation of findings can be seen from the number of samples used in the experiments. In [69], 125 java bean components are used to validate the approach. These components were selected from single source. In another study i.e. [27] 12 interfaces are used for validation. In this thesis 77 packages and two software; comprising of 6 and 10 versions (68 packages) are analyzed for validation. The number packages in them are 4 and 8 respectively. However, the results cannot be compared with these studies due to the fact that these approaches are black-box approaches and the approach used in the thesis is white-box approach. The comparison is presented in Table 3.7.

Table 3.5 Experiment sample comparison-1

Study Reference	Classes used
(Etzkorn et al., 2001)	18 in one experiment and 25 in second experiment
This Thesis	103

Table 3.6 Experiment sample comparison-2

Study Reference	Components used	LOC
(Gui and Scott, 2007)	HTML parser	994
	Lexical tokenizer	1288
	Barcode generator	2323
This Thesis	Jasmin (6 versions)	8256 in (V1.0) to 11467 in (V2.4)
	pBeans (10 versions)	1497 in (V1.0) to 1057 in (V2.0.4)

In chapter five the results of experiment-1 are presented, which are obtained using a sample of 103 classes. In a similar study i.e. [67], 18 and 25 classes are used in two experiments carried out to validate the reusability assessment approach. However, the context of their study was different; their approach is meant for software developed in C language.

In [28], coupling metrics are used to assess the reusability. The approach is validated using 3 components. These components have 994, 1288 and 2323 lines of code. In this thesis, multiple versions of two software are used to analyze the proposed approach. The sizes of software (Table 3.6) are fairly large than the earlier study.

Table 3.7 Experiment sample comparison-3

Study Reference	Component used
(Washizaki et al., 2003)	125 from one source (jar)
(Boxall and Araban, 2004)	12 interfaces
(Eun Sook et al., 2001)	1 case study (application)
This Thesis	77 packages from multiple sources Jasmin (3 packages X 6 versions = 18 packages) (1 package X 4 versions = 4 packages) pBeans (3packages X 10 versions = 30 packages) (4 packages X 3 versions= 12 packages) (1 package X 4 versions= 4 packages) Total = 77 + (18+4+30+12+4) = 145 packages

3.14.3 Validation of Mixed Method Findings

Mixed method studies use both qualitative and quantitative methods. So, all the validity types of qualitative and quantitative methods are applicable to the mixed

method studies. Apart from these validations, researchers have identified some validity types for mixed method studies.

The results are not affected by the sample integration validity threat. Sample integration threat arises in situation when opinions of different groups, collected using different methods are mixed. In the case of this thesis, the methods are applied in a sequential manner. Interviews were conducted with the experts and then a questionnaire was formed on the basis of the results of the prior interviews. Secondly, all respondents represented the same role (software engineer who uses OSS). This ensures that the perception or viewpoint of all respondents is the same.

The inside-outside validity is maintained during the study using a peer review method. The conclusions and interpretations were reviewed by fellow researchers (co-supervisor/supervisor).

The weakness of the results is minimized by employing different methods. The findings of the qualitative study cannot be generalized. So, a survey was conducted to have more confidence in the results. Furthermore, statistical techniques were used to test the hypothesis based on the findings of the interview and survey.

The sequential validity of the results was kept in mind during the study. The sequence of application of methods was decided prior to the commencement of research. The nature of study is exploratory, as the phenomenon under investigation has not been much worked on by the previous researchers. So, qualitative method was used to lay down the basis for the quantitative methods.

The results of the study are not affected by the conversion validity threat. The reason is during the analysis qualitizing or quantitizing of data is not performed.

The pragmatic validity of the results is ensured by using the pure assumptions of qualitative and quantitative methods. The conclusions are drawn on the basis of the results acquired by different components of the study.

The multiple validates validity is exhibited by the results as the validity threat of qualitative and quantitative methods are kept in view. As argued in the previous sections, the political and commensurability validity threats are not the issue in this

research. The research is conducted by only one researcher, which minimizes the political validity threats. However, the opinions and some reviews were conducted by the fellow researchers; the methods were applied sequentially. So, there is a switching of viewpoints starting with the qualitative and ending at quantitative. This switching provided a mixed worldview, which is necessary to achieve the commensurability validity.

3.15 Summary

Research is a process to acquire knowledge. Epistemology is the philosophy of knowledge. It deals with the underlying assumptions of knowledge that ‘what is knowledge?’ and ‘how to acquire it?’ Epistemology is related to different theoretical perspectives or paradigms. Pragmatism is an emerging research paradigm which is based on the mixing of qualitative and quantitative methods. Studies conducted within the pragmatism paradigm are referred as mixed method studies.

This chapter includes a brief overview of the qualitative and quantitative methods employed in this study. The qualitative method used in this study is interview; a face to face conversation between the researcher and the subject. The quantitative methods used in this study include survey and statistical experiment. Survey, which is a comprehensive data collection mechanism based on questionnaires filled by the survey population. Statistical experiment is the use of appropriate statistical techniques to interpret the quantitative data. In this study scatter diagrams, pie charts, and pearsons’ correlation analysis were used to understand the data.

The last section of this chapter provides a brief account on the validity of research results. The validity of research shows soundness and credibility of results. Validity in the context of qualitative research is related to descriptive, theoretical and interpretive validity. The validity of quantitative results is based on the acceptance or rejection of null hypothesis.

CHAPTER 4

IDENTIFICATION OF CATEGORIES & DIMENSIONS IN REUSING OSS

All the great truths are simple in final analysis, and easily understood; if they are not, they are not great truths. (Napoleon Hill, 1883-1970)

4.1 Overview

In this chapter the categories and dimensions of reusing OSS in reuse intensive software are presented. One of the identified dimension i.e. factor affecting reusability is carried to the quantitative phase and a questionnaire is developed for survey. The results of survey are presented in this chapter. The second section contains the comprehensive analysis of variability implementation mechanisms. The variability implementation mechanisms are mapped to the variability types, scope, and binding time.

4.2 Categories & Dimensions of Reusing OSS in Reuse Intensive Environment

The findings of the interviews are presented in this section. These findings emerged in seven categories. The names and descriptions of these categories are provided in Table 4.1.

CHAPTER 4

IDENTIFICATION OF CATEGORIES & DIMENSIONS IN REUSING OSS

All the great truths are simple in final analysis, and easily understood; if they are not, they are not great truths. (Napoleon Hill, 1883-1970)

4.1 Overview

In this chapter the categories and dimensions of reusing OSS in reuse intensive software are presented. One of the identified dimension i.e. factor affecting reusability is carried to the quantitative phase and a questionnaire is developed for survey. The results of survey are presented in this chapter. The second section contains the comprehensive analysis of variability implementation mechanisms. The variability implementation mechanisms are mapped to the variability types, scope, and binding time.

4.2 Categories & Dimensions of Reusing OSS in Reuse Intensive Environment

The findings of the interviews are presented in this section. These findings emerged in seven categories. The names and descriptions of these categories are provided in Table 4.1.

Table 4.1 Categories and their description

Category ID	Category Name	Description
Cat-1	Challenges in OSS	These challenges are wide-ranging: the customer's viewpoint; the end user's viewpoint; commercial and secure application development issues.
Cat-2	Current reuse practices	Knowledge about current reuse practices as employed in industry are combined in this category. This knowledge is based on the experience of the respondents.
Cat-3	Using OSS in SPL	The views of the respondents on the use of open source software in product lines are put together in this category.
Cat-4	Role of OSS in promoting reuse	This category is based on the role of OSS in the promotion of reuse, i.e. why OSS is influencing reuse intense software development?
Cat-5	Factors affecting reusability	The factors of reusability are assembled under this category.
Cat-6	Desirable characteristics of OSS	The desirable characteristics of OSS, identified during the study, are presented in this category.
Cat-7	Suggestions	The suggestions provided by the respondents are presented in this category.

4.2.1 Challenges in OSS

In this section the challenges in OSS are presented, and categorized on the basis of the opinion of the respondents. These challenges fall in different dimensions. The list of challenges (sub categories) and their corresponding representative quotes are presented in Table 4.2.

4.2.1.1 SC-1-1 Finding OSS

The very first challenge in OSS is searching for it. Searching facilities are improving with the emergence of new search engines. Furthermore, enormous contributions are being made by numerous software engineers. The availability/accessibility of OSS has improved but it is still a challenge to find specific OSS. One of the reasons is that different cataloguing standards are employed by search engines. This lack of a standard makes it difficult for a new user (i.e. a software engineer) to search for a component using different search engines.

4.2.1.2 SC-1-2 Evaluating OSS

The evaluation of the OSS is another challenge. For example in case, in the first step (finding an OSS) when a required component is found, then the decision whether to use it or not is related to the evaluation of the OSS. The practices to evaluate OSS differ in different organizations. The evaluation of OSS prior to using it is the discretion of software engineer in small organizations. In such environments this evaluation depends on the knowledge and expertise of the software engineer.

4.2.1.3 SC-1-3 Lack of Documentation

Lack of documentation is related to the understandability of software. Lack of documentation affects the understandability/analysis of the software. So, without having appropriate documentation it is difficult for software engineers to use it. One of the reasons for this issue is; a large number of developers contribute to many OSS components, and this complicates the provision of documentation.

The respondents consider documentation as the most important quality of OSS. The quality of documentation reflects the quality of the software. Another aspect of documentation is that it provides a record of the changes made to the OSS, i.e. it gives its history.

amounts of time understanding others code, software engineers would prefer to write their own.

4.2.1.7 SC-1-7 Security

There is a security concern when using OSS in critical and highly secure application domains, such as defense, government and financial sectors. In such situations there should be mechanisms for code scanning to ensure that the code is clean, meaning that there is nothing malicious in the code.

4.2.1.8 SC-1-8 Improper Reviewing/Comments

Reviewers'/users' comments about OSS play an important role for the potential user of OSS. One can learn about the software prior to downloading and using it. There is a huge amount of code (software) available over the Internet. Several users have written comments about some of it. However, the issue is that there is no standard for reviewing code and writing comments about it. The person who writes a comment shares a personal experience with a particular piece of software. Sometimes, the context of its use is not clear, which raises questions in the reader's mind. It is suggested by the respondents that there should be standards for writing comments about software so that OSS users can easily extract the required information.

4.2.1.9 SC-1-9 Fear of Losing Market Share

This issue is more relevant to product line and domain based software development, where companies target a specific domain and group of potential customers. In such situations the software organizations may use OSS but do not want to contribute software to an OSS repository because they want to keep their innovations to themselves. In this way they sustain themselves in a particular domain. There is a fear associated with sharing - if they share code they will risk losing their position in the market.

Table 4.2 Sub categories of challenges in OSS

Cat-1		Challenges in OSS	
Sub Category ID	Sub Category Name	Representative Quote	
SC-1-1	Finding OSS	“Finding an OSS component is one of the challenges”.	
SC-1-2	Evaluating OSS	“If I find a required OSS component then its evaluation is a challenge”.	
SC-1-3	Lack of documentation	<p>“...without proper information it is difficult to understand it”.</p> <p>“If there is no proper documentation then others cannot understand the software neither can change nor modify it”.</p> <p>“The challenge in the context of the open source is analyzing, usually OS comes along with source code without many documentation. So, it is very difficult to analyze without documentation.”</p>	
SC-1-4	Reluctance for developers to make their software OSS	<p>“They don’t want to contribute to the open source because they want to run their software house ...”</p> <p>“they are willing to use the OS but not to contribute to the OS because of their limitation and because of the market competition”</p>	

Table 4.2 Sub categories of challenges in OSS (cont.)

Cat-1		Challenges in OSS (cont.)	
Sub Category ID	Sub Category Name	Representative Quote	
SC-1-5	Lack of information about intellectual property rights /copyright	“The developers have lack of information about the intellectual property rights in OSS, so they are afraid to share code”. “Lineage of the software ensuring that no intellectual property so that is the biggest hesitation”.	
SC-1-6	Lack of adherence to coding convention /standard	“There should be some specific rules, common rules for each for the whole developer community or those contributing to the OSS”. “If some immature developer is developing the software defiantly the code would be different from the professional developer”.	
SC-1-7	Security	“Any secure system which included OS but still there could be certain measures if the OSS. They did a scan on the source code rather than they incorporated as binary and they could do the necessary analysis to know that the OSS does not have any malicious code, entered in the software”.	
SC-1-8	Improper reviewing /comments	“I have seen some customers/users of OSS review but the main problem is there are no rules for writing a review, every reviewer is writing the review in their own context in their own way”.	
SC-1-9	Fear of losing market share	“If they develop a tool or software and they contribute of float it as OSS there are chances that they can’t further work/earn”.	

4.2.2 Current Reuse Practices

The findings which fall under this category are related to the knowledge about current reuse practices as employed in industry are combined in this category. This knowledge is based on the experience of the respondents. The list of sub categories is presented in Table 4.3 with their corresponding representative quotes.

4.2.2.1 SC-2-1 Knowledge Reuse

There is a form of reuse in which knowledge is reused. Imagine a situation where a software engineer is searching for a component written in one language but can only find it written in another language. The software engineer can reuse the knowledge inherent in the logic but rewrite the code. This form of reuse is helpful where the bulk of the logic remains the same and objectives of reuse are achieved with some changes / adaptation. Demo version of software is also related to the knowledge reuse, as explained in next section.

4.2.2.2 SC-2-2 Demo

Much of OSS comes with a demo version; this is very helpful for understanding the software. The potential user of OSS can base his/her decision of whether or not to use the component on the success of using the demo version. In other cases, the software engineer wishes to use a component following some modifications. A demo aids the software engineer in thinking how best to make the modifications. Software engineers can sometimes gain a better idea of the functionality through using a demo rather than studying a large amount of code. In this way target areas of the program requiring modification can sometimes be more easily identified.

Table 4.3 Sub categories of current reuse practices

Cat-2 Current reuse practices		
Sub Category ID	Sub Category Name	Representative Quote
SC-2-1	Knowledge reuse	“We can just take an idea and we can reuse the idea”.
SC-2-2	Demo	“Demo could be helpful for programmers they can understand the software by a demo of the software, demo gives an idea to the programmer...”.
SC-2-3	Not started from scratch	<p>“Now I think this concept that you start developing a product from scratch is impractical because right now the time is scarce in the world”.</p> <p>“Software product lines are started with having some component in hand means the company is already working in this domain and that is why OSS may help them to start product line or to add new product into the line”.</p>

4.2.2.3 SC-2-3 Not Started From Scratch

There is an opinion that starting a product ‘from scratch’ is impractical. Software engineers start developing with having some components at hand. It saves time and other resources. At a higher level the same goes for software product lines. Product lines are started after having prior knowledge and an awareness of their applications

in the domain. Reuse as an aid to getting started can facilitate a competitive advantage, allowing a new software product to be developed in a short period of time.

The association of an organization with a specific domain helps to develop trust in its new products. The customer prefers software products from companies that are already established in a specific domain.

4.2.3 Using OSS in SPL

This category is based on the views of the respondents on the use of open source software in product lines. The sub categories are presented in Table 4.4 with their corresponding representative quotes.

4.2.3.1 SC-3-1 Fast Transition

The use of OSS to develop a product line provides an opportunity to develop a new product in less time. On the other hand, it hastens the transition of manual to automated systems. Many of the systems which we interact with in our daily life are similar in nature. SPLs deal with such similarities (commonalties). The infrastructure of OSS core assets can be used to initialize many specialized products.

4.2.3.2 SC-3-2 OSS is Attracting SPL Community

OSS is attracting the product line community in the sense of starting a new product. An organization can develop a new product in less time using OSS. On the other hand, product lines are seldom started from scratch. So, OSS provides a good start to the SPL community. Obviously the standards and quality of OSS is an issue in the case of OSS based SPLs.

Table 4.4 Sub categories of using OSS in an SPL

Cat-3 Using OSS in SPL		
Sub Category ID	Sub Category Name	Representative Quote
SC-3-1	Fast transition	“If such setup is developed (using open source to build family of systems) then the transition from manual to computerize will be much faster.”
SC-3-2	OSS is attracting SPL community	“OSS is very attractive to product line community”.
SC-3-3	Improvement in quality	“The number of times a component is reused its quality is improved”. “Reuse also refines the product”.
SC-3-4	Provides opportunities	“It’s a good opportunity that you have idea or free code and then you develop product lines”.

4.2.3.3 SC-3-3 Improvement in Quality

The reuse of software improves its quality. However, in the case of SPLs it is even more beneficial. The core assets are reused in multiple applications; this reuse refines the components. The more times a component is reused the greater is its quality. Another aspect of this is that the end user/customer is in a better position to state his requirements and comment on a system after using a similar one.

4.2.3.4 SC-3-4 Provides Opportunities

The respondents considered OSS based SPLs as a window of opportunity. This reuse may lead to the inter-organizational reuse of the components. Reuse will be moved to a higher level, one that finds commonalities among domains. Such types of core assets will be developed which are used by multiple domains.

4.2.4 Role of OSS in promoting reuse

This category is based on the role of OSS in the promotion of reuse, i.e. why OSS is influencing reuse intense software development? The sub categories and their corresponding representative quotes are presented in Table 4.6.

4.2.4.1 SC-4-1 Saves Time

The top most benefit of reuse is that it saves time. Software engineers prefer to reuse the component if it is already available. On the organizational level and in product line practices reuse results in a short delivery time. New products can be launched in a shorter time. It provides a competitive edge and there is time available for experimentation and innovation to enhance the features of the product.

4.2.4.2 SC-4-2 Less Effort Required

Reuse also results in saving effort. Less effort is required to develop software using OSS as compared to starting from scratch. In the context of SPL, product lines are seldom started from scratch. The organizations which move towards the development of SPL already have experience of that particular domain.

4.2.4.3 SC-4-3 Ease of Development

OSS allows for easier development; that is why its use is encouraged in the software industry. Developing a new product is comparatively easy using OSS. A developer, who is new to the domain, can get domain knowledge by reusing the components.

4.2.4.4 SC-4-4 Market Trust

The reuse of OSS enables a company to launch a product more rapidly. A potential customer prefers products of a company that is already developing software in that particular domain. So, indirectly, OSS usage develops the trust of the customer.

Table 4.5 Sub categories of role of OSS in promoting reuse

Cat-4 Role of OSS in promoting reuse		
Sub Category ID	Sub Category Name	Representative Quote
SC-4-1	Saves time	"...it saved time otherwise it will take weeks to develop it".
SC-4-2	Less effort required	"It saves around 80% of the time and less effort is required".
SC-4-3	Ease of development	"Nowadays open source development is encouraged within the software development community, and people are going towards open source, they feel very easy to develop".
SC-4-4	Market trust	"The company which reuse can develop a system in months and other may take a year... the company which is running a product line in some domain it also develop a trust".

4.2.5 Factors Affecting Reusability

The factors identified as the attributes of reusability are assembled under this category. The subcategories and corresponding quotes are presented in Table 4.7.

4.2.5.1 SC-5-1 Flexibility

Flexibility is related to reusability in two ways. First, it is the ability of a component to be used in multiple configurations. Second, it is a necessary attribute concerning future requirements and enhancements.

4.2.5.2 SC-5-2 Maintainability

Maintainability is related to reuse in terms of error tracking and debugging. If the component is maintainable it is more likely to be reused. In cases where OSS components are running on systems connected to another system then a bug is particularly problematic. Sometimes debugging a component on one configuration may not work on other configurations. On the other hand, in black box reuse, maintainability is not considered a factor of reusability.

4.2.5.3 SC-5-3 Portability

Portability is considered a factor in the sense that a cohesive component is more portable. A component having all the necessary information within it or having less interaction with another module during its execution is more reusable. Again in the case of black box reuse it is not a factor.

4.2.5.4 SC-5-4 Scope Coverage

Another characteristic of the open source components explored is the extent of its scope. A developer would prefer a component to cover as much of the application's functionality as possible. Size is a concern in large components as it often means a

high level of complexity and poor understandability. Furthermore, scope coverage is important in situations where future enhancements have already been envisioned, or where there is the likelihood that more features will be added in the future.

4.2.5.5 SC-5-5 Stability

The respondents regard stability as an important factor to be considered while making decisions. Stability of a component refers to its quality of being error free. Here, the term ‘stability’ can be linked to ‘safety in numbers’, that is, a reasonable number of developers has contributed to the component and, furthermore, a reasonable number has used it. Stability is also related to the usage history of the component.

4.2.5.6 SC-5-6 Usage History

Usage history provides a hint about the usefulness of the component. Another side of usage history is the maturity of the component. The component can be considered mature, if it is used in many applications. The use of component in many applications also reflects its quality of interoperability. It provides a confidence to the potential user that component can be easily adapted. Another aspect of usage history is that the use of a particular OSS in different applications provides an example of usage of the component. This example can be effective for learning purpose.

4.2.5.7 SC-5-7 Understandability

The respondents also have a consensus of opinion on the understandability attribute. It is also related to the maintainability of the component; a component that is easy to understand is easy to maintain. Understandability also affects the reliability of a component.

Table 4.6 Identified factors and representative quotes

Cat-5 Factors Affecting Reusability			
Sub Category ID	Sub Category Name	Representative Quote	
SC-5-1	Flexibility	<p>“Flexibility refers to the ability to use it in multiple configurations”.</p> <p>“In order to reuse some component source code it should be flexible enough to be used in several contexts”.</p> <p>“Flexibility is necessary because there are changes required with the passage of time, so it saves you not to be bound”.</p>	
SC-5-2	Maintainability	<p>“Maintainability is a large problem in such situations when you use OSS and we are running the system with connectivity with other systems; so every time there are some bugs and removing the bugs in other code that is developed by some else is very difficult for the developer”.</p>	
SC-5-3	Portability	<p>“Portability is also related to the install ability, it should be taken care and portability should be economical we don’t have to install other software to run a component in other systems”.</p>	
SC-5-4	Scope Coverage	<p>“That depends on the situation but normally we choose the more coverage component as compared to the less covered one”.</p> <p>“... it depends on the application if we want to extend further our application then we will go for more features”.</p>	

Table 4.6 Identified factors and representative quotes (cont.)

Cat-5 Factors Affecting Reusability (cont.)		
Sub Category ID	Sub Category Name	Representative Quote
SC-5-5	Stability	“Stable meaning reasonably error free and it could be used with confidence that there is no bug”.
SC-5-6	Understand-ability	<p>“If I don’t understand it then I can’t show that it is reliable and prove it to myself then I am not going to use it”.</p> <p>“Size can be managed but if it is not understandable then it is difficult to reuse”</p>
SC-5-7	Usage History	<p>“Usage history also shows the maturity of the component and how many people have used and made changes to it”.</p> <p>“In many cases open source software is used by many people many engineers, already proven its usefulness”.</p>
SC-5-8	Variability	“Variability is a two edge sword in other words there are advantages and disadvantages”.
SC-5-9	Documentation	<p>“If there is lack of documentation then I mean it creates hurdles to understand the code for any other developer or the software engineer”.</p> <p>“If there is no proper documentation then others cannot understand the software neither can change nor modify it”.</p>

4.2.5.8 SC-5-8 Variability

Variability is one of the factors identified. Variability is also seen as the configurability of a component, that it can be configured in multiple configurations. Variability is also related to the scalability property of component that it can be scaled up whenever required.

4.2.5.9 SC-5-9 Documentation

The respondents consider documentation as one of the important factors that affect the flexibility, understandability and reusability. The issue of documentation is multifaceted. Usually, OSS comes without much documentation. OSS is developed and contributed by many developers. The number of developers may reach up to thousands, like in the case of Linux. The code size increases rapidly. So, it is very difficult to analyze code without documentation.

Documentation is associated with understandability. The lack of documentation or poorly maintained documentation hinders understandability. Documentation also provides a record of the component, component history can be known by seeing the documentation.

4.2.6 Desirable Characteristics of OSS

The desirable characteristics of OSS, identified during the study are presented in this category.

4.2.6.1 SC-6-1 Academic Perspective

The desirable characteristics of OSS, from an academic perspective include the availability of test cases with the open source software. The primary focus of an OSS developed in academia is innovation and functionality. OSS in academic settings is intended to extend the body of knowledge. There is a room for experimentation in academia.

4.2.6.2 SC-6-2 Industrial Perspective

The desired characteristics of software from business point of view differ from that of academic perspective. Firstly, there is no room for experimentation in business environments or commercial software development. The critical factor in business environment is risk aversion. Several methods are used for assessment and mitigation of potential risks.

4.2.6.3 SC-6-3 Maintenance Support

Maintenance is one of the issues in OSS. This is because of the shared/lack of ownership of software. The potential user of OSS looks for its maintenance support. This factor is important such that it influences the decision to use a particular OSS. The OSS with maintenance support is preferred over the other which doesn't have this support.

4.2.6.4 SC-6-4 Maintenance Agreement

In some situations companies may opt for a maintenance agreement with the developing organization. This kind of agreements covers extensions or changes in software. On one hand, the customer prefers maintenance agreements for their future needs or enhancements in the systems. On the other hand, software developing companies earn additional revenues from these agreements.

4.2.6.5 SC-6-5 Infrastructure Support

Infrastructure support by the OSS is one of it's a desirable characteristic. Here, the term infrastructure refers to operating system, web application server or the graphical user interface. So, the OSS with better infrastructure support is preferable. Better infrastructure support makes the OSS an affordable choice under most circumstances.

Table 4.7 Desirable characteristics of OSS and representative quotes

Cat-6 Desirable characteristics of OSS		
Sub Category ID	Sub Category Name	Representative Quote
SC-6-1	Academic Perspective	“In academics because you are not delivering the PL for business purposes and the value is or the basis is extending the body of knowledge and helping other researchers to develop or break through in new area of software capabilities or demonstrating new algorithms or infrastructures whatever its purely functionality and functionality could be the number of test cases delivered with the open source product and the ease of use.”
SC-6-2	Industrial Perspective	“...the critical business importance is not to take risks this is known as the risk assessment, how risky it is and for risk there are several ways of determining risks”.
SC-6-3	Maintenance Support	“...support for the open source software is the number one characteristic we look for”

Table 4.7 Desirable characteristics of OSS and representative quotes (cont.)

Cat-6 Desirable characteristics of OSS (cont.)		
Sub Category ID	Sub Category Name	Representative Quote
SC-6-4	Maintenance Agreement	“companies may opt for maintenance agreement with the developer company, at any time during the agreement if there is a need of extensions or change company can provide support”
SC-6-5	Infrastructure Support	“I would be looking for the software that has the capabilities that I want to include in my product line but also the capabilities that I need to support the services or the infrastructure”
SC-6-6	Maturity of OSS	“...we look for is how mature it is and if we have to change anything to make it work, how many examples of it are being used in software community”
SC-6-7	Error Handling Mechanism	“When an error happens, what type of interrupt /what type of message is passed back? What type of parameters is required to handle the error?”
SC-6-8	Scalability	“The biggest variability parameter, I am concerned with is the scale of the work... in other words, will this component scale?”

4.2.6.6 SC-6-6 Maturity of OSS

Maturity of OSS also plays an important role while making the selection decision. One way to know the maturity of an OSS is to look for the usage of a particular OSS in different scenarios. These examples of use provide an idea to potential user. The potential user may find similarities or differences in examples and his situations. The comparison helps him to build a confidence in a particular OSS. The potential user may identify related threats and risks.

4.2.6.7 SC-6-7 Error Handling Mechanism

The availability of error handling mechanism is a desired characteristic of OSS. Error handling mechanism includes the knowledge about the error types, related messages and their remedies. The remedy may include the types of parameters required to remove the error.

4.2.6.8 SC-6-8 Scalability

The capability of OSS to be scaled is considered a desirable characteristic. The scalability of OSS is its ability to handle the growing needs of the organization. It is also considered a variability parameter, that new functionality can be added or extension of current functionality.

4.2.7 Suggestions

The suggestions provided by the respondents are presented in this category.

4.2.7.1 SC-7-1 Inter Language Reuse

Inter language reuse of OSS is one of the suggestions. It can be viewed as a challenge to software engineering. The generic artifact like design documents, requirement specification can be implemented in different languages. It is due to their abstract nature. However, the code assets lack this level of abstractness. One of the possible

solutions is the conversion of code of one language to another with the help of some intermediating software.

4.2.7.2 SC-7-2 Software Agents

The development of software agents is suggested by the respondents. These agents should be capable of guiding the software developer as to which kind of changes is required when adapting a particular component. These agents may help users by creating a meta-data file, containing the details of structures, classes, their types and relationships. So that the developer can see what changes are required, and it may help him to make a decision whether to use or not to use a particular component.

Table 4.8 Suggestions and representative quotes

Cat-7		
Suggestions		
Sub Category ID	Sub Category Name	Representative quote
SC-7-1	Inter Language Reuse	“...for example if I want to extract python code, if I use python code for example I am a C# programmer that use dot net technology so how can I use it ... for example I use python for text processing, how can I use python code? How can it be portable or how can it be easily used in java or C?”
SC-7-2	Software Agents	“...for software development there is such type of agent, that they can easily see other things also, if they are using some other code, agent can suggest which kind of change is required and which variable constant you should change or which type of features/ classes”.

4.2.8 Other Considerations

A few of the ‘open codes’ cannot be associated with the categories due to the contextual differences. Therefore, these thoughts of the respondents are presented separately in this section.

Table 4.9 Other considerations and representative quotes

Cat-8	Other Considerations	
Sub Category ID	Sub Category Name	Representative quote
SC-8-1	Future of OSS and PL	“...coming time is of OSS because, open source will be more common, yes both of these fields are promising...”
SC-8-2	Importance of Measurement	“...measuring these factors is very useful to understand...”
SC-8-3	White Box Taking Advantage	“...it depends on the situation but in the setup of using open source, white box is taking advantage”.

4.2.8.1 SC-8-1 Future of OSS and PL

The respondents see a promising future of OSS, SPL and the development of SPL using OSS. Now a day’s time is the most important resource. OSS is enabling organizations to reuse software and save time. Secondly, OSS is providing great support in development, provided that a right choice is made. This support is in terms of finance, time and resources.

4.2.8.2 SC-8-2 Importance of Measurement

Measurement in software engineering is as important as in other fields. The measurement of factors affecting reusability is also important. The impact of measurements can be seen at different levels like a developer can assess the quality of

his work. The higher management can assess reusability and may use the assessment results to compare software for making decisions or to prepare project plans.

4.2.8.3 SC-8-3 White Box Taking Advantage

The decision regarding either black box or white box usage of OSS component depends on many factors. The organizations may think about their human resource capabilities before making a white box reuse decision. The reuse policies of organizations are also important while making such decisions. However, in any situation white box reuse is taking advantage of customizability and provides more freedom.

In previous section the findings of the qualitative study are presented. These interview data was analyzed using the content analysis approach. Findings are presented in seven categories and 39 sub categories. In next section the results obtained by the quantitative study making use of survey are presented.

4.3 Attribute Ranking Survey

This section contains the findings of the quantitative study (survey). The details of the survey including sampling technique, sample size, confidence level and confidence interval were presented in chapter 3. After executing the survey research method these results are gathered. The results of the survey are presented in the form of pie charts. The percentages of population selected a specific scales are presented in figures Figure 4.1 to Figure 4.9.

The pie chart of attribute flexibility shows a concentrated data at scale four (agree) and five (strongly agree). It shows a consensus that flexibility is one of the attribute of reusability.

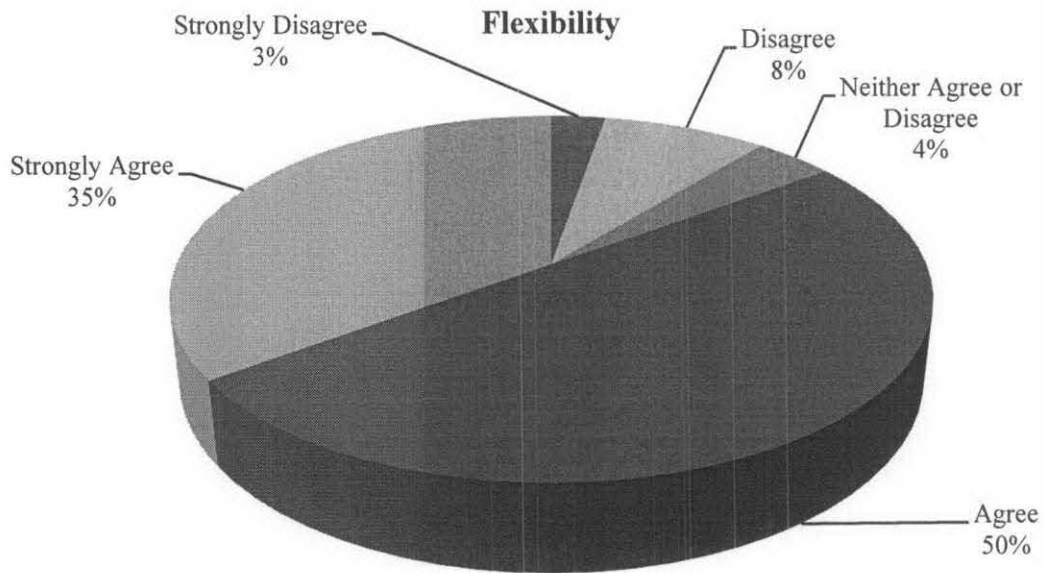


Figure 4.1 Frequency distribution of the scales assigned to flexibility

Table 4.10 Flexibility rankings

Flexibility	
Scale	Confidence Interval
Strongly Disagree	*0% - 7.69%
Disagree	3.42% - 13.24%
Neither Agree or Disagree	*0% - 8.95%
Agree	45.09% - 54.91%
Strongly Agree	29.94% - 39.76%

* The lower limit being below zero is rounded to zero, following the guidelines presented in [142].

The pie chart of scope coverage shows a less consensus of population that it is an attribute of reusability. It is visible by the percentage of population opted for the scale (disagree).

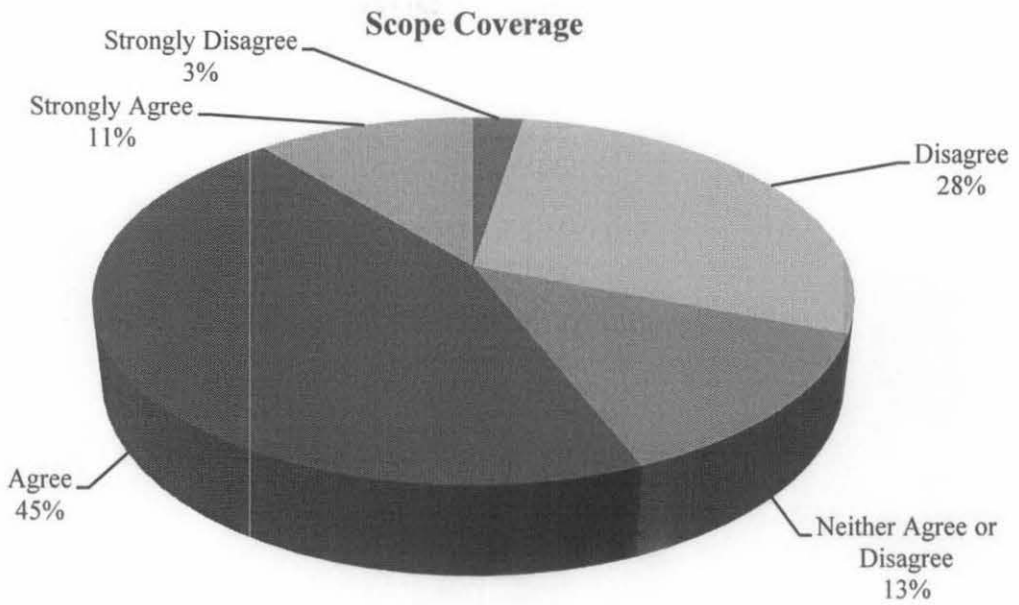


Figure 4.2 Frequency distribution of the scales assigned to scope coverage

Table 4.11 Scope Coverage rankings

Scope Coverage	
Scale	Confidence Interval
Strongly Disagree	*0% - 7.44%
Disagree	23.37% - 33.19%
Neither Agree or Disagree	8.22% - 18.04%
Agree	40.04% - 49.86%
Strongly Agree	6.20% - 16.02%

* The lower limit being below zero is rounded to zero, following the guidelines presented in [142].

The pie chart of attribute portability shows a concentrated data at scale four (agree) and five (strongly agree). It shows a consensus that portability is one of the attribute of reusability.

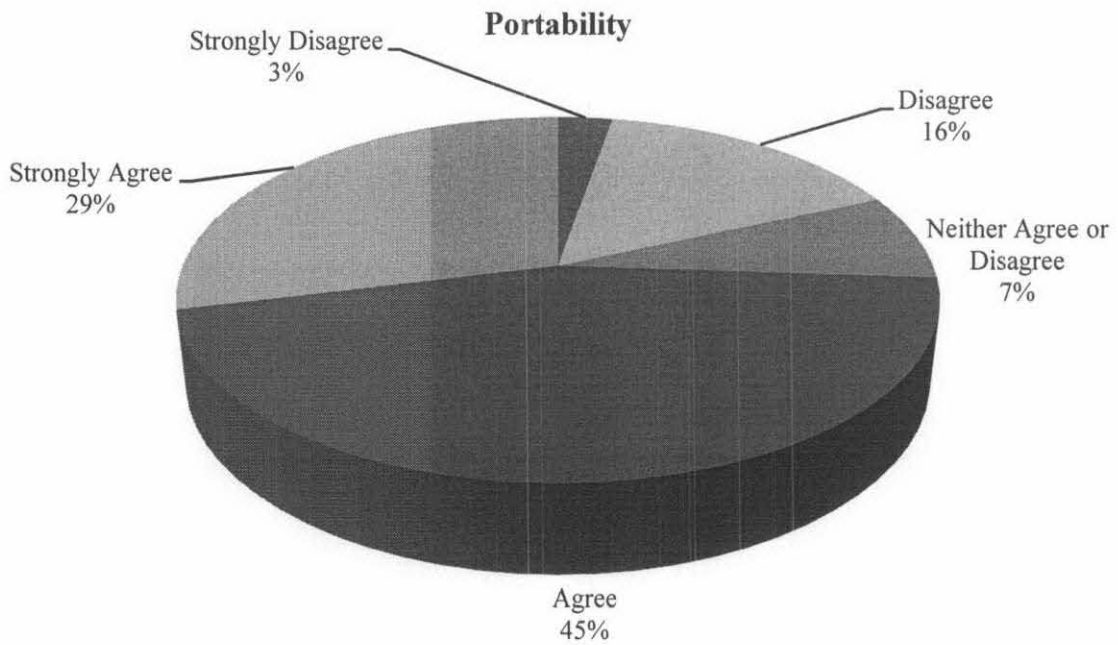


Figure 4.3 Frequency distribution of the scales assigned to portability

Table 4.12 Portability rankings

Portability	
Scale	Confidence Interval
Strongly Disagree	*0% - 7.69%
Disagree	10.75% - 20.57%
Neither Agree or Disagree	2.67% - 12.49%
Agree	40.29% - 50.11%
Strongly Agree	23.88% - 33.70%

* The lower limit being below zero is rounded to zero, following the guidelines presented in [142].

The pie chart of attribute maintainability shows a concentrated data at scale four (agree) and five (strongly agree). It shows a consensus that maintainability is one of the attribute of reusability.

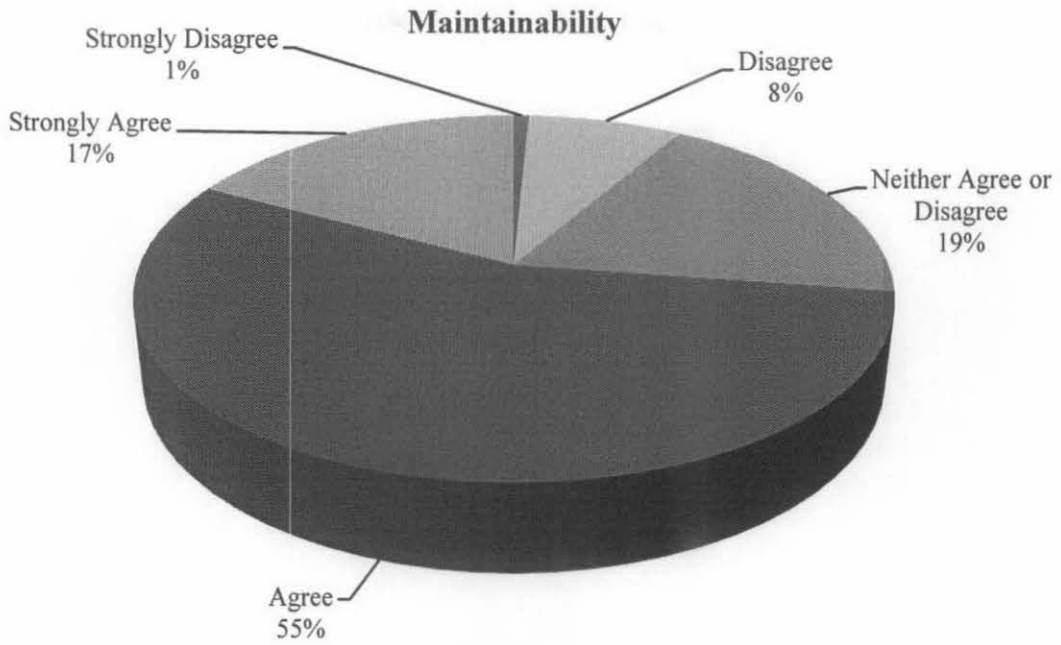


Figure 4.4 Frequency distribution of the scales assigned to maintainability

Table 4.13 Maintainability rankings

Maintainability	
Scale	Confidence Interval
Strongly Disagree	*0% - 5.67%
Disagree	2.92% - 12.74%
Neither Agree or Disagree	13.78% - 23.60%
Agree	50.39% - 60.21%
Strongly Agree	12.51% - 22.33%

* The lower limit being below zero is rounded to zero, following the guidelines presented in [142].

The pie chart of attribute variability shows that 30% of the population opted for neither agree nor disagree. One of the possible reasons for this indecisiveness of population is lack of knowledge about the variability.

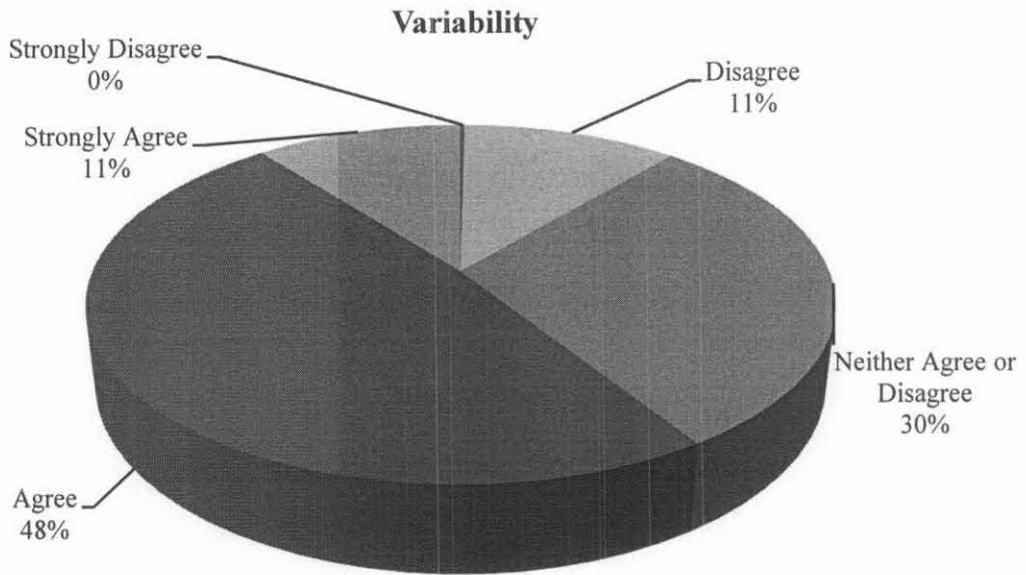


Figure 4.5 Frequency distribution of the scales assigned to variability

Table 4.14 Variability rankings

Variability	
Scale	Confidence Interval
Strongly Disagree	*0% - 5.16%
Disagree	6.20% - 16.02%
Neither Agree or Disagree	24.64% - 34.46%
Agree	43.57% - 53.39%
Strongly Agree	5.70% - 15.52%

* The lower limit being below zero is rounded to zero, following the guidelines presented in [142].

The pie chart of attribute understandability shows a concentrated data at scale four (agree) and five (strongly agree). It shows a consensus that understandability is one of the attribute of reusability.

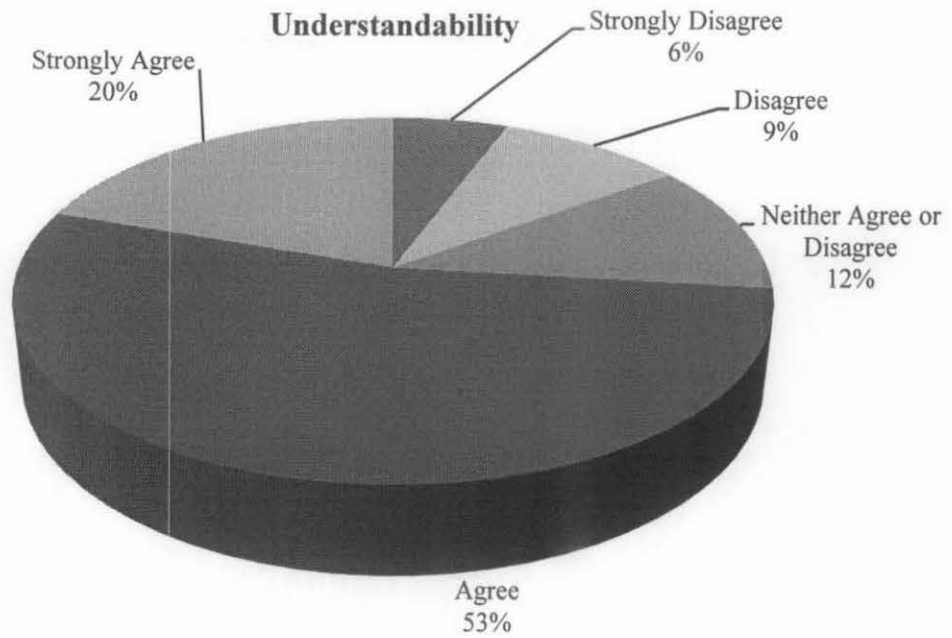


Figure 4.6 Frequency distribution of the scales assigned to understandability

Table 4.15 Understandability rankings

Understandability	
Scale	Confidence Interval
Strongly Disagree	*0% - 10.72%
Disagree	4.43% - %14.25
Neither Agree or Disagree	6.71% - %16.53
Agree	48.37% - %58.19
Strongly Agree	15.04% - 24.86%

* The lower limit being below zero is rounded to zero, following the guidelines presented in [142].

The pie chart of documentation shows that a large number of populations opted for scales; agree (31%) and strongly agree (31%) population that documentation is an attribute of reusability. On the other hand 24% of population opted for neither agree nor disagree and 14% of population is having a disagreement that documentation affects the reusability of software.

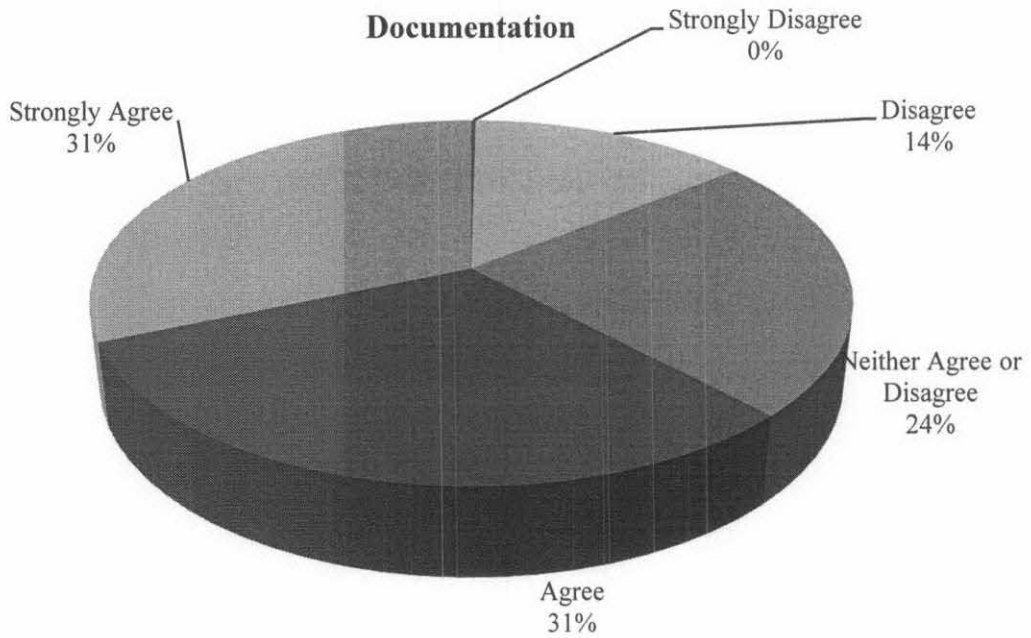


Figure 4.7 Frequency distribution of the scales assigned to documentation

Table 4.16 Documentation rankings

Documentation	
Scale	Confidence Interval
Strongly Disagree	*0% - 5.16%
Disagree	9.23% - 19.05%
Neither Agree or Disagree	18.57% - 28.39%
Agree	25.90% - 35.72%
Strongly Agree	26.40% - 36.22%

* The lower limit being below zero is rounded to zero, following the guidelines presented in [142].

The pie chart of attribute usage history shows a concentrated data at scale four (agree) and five (strongly agree). It shows a consensus that usage history is one of the attribute of reusability.

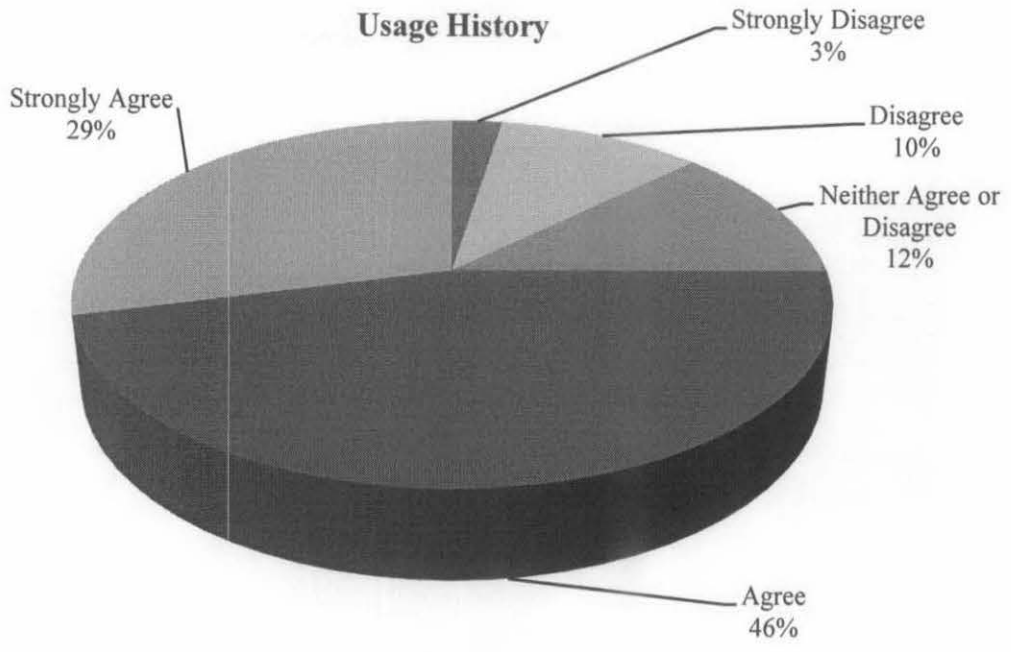


Figure 4.8 Frequency distribution of the scales assigned to usage history

Table 4.17 Usage History rankings

Usage History	
Scale	Confidence Interval
Strongly Disagree	*0% - 7.44%
Disagree	5.44% - 15.26%
Neither Agree or Disagree	7.21% - 17.03%
Agree	41.05% - 50.87%
Strongly Agree	24.13% - 33.95%

* The lower limit being below zero is rounded to zero, following the guidelines presented in [142].

The pie chart of reusability attribute stability shows concentrated data at scale four (agree) and five (strongly agree). It shows a consensus that stability is one of the attribute of reusability.

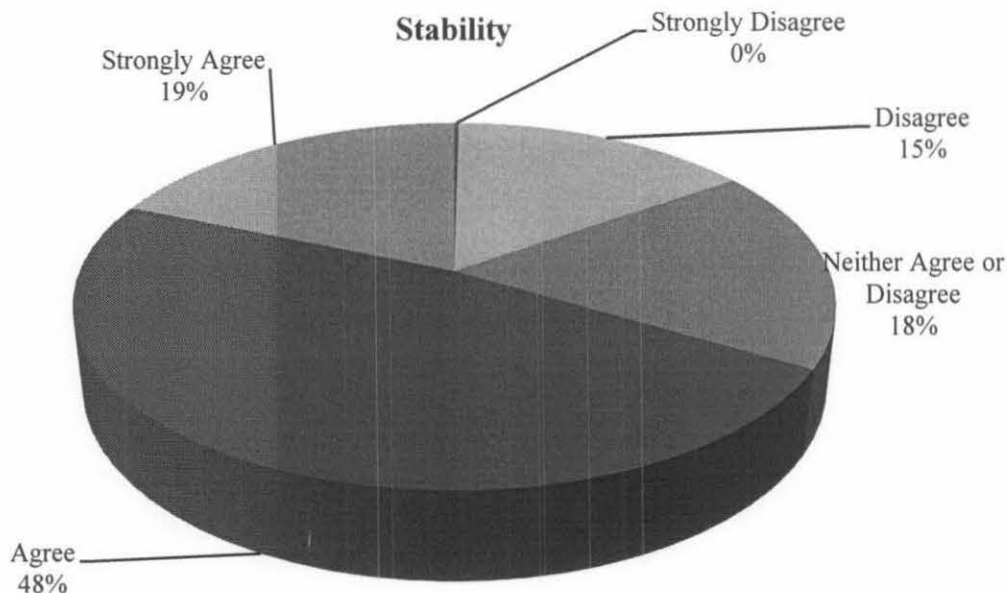


Figure 4.9 Frequency distribution of the scales assigned to stability

Table 4.18 Stability rankings

Stability	
Scale	Confidence Interval
Strongly Disagree	*0% - 5.16%
Disagree	10.24% - 20.06%
Neither Agree or Disagree	13.02% - 22.84%
Agree	43.07% - 52.89%
Strongly Agree	13.78% - 23.60%

* The lower limit being below zero is rounded to zero, following the guidelines presented in [142].

In the second section of this chapter variability and its implementation mechanisms are discussed.

4.3.1 Theoretical Analysis of Variability Implementation Mechanism

In this section, a classification of available variability mechanisms is presented. This classification is in terms of type, scope, and the artifact to be targeted. Examples to illustrate the points under discussion are given in the form of Java code.

It was stated earlier that variability management is one of the success factors in software product line development. It is a non-trivial activity due to multiple factors [43]. Variability management has many facets not only at architecture and coding level, but also in the development process, as different tools can be used at different phases.

In this study only those mechanisms for managing variability are discussed which are related to the implementation phase. In software engineering terminology, ‘implementation’ refers to the production of source code or translation of design into software components [3].

Our focus during this discussion is on the mainstream product line implementation technology, such as object oriented development, and specifically the implementations based on the Java language. Another point to consider at this stage is that variability management of requirements, design artifacts, and test cases is out of the scope of this study.

The term ‘variability realization technique’ refers to the mechanism which is used for implement at the variation point [43].

A variation point specifically identifies the part of a variable requirement that is subject to change. A variant is an instance of a variable requirement. A variant can be implemented in different ways as software entities and these entities may include components, classes, a set of classes or lines of code [43].

Variability can be introduced at different stages of software development such as during architectural design, during detailed design, during implementation, and when compiling or linking [43].

Different software entities are relevant at each of these levels. Since our work is concerned with implementation level variability, the software entities which we will focus on are individual classes and lines of code.

The following variability realization techniques are discussed based on [39] and [40].

4.3.1.1 Aggregation / Delegation

In object oriented programming, aggregation defines a “has-a” relationship or whole-part relationship between the objects. For example, A and B are two classes, an object of class A has an object of class B if B is part of A. This phenomenon is also referred to as containment - an object of class A contains an object of class B.

Variability can be handled by aggregation [39, 143]. The technique of aggregation enables an object to support a functionality which it cannot support normally. The object that sends a request to perform the function is delegating it to another object. The object which performs the function, at the request of a delegating object, is the delegated object.

Mandatory functionality can be handled without delegation; variant functionality is handled by delegation. Aggregation is also relevant to optional features. Typically, aggregation causes the variability to be resolved at compile time [39]. Table 4.11 lists the characteristics of this mechanism.

A class can facilitate the functionality which is not originally supported by it. An example is a bank account, with an option of providing a free locker with the account. The optional functionality can be added to this class by aggregation. An object of *Locker* class is added to the *Bank_acc* class. In this way aggregation makes it possible for class *Bank_acc* to provide the optional functionality of a locker.

For example,

```
public class Bank_acc {  
    private Locker lc;
```

Table 4.19 Characteristics of aggregation

Aggregation	
Type (functionality)	Positive, negative, optional
Works with	Attribute
Type (effect)	Attribute variability
Scope	Open
Feature	Optional
Binding Time	Run-time

4.3.1.2 Inheritance

Inheritance is a relationship between two classes, where one class is a parent class (super class) of the other class (sub class). In this scenario, the common functionality is handled by the super class and variable functionality is handled by the sub class. In class based inheritance, a sub class can introduce new attributes and operations. These newly defined attributes and operations can be overwrite or added to the existing ones. Table 4.12 summarizes the characteristics of this mechanism.

Following is an example. Both classes *Saving_acc* and *Fixdeposit_acc* have the same base class - *Bank_acc*.

```
public class Fixdeposit_acc extends Bank_acc{
// class definition
}
public class Saving_acc extends Bank_acc{
// class definition}
```

Table 4.20 Characteristics of inheritance

Inheritance	
Type (functionality)	Positive, negative, optional, alternative
Works with	Class, method, attribute
Type (effect)	Attribute, workflow variability
Scope	Open
Feature	Mutually inclusive
Binding Time	Compile-time

4.3.1.3 Parameterization

Parameterization is a mechanism that is used to handle variability at both the design level [143] and the implementation level [39-40]. This mechanism can be used with any of the object, method, class or package levels. The behavior of the parameterized entity, such as a class or method, can be manipulated by setting the values of parameters. Parameterization is a preplanned variation mechanism and sets constraints on the code that is implemented.

4.3.1.4 Generics

‘Generics’ refers to a mechanism to parameterize classes and functions. A collection class is one area where generics could be used. As an example, we could use it to implement a stack for containing elements of different data types. Table 4.13 summarizes the characteristics of this mechanism.

Table 4.21 Characteristics of parameterization/ generalization

Parameterization/Generalization	
Type (functionality)	Positive, negative, optional, alternative
Works with	Class, method, attribute
Type (effect)	Attribute variability
Scope	Selection
Feature	Optional , alternative
Binding Time	Compile-time

Following is an example:

```
public class Member <T>{
    private T id;

    public T getId() {
        return id;
    }
    public void setMyList (List <String> list) {
        this.MyList = list ;
    }
}
```

4.3.1.5 Overloading

Overloading refers to using a symbol or name to refer to multiple entities. In general, these entities can be data types, procedures, operators, and functions. Overloading can be used as a variability mechanism [39], as a ‘method’ of the base class can be overloaded and variability in the implementation can be introduced. Table 4.14 summarizes the characteristics of this mechanism.

Table 4.22 Characteristics of overloading

Overloading	
Type (functionality)	Positive, negative, alternative
Works with	Method
Type (effect)	Logic, workflow variability
Scope	Open
Feature	Optional
Binding Time	Compile-time

Following is an example. In both classes *Saving_acc* and *Fixdeposit_acc* have the same base class - *Bank_acc*. Both classes are overloading the *cal_interest* method of the base class.

```
public class Fixdeposit_acc extends Bank_acc{

public double cal_interest(// parametric change ){

// Logical change
}

public class Saving_acc extends Bank_acc{

public double cal_interest(//parametric change ){
// Logical change
}
```

4.3.1.6 Aspect-oriented Programming

Aspect-oriented techniques provide a solution to the crosscutting ‘concerns’ problem. A system can have different ‘concerns’; a ‘concern’ is an area of interest or property of a system that must be implemented in order for there to be a successful solution to

a problem [144]. Traditional software engineering is involved in the identification of concerns and these concerns are used to modularize a system.

The concept of encapsulation in software engineering leads us to attempt to implement each concern as a separate module. This phenomenon is also known as ‘the separation of concern’. It is not always possible to map each concern to a separate module.

[145] describes some of the points relating to concerns. The implementation of a single concern over more than one module is termed ‘crosscutting’; it creates the problem of concern/code tangling and scattering. Concern tangling is the situation when more than one concern is implemented in a single module and concern scattering is when one concern is implemented in multiple modules. Crosscutting concerns could be, for example, the system wide quality requirements. Kiczales divides concerns into two categories: Aspects and Components. If a concern can be cleanly encapsulated in a module it will be a component, and it will be an aspect if the concern crosscuts and cannot be cleanly implemented in a single module; these separate specifications of aspects and components are then combined to provide the solution by the process of weaving. Aspect-Oriented Software Engineering introduces a new mechanism to modularize a system and separate the crosscutting concerns. Table 4.15 summarizes the characteristics of this mechanism.

Following is an example. In it, *pointcut* is set to the *Open_acc* method whenever this method is called. Before this method executes, an advice of verification will run to verify the details.

```
pointcut PCAccount( ):call (void Open_acc( Name, Id) ;
    before ( ): PCAccount( ){
        // Verification of name and id }
```

Table 4.23 Characteristics of AOP

Aspect-oriented programming	
Type (functionality)	Positive, optional, alternative
Works with	Method
Type (effect)	Work flow, logic variability
Scope	Open
Feature	Optional
Binding Time*	Compile-time, Run-time or Load-time

* Aspects and their corresponding ‘advices’ are integrated into the system through the process of weaving. Weaving can be performed at compile time, run time or load time.

4.4 Summary

The findings of qualitative results of this study are categorized in seven categories. These categories include challenges in OSS, current reuse practices, using OSS in SPL, role of OSS in promoting reuse, factors affecting reusability, desirable characteristics of OSS and suggestions. Several dimensions of these categories are identified and listed in this chapter. The results of a survey are also presented to rank the reusability attributes.

A theoretical analysis of variability implementation mechanisms has been conducted. The variability types, scope, effect, related artifact and binding times are associated. This work complements the previous works in this area.

CHAPTER 5

CORRELATION STUDY OF FACTORS AFFECTING REUSABILITY

Realists do not fear the results of their study. (Fyodor Dostoyevsky, 1821-1881)

5.1 Overview

This chapter presents the proposed conceptual model for the inclusion of OSS in SPL. The proposed reusability attribute model is also presented in this chapter along with the description of attributes and metrics. The second section of this chapter includes the results of application of the proposed model at class level and package level. An evolutionary analysis of reusability is conducted for two open source software. The results are presented in his chapter.

5.2 Reusability Assessment Conceptual Model

A software organization developing software using OSS follows a process of following steps [35].

- Identification of potential OSS components
- Selection of OSS component
- Adaptation of OSS component (if necessary)

There could be several ways of identification of components, which include the use of search engines or OSS provider's web sites. After identifying the potential components, organizations decide which of the identified component to be used. At the selection stage the different criteria are used. These may include the legal aspects such as licence type or maintenance support for the component. In the context of this

thesis, a particular aspect i.e. reusability of component is assessed to facilitate the decision process.

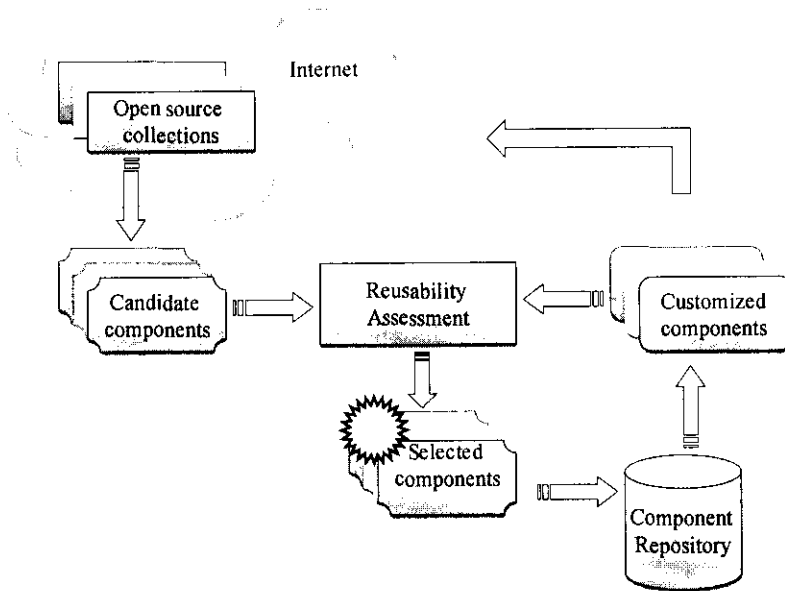


Figure 5.1 Conceptual model of reusability assessment

A conceptual model of our proposed approach is presented in Figure 5.1. It complements the approaches proposed for inclusion of components [64] and specifically OSS based software product lines [23]. The components are searched from the open source search engines. The candidate components are selected on the basis of requirements. These selected components are assessed to know their reusability using the model and metrics presented in section 5.3 and 5.5. The selected components are a part of component repository. These components are used ‘as it is’ or may be customized to serve the specific needs. These customized components are again assessed to know the reusability and if selected then again saved in the component repository. In some cases, customized components are contributed to the open source collection of components.

5.3 Proposed Class Level Reusability Attribute Model

The attributes of reusability are identified in chapter four. These identified attributes are used in the GQM model. The formation of this model helps to identify the suitable measures for these attributes. Although in the interview we have identified 9 factors, however, in the attribute model only six of the attributes are considered. Three of the

attributes, namely stability, documentation and usage history, are not included in the model due to their subjective nature, which does not match the objective measurement of source code.

5.3.1 Reusability GQM Model

The GQM model is presented in this section.

Table 5.1 GQM Model Class Reusability

Object of study: Class	
Purpose: Assessment	
Quality focus: Effort required reusing	
Viewpoint: Developer	
Environment: Development of software in a reuse intensive environment	
Goal: Assessment of object oriented systems to predict reusability from the viewpoint of a developer.	
Question	1. How easy is it to reuse the class?
Question	1.1. How much variability is there in the component?
Question	1.1.1. What is the average number of methods per class?
Metric	1.1.1.1. $\text{Number of methods} \div \text{Total number of classes}$
Question	1.1.2. What is the average number of children per class?
Metric	1.1.2.1. $\text{Number of children} \div \text{Total number of classes}$
Question	1.2. How easy is it to understand the class?
Question	1.2.1. What is the size of the class?
Metric	1.2.1.1. Number of methods (NOM)
Metric	1.2.1.2. Lines of code (LOC)
Question	1.2.2. How much coupling is there in the class?
Metric	1.2.2.1. Coupling between objects (CBO)
Question	1.2.3. How much cohesion is there in the class?
Metric	1.2.3.1. Lack of cohesion in methods (LCOM)
Question	1.2.4. How many comment lines are there in the class?
Metric	1.2.4.1. No. of comments
Question	1.3. How easy is it to maintain the system?
Metric	1.3.1. Maintainability Index (MI)
Metric	1.3.2. Cyclomatic Complexity (CC)

Question	1.4. How much flexibility is there in the class?
Question	1.4.1.How much coupling is there in the class?
Metric	1.4.1.1. CBO
Question	1.4.2.How much cohesion is there in the class?
Metric	1.4.2.1. LCOM
Question	1.5. How portable is the class?
Question	1.5.1.How independent is the class?
Metric	1.5.1.1. Depth of inheritance tree (DIT)
Question	1.6. How much of the scope is covered by the class?
Question	1.6.1.How many features are covered by the class?
Metric	1.6.1.1 NOM/Total number of methods in all classes

5.3.2 Attributes

The definitions of attributes from the literature are presented in this section.

5.3.2.1 *Understand-ability*

It is defined as “the ease with which a system can be comprehended at both the system-organizational and detailed statement levels” [3]. In [69, 82] understand-ability is considered as an attribute of reusability.

5.3.2.2 *Flexibility*

It is defined as “the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed” [3]. In [31, 61, 82] flexibility is considered as an attribute affecting the reusability of a component. In the context of an SPL, the flexibility characteristic is necessary for a core asset as it is intended to be reused in the development of other products.

5.3.2.3 *Portability*

It is defined as “the ease with which a system or component can be transferred from one hardware or software environment to another” [3]. The portability of a component depends on its independence, i.e. the ability of the component to perform its functionality without external support. In a scenario where an open source component is used in SPL development, the component should have the characteristic of portability. The component, being a core asset, may be used in the development of another product/family member within the product line/family.

5.3.2.4 *Maintainability*

In [3], maintainability is defined as “the ease with which a software system or component can be modified to change or add capabilities, correct faults or defects, improve performance or other attributes, or adapt to a changed environment”. Two metrics, CC and MI, are used to measure maintainability.

5.3.2.5 *Scope Coverage*

It is the attribute that measures the number of features provided by the component against the total number of features in the SPL scope.

5.3.2.6 *Variability*

Variability management (VM) is an important activity in a reuse intense software development environment. It is a non trivial activity and has many facets as not only both architecture and coding are variable, but so is the development process, as different tools can be used.

VM in a product line context refers to the identification, modeling, resolution, storage and instantiation of variability [146]. VM is the distinguishing feature of software product line development [147]. Efficient VM is one of the key success factors in a reuse intense software development environment. In product line

development, all the artifacts developed are considered core assets. Variability is considered as a characteristic of a reusable core asset [148].

In [41] variability types are defined. The types include attribute, logic, workflow, persistency, interface, and combined. Regarding attribute variability, an attribute is supposed to be a placeholder for values to be stored – such as constants, variables or data structures. Three cases are presented. First is when the number of attributes varies between members of a product family. Second is the variation in the data type of the values assigned to the attributes, and the third case represents the variation of the value assigned to the attribute that is persistent.

Logic variability is the variation of the algorithm or logical procedure. There are several cases of logic variability, each case dependent upon the entity that varies, be it the procedural flow, the post condition, the exception handling, or the side effects between family members. Workflow variability is variation in the order, type and number of methods invoked by family members when carrying out a common task.

Persistency variability refers to the variation in the values of attributes that are stored in secondary storage. Interface variability is the variation in the signature of the interface method, i.e. to implement the same requirement; different members of a family implement their methods in different ways. These are distinguished by the name, return type, and order and type of parameters. Combined variability is where a variation point has more than one variability type.

In [42] variability is categorized as follows: positive - when some functionality is added; negative - when there is a withdrawal of functionality; optional - when code is added; alternative - when code is removed; function - when functionality is changed; platform/environment - when the platform or environment is changed.

Our focus during this discussion is on mainstream product line implementation technology, such as object oriented development, and specifically implementations based on the Java language. Another point to consider at this stage is that VM of requirements; design artifacts, and test cases are out of the scope of this study.

The term ‘variability realization technique’ refers to the mechanism which is used to implement at the variation point [43]. A variation point specifically identifies the part of a variable requirement that is subject to change. A variant is an instance of a variable requirement. A variant can be implemented in different ways, affecting different software entities and these entities may include components, classes, a set of classes or lines of code [43]. Variability can be introduced at different stages of software development such as during architectural design, during detailed design, during implementation, and when compiling or linking [43].

5.3.2.7 State of the Art (Variability Metrics)

A systematic review [66] presents the state of the art in the area of software measurement. The results of the review show that there is no measure available for variation. This shortage of metrics to measure variability, specifically at the implementation level, is also recognized in another study [149]. In our work, we acknowledge this gap and propose metrics to assess the variability of software components.

5.3.2.8 Proposed Variability Metrics

In [41] types of variability are defined on the basis of component reference models, namely CORBA and EJB. The building blocks of a component are defined as classes, work flow among classes, and interfaces.

We can consider the entities involved in object oriented programming. In Java these comprise the classes, interfaces, packages and Java beans. From the viewpoint of reuse, using Java beans is considered to be a black box approach. However, our work is concerned with a white box approach to the reuse of components.

An object oriented class consists of attributes, which hold data, and methods that exhibit behavior. An abstract class is used as super-class for a class hierarchy; it cannot be instantiated.

In [41] the following variability types are listed: attribute, logic and workflow. Another view of variability types is presented in [43], where variability is categorized as positive, negative, optional, function and platform/environment. All of the variability types given in [41] can be mapped to the variability types given in [43], for instance, the 'attribute' variability type is a 'positive' variable type when a new attribute is added.

Attribute variability can be implemented using any of the following techniques: inheritance; aggregation; parameterization /generics; overloading. Further cases of attribute variability are defined in [41]. One of these is the variation in the number of attributes. This type of variability is supported by inheritance and aggregation. Another type of attribute variability is variation in the data types of the attributes; this variability is supported by parameterization/generics.

As described earlier, inheritance is one of the mechanisms to handle attribute variability. In our work we propose variability metrics on the basis of the theory and mechanism of inheritance.

With inheritance the subclass inherits all the methods and attributes of the super-class. The subclass can define its own attributes in addition to those it inherits from the super-class, which causes the attribute variability. The other mechanism associated with inheritance is overloading which causes logic and work flow variability. So, a class that is higher in the hierarchy, and therefore having more accessible attributes and methods, has more variability.

The Number Of Children (NOC) metric is defined in [150] and the Number Of local Method metric is defined in [151]. The proposed metrics make use of these established metrics and associate the concept of variability with them. The relationship of variability and metrics is presented in Figure 5.2.

Following is the definition of metrics based on GQM approach.

Table 5.2 GQM for class variability

Object of study: Class	
Purpose: Assessment	
Quality focus: Effort required reusing	
Viewpoint: Developer	
Environment: Development of software in a reuse intensive environment	
Goal: Assessment of object oriented systems to predict variability from the view point of the developer.	
Question	1. How much variability is there in the component?
Question	1.1. What is the ratio of method per class?
Metric	1.1.1. Number of methods ÷ Total number of methods in component
Question	1.2. What is the ratio of number of child per class?
Metric	1.2.1 Number of child ÷ Total number of classes

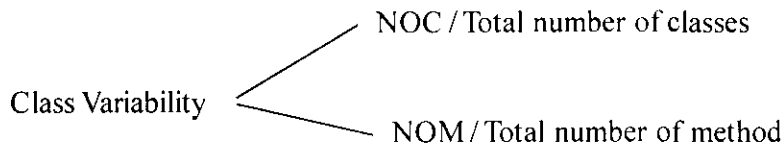


Figure 5.2 Relationship of variability with metrics

At package level the use of Abstractness metric (A) is proposed to assess the variability of a package. Abstractness of a package is the ratio of abstract classes and interfaces to the total number of classes [152]. Its domain is a set of integers, value ranges from 0 to 1 i.e. [0, 1), where zero refers to a concrete package i.e. absence of abstract class or interface and 1 refers to an abstract package i.e. all of its classes are abstract.

$$\text{Abstractness (A)} = \text{Number of abstract classes + interfaces} / \text{Total number of classes}$$

These two constructs of object oriented paradigm, i.e. abstract classes and interfaces, support two variability mechanisms which are ‘inheritance’ and ‘overloading’. These mechanisms facilitate the implementation of positive, negative,

optional and alternative type of variability. The variability can be introduced at class, method and attribute level by using these mechanisms. These mechanisms can support open scope variability at compile time.

Abstractness metric computes the ratio of abstract classes and interfaces to the total number of classes. It can be seen as an indirect measure of variability. So, the variability capability of a package or openness of a package towards variability can be assessed using the abstractness metric.

5.3.3 Sub Attributes

The definitions of sub attributes are presented in this section.

5.3.3.1 Independence

The term 'independence' is introduced to reflect the property of the system concerning the ability of a class to perform its responsibilities on its own. Independence is measured by DIT. The classes lower in the hierarchy are inherited by other classes; these classes depend on their ancestors to perform their functionalities.

5.3.3.2 Size Metrics

In [84] the aspect of the software dealing with its physical size is named the 'length' of the software. The metric used for size is lines of code (LOC). It counts the lines of source code. The second metric used to measure size is number of methods (NOM).

5.3.3.3 Coupling and Cohesion Metrics

Coupling and cohesion are two key concepts in object oriented software engineering. Both of these are related to interaction between the entities. The higher the level of interaction, the higher is the level of dependency. The lower the level of interaction, the higher is the level of cohesion. Cohesion refers to the extent to which an entity can

perform its responsibilities on its own. The metric used for coupling is CBO and the one used for cohesion is LCOM.

5.3.4 Class Level Metrics

The metrics used to assess the attributes of reusability are defined in this section.

5.3.4.1 Coupling between Objects (CBO)

These metrics count the number of classes to which a class is coupled [150]. Coupling prevents a class from performing its responsibility on its own, i.e. the class having a higher CBO value is more dependent on other classes. This dependence of a class on other classes decreases its understand-ability and flexibility. It is measured on an absolute scale; its domain is the set of integers $[0, \infty)$.

5.3.4.2 Lack of Cohesion Metric (LCOM)

Cohesiveness is the property that enhances encapsulation. LCOM metrics indicate the lack of cohesion; lack of cohesion decreases understandability and flexibility [150]. It is measured on an absolute scale; its domain is the set of integers $[0, \infty)$.

5.3.4.3 Depth of Inheritance Tree (DIT)

This is a measure that indicates the depth of a class within a hierarchy [150]. The class lower in the hierarchy depends on all the ancestor classes; it hinders its ability to be independent. A higher value of DIT reduces the independence which results in decreased portability. It is measured on an absolute scale; its domain is the set of integers $[0, \infty)$.

5.3.4.4 *Lines of Code (LOC)*

This is a measure of the lines of source code [84]. It is a size indicator of the entity. The size of the software affects its understandability. It is measured on an absolute scale; its domain is the set of integers $[0, \infty)$.

5.3.4.5 *Number of Methods (NOM)*

This metric is introduced in [151]. It measures the number of methods declared within the class. It is an indicator of the size of a class. It is measured on an absolute scale; its domain is the set of integers $[0, \infty)$.

5.3.4.6 *Number of Child (NOC)*

NOC is the measure that counts the children of a class [150]. NOC itself shows the reuse of a class. A large number of children mean that the functionality of the class is reused through inheritance. It is measured on an absolute scale; its domain is the set of integers $[0, \infty)$.

5.3.4.7 *Maintainability Index (MI)*

Maintainability index (MI) [153] value is the representative of the relative maintainability of the code [154]. Maintainability index is calculated by making use of lines of code, Mc Cabe complexity metric and Halstead measures. The maintainability index is calculated by the following formula:

$$MI = 171 - 5.2 \ln(aV) - 0.23 aV(g') - 16.2 \ln(aLOC) + 50 \sin[(2.4 * \text{perCM})^{1/2}]$$

Where

aV = average Halstead volume per module

$aV(g')$ = average extended cyclomatic complexity per module

$aLOC$ = average count of lines of code per module

perCM = average percent of lines of comments per module

5.3.4.8 Cyclomatic Complexity

Cyclomatic complexity metric is the measure of control structure complexity [155]. It counts the linear independent paths i.e. minimum number of paths during the execution. It is measured on an absolute scale and its domain is integers $[1, \infty)$.

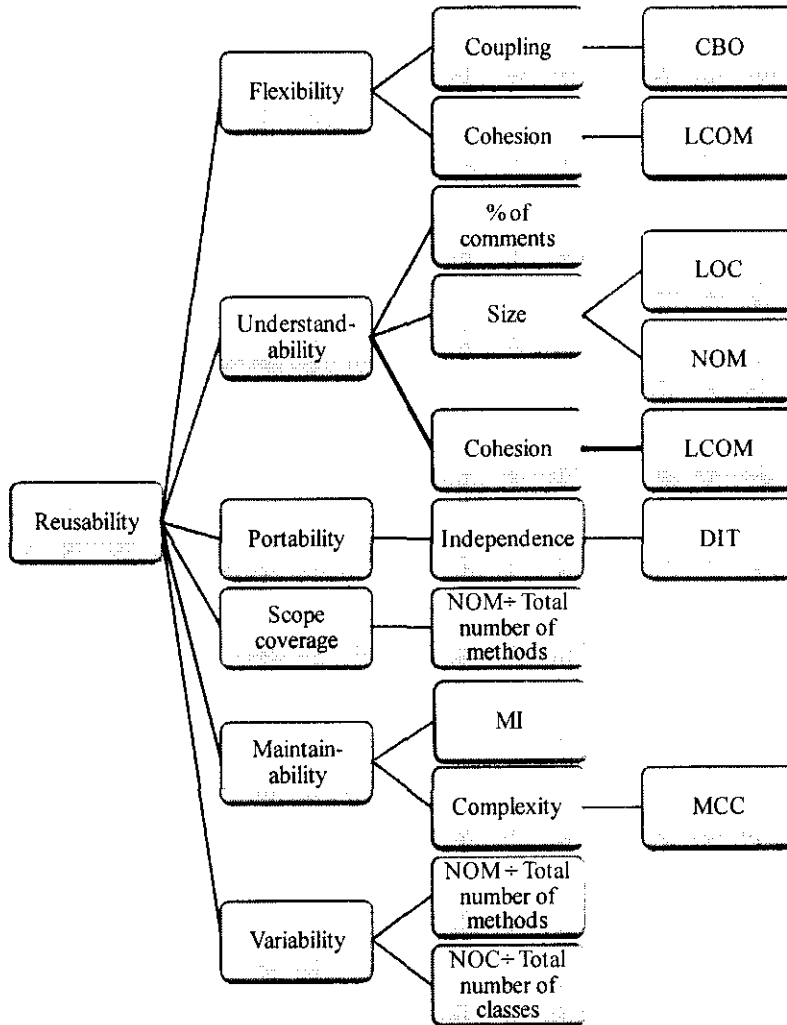


Figure 5.3 Reusability attribute model (class level)

5.4 Metrics Threshold Values and Equations

The threshold or reference values for metrics are required to understand a metric's value. The identification of threshold values for software metrics is an ongoing research area. Several efforts have been made to identify threshold values for metrics such as [156] and [157-158].

In [157] reference values for LCOM, DIT and NOM are identified. The threshold values for CBO, RFC and WMC are identified in [159]. In [160] statistical based thresholds for three metrics (LOC, NOM, CYCLO) are presented. The threshold value for MI is identified in a Hewlett-Packard study [161]. The value of cyclomatic complexity is categorized by the Software Engineering Institute (SEI). The values of metrics are adjusted using the threshold values.

Following are the equations used to calculate reusability and its attribute values.

$$\text{Reusability of Class} = 0.16 \times \text{Flexibility} + 0.16 \times \text{Understandability} + 0.16 \times \text{Portability} + 0.16 \times \text{Scope coverage} + 0.16 \times \text{Maintainability} + 0.16 \times \text{Variability} \quad (1)$$

$$\text{Flexibility} = 1 - [(0.5 \times \text{Coupling}) + (0.5 \times \text{Cohesion})] \quad (2)$$

$$\text{Coupling} = \text{adjusted CBO}, \text{ Cohesion} = \text{adjusted LCOM}$$

$$\text{Understandability} = 1 - [(0.25 \times \text{Coupling}) + (0.25 \times \text{Cohesion}) + (0.25 \times \text{Comments}) + (0.25 \times \text{Size})] \quad (3)$$

$$\text{Size} = (0.5 \times \text{adjusted LOC}) + (0.5 \times \text{adjusted NOM})$$

$$\text{Portability} = \text{Independence} = 1 - \text{adjusted DIT} \quad (4)$$

$$\text{Scope coverage} = \text{NOM} \div \text{Total number of methods in all classes} \quad (5)$$

$$\text{Maintainability} = (0.5 \times \text{adjusted MCC}) + (0.5 \times \text{adjusted MI}) \quad (6)$$

$$\text{Variability} = 0.5 \times (\text{NOC} \div \text{Total number of classes}) + 0.5 \times (\text{NOM} \div \text{Total number of methods in all classes}) \quad (7)$$

As a starting point, equal weights/coefficients are assigned to each of the attributes and sub attribute. Equal weights are also used in [67] and it is stated that the linear combination of equal weights works well in most cases.

In equation 1, the LCOM and CBO metrics are used to assess the flexibility. Both the CBO and LCOM have a negative impact on flexibility. So, the adjusted values of

these metrics are subtracted by 1. Same is the case in equation 2, where all the metrics have a negative relationship with understandability. In equation 4, DIT is subtracted by 1 to cater for its negative impact on the independence of class. In equation 6, value of complexity has a negative impact on maintainability. The value of complexity is adjusted in the code (provided in appendix B).

Table 5.3 Reusability attributes sub-attributes and metrics

Attribute	Sub-attribute	Metrics
Flexibility	Coupling, Cohesion	CBO, LCOM
Understandability	Coupling, Cohesion, Size	CBO, LCOM, %comments, LOC, NOM
Portability	Independence	DIT
Scope Coverage		NOM ÷ Total number of methods
Maintainability	Complexity	CC, MI
Variability		NOC ÷ Total number of classes, NOM ÷ Total number of methods

5.5 Proposed Package Level Reusability Attribute Model

The attributes of reusability are identified in chapter four. These identified attributes are used in the GQM model. The formation of this model helps to identify the suitable measures for these attributes. Although in the interview we have identified 9 factors, but in the attribute model only five of the attributes are considered.

Three of the attributes, namely stability, documentation and usage history, are not included in the model due to their subjective nature, which does not match the objective measurement of source code. The attribute scope coverage is also not considered at package level because the scope coverage of a package can be measured against the application in which this package is going to be used. The assessment in

this thesis is limited to the information which can be collected from the code of the package.

$$\text{Reusability of Package} = 0.2 X \text{ Understandability} + 0.2 X \text{ Portability} + 0.2 X \text{ Maintainability} + 0.2 X \text{ Variability} + 0.2 X \text{ Flexibility} \quad (8)$$

$$\text{Understandability} = 1 - [(0.5 X \text{ Lack of Comments}) + (0.5 X \text{ Size})] \quad (9)$$

$$\text{Size} = (0.33 X \text{ adjusted LOC}) + (0.33 X \text{ adjusted NOM}) + (0.33 X \text{ adjusted Number of classes}) \quad (10)$$

$$\text{Portability} = \text{Independence} = 1 - \text{adjusted Fan out} \quad (11)$$

$$\text{Maintainability} = (0.5 X \text{ adjusted CC}) + (0.5 \text{ adjusted MI}) \quad (12)$$

$$\text{Variability} = \text{adjusted Abstractness} \quad (13)$$

$$\text{Flexibility} = 1 - \text{adjusted Instability} \quad (14)$$

As a starting point, equal weights/coefficients are assigned to each of the attributes and sub attribute. Equal weights are also used in [67] and it is stated that the linear combination of equal weights works well in most cases.

In equation 9, the lack of comments and size are subtracted from understandability due to their negative impact on it. In equation 11, fan-out metrics shows a negative impact on the independence of package. In equation 12, value of complexity has a negative impact on maintainability and MI has a positive impact on maintainability. So, the negative impact of complexity is catered for in the code. In equation 14, lack of instability represents the flexibility of package.

Table 5.4 Package level attributes, sub attributes and metrics

Attribute	Sub-attribute	Metrics
Flexibility	Instability	I
Understand ability	Comments, Size	Number of classes, %comments, LOC, NOM
Portability	Independence	Fan-out
Maintainability	Complexity	CC, MI
Variability	Abstractness	A

An attribute model of reusability is defined by making use of GQM approach.

Table 5.5 GQM Model Package Reusability

Object of study: Package	
Purpose: Assessment	
Quality focus: Effort required reusing	
Viewpoint: Developer	
Environment: Development of software in a reuse intensive environment	
Goal: Assessment of Object oriented systems to assess reusability from the view point of developer.	
Question	1. How easy is it to reuse the package?
Question	1.1. How much variability is there in the package?
Question	1.1.1.What is the value of abstractness?
Metric	1.1.1.1. Abstraction
Question	1.2. How easy is it to understand the package?
Question	1.2.1.What is the size of the package?
Metric	1.2.1.1. Total number of classes
Metric	1.2.1.2. Lines of code (LOC)
Metric	1.2.1.3. No. of methods
Question	1.2.2.How many comment lines are there in the package?
Metric	1.2.2.1. No. of comments
Question	1.3. How easy is it to maintain the package?
Metric	1.3.1.What is the value of Maintainability Index (MI)?
Metric	1.3.1.1. Value of MI
Question	1.3.2.What is the value of Cyclomatic Complexity (CC)?
Metric	1.3.2.1. Value of CC
Question	1.4. How much flexibility is there in the package?
Question	1.4.1.How much resilience is there to change?
Metric	1.4.1.1. Instability of package
Question	1.5. How portable is the package?
Question	1.5.1.How independent is the package?
Metric	1.5.1.1. Fan-out

5.5.1 Package Level Metrics

The following package level metrics are employed in this study. The package level metrics differs from the class level metrics due to the difference in the nature of these artifacts. Although in the interview we have identified 9 factors, however, in the attribute model only five of the attributes are considered. Three of the attributes, namely stability, documentation and usage history, are not included in the model due to their subjective nature, which does not match the objective measurement of source code. The attribute scope-coverage is not included due to the limited information. In case, where the total number of features provided by the component is known, scope-coverage of a specific package can be calculated following the class level as example.

5.5.1.1 *Number of Classes / Interfaces*

It is the measure of total number of classes in a package. The size of package effects the understand ability. Number of classes is measured on an absolute scale; its domain is the set of integers $[0, \infty)$.

5.5.1.2 *Abstractness (A)*

Abstractness of a package is the ratio of abstract classes and interfaces to the total number of classes [152]. Its domain is a set of integers, value ranges from 0 to 1 i.e. $[0, 1)$, where zero refers to a concrete package i.e. absence of abstract class or interface and 1 refers to an abstract package i.e. all of its classes are abstract.

Abstractness (A) = Number of abstract classes + interfaces / Total number of classes

5.5.1.3 *Fan-out / Efferent Coupling*

The metric fan-out measures the total number of external classes coupled to classes of a package. It counts the number of classes outside the package referenced by the class of a given package [155]. Fan-out is equivalent to efferent coupling. Each class is counted for one time only. The value of metric is zero, if package is not using any

class of an external package. It is measured on an absolute scale; its domain is the set of integers $[0, \infty)$. Increased value of fan-out represents a high dependability of the package on other packages.

5.5.1.4 Fan-in / Afferent Coupling

The metric fan-in measures the total number of external classes coupled to classes of a package. It counts the number of references made towards the class of a given package [155]. Each class is counted for one time only. The value of metric is zero, if there is no external package which uses the classes of this package. This metric is equivalent to afferent coupling. It is measured on an absolute scale; its domain is the set of integers $[0, \infty)$. Increased value of fan-in represents a high dependability of other packages on the given package.

5.5.1.5 Instability (I)

Instability of a package is counted by counting the number of dependencies which enter or leave a package [152, 162]. Instability of a package is the ratio of efferent coupling (Fan-out) to the total coupling (Fan-out + Fan-in). It is represented by 'I' and its domain is the set of integers $[0, 1)$.

$$\text{Instability (I)} = Ce / Ca + Ce$$

Instability metric is also an indicator of resilience to change. The value of $I = 0$, represents a stable package i.e. a package that is less affected by change. The value of $I = 1$, represents an unstable package i.e. a package that is more affected by change.

5.5.1.6 Number of Method (NOM)

This metric is introduced in [151]. It measures the number of methods declared within the class. It is an indicator of the size of a class. It is measured on an absolute scale; its domain is the set of integers $[0, \infty)$.

5.5.1.7 Lines of Code (LOC)

This is a measure of the lines of source code [84]. It is a size indicator of the entity. The size of the software affects its understandability. It is measured on an absolute scale; its domain is the set of integers $[0, \infty)$.

5.5.1.8 Lines of Comments

It is the measure of total number of comment lines in the package, measured on an absolute scale and its domain is set of integers $[0, \infty)$. The comments have a positive effect on the understandability of a code asset.

5.5.1.9 Cyclomatic Complexity

Cyclomatic complexity metric is the measure of control structure complexity [155]. It counts the linear independent paths i.e. minimum number of paths during the execution. It is measured on an absolute scale and its domain is set of integers $[1, \infty)$.

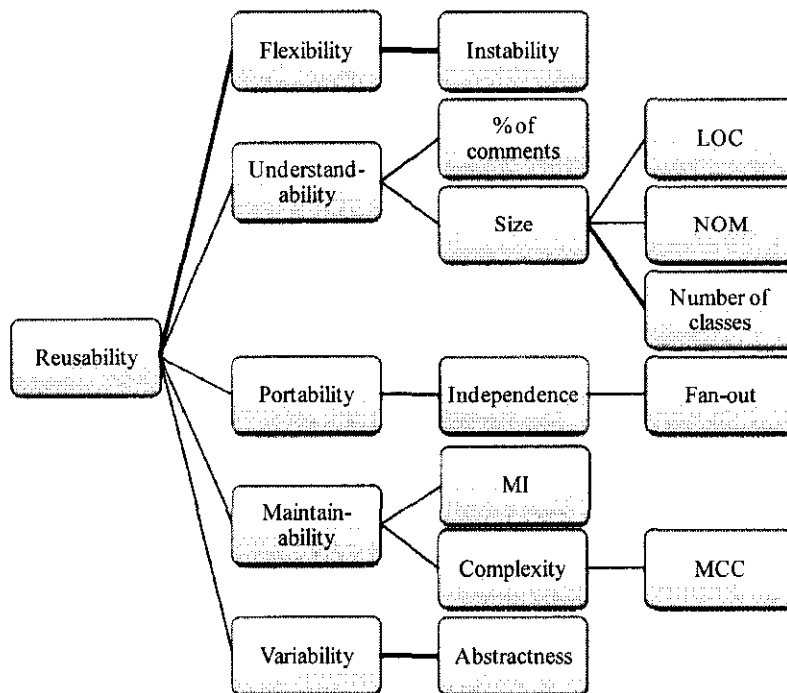


Figure 5.4 Reusability attribute model (package level)

5.5.1.10 Maintainability Index (MI)

Maintainability index (MI) [153] value is the representative of the relative maintainability of the code [154]. Maintainability index is calculated by making use of lines of code, Mc Cabe complexity metric and Halstead measures. The maintainability index is calculated by the following formula:

$$MI = 171 - 5.2 \ln(aV) - 0.23 aV(g') - 16.2 \ln(aLOC) + 50 \sin[(2.4 * \text{perCM})^{1/2}]$$

Where

aV = average Halstead volume per module

$aV(g')$ = average extended cyclomatic complexity per module

$aLOC$ = average count of lines of code per module

perCM = average percent of lines of comments per module

5.6 Reusability Assessment at Class Level

In this section the results of experiment 1 are presented. This experiment is intended to test the hypotheses formulated as a result of the interview and survey. In this experiment the hypotheses related to the class level reusability attribute model are tested. The values of reusability are calculated using the equations stated earlier in this chapter. Pearson's correlation analysis is conducted using the statistical software. The results are presented in the form of scatter plots along with the correlation values between (i) attributes and metrics and (ii) reusability and its attributes.

5.6.1 Metrics and Attributes Analysis

The relationships of sub-attribute to the attributes are tested in the form of hypotheses in this section.

5.6.1.1 MI, CC, Maintainability

The following hypotheses about MI and maintainability are tested:

H_{0_1} = MI of software has no effect on its maintainability

H_{1_1} = MI of software has an effect on its maintainability

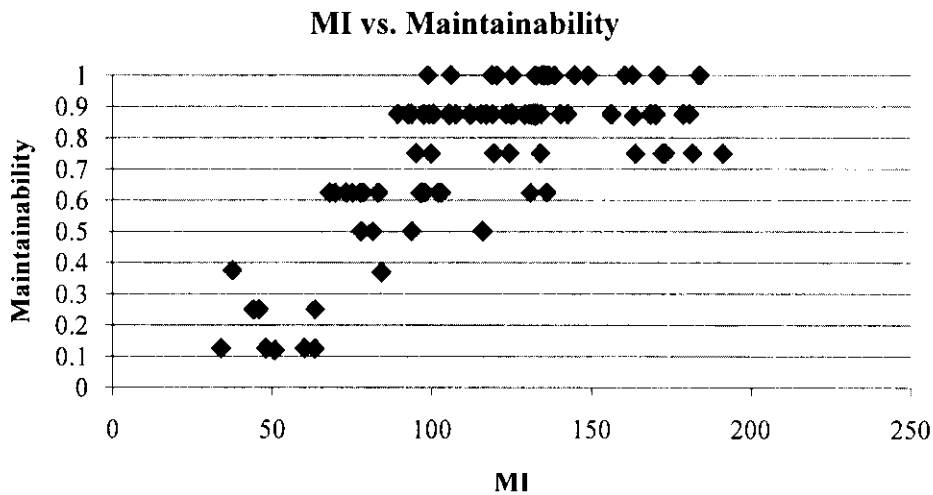


Figure 5.5 Scatter plot of MI vs. Maintainability

The correlation between MI and maintainability is $r(103) = 0.716, p = 0$. It shows a strong positive correlation between MI and maintainability. So, the null hypothesis is rejected and it can be concluded that MI is positively correlated to maintainability. An increase in the value of MI increases maintainability.

The following hypotheses about complexity and maintainability are tested:

H_{0_2} = CC of software has no effect on its maintainability

H_{1_2} = CC of software has an effect on its maintainability

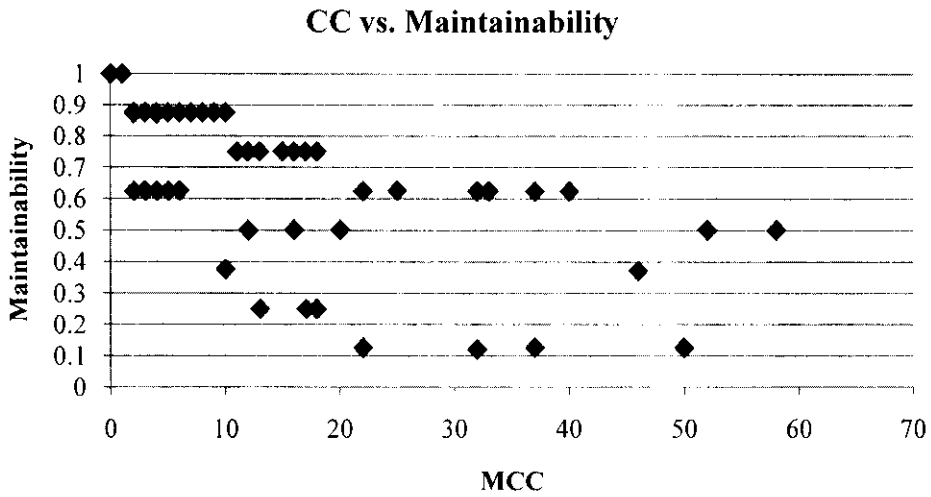


Figure 5.6 Scatter plot of CC vs. Maintainability

The correlation between CC and maintainability is $r(103) = -0.664, p = 0$. It shows a strong negative correlation between CC and maintainability. So, the null hypothesis is rejected and it can be concluded that CC is negatively correlated to maintainability. An increase in the value of CC decreases maintainability.

Table 5.6 Pearson's correlation values MI, CC and maintainability

Pearson's Correlations			
		MI	CC
Maintainability	Pearson Correlation	.716**	-.664**
	Sig. (2-tailed)	.000	.000
	N	103	103
**. Correlation is significant at the 0.01 level (2-tailed).			

5.6.1.2 LCOM, CBO, Flexibility

The following hypotheses about CBO and flexibility are tested:

H_{03} = CBO of software has no effect on its flexibility

H1₃= CBO of software has an effect on its flexibility

The correlation between CBO and flexibility is $r(103) = -0.751, p = 0$. It shows a strong negative correlation between CBO and flexibility. So, the null hypothesis is rejected and it can be concluded that CBO is negatively correlated to flexibility. An increase in the value of CBO decreases flexibility.

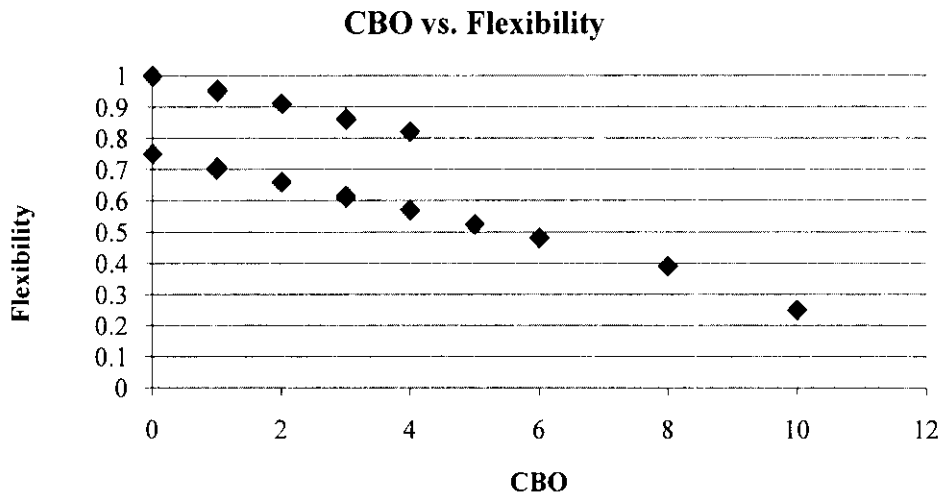


Figure 5.7 Scatter plot of CBO vs. Flexibility

The following hypotheses about LCOM and flexibility are tested:

H0₄ = LCOM of software has no effect on its flexibility

H1₄ = LCOM of software has an effect on its flexibility

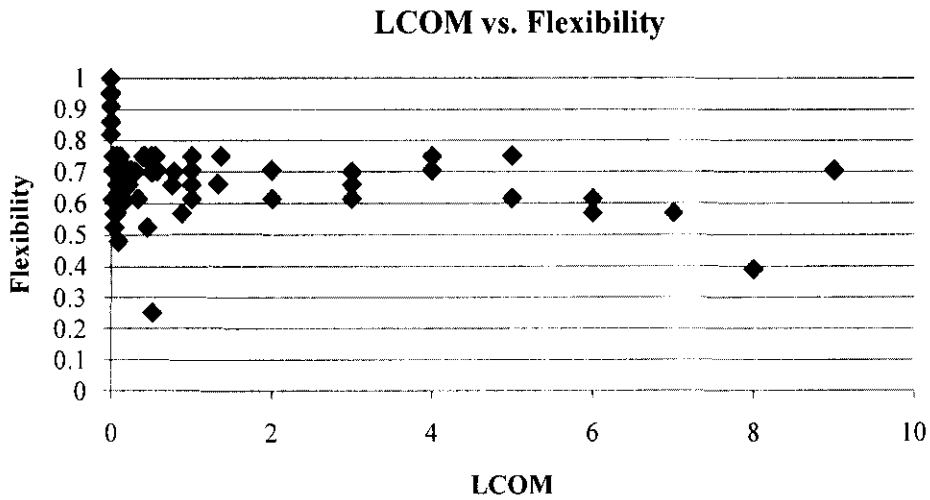


Figure 5.8 Scatter plot of LCOM vs. Flexibility

The correlation between LCOM and flexibility is $r(103) = -0.357, p = 0$. It shows a negative correlation between LCOM and flexibility. So, the null hypothesis is rejected and it can be concluded that LCOM is negatively correlated to flexibility. An increase in the value of LCOM decreases flexibility.

5.6.1.3 CBO, LCOM, Understandability

The following hypotheses about CBO and understandability are tested:

H_0 = CBO of software has no effect on its understandability

H_1 = CBO of software has an effect on its understandability

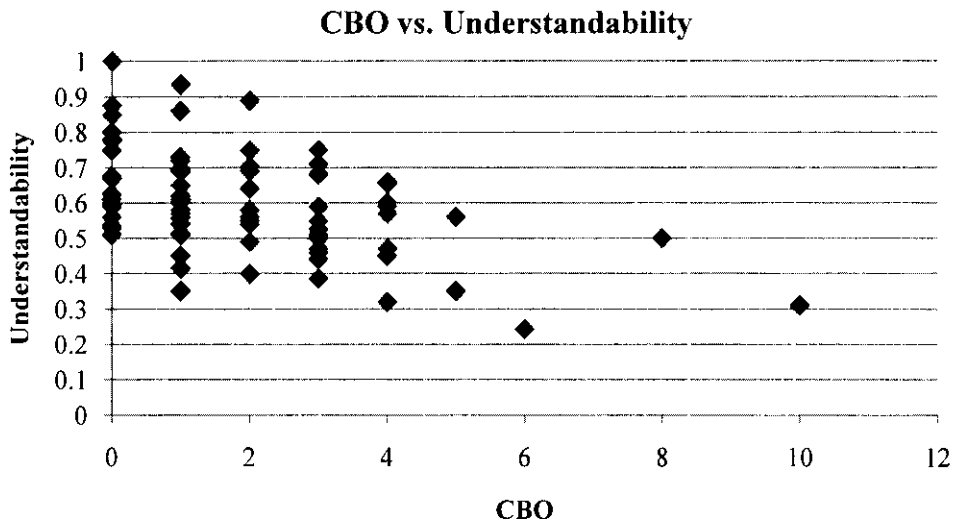


Figure 5.9 Scatter plot of CBO vs. Understandability

The correlation between CBO and understandability is $r(103) = -0.529$, $p = 0$. It shows a negative correlation between CBO and understandability. So, the null hypothesis is rejected and it can be concluded that CBO is negatively correlated to understandability. An increase in the value of CBO decreases understandability.

The following hypotheses about LCOM and understandability are tested:

H06 = LCOM of software has no effect on its understandability

H16 = LCOM of software has an effect on its understandability

Table 5.7 Pearson's correlation values CBO, LCOM and Understandability

Pearson's Correlations			
		CBO	LCOM
Understandability	Pearson Correlation	-.751**	-.357**
	Sig. (2-tailed)	.000	.000
	N	103	103
**. Correlation is significant at the 0.01 level (2-tailed).			

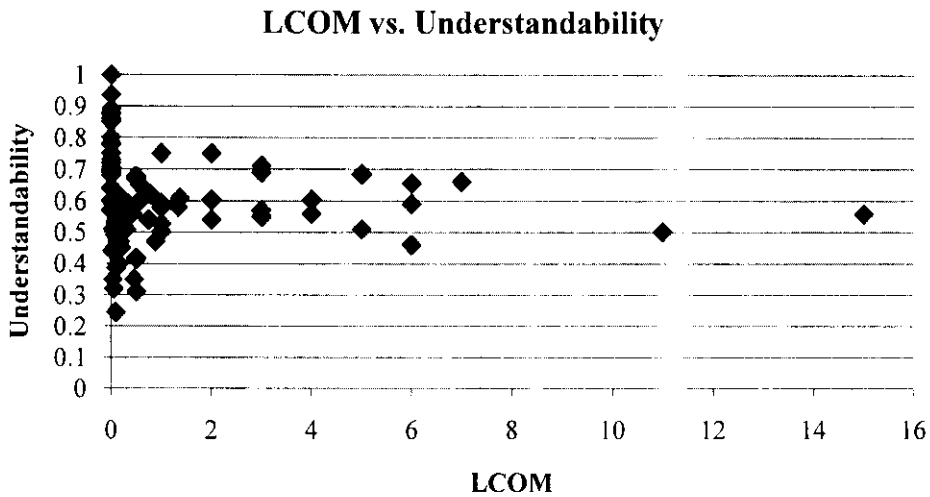


Figure 5.10 Scatter plot of LCOM vs. Understandability

The correlation between LCOM and understandability is $r(103) = -0.108$, $p = 0.278$. The r value shows a weak negative correlation between LCOM and understandability. However, the inequality $p > 0.05$ shows how insignificant this relationship is. Therefore, the alternate hypothesis is rejected and it can be concluded that LCOM is not related to understandability in this context.

5.6.1.4 Comments, Understandability

The following hypotheses about comments and understandability are tested:

H_{07} = Comments of software has no effect on its understandability

H_{17} = Comments of software has an effect on its understandability

The correlation between comments and understandability is $r(103) = -0.144$, $p = 0.146$. The r value shows a weak negative correlation between comments and understandability. However, the inequality $p > 0.05$ shows how insignificant this relationship is. Therefore, the alternate hypothesis is rejected and it can be concluded that comments are not related to understandability in this context.

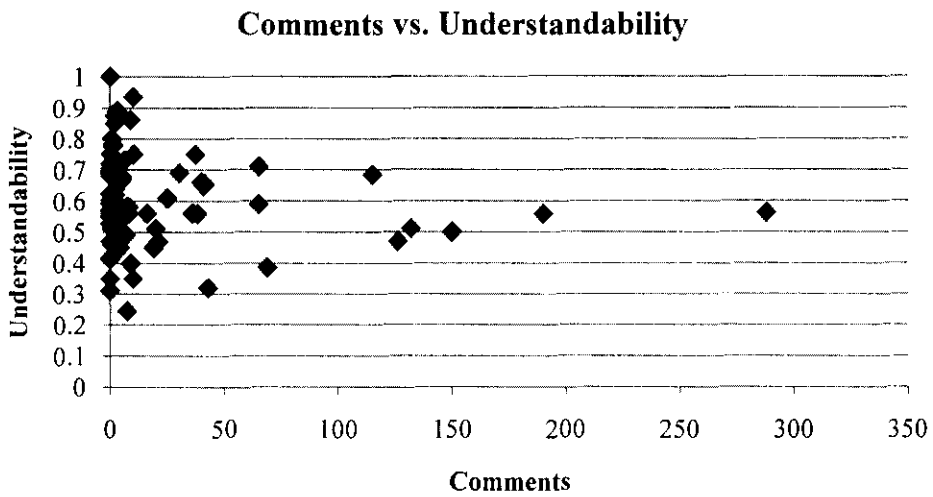


Figure 5.11 Scatter plot of Comments vs. Understandability

5.6.1.5 LOC, NOM, Understandability

The following hypotheses about LOC and understandability are tested:

H_{0_8} = LOC of software has no effect on its understandability

H_{1_8} = LOC of software has an effect on its understandability

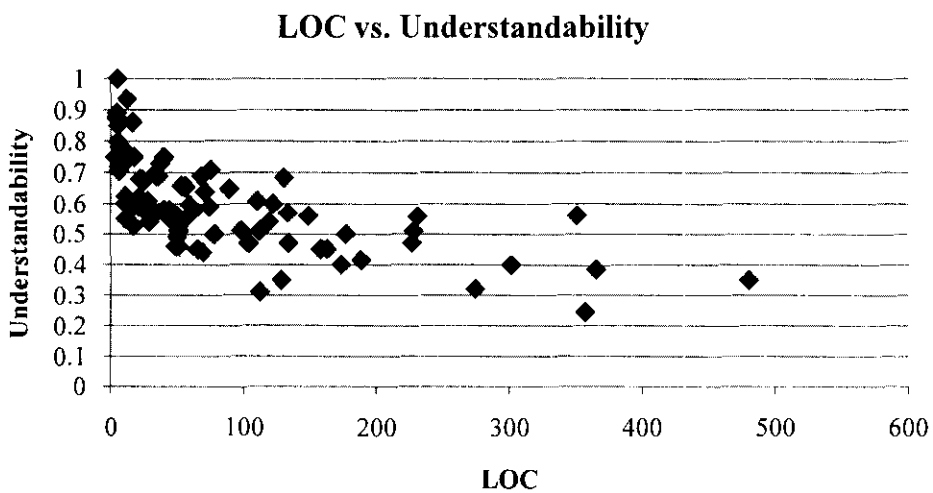


Figure 5.12 Scatter plot of LOC vs. Understandability

The correlation between LOC and understandability is $r(103) = -0.668$, $p = 0$. It shows a strong negative correlation between LOC and understandability. So, the null

hypothesis is rejected and it can be concluded that LOC is negatively correlated to understandability. An increase in the value of LOC decreases understandability.

The following hypotheses about NOM and understandability are tested:

H_0 = NOM of software has no effect on its understandability

H_1 = NOM of software has an effect on its understandability

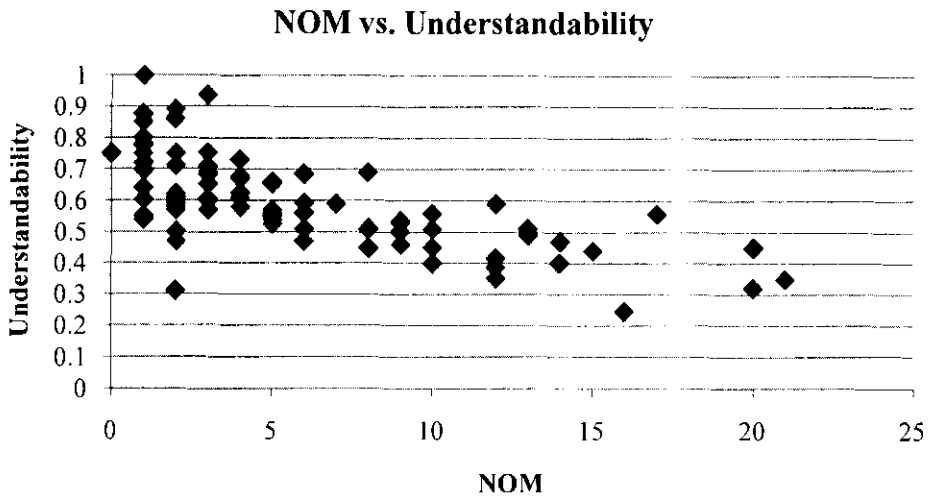


Figure 5.13 Scatter plot of NOM vs. Understandability

The correlation between NOM and understandability is $r(103) = -0.701, p = 0$. It shows a strong negative correlation between NOM and understandability. So, the null hypothesis is rejected and it can be concluded that NOM is negatively correlated to understandability. An increase in the value of NOM decreases understandability.

Table 5.8 Pearson's correlation values Understandability and its attributes

Pearson's Correlations						
		CBO	LCOM	Comments	LOC	NOM
Understandability	Pearson Correlation	-.529**	-.108	-.144	-.668**	-.701**
	Sig. (2-tailed)	.000	.278	.146	.000	.000
	N	103	103	103	103	103
**. Correlation is significant at the 0.01 level (2-tailed).						
*. Correlation is significant at the 0.05 level (2-tailed).						

5.6.2 Attribute Analysis

The analysis of attributes of reusability is presented in this section.

5.6.2.1 Flexibility, Reusability

The following hypotheses about flexibility and reusability are tested:

H0₁: Flexibility of software has no effect on its reusability

H1₁: Flexibility of software has an effect on its reusability

The correlation between flexibility and reusability is $r(103) = 0.762, p = 0$. It shows a strong positive correlation between flexibility and reusability. So, the null hypothesis is rejected and it can be concluded that flexibility is positively correlated to reusability. An increase in the value of flexibility increases reusability.

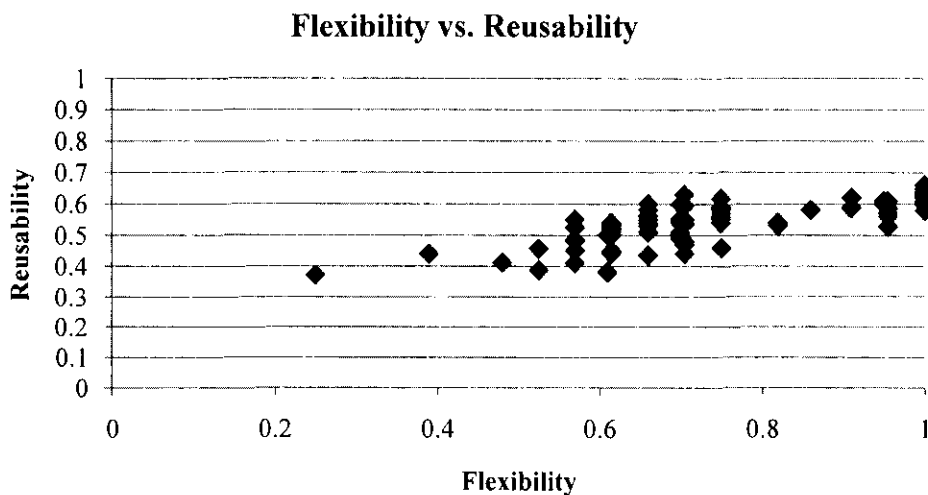


Figure 5.14 Scatter plot of flexibility vs. reusability

5.6.2.2 Understandability, Reusability

The following hypotheses about understandability and reusability are tested:

H0₂: Understandability of software has no effect on its reusability

H1₂: Understandability of software has an effect on its reusability

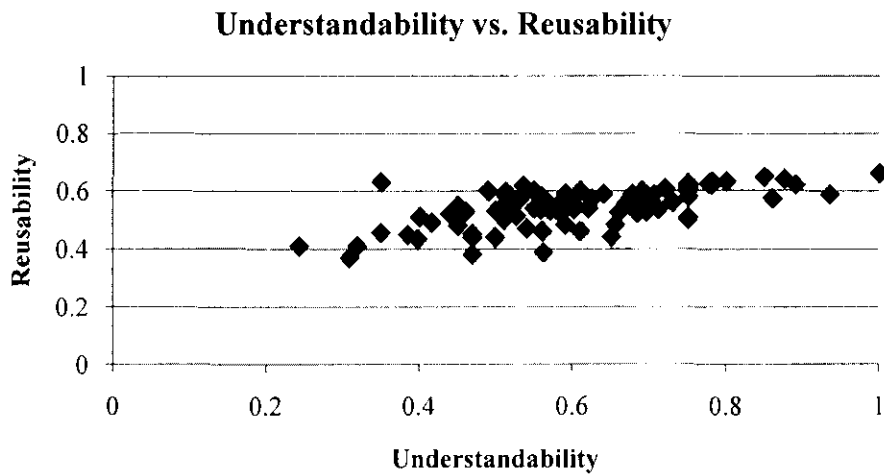


Figure 5.15 Scatter plot of Understandability vs. Reusability

The correlation between understandability and reusability is $r(103) = 0.669$, $p = 0$. The value of r shows a strong positive correlation between understandability and reusability. We reject the null hypothesis and it can be concluded that an increase in the value of understandability increases reusability.

5.6.2.3 Scope-coverage, Reusability

The following hypotheses about scope-coverage and reusability are tested:

H0₃: Scope-coverage of software has no effect on its reusability

H1₃: Scope-coverage of software has an effect on its reusability

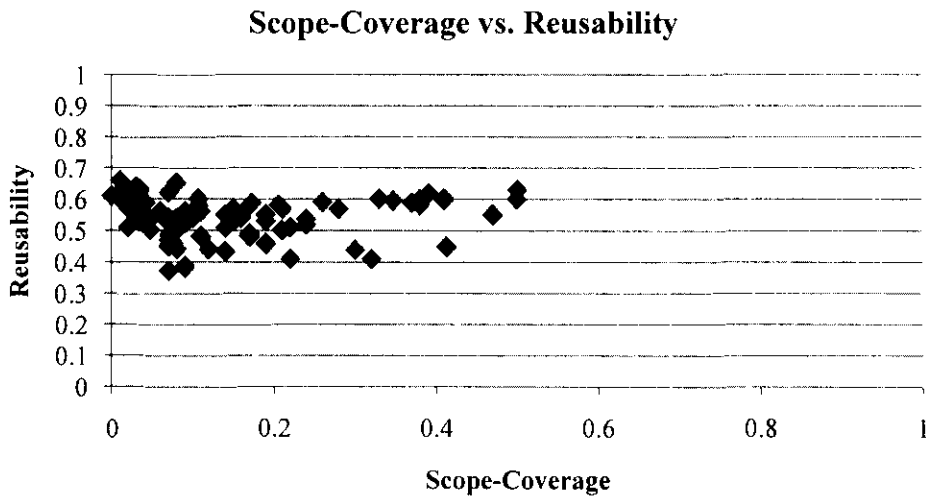


Figure 5.16 Scatter plot of scope-coverage vs. reusability

The correlation between scope coverage and reusability is $r(103) = -0.051$, $p = 0.609$. The r value shows a weak negative correlation between scope coverage and reusability. However, the inequality $p < 0.05$ shows how insignificant this relationship is. Therefore, the alternate hypothesis is rejected and it can be concluded that scope coverage is not related to reusability in this context. These results demand further investigation.

5.6.2.4 Variability, Reusability

The following hypotheses about variability and reusability are tested:

H0₄: Variability of software has no effect on its reusability

H1₄: Variability of software has an effect on its reusability

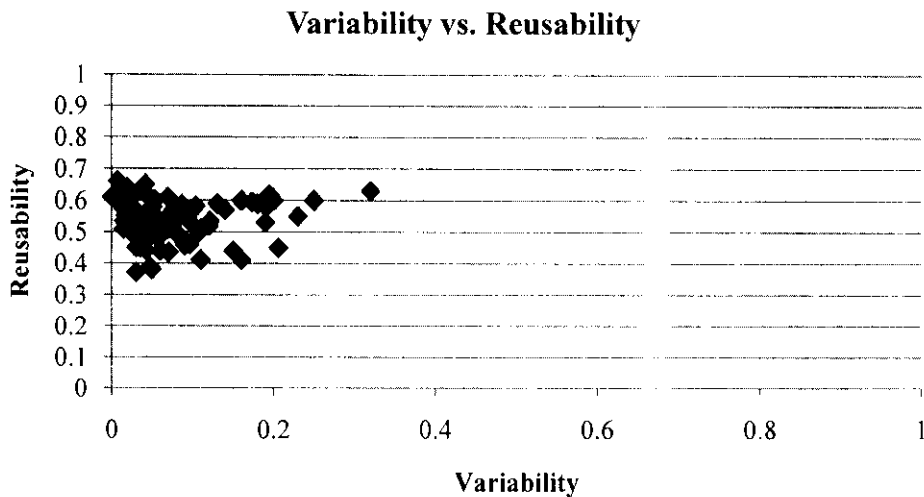


Figure 5.17 Scatter plot of variability vs. reusability

The correlation between variability and reusability is $r(103) = -0.042$, $p = 0.671$. There is a weak negative correlation between variability and reusability; further, $p > 0.05$ shows how insignificant the link is between variability and reusability. The correlation analysis leads to the rejection of the alternate hypothesis. It is concluded that variability is not related to reusability in this context. This conclusion demands further validation which may mean going back and rethinking about the variability metrics.

5.6.2.5 Maintainability, Reusability

The following hypotheses about maintainability and reusability are tested:

H_0 : Maintainability of software has no effect on its reusability

H_1 : Maintainability of software has an effect on its reusability

The correlation between maintainability and reusability is $r(103) = 0.797$, $p = 0$. The r value shows a strong positive correlation between maintainability and reusability. The null hypothesis is rejected and it can be concluded that an increase in maintainability increases reusability.

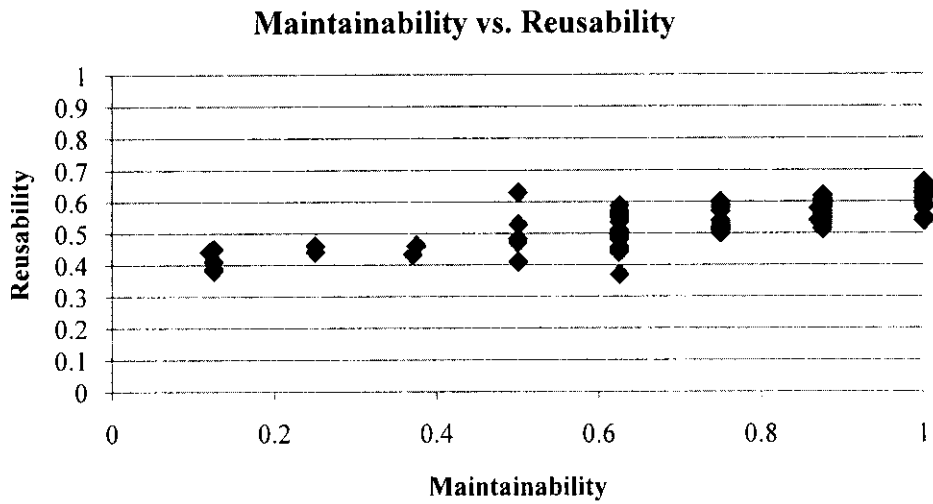


Figure 5.18 Scatter plot of maintainability vs. reusability

5.6.2.6 Portability, Reusability

The following hypotheses about portability and reusability are tested:

H0₆: Portability of software has no effect on its reusability

H1₆: Portability of software has an effect on its reusability

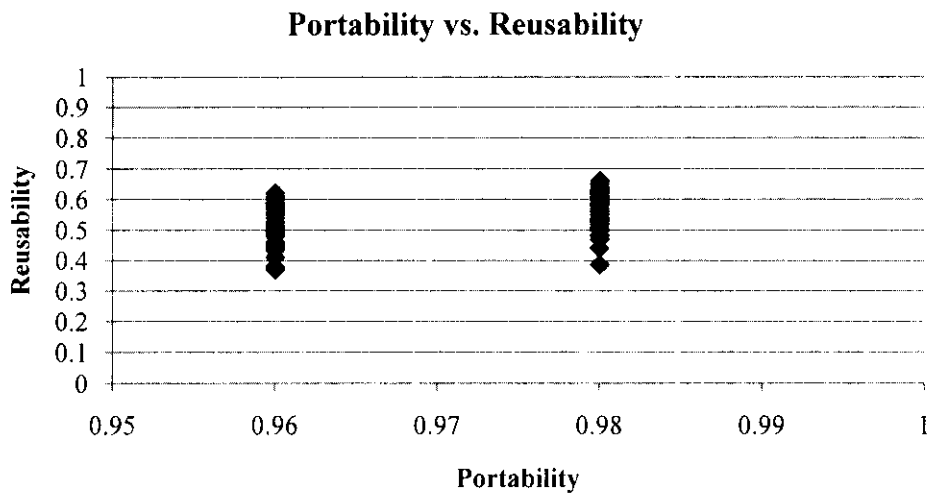


Figure 5.19 Scatter plot of portability vs. reusability

The correlation between portability and reusability is $r(103) = 0.404, p = 0$. The r value shows a weak positive correlation between portability and reusability. The value of p is 0, which leads to the rejection of null hypothesis. It can be concluded that there is a positive effect of portability on reusability. Increase in the value of portability increases reusability.

Table 5.9 Pearson's correlation values of Reusability and its attributes (class level)

Pearson's Correlations							
		Flexibility	Understand ability	Scope Coverage	Variability	Maintain ability	Port ability
Reusability	Pearson Correlation	.762**	.669**	-.051	-.042	.797**	.404**
	Sig. (2-tailed)	.000	.000	.609	.671	.000	.000
	N	103	103	103	103	103	103
**. Correlation is significant at the 0.01 level (2-tailed).							
*. Correlation is significant at the 0.05 level (2-tailed).							

5.7 Reusability Assessment at Package Level

In this section the results of experiment 2 are presented. This experiment is intended to test the hypotheses formulated as a result of the interview and survey. In this experiment the hypotheses related to the package level reusability attribute model are tested. The values of reusability are calculated using the equations stated earlier in this chapter. Pearson's correlation analysis is conducted using the statistical software. The results are presented in the form of scatter plots along with the correlation values between reusability and its attributes.

5.7.1 Attribute Analysis

In this section the relationship between reusability and its attributes is tested and presented in form of scatter plots.

5.7.1.1 Flexibility, Reusability

The following hypotheses about flexibility and reusability are tested:

H₀: Flexibility of package has no effect on its reusability

H₁: Flexibility of package has an effect on its reusability

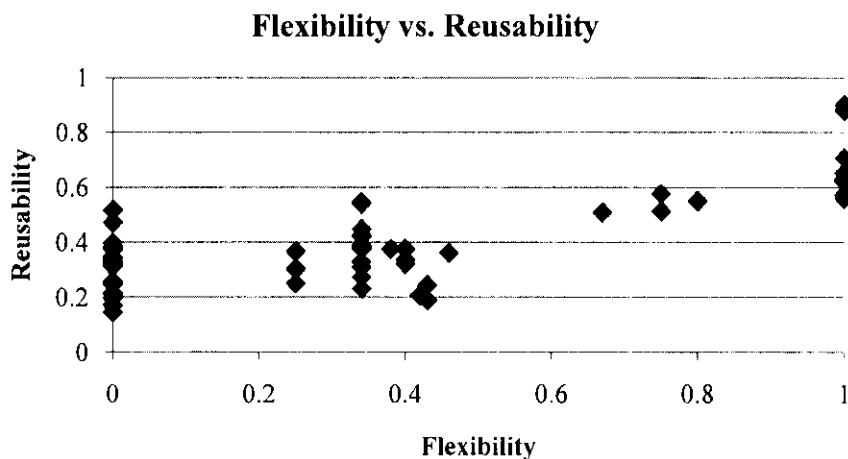


Figure 5.20 Scatter plot of flexibility vs. reusability

The correlation between flexibility and reusability is $r(77) = 0.789$, $p = 0$. It shows a strong positive correlation between flexibility and reusability. So, the null hypothesis is rejected and it can be concluded that flexibility is positively correlated to reusability. An increase in the value of flexibility increases reusability.

5.7.1.2 Variability, Reusability

The following hypotheses about variability and reusability are tested:

H0₂: Variability of package has no effect on its reusability

H1₂: Variability of software has an effect on its reusability

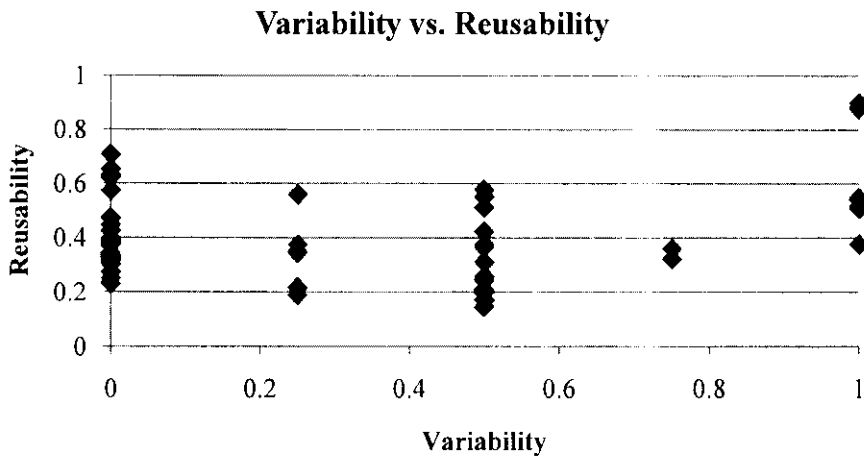


Figure 5.21 Scatter plot of variability vs. reusability

The correlation between variability and reusability is $r(77) = 0.674$, $p = 0$. The r value shows a weak positive correlation between variability and reusability. The value of p is 0, which leads to rejection of the null hypothesis. It can be concluded that there is a positive effect of variability on reusability. Increase in the value of variability increases reusability.

5.7.1.3 Portability, Reusability

The following hypotheses about portability and reusability are tested:

H0₃: Portability of package has no effect on its reusability

H1₃: Portability of package has an effect on its reusability

The correlation between portability and reusability is $r(77) = 0.693$, $p = 0$. The value of r shows a strong positive correlation between portability and reusability. We reject the null hypothesis and it can be concluded that an increase in the value of portability increases reusability.

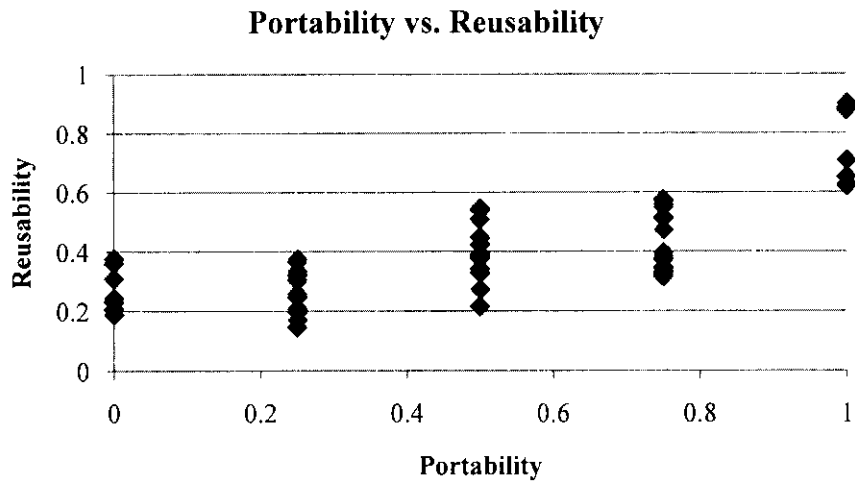


Figure 5.22 Scatter plot of portability vs. reusability

5.7.1.4 Maintainability, Reusability

The following hypotheses about maintainability and reusability are tested:

H0₄: Maintainability of package has no effect on its reusability

H1₄: Maintainability of package has an effect on its reusability

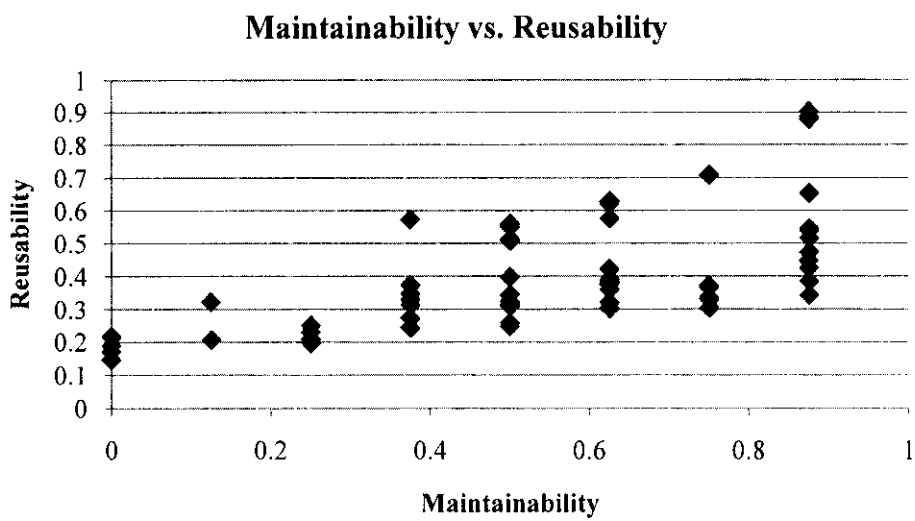


Figure 5.23 Scatter plot of maintainability vs. reusability

The correlation between maintainability and reusability is $r(77) = 0.667$, $p = 0$. The r value shows a strong positive correlation between maintainability and reusability. The null hypothesis is rejected and it can be concluded that an increase in maintainability increases reusability.

5.7.1.5 Understandability, Reusability

The following hypotheses about understandability and reusability are tested:

H_0 : Understandability of package has no effect on its reusability

H_1 : Understandability of package has an effect on its reusability

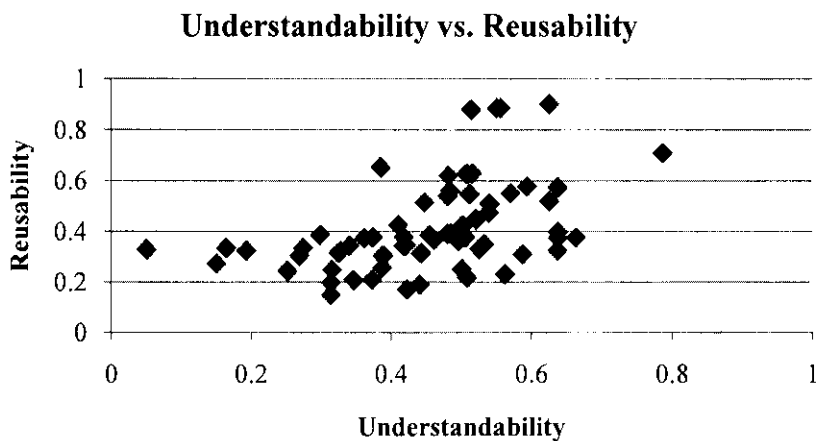


Figure 5.24 Scatter plot of understandability vs. reusability

The correlation between understandability and reusability is $r(77) = 0.417$, $p = 0$. The r value shows a weak positive correlation between understandability and reusability. The value of p is 0, which leads to rejection of the null hypothesis. It can be concluded that there is a positive effect of understandability on reusability. Increase in the value of portability increases reusability.

Table 5.10 Pearson's correlation values of reusability & attributes (package level)

Pearson's Correlations						
		Understand- ability	Flexibility	Portability	Variability	Maintain ability
Reusability	Pearson Correlation	.417**	.789**	.693**	.674**	.667**
	Sig. (2-tailed)	.000	.000	.000	.000	.000
	N	77	77	77	77	77
**. Correlation is significant at the 0.01 level (2-tailed).						

5.8 Evolutionary Reusability Analysis at Package Level

In this experiment, two open source software Jasmin and pBans are analyzed using the proposed reusability attribute model. During the analysis, six versions of jasmin and ten versions of pBeans are analyzed. The results are presented and discussed in the next sections.

5.8.1 Reusability Analysis of Jasmin

Jasmin software has evolved from version 1.0 to version 2.4 (6 versions). The number of packages remains the same that is four in all six versions. The number of classes evolved from 99 (in version 1.0) to 118 (in version 2.4). The number of methods increased from 618 (in version 1.0) to 792 (in version 2.3).

The detailed of the assessment of reusability and its attributes are presented in the following section.

5.8.1.1 Analysis of Package-1 (Jasmin)

The value of reusability of Jasmin package is 0.17 in version 1.0, which is the lowest. It keeps on increasing up to 0.26 in version 2.1. Version 2.1 shows the highest reusability value of Jasmin package i.e. 0.26. It can be observed that the contributing factor in this increase of value is maintainability. However, slight changes in the values of understandability and variability can also be seen.

Starting from version 2.2, the value of reusability is decreasing. Here, the factors that contribute to this decrease of values are understandability and maintainability. The values of reusability and respective attribute are presented in Table 5.11 and a graph plot for these values is presented in Figure 5.25.

Table 5.11 Version wise values of reusability and its attributes (package-1)

Jasmin							
	Versions						
	1.0	2.0	2.1	2.2	2.3	2.4	
Understandability	0.42	0.37	0.39	0.31	0.31	0.31	
Variability	0.18	0.17	0.15	0.17	0.17	0.17	
Maintainability	0	0.25	0.50	0.50	0.25	0	
Flexibility	0	0	0	0	0	0	
Portability	0.25	0.25	0.25	0.25	0.25	0.25	
Reusability	0.17	0.21	0.26	0.25	0.20	0.15	

Reusability and Attributes Values Package -1 (jasmin)

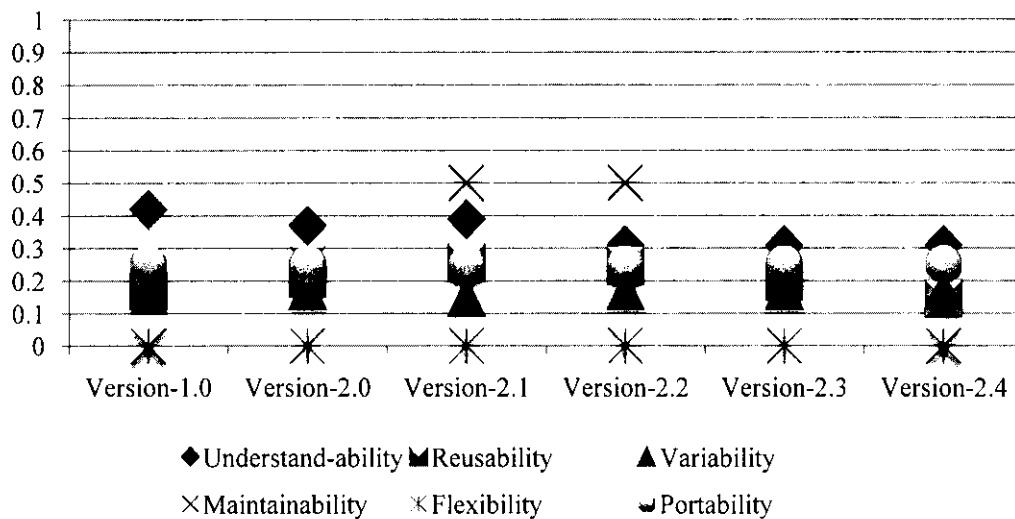


Figure 5.25 Reusability and its attribute values for package-1

In versions 1.0, 2.0 and 2.1 there is a significant increase in LOC, NOM and comments. This increase has an inverse effect on the understandability. The understandability value keeps on decreasing till version 2.2. In subsequent versions i.e. 2.3 and 2.4 there is no change in the understandability value. It is due to the minor difference in the values of LOC, NOM and comments. The version wise values of understandability, LOC, NOM and comments are presented in Table 5.12.

Table 5.12 Version wise values of understandability and its attributes (package-1)

Jasmin	Versions					
	1.0	2.0	2.1	2.2	2.3	2.4
LOC	2145	2759	3860	3890	3902	3911
NOM	82	105	125	130	130	131
Comments	1004	1363	2022	1964	1961	1963
Understandability	0.42	0.37	0.39	0.31	0.31	0.31

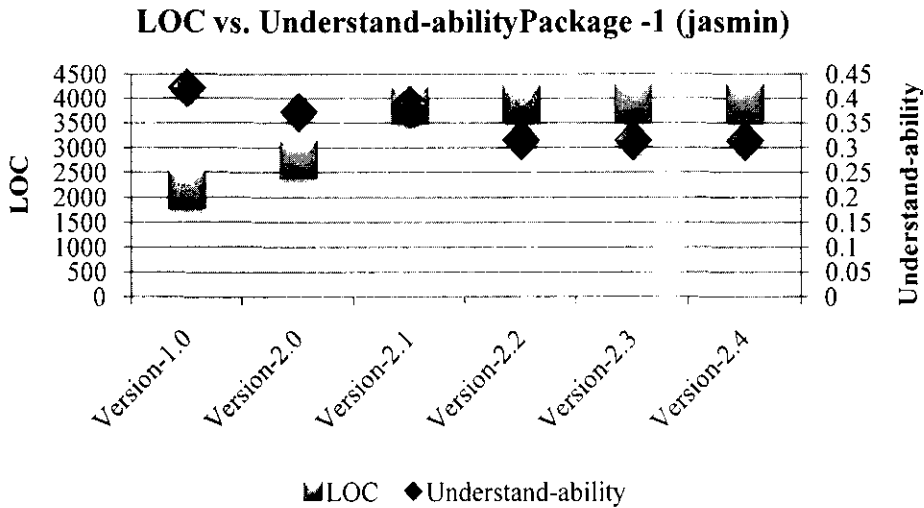


Figure 5.26 Graph plot of values of LOC and understandability package-1

In Figure 5.26 the increase in LOC and its inverse effect on understandability is quite visible. The value of understandability is highest in version 1.0 and lowest in version 2.4. On the other hand the value of LOC is lowest in version 1.0 and highest in version 2.4.

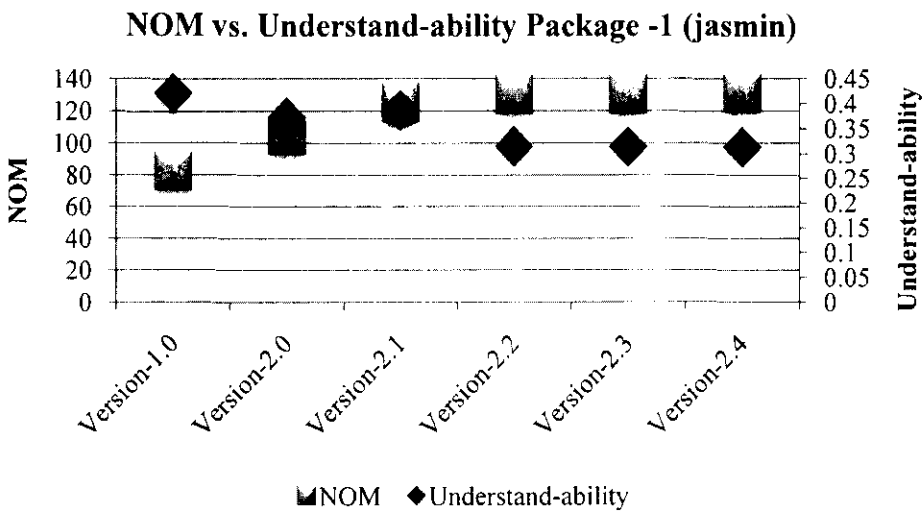


Figure 5.27 Graph plot of values of NOM and understandability package-1

The increase in NOM causes increase in size which decreases understandability. The value of NOM is lowest in version 1.0, and the value of understandability is the highest in this version. The trend can be viewed in Figure 5.27, where all the versions are showing the same response of increase in NOM. The number of methods keeps on increasing as a result of it understandability keeps on decreasing.

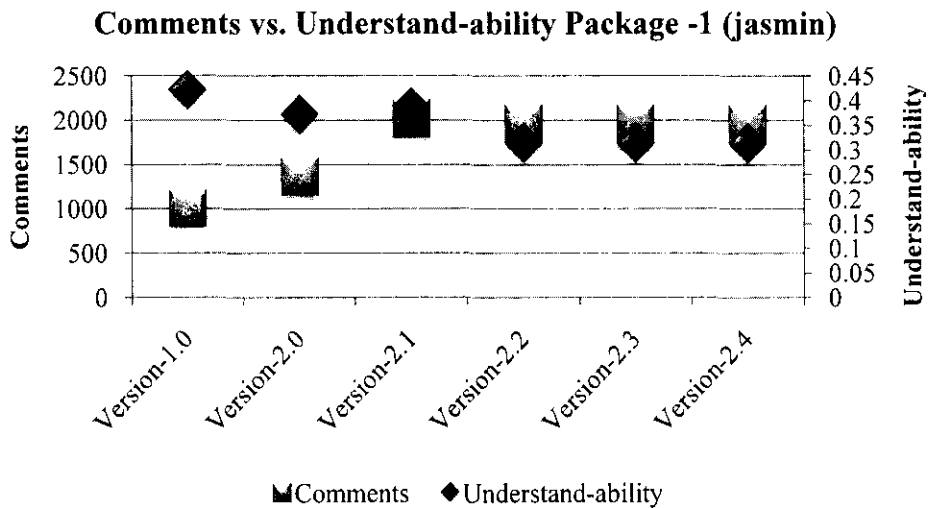


Figure 5.28 Graph plot of values of comments and understandability package-1

The value of comments and its effect on reusability is not consistent to the theory which states that increase in comments increases understandability. In Figure 5.28, it can be seen that the number of comments is increasing in version 1.0 to version 2.0. However, the value of understandability is continuously decreasing. The underlying reason can be understood by having a look at Figure 5.26 and Figure 5.27. The number of lines of code and number of methods increase significantly in every version, which overshadows the effect of increase in comments.

Table 5.13 Version wise values of maintainability and its attributes (package-1)

Jasmin		Versions					
		1.0	2.0	2.1	2.2	2.3	2.4
MI		59.91	68.63	95.21	85.99	76.1	62.99
CC		73	77	85	87	87	88
Maintainability		0	0.25	0.50	0.50	0.25	0

The value of maintainability of Jasmin package increased from 0 to 0.5 in version 2.2, which is the maximum value. The contribution of rise in the value of MI can be seen from version 1.0 to 2.2. The value of CC keeps on increasing which has a negative effect on maintainability. After version 2.1, the value of MI is decreasing

which is causing a decrease in the value of maintainability. The values of MI, CC and maintainability are presented in Table 5.13, Figure 5.29 and Figure 5.30.

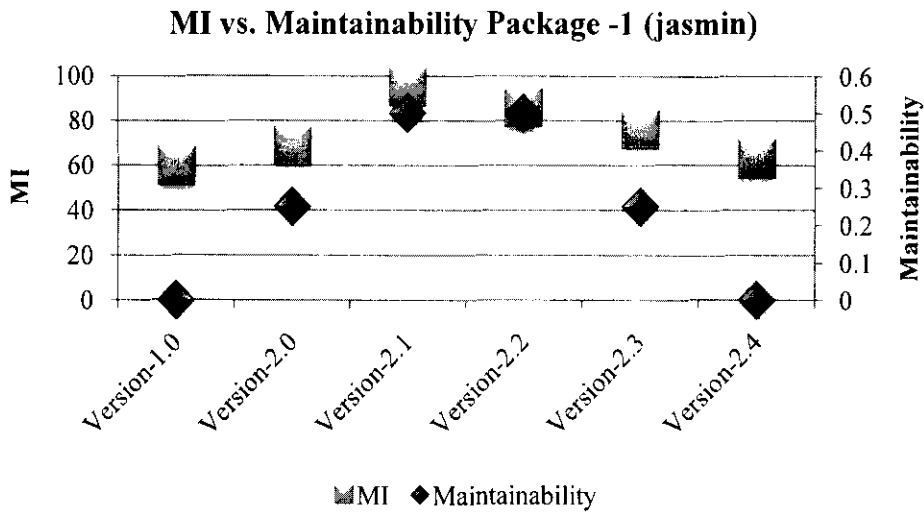


Figure 5.29 Graph plot of values of MI and maintainability package-1

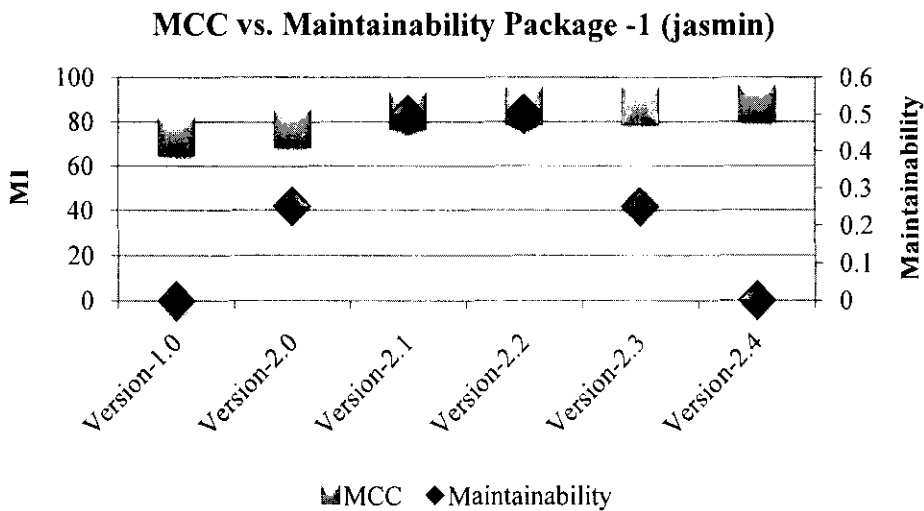


Figure 5.30 Graph plot of values of CC and maintainability package-1

5.8.1.2 Analysis of Package-2 (java_cup.runtime)

The package `java_cup.runtime` has evolved from version 1.0 to version 2.4. The highest value of reusability can be observed in version 1.0. It is due to the contribution of flexibility value. The reusability decreased in version 2.0 and 2.1. In version 2.1 the reusability value reached the lowest point that is 0.37. This decrease is due to the value of maintainability. Afterwards in versions 2.2, 2.3 and 2.4, the value

remains the same i.e. 0.37, because of no change in the values of the attributes. It shows that there is no significant change in the package in these versions. The values are presented in

Table 5.14 and Figure 5.31.

The value of understandability remains constant in the versions of package `java_cup.runtime`. There is no change in the lines of code, number of methods or number of comments.

Table 5.14 Version wise values of reusability and its attributes (package-2)

java_cup.runtime						
	Versions					
	1.0	2.0	2.1	2.2	2.3	2.4
Understandability	0.64	0.64	0.64	0.64	0.64	0.64
Flexibility	1.00	0.00	0.00	0.00	0.00	0.00
Portability	0.75	0.75	0.75	0.75	0.75	0.75
Variability	0.10	0.10	0.10	0.10	0.10	0.10
Maintainability	0.38	0.50	0.38	0.38	0.38	0.38
Reusability	0.57	0.40	0.37	0.37	0.37	0.37

**Reusability and Attributes Values Package -2
(java_cup.runtime)**

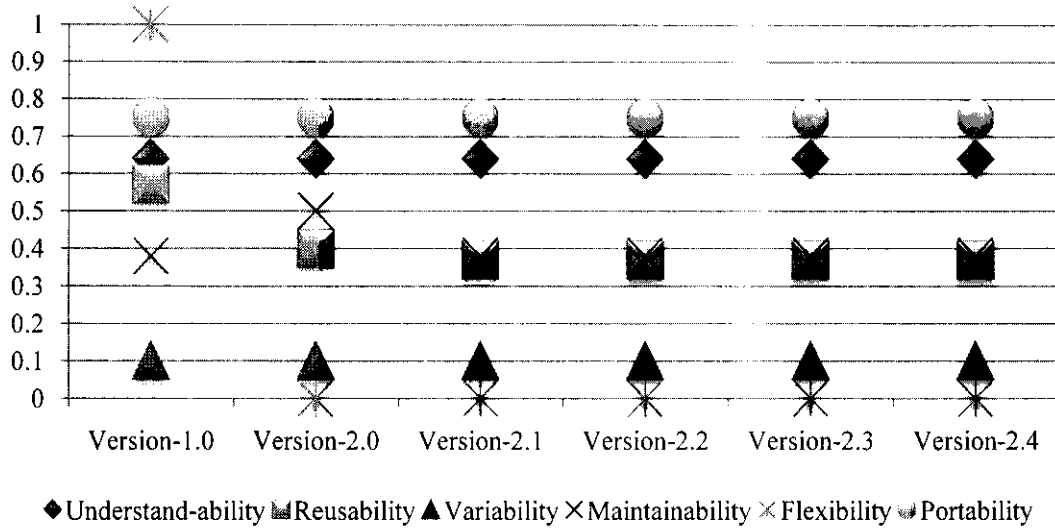


Figure 5.31 Reusability and its attribute values for package-2

The maintainability value of java_cup.runtime package increased in version 2.0 i.e. 0.50, which is the highest one. This increase is due to the decrease in complexity. In the subsequent versions 2.2 to 2.4 the value of maintainability remains the same due to no change in the values of the attributes. The values of MI, CC and maintainability are presented in Table 5.15, Figure 5.32 and Figure 5.33.

Table 5.15 Version wise values of maintainability and its attributes (package-2)

java_cup.runtime	Versions					
	1.0	2.0	2.1	2.2	2.3	2.4
MI	73.48	73.48	73.48	73.48	73.48	73.48
CC	24	13	24	24	24	24
Maintainability	0.38	0.50	0.38	0.38	0.38	0.38

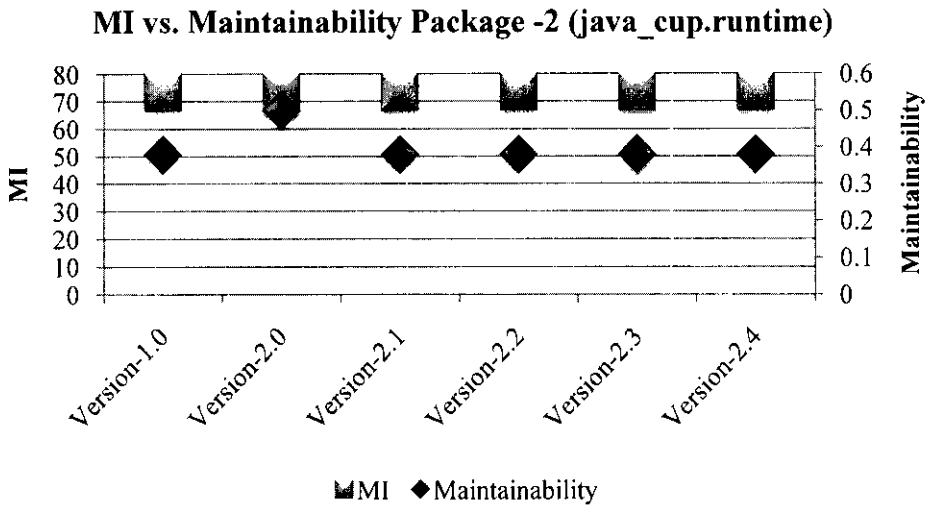


Figure 5.32 Graph plot of values of MI and maintainability package-2

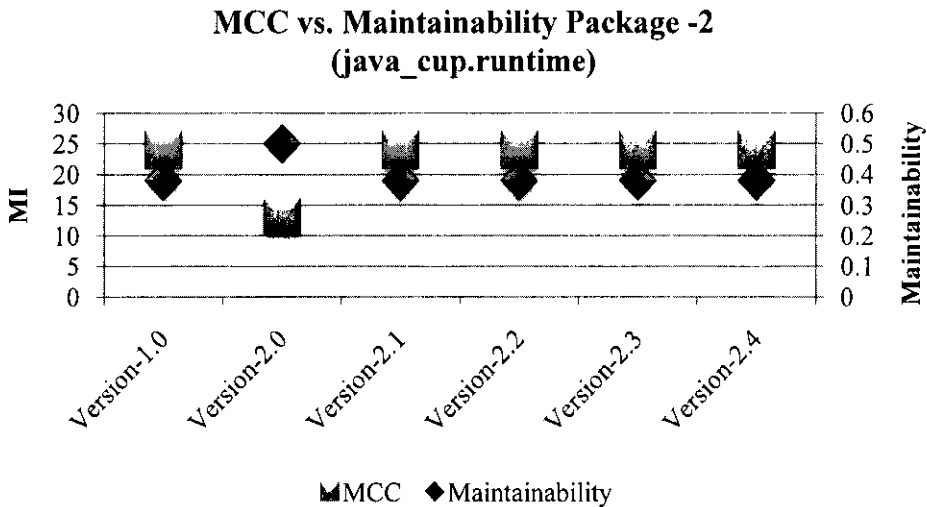


Figure 5.33 Graph plot of values of CC and maintainability package-2

5.8.1.3 Analysis of Package-3 (Jas)

The package Jas is part of Jasmin throughout in all versions. However, the change in the reusability value can be seen in version 2.0. The highest value of reusability is observed in version 1.0 i.e. 0.56. The value of reusability decreased to 0.38 in version 2.0 and remains the same in all subsequent versions. The values of reusability and attributes are presented in Table 5.16 and Figure 5.34. This constant value of

reusability is due to the fact that the values of attributes remain unchanged after version 2.0.

Table 5.16 Version wise values of reusability and its attributes (package-3)

Jas	Versions					
	1.0	2.0	2.1	2.2	2.3	2.4
Understandability	0.48	0.42	0.42	0.42	0.42	0.42
Flexibility	1.00	0.00	0.00	0.00	0.00	0.00
Portability	0.75	0.75	0.75	0.75	0.75	0.75
Variability	0.06	0.05	0.05	0.05	0.05	0.05
Maintainability	0.50	0.50	0.50	0.50	0.50	0.50
Reusability	0.56	0.34	0.34	0.34	0.34	0.34

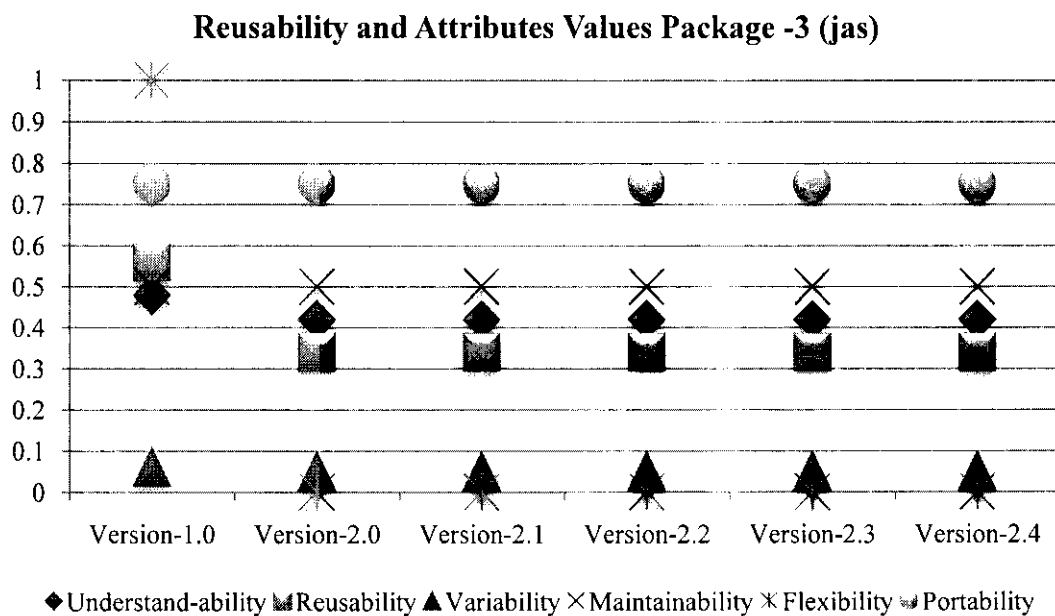


Figure 5.34 Reusability and its attribute values for packege-3

The values of LOC, NOM, comments and understandability are represented in Table 5.17, Figure 5.35, Figure 5.36 and Figure 5.37. The value of understandability decreased from 0.48 to 0.42. It remains the same in subsequent versions due to a slight increase in the other factors. The increase in NOM is significant in version 2.0, which is the cause of decrease in the value of understandability. The increase in LOC,

NOM and comments can be seen in the later versions. However, the increase in LOC, NOM and comments is not significant. So, this increase has no effect on the value of understandability.

Table 5.17 Version wise values of understandability and its attributes (package-3)

jas	Versions					
	1.0	2.0	2.1	2.2	2.3	2.4
LOC	1708	2177	2937	3101	3102	3105
NOM	191	244	308	318	318	319
Comments	157	198	249	273	273	273
Understandability	0.48	0.42	0.42	0.42	0.42	0.42

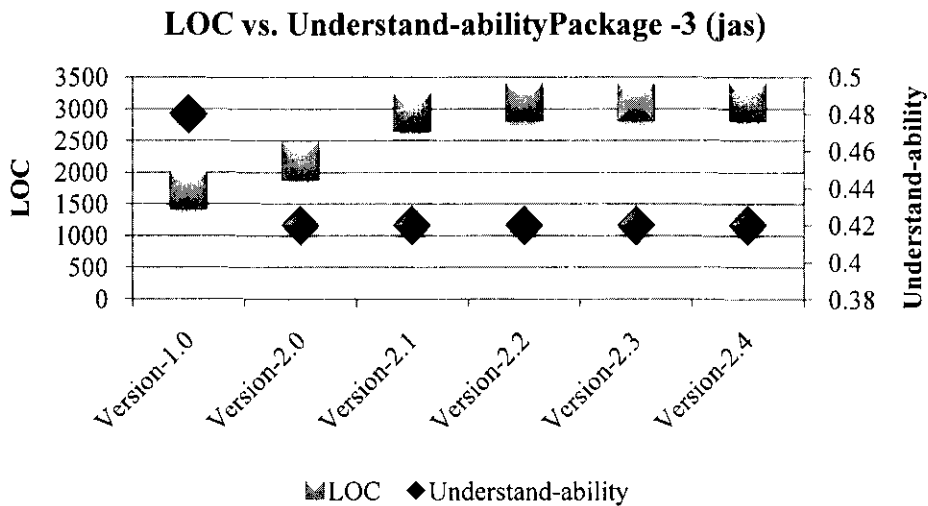


Figure 5.35 Graph plot of values of LOC and understandability package-3

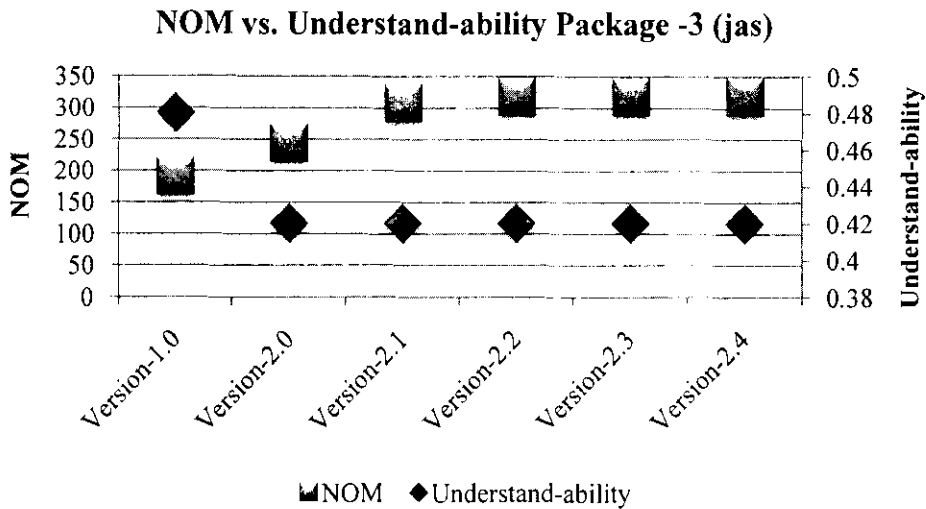


Figure 5.36 Graph plot of values of NOM and understandability package-3

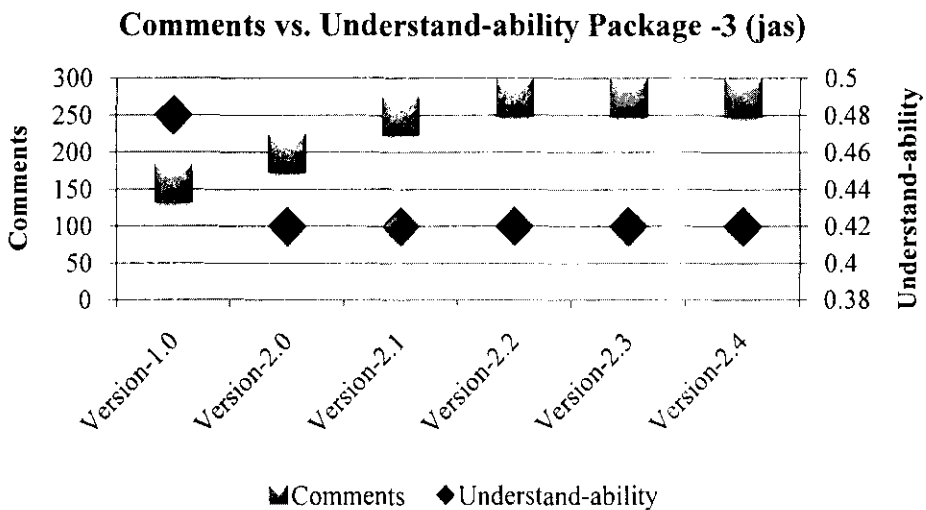


Figure 5.37 Graph plot of values of comments and understandability package-3
The values of CC, MI and maintainability are presented in

Table 5.18, Figure 5.38 and Figure 5.39. The value of CC increased from 99 to 145 in version 2.0. The value of CC further increased to 231 in version 2.1. There is a slight increase in the values of MI. However, the increase in the values of CC and MI is neglect able. Therefore, no effect of change in CC and MI values is observed on maintainability value.

Table 5.18 Version wise values of maintainability and its attributes (package-3)

jas	Versions					
	1.0	2.0	2.1	2.2	2.3	2.4
MI	135.62	135.67	132.32	131.87	131.58	151.15
CC	99	145	231	252	252	252
Maintainability	0.50	0.50	0.50	0.50	0.50	0.50

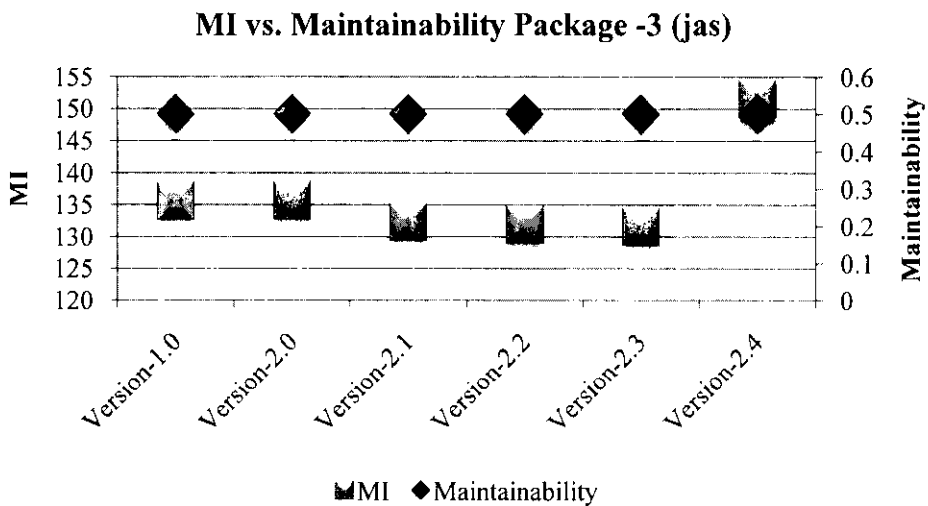


Figure 5.38 Graph plot of values of comments and understandability package-3

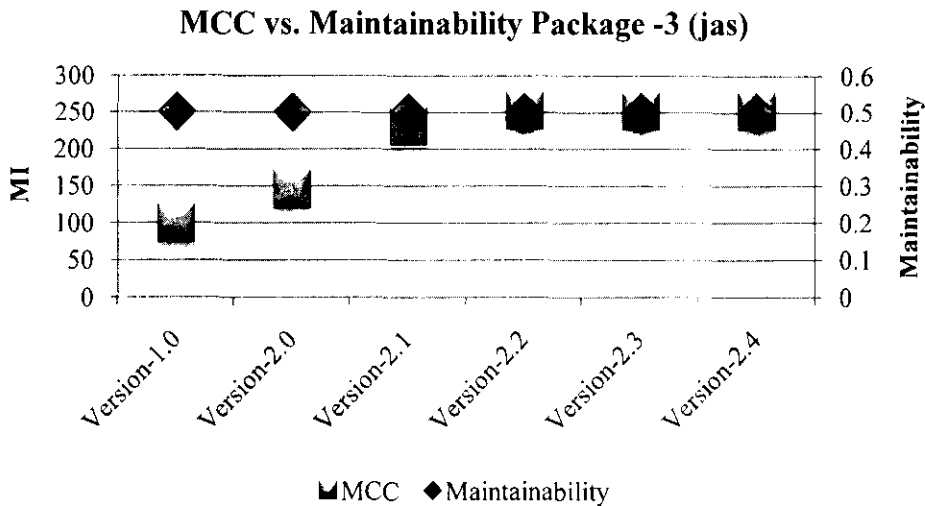


Figure 5.39 Graph plot of values of CC and maintainability package-3

5.8.2 Reusability Analysis of pBeans

pBeans package has evolved from version 1.0 to version 2.0.2 (10 versions). The number of packages increased from three to eight. The number of classes evolved from 28 (in version 1.0) to 49 (in version 2.0.2). The number of methods increased from 161 (in version 1.0) to 341 (in version 2.0.2).

The details of the assessment of reusability and its attributes are presented in the following section.

5.8.2.1 Analysis of Package-2 (pbean)

The pbean package is part of the software in all of its versions. The reusability and attribute values are presented in Table 5.19 and Figure 5.40. It can be observed that there is a significant difference in the reusability value of pbean package 1.3.0.

Table 5.19 Version wise values of reusability and its attributes (package-2)

pbean	Versions									
	1.0	1.1	1.2	1.2.1	1.2.2	1.3.0	1.3.1	2.0	2.0.1	2.0.2
Understand ability	0.94	0.94	0.88	0.88	0.87	0.86	0.89	0.38	0.87	0.87
Flexibility	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
Portability	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50
Variability	0.45	0.45	0.38	0.38	0.38	0.38	0.41	0.67	0.67	0.67
Maintain ability	0.38	0.38	0.38	0.38	0.38	0.38	0.38	0.63	0.63	0.63
Reusability	0.45	0.45	0.43	0.43	0.43	0.62	0.43	0.43	0.53	0.53

The reusability value increased to 0.62 in version 1.3.0, which is the highest one. This increase is due to the increase in the value of flexibility. Another increase in the value can be observed in version 2.0.1, which is due to the increase in understandability value. The attribute values show that the package has not been changed much up to version 1.2.2. The major changes can be observed in version 1.3.0 and version 2.0.1.

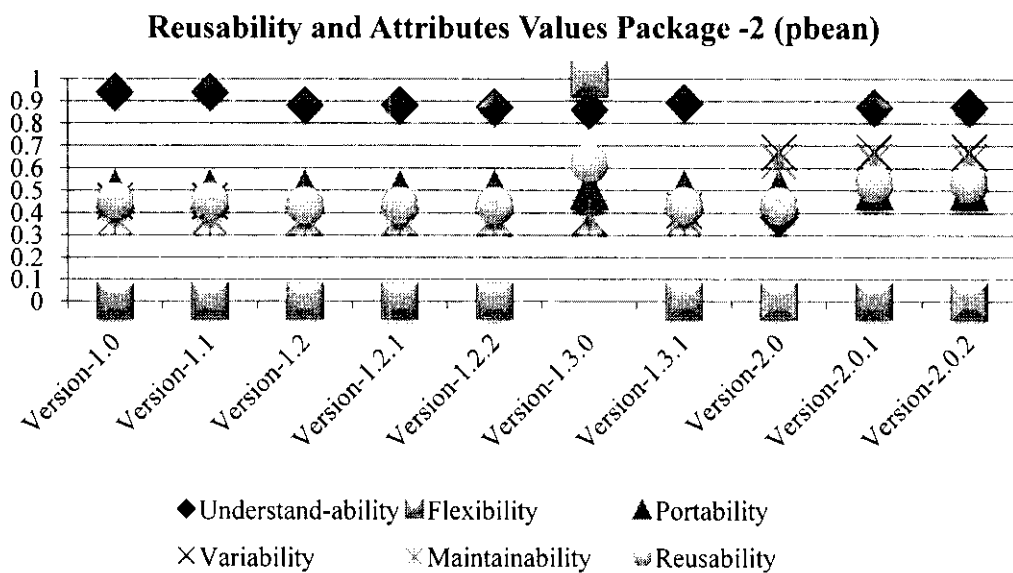


Figure 5.40 Reusability and its attribute values for package-2

The values of LOC, NOM and comments are presented in Table 5.20, Figure 5.41, Figure 5.42 and Figure 5.43. The values show a decrease in understandability value in version 1.2, which is because of the increase in the number of LOC and NOM. The increase in size has an inverse effect on understandability.

A significant decrease in understandability value can be seen in version 2.0, which is due to the increase in LOC, NOM and lack of comments. In the version 2.0.1, the value of understandability increased to 0.87. This increase is contributed by the increase in comments and decrease in LOC.

Table 5.20 Version wise values of understandability and its attributes (package-2)

pbean	Versions									
	1.0	1.1	1.2	1.2.1	1.2.2	1.3.0	1.3.1	2.0	2.0.1	2.0.2
LOC	364	375	437	437	437	437	558	638	614	614
NOM	43	43	53	53	53	53	68	106	107	107
Comments	364	375	437	437	433	425	500	1	604	604
Understandability	0.94	0.94	0.88	0.88	0.87	0.86	0.89	0.38	0.87	0.87

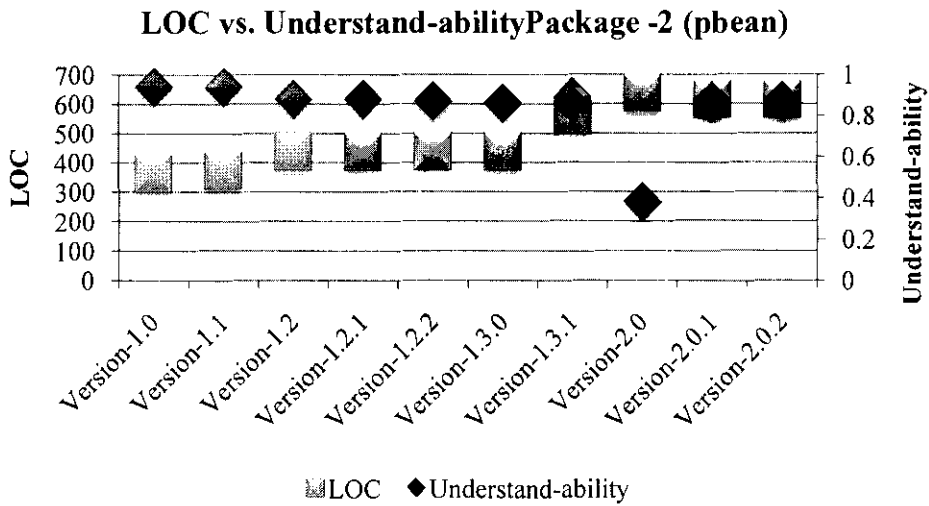


Figure 5.41 Graph plot of values of LOC and understandability package-2

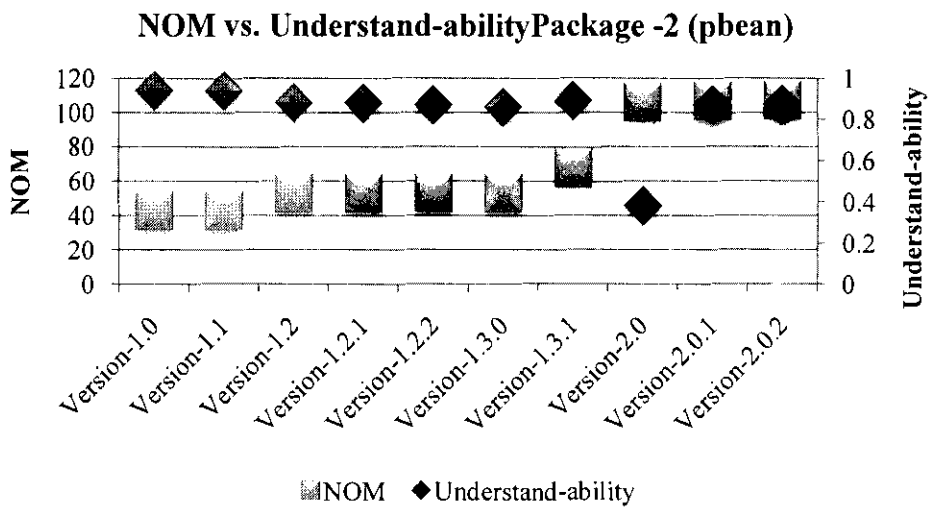


Figure 5.42 Graph plot of values of NOM and understandability package-2

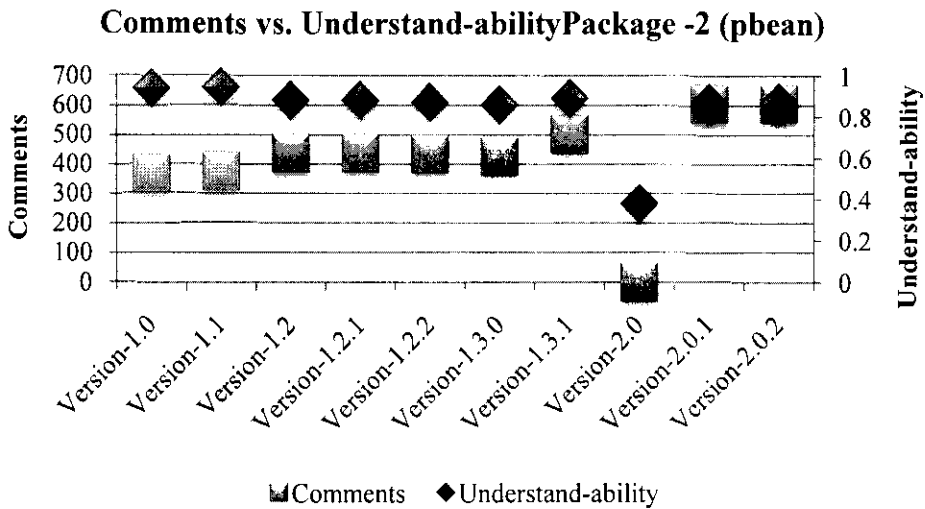


Figure 5.43 Graph plot of values of comments and understandability package-2

The value of maintainability remains the same in version 1.0 to version 1.3.1. A decrease in the value of complexity can be observed in version 2.0. This decrease in complexity and a significant increase in the value of MI increased the value of maintainability to 0.63. The value of maintainability remains the same for the subsequent versions. It shows major changes in the package in versions 1.3.1 and 2.0.

Table 5.21 Version wise values of maintainability and its attributes (package-2)

pbean	Versions									
	1.0	1.1	1.2	1.2.1	1.2.2	1.3.0	1.3.1	2.0	2.0.1	2.0.2
MI	67.6	67.1	69.1	69.1	69.1	69.1	67.6	130.3	98.1	98.1
	9	1	1	1	1	1	5	5	2	2
CC	36	36	39	39	39	39	46	34	34	34
Maintai	0.38	0.38	0.38	0.38	0.38	0.38	0.38	0.63	0.63	0.63
n-ability										

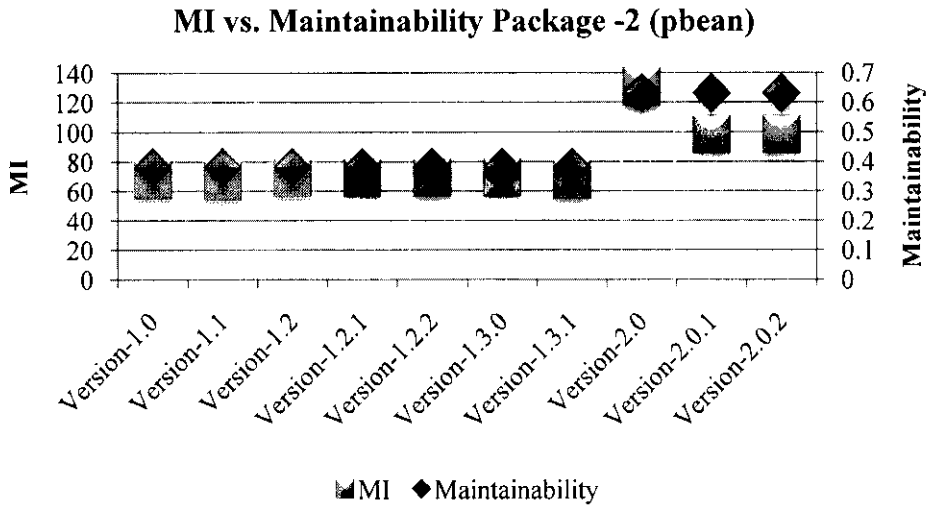


Figure 5.44 Graph plot of values of MI and maintainability package-2

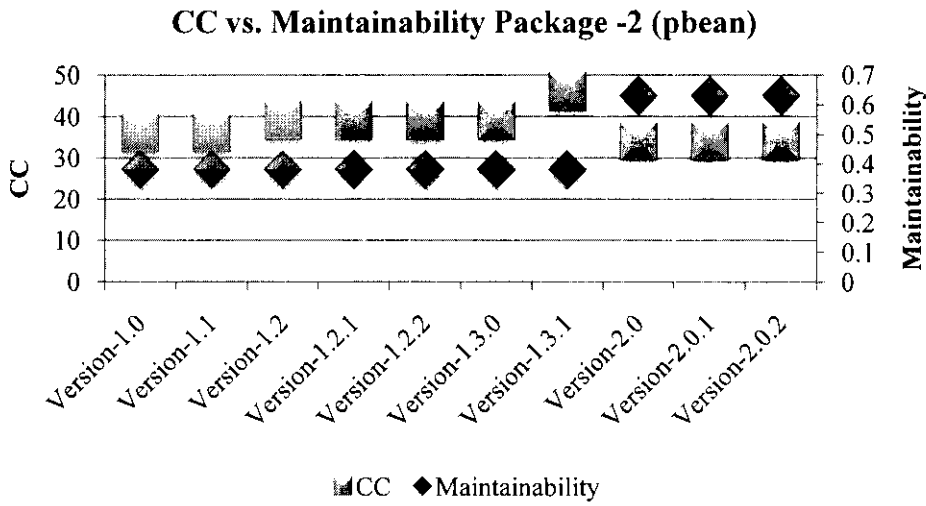


Figure 5.45 Graph plot of values of CC and maintainability package-3

5.8.2.2 Analysis of Package-3 (pbean.data)

The reusability and attributes values for pbean.data package are presented in

Table 5.22 and Figure 5.46. The highest reusability value is observed in version 1.3.1. This highest value is due to the high values of understandability and maintainability. The value of reusability is lowest in versions 1.2 and 1.3.0. It can be said that version 1.3.0 has undergone major changes which improved its reusability value.

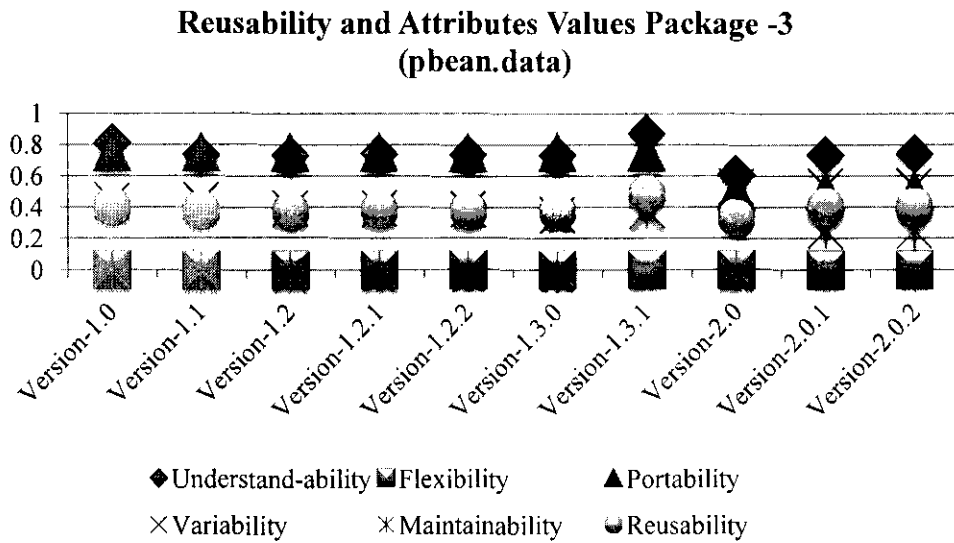


Figure 5.46 Reusability and its attribute values for package-3

Table 5.22 Version wise values of reusability and its attributes (package-3)

pbean.data	Versions									
	1.0	1.1	1.2	1.2.1	1.2.2	1.3.0	1.3.1	2.0	2.0.1	2.0.2
Understand-ability	0.81	0.73	0.73	0.74	0.74	0.73	0.87	0.61	0.73	0.74
Flexibility	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Portability	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.50	0.50	0.50
Variability	0.44	0.44	0.39	0.39	0.39	0.35	0.38	0.53	0.53	0.53
Maintain-ability	0.00	0.00	0.00	0.00	0.00	0.00	0.38	0.00	0.25	0.25
Reusability	0.40	0.38	0.37	0.38	0.38	0.37	0.48	0.33	0.40	0.40

The values of LOC, NOM, comments and understandability are presented in Table 5.23, Figure 5.47, Figure 5.48 and Figure 5.49. Version 1.0 of pbeans.data has the highest value of understandability i.e. 0.81. The value is decreased to 0.73 in version 1.1, where there is an increase in NOM and LOC. There are slight changes in the values in versions 1.2, 1.2.1, 1.2.2 and 1.2.3. However, the changes are insignificant and their effect cannot be seen on the values of understandability.

Table 5.23 Version wise values of understandability and its attributes (package-3)

Pbean.data	Versions										
	1.0	1.1	1.2	1.2.1	1.2.2	1.2.3	1.3.0	1.3.1	2.0.0	2.0.1	2.0.2
LOC	109	113	134	137	137	150	834	176	175	17	
	7	6	7	5	5	2		8	8	77	
NOM	116	120	138	142	142	154	105	145	148	14	9
Comments	109	110	130	135	134	145	830	127	170	17	
	7	0	0	0	5	6		2	0	50	
Understandability	0.81	0.73	0.73	0.74	0.74	0.73	0.87	0.61	0.73	0.7	4

In version 1.3.1 there is a significant decrease in NOM and LOC which resulted in an increase in the understandability value. Afterwards, in version 2.0 the value of understandability decreased to 0.61, due to an increase in LOC and NOM. It can be observed that the numbers of comments are also increasing with LOC and vice-versa.

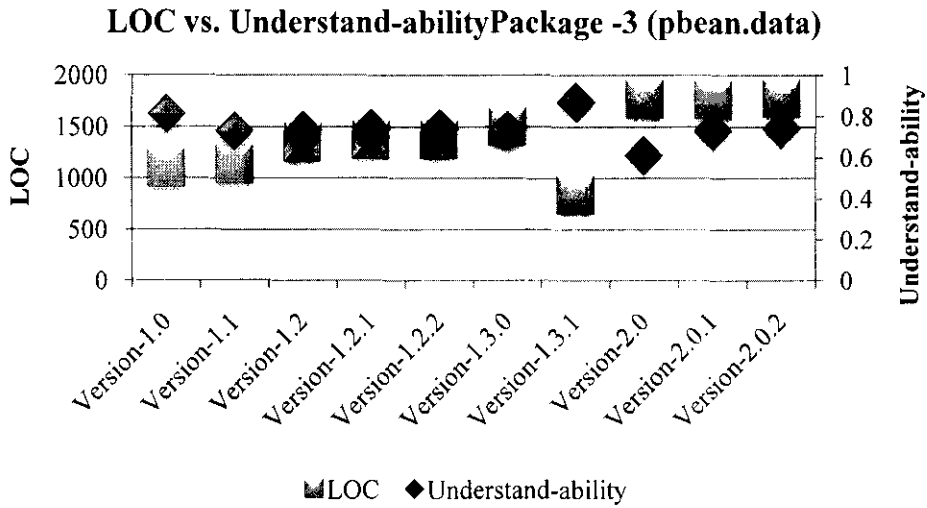


Figure 5.47 Graph plot of values of LOC and understandability package-3

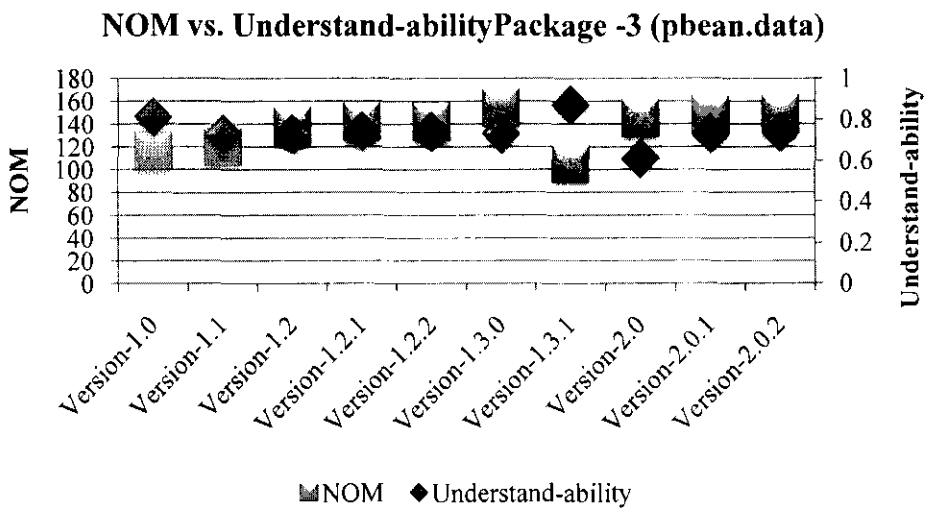


Figure 5.48 Graph plot of values of NOM and understandability package-3

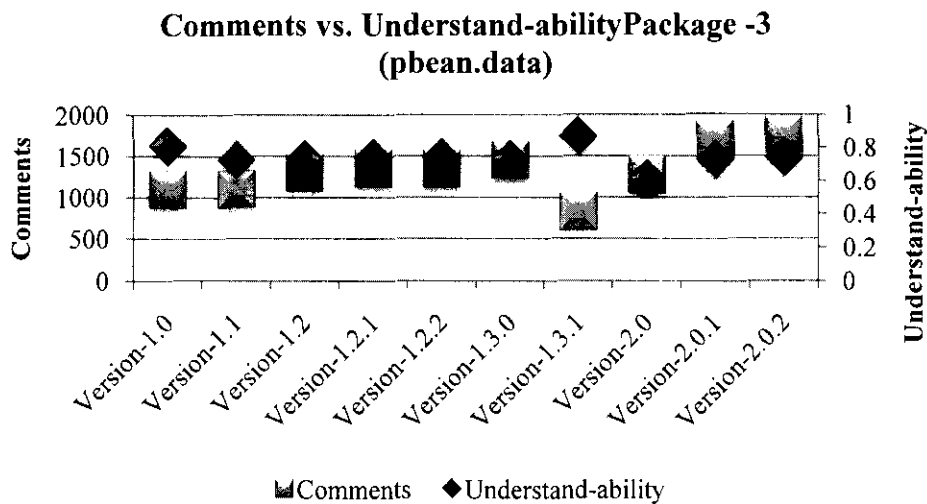


Figure 5.49 Graph plot of values of comments and understandability package-3

The values of MI, complexity and maintainability are presented in Table 5.24, Figure 5.50 and Figure 5.51. A significant decrease in the value of complexity can be seen in the version 1.3.1. Therefore, in version 1.3.1, pbean.data package has the highest value for maintainability. In subsequent versions, the complexity value increased up to 153. However, its effect is normalized by the MI value which has increased in versions 2.0.1 and 2.0.2.

Table 5.24 Version wise values of maintainability and its attributes (package-3)

Pbean.data	Versions									
	1.0	1.1	1.2	1.2.1	1.2.2	1.3.0	1.3.1	2.0	2.0.1	2.0.2
MI	63.63	63.7	62.9	63.0	63.0	63.1	70.4	55.7	66.3	66.3
CC	77	77	98	98	98	106	48	149	153	153
Maintainability	0.00	0.00	0.00	0.00	0.00	0.00	0.38	0.00	0.25	0.25

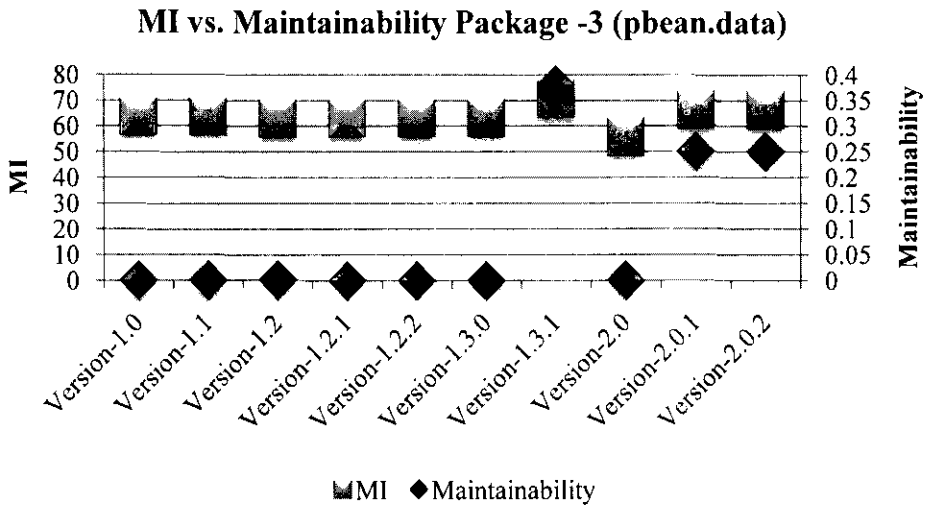


Figure 5.50 Graph plot of values of MI and maintainability package-3

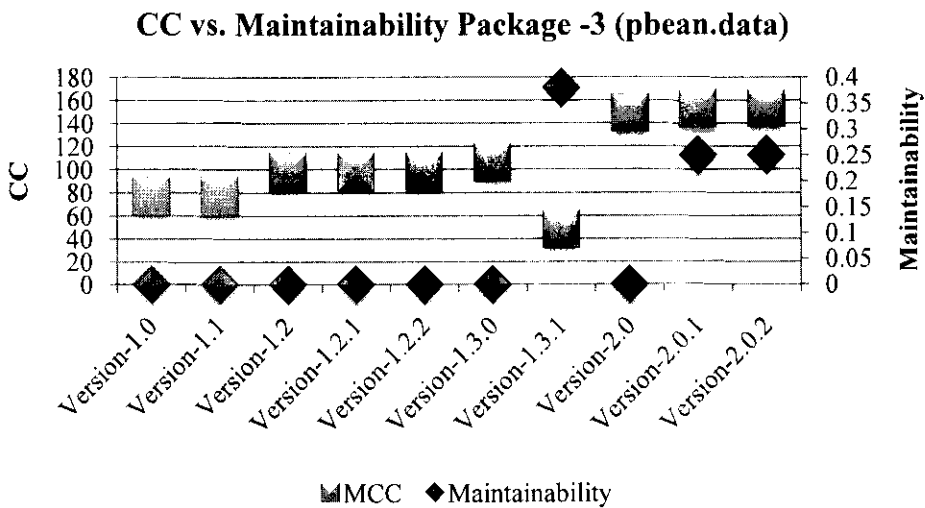


Figure 5.51 Graph plot of values of CC and maintainability package-4

5.8.2.3 Analysis of Package-4 (data.mysql)

The package data.mysql is added to the pBeans in version 2.0. Its reusability value is 0.36, which increased to 0.40 in the subsequent versions. The increase in the reusability value is due to the increase in the value of understandability. The values of reusability and attributes are presented in Table 5.25 and Figure 5.52.

Table 5.25 Version wise values of reusability and its attributes (package-4)

data.mysql	Versions		
	2.0	2.0.1	2.0.2
Understand ability	0.19	0.65	0.65
Flexibility	0.00	0.00	0.00
Portability	0.75	0.75	0.75
Variability	0.00	0.00	0.00
Maintain ability	0.88	0.63	0.63
Reusability	0.36	0.40	0.40

Reusability and Attributes Values Package -4
(data.mysql)

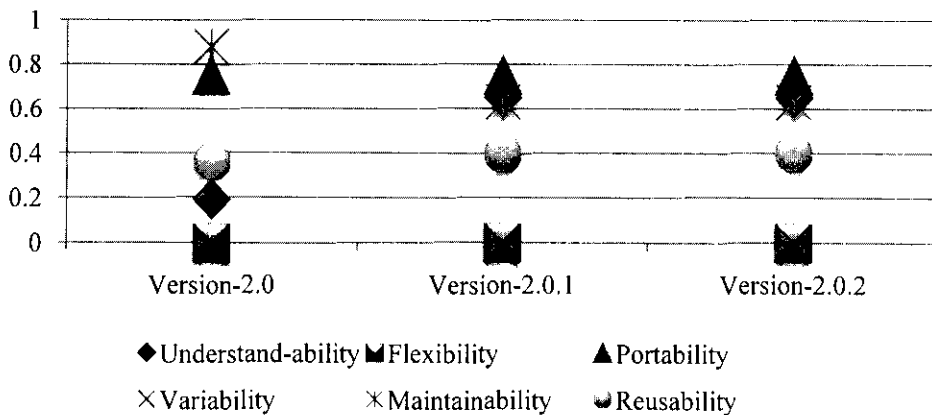


Figure 5.52 Reusability and its attribute values for package-4

The values of LOC, NOM, comments and understandability are presented in Table 5.26, Figure 5.53, Figure 5.54 and Figure 5.55. The value of understandability increased from 0.19 in version 2.0, to 0.65 in subsequent versions. The increase in the value of understandability is due to the increase in the number of comments.

Table 5.26 Version wise values of understandability and its attributes (package-4)

	Versions		
	2.0	2.0.1	2.0.2
LOC	102	102	109
NOM	15	15	15
Comments	0	100	100
Understandability	0.19	0.65	0.65

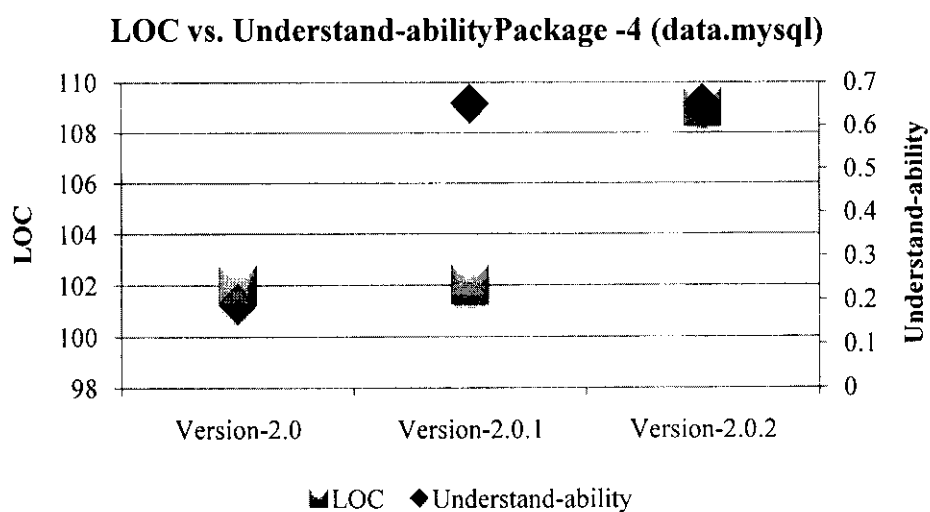


Figure 5.53 Graph plot of values of LOC and understandability package-4

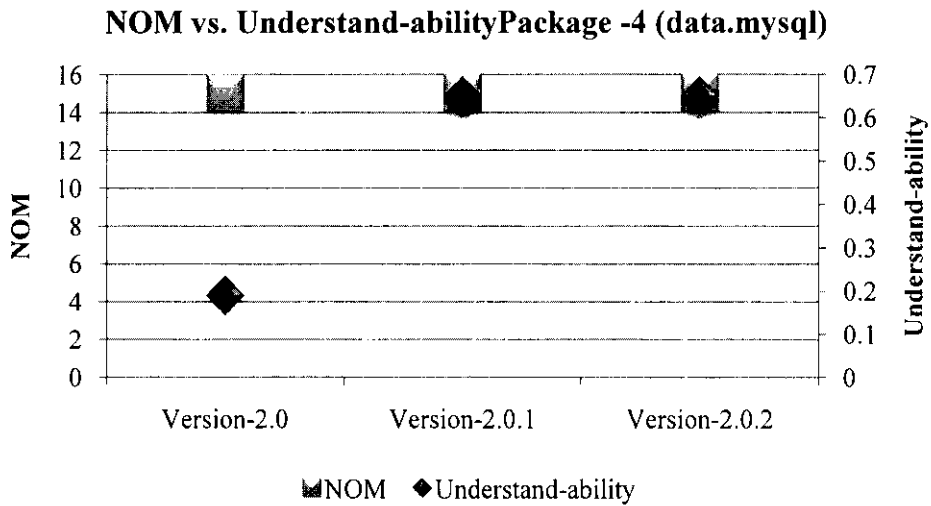


Figure 5.54 Graph plot of values of NOM and understandability package-4

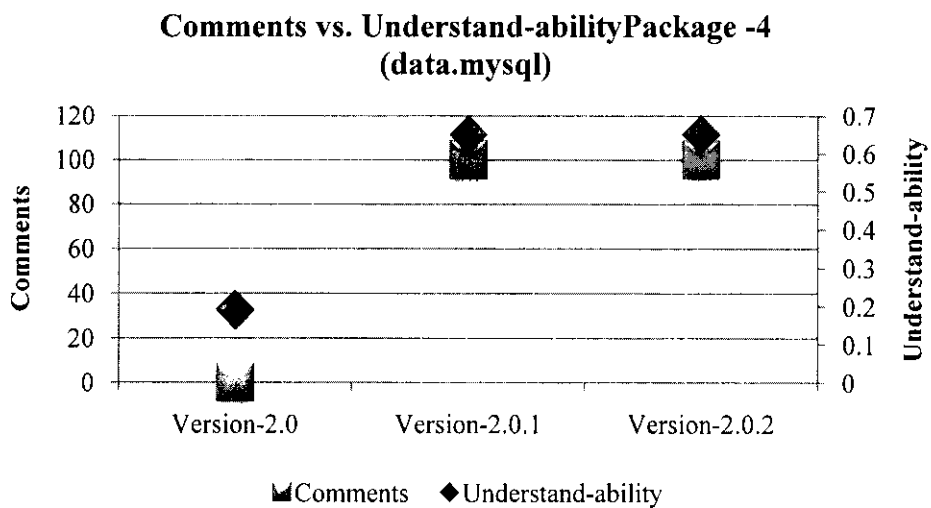


Figure 5.55 Graph plot of values of comments and understandability package-4

The values of MI, complexity and maintainability are presented in

Table 5.27, Figure 5.56 and Figure 5.57. The value of maintainability decreased in version 2.0.1. The decrease in maintainability value is due to the decrease in the value of MI. The maintainability index is directly related to maintainability.

Table 5.27 Version wise values of maintainability and its attributes (package-4)

	Versions		
	2.0	2.0.1	2.0.2
MI	134.79	80.04	80.04
CC	7	7	7
Maintainability	0.88	0.63	0.63

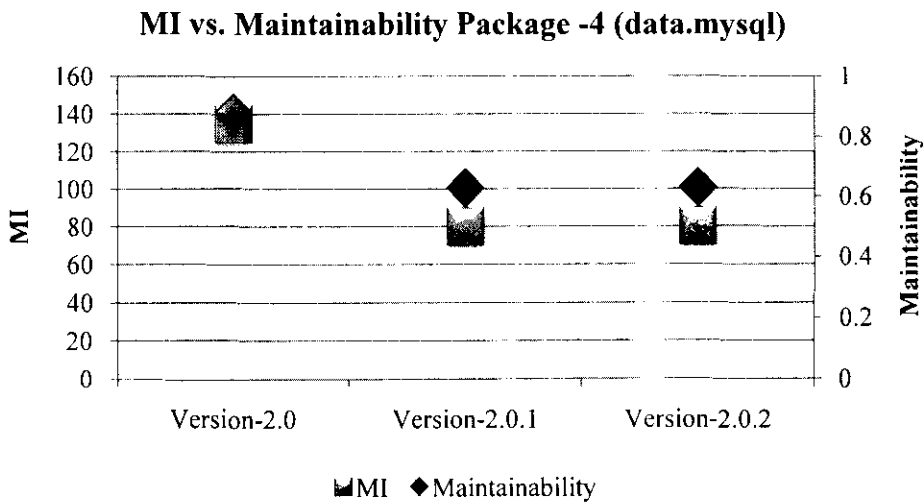


Figure 5.56 Graph plot of values of MI and maintainability package-4

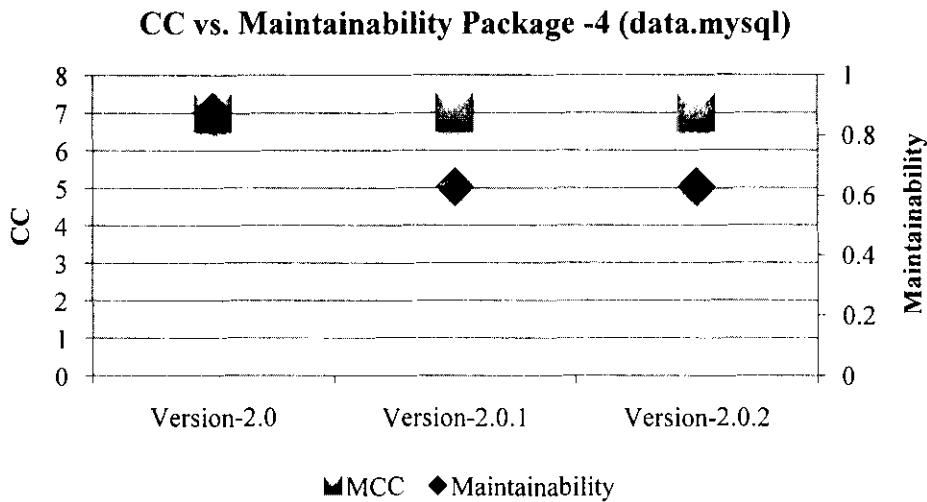


Figure 5.57 Graph plot of values of CC and maintainability package-5

5.9 Summary

The emergence of OSS has influenced the decision process of components selection. Now, while identifying components, OSS is also considered as a third choice other than developing or buying a component. The inclusion of OSS in systematic reuse based development such as SPLs is proposed by software engineering researchers. The proposals include frameworks and model of OSS based product line development. In this chapter a conceptual model of reusability assessment has been presented. The reusability attribute models are proposed by making use of GQM model.

At class level six attributes are included which are understandability, flexibility, maintainability, scope coverage, portability and variability. With the exception of variability and scope coverage, the metrics to assess these attributes are selected from the literature. While the metrics for variability and scope coverage are proposed, due their non existence in the literature. The following five attributes are considered at package level viz. flexibility, understandability, portability, maintainability and variability.

These model and metrics have been applied on 103 classes and 77 packages. The results were statistically analyzed. The analysis revealed that four out of six attributes

at class level have significant correlation with reusability. In packages all the five attributes have significant correlation with reusability. An evolutionary reusability analysis of packages of two open source software was conducted to demonstrate the potential application of the proposed approach.

CHAPTER 6

DISCUSSIONS

6.1 Overview

In this chapter the contributions are highlighted in the context of current research in this field. A discussion is presented and comparisons of the results are conducted where possible. Implications of the key findings of this study are also part of this chapter.

6.2 Key Findings of Research

In this section the key findings of this research study are presented / summarized. During the discussion relevant contemporary literature is referred to signify the importance of the findings of this study.

6.2.1 Review of Reusability Assessment Approaches

A review of reusability assessment approaches has been conducted during the study. The results are presented in chapter two. The review has been conducted to answer the following questions about the proposed approaches.

- What approaches have been introduced to assess software component reusability?
- What is the applicability of these approaches?
- What is the procedure adopted for validating the approach?

A search was made prior to starting the review process, it revealed that a similar study i.e. [163] is available in this area. However, this study was conducted in 2004. Therefore, it is not representative of the current state of the art in this area and it doesn't contain the latest approaches. The difference can be observed from the number of studies reviewed. The authors of [163], have reviewed and presented nine approaches. The number of publications reviewed and presented in this thesis is twenty. The number is fairly greater than the previous study which shows a continuous interest of the software community in this research area.

Table 6.1 Comparison of reviewed studies

[164]	This Thesis
Reviewed 9 approaches	20 approaches are reviewed

The results of the review show that the majority of the approaches are based on metrics (70%). The applicability of majority of the approaches is object-oriented paradigm (70%). The implementation language which is targeted in most of the approaches is java (71%). The intention of (60%) of the approaches is white box measurement. These figures show that the software development community is more interested in object-oriented paradigm and java based implementations.

One of the issue rose after the literature review, which pointed the lack of validation of the proposed approaches in most of the previous works. The results show that (30%) of the proposed approaches lacks the validation of results. The software research community needs to give attention to validation as it is necessary to validate results in order to gain the confidence of software practitioners.

6.2.2 Analysis of Variability Implementation Mechanisms

A theoretical analysis of variability mechanisms has been conducted and the results are presented in chapter four. In this analysis, the variability implementation mechanisms are mapped to their corresponding type, scope, artifact, and feature. A similar kind of mapping is provided in [39] in 2001. However, in this thesis the latter work of Svahnberg et al. ; [43], Kim et al. ; [41] and Pohl et al. ; [40] are also

considered. On the basis of the latest works a comprehensive mapping of variability mechanisms is provided in this thesis.

6.2.3 Identification of Challenges in OSS

In this thesis, challenges in OSS are identified through qualitative method. These challenges include finding and evaluating OSS, lack of documentation, developers' reluctance to make their software OSS. The developers do not have appropriate information about the intellectual property rights / copyrights. Another issue is the lack of adherence to the coding standards. The security of OSS is also one of the major challenges. In relation to the challenge of finding OSS, improper reviewing and comments is one of the issues that needs attention. At the organizational level there is a fear of losing market share, which poses a challenge for the OSS community.

A study, [24] has also discussed the challenges to the OSS. The authors made an argument that "there has been no systematic synthesis of the OSS challenges reported in the literature" [24]. However, their study was based on the literature survey. In the case of this thesis, the findings are based on the interviews with the experts, researchers and practitioners.

The common findings of this thesis and [24] are the challenges of finding and evaluating components, poor documentation, legal aspects such as copyrights and intellectual property rights. The findings presented in this thesis are based on the view point of user of the OSS i.e. software engineer. In this regard, the prominent findings are issues of security, fear of losing market share at the organizational level and fear of losing job at the individual software engineer's level.

Table 6.2 Comparison of findings-1

Reference	Method	Common Findings
[35]	Focus group	Security

The issue of security has more importance among the others. In a recent focus group study [35], security of component has been identified as an important technical factor that influences the selection of component. Currently, software engineering

researchers are working on empirical studies on open source and closed source software such as, [165]. This study [165] concludes that there is no significant difference in both open and closed source software development in terms of security.

Table 6.3 Comparison of findings-2

Reference	Method	Common Findings
[24]	Literature Review	<ul style="list-style-type: none"> ➤ Finding and evaluating components ➤ Poor documentation ➤ Legal aspects such as copyrights and intellectual property rights
This Thesis	Interviews	<p>Prominent Findings</p> <ul style="list-style-type: none"> ➤ Issues of security ➤ Fear of losing market share at the organizational level ➤ Fear of losing job at the individual software engineer's level

The challenge of improper reviewing and comments can be related to the challenge of several description of the same documents, which is identified in [166]. Both of these factors complicate the searching process.

Table 6.4 Comparison of findings-3

[24]	This Thesis
Several description of same documents	One of the finding is that Improper reviewing and comments is a challenge

6.2.4 Identification of Current Reuse Practices

The current reuse practices are identified in this thesis which include knowledge reuse, looking at the demonstration of software and regarding the initialization of SPL.

In [167] and [168] it is stated that the OSS developers reuse the existing code in three forms. These three forms include reuse inform of component, single line of code, algorithm / method. In this thesis, it is identified that another form of reuse is the translation of logic from one programming language to another. It is the situation when a developer searches for a specific code in a specific language but she finds the same functionality in some other language. Then she tries to translate it in the desired language. In other words creates a replica of the program in other language.

Demo versions of OSS are used to assess the effort required to modify or the number of modifications in the code. Furthermore, the modules where the changes are required can be easily identified by using the demo version.

Table 6.5 Comparison of findings-4

Reuse Practices	
[167] and [168]	This Thesis
Inform of component, single line of code, algorithm / method	<ul style="list-style-type: none"> ➤ Rewriting the code in other language with adaptation (replica in other language) ➤ Using demo version to extract knowledge ➤ Identification of module which require modifications

6.2.5 Using OSS in an SPL

The use of OSS in SPL facilitates fast transition towards automation and entering into new markets. OSS is such a platform which provides components, it attracts the SPL community. The attraction is due to the benefits of the OSS. The use of OSS in SPL improves the quality of the software. OSS is opening opportunity for the SPL community to add more innovations to their product lines.

The findings are in line with the other available studies. The improvement in quality can be credited to the fewer defects per line of code [46], and reliability [47-48] of OSS.

6.2.6 Role of OSS in Promoting Reuse

The role of OSS in promoting reuse has four dimensions. These dimensions include time and efforts saving aspect, ease of development and market trust. The market trust or in other words the trust of the customers is gained by the organization which is useful while entering into new domains by using OSS.

The results of a survey based study [52] also reports a similar finding that by reusing OSS, developer can save time for other important tasks of the project.

Table 6.6 Comparison of findings-5

Reference	Method	Common Findings
[52]	Survey	Developer can save time

6.2.7 Identification of Factors Affecting Reusability

The factors affecting reusability of software are identified using qualitative method. These factors include flexibility, maintainability, portability, scope coverage, stability, understandability, usage history, variability and documentation. The details of the factors are presented in chapter four.

The findings reported in this thesis have extended the body of knowledge by adding new attributes of reusability. A review of the proposed reusability assessment approaches is provided in chapter two. The salient feature of this study is the identification of the reusability attributes from the perspective of software developer. This identification is based on the interviews. The use of this method of inquiry is motivated by another view of reusability. This view is presented in [102], it states that reusability is a form of usability from the perspective of the software developer. Interview is one of the suitable methods in such situation.

The results acquired using the qualitative method were passed to the quantitative phase which includes a survey on the identified reusability attributes. In this survey, relative importance of attributes is identified. The results of the survey show high values for understandability, flexibility, maintainability, portability and usage history.

The following attributes received relatively less ranking; these include scope coverage, documentation, variability and stability.

In a recent focus group study [35] has also reported that security, documentation and maintainability are among the important technical factors. These factors should be taken into account while selecting OSS component.

In [51], it is recommended to consider documentation of OSS as a criterion to select a candidate OSS. The finding presented in this thesis that documentation is one of the factors which affect reusability is in line with [51].

Table 6.7 Comparison of findings-6

Reference	Method	Common Findings
[35]	Focus group	➤ Documentation ➤ Maintainability
[51]	Experience report	➤ Documentation

Stability is one of the identified factors of reusability in this thesis. In [35], the concept of stability is linked to the “ad hoc standard”. The explanation of the term ‘ad hoc standard’ is stated as “that they are used in many products of that kind” [35]. Our notion to explain this phenomenon is ‘safety in numbers’. An OSS contributed by many developers and used in many applications is more stable.

In this thesis ‘usage history’ is identified as one of the factor that influences reusability. A similar concept i.e. ‘release history’ is mentioned in [51]. The release history refers to “how often the new releases come out” [51].

Table 6.8 Comparison of findings -7

[35]	This Thesis
“ad hoc standard”	‘safety in numbers’
Maturity of community	How many examples of it are being used in software community?

Some of these factors are considered as the attributes of reusability while proposing the reusability attribute model.

Table 6.9 Comparison of findings-8

[51]	This Thesis
'release history'	'usage history'
Maturity of community is considered to gauge the maturity of OSS	How many examples of it are being used in software community?

6.2.8 Identification of Desirable Characteristics of OSS Components

In this thesis, the desirable characteristics of OSS are seen from the perspective of academics and industry. The desirable characteristics include the availability of test cases. This finding is in line with [35], where availability of test cases is considered as one of the plus points.

The maturity of OSS is identified as a desirable characteristic in this thesis. It is also identified by [35] and [51], where the maturity of a community is considered to gauge the maturity of OSS.

Table 6.10 Comparison of findings-9

Reference	Method	Common Findings
[35]	Focus group	➤ Availability of test cases
[51]	Experience report	➤ Infrastructure support

Infrastructure support is identified as one of the desirable characteristic of OSS in this thesis. This dimension of desirable characteristics is also mentioned in [51], as

one of the criteria to choose OSS. It is suggested to ask a question that “Are they (OSS) compatible with the rest of your infrastructure?”[51].

6.2.9 Proposed Reusability Assessment Model

The proposed reusability model in this study cannot be compared to the earlier studies such as [82], which is for the aspect oriented systems. Another study is [69], which is proposed for black box reuse. The studies [68] and [67] are relevant, they also assessed reusability. However, the implementation language targeted in those studies is C++. In this thesis, java based implementation is considered and ‘variability’ is identified as an attribute of reusability. Variability is not considered as an attribute of reusability at implementation level in any of the above studies.

Our work involves identifying reusability assessment metrics. Some of these are known, whereas others have been introduced by us. In some other research works, metrics are presented but not validated e.g. [73]. In our work, however, we both present the metrics and validate them empirically. In [28] and [77] reusability was assessed based on the degrees of coupling and cohesion. In comparison, our work considers these as well as other factors. Our work focuses on components written in java. The metrics that we have selected are to a certain extent dependent on java.

The list of factors affecting reusability was arrived at following the interviews with experts and survey. Next, the metrics applicable to these was decided upon. Most of these came from literature review, however, a small number were devised in this thesis. Finally, we took a number of classes / packages and assessed their reusability, the results are statistically analyzed. The proposed reusability attribute model differs from the earlier works such as [82], [69], [68] and [67], due to the inclusion of a two new attributes of reusability. These attributes are analyzed statistically and one of them i.e. ‘variability’ is correlated to reusability. A comparison of sample size used in this thesis with the earlier works is presented in section 3.14.2.

6.2.10 Statistical Results

The statistical results are presented in chapter five. The results include the correlation analysis between reusability and attribute values. The analysis was conducted at two levels. These levels include the class level and package level. At class level, the analysis was conducted to know the correlation between the metrics and the attribute which was measured using more than one metric.

At class level, 103 classes were assessed for their reusability. This assessment was based on the metrics related to the identified reusability attributes. Flexibility was assessed by CBO and LCOM. The results show that both of these metric values have a statistically significant relationship with flexibility value. CBO has a strong negative correlation i.e. $-.751$. LCOM is also negatively correlated to flexibility. However, the magnitude of the relationship is a bit weak i.e. $-.357$.

Maintainability was assessed by using MI and complexity metric. The results show statistically significant relationship between maintainability, MI and CC. The results show that MI has a strong positive correlation with maintainability. The magnitude of correlation between MI and maintainability is $.716$, while cyclomatic complexity has a negative correlation of magnitude $-.664$.

The reusability attribute understandability is based on the metrics of CBO, LCOM, comments, LOC and NOM. The metrics CBO, LCOM and NOM have statistically significant relationship with understandability. The results show that there is a negative correlation between NOM, LOC, CBO and understandability i.e. $-.701$, $-.668$, $-.529$ respectively.

The relationship between reusability and flexibility, maintainability, portability and understandability is statistically significant. The results show a strong correlation between understandability, maintainability, flexibility, and reusability values i.e. $.669$, $.797$, and $.762$. The relationship between reusability and portability is statistically significant. However, the magnitude of this relation is weak as compared to the other attributes i.e. $.404$. The two attributes variability and scope coverage has no statistically significant relationship with reusability at class level.

At package level, 77 packages were assessed for their reusability. This assessment was based on the metrics related to the identified reusability attributes. The results show statistically significant relationship between reusability and the other attributes. These attributes include understandability, flexibility, portability, maintainability and variability.

The attributes flexibility, portability, variability and maintainability have strong positive correlation with reusability. The strength of the relationship is .789, .693, .674, .667 respectively. Understandability is also positively correlated to reusability; the magnitude of the relationship is .417, which is weak as compared to the other attributes.

6.2.11 Evolutionary Analysis of Reusability

A evolutionary analysis of reusability has been conducted and results are presented in chapter five. The software on which the analysis was conducted includes Jasmin and pBeans. Six versions of Jasmin and ten versions of pBeans were analyzed. The analysis was conducted at package level.

The results of packages of Jasmin and pBeans software show an inverse relationship between LOC and understandability, and NOM and understandability. The relationship between comments and understandability is not significant, and in some cases it is overshadowed by the lines of code and number of methods.

The results regarding MI, complexity and maintainability show an inverse relationship between complexity and maintainability, however, there is a direct relationship observed between MI and maintainability.

6.3 Synthesis of the Qualitative and Quantitative Results

The results acquired during this research study are presented in chapters four and five. These results include the findings of the qualitative method i.e. interview with seven respondents. The findings of the interview include seven categories. The findings

regarding the category, factors affecting the reusability of software are carried forward into the survey. These methods are followed by statistical analysis and evolutionary reusability analysis of two software. In this section a synthesis of the results is presented.

6.3.1 Flexibility

The first factor identified as the attribute of reusability is flexibility of software. The results include the following opinions, 50% of population 'agree', 35% of population 'strongly agree', 15% of population opted for 'neither agree nor disagree', 'disagree' and 'strongly disagree'. The strength of correlation between flexibility and reusability is .762 at class level and .789 at package level.

The impact of flexibility can be seen in packages 2 and 3 of Jasmin software, where a decline in the flexibility value negatively influenced the reusability of package. The above mentioned empirical evidences point towards the importance of flexibility as an attribute of reusability in the scenario studied in this research.

6.3.2 Maintainability

The second factor identified as the attribute of reusability is maintainability of software. The survey results include the following opinions, 55% of population 'agree', 17% of population 'strongly agree', 28% of population opted for 'neither agree nor disagree', 'disagree' and 'strongly disagree'. The strength of correlation between maintainability and reusability is .797 at class level and .667 at package level.

The impact of maintainability can be seen in package 1 of Jasmin and package 3 of pBeans software, where a decline in the maintainability value negatively influenced the reusability of package. The above mentioned empirical evidences point towards the importance of maintainability as an attribute of reusability in the scenario studied in this research.

6.3.3 Portability

The third factor identified as the attribute of reusability is portability of software. The results include the following opinions, 45% of population 'agree', 29% of population 'strongly agree', 26% of population opted for 'neither agree nor disagree', 'disagree' and 'strongly disagree'. The strength of correlation between portability and reusability is .404 at class level and .693 at package level.

The impact of portability can be seen in package 3 of pBeans software, where a rise in the portability value positively influenced the reusability of package. The aforementioned empirical evidences point towards the importance of portability as an attribute of reusability in the scenario studied in this research.

6.3.4 Scope Coverage

The fourth factor identified as the attribute of reusability is scope coverage of software. The results include the following opinions, 45% of population 'agree', 11% of population 'strongly agree', 44% of population opted for 'neither agree nor disagree', 'disagree' and 'strongly disagree'. The strength of correlation between scope coverage and reusability is not significant at class level and it is not included at package level.

6.3.5 Stability

The fifth factor identified as the attribute of reusability is stability of software. The results include the following opinions, 48% of population 'agree', 19% of population 'strongly agree', 33% of population opted for 'neither agree nor disagree', 'disagree' and 'strongly disagree'. The above mentioned empirical evidences point towards the importance of stability as an attribute of reusability in the scenario studied in this research.

6.3.6 Understandability

The sixth factor identified as the attribute of reusability is understandability of software. The results include the following opinions, 53% of population 'agree', 20% of population 'strongly agree', 27% of population opted for 'neither agree nor disagree', 'disagree' and 'strongly disagree'. The strength of correlation between understandability and reusability is .669 at class level and .417 at package level.

The impact of understandability can be seen in packages 2, 3 and 4 of pBeans and package 1 of Jasmin software, where a decline in the understandability value negatively influenced the reusability of package and vice versa. The above mentioned empirical evidences point towards the importance of understandability as an attribute of reusability in the scenario studied in this research.

6.3.7 Usage History

The seventh factor identified as the attribute of reusability is usage history of software. The results include the following opinions, 46% of population 'agree', 29% of population 'strongly agree', 25% of population opted for 'neither agree nor disagree', 'disagree' and 'strongly disagree'. The above mentioned empirical evidences points towards the importance of usage history as an attribute of reusability in the scenario studied in this research.

6.3.8 Variability

The eighth factor identified as the attribute of reusability is variability of software. The results include the following opinions, 48% of population 'agree', 11% of population 'strongly agree', 41% of population opted for 'neither agree nor disagree', 'disagree' and 'strongly disagree'. The strength of correlation between variability and reusability is not significant at class level. However, its value is .674 at package level, which is significant.

The impact of variability can be seen in packages 2 and 3 of pBeans software, where a decline in the variability value negatively influenced the reusability of package. The aforementioned empirical evidences point towards the importance of variability as an attribute of reusability in the scenario studied in this research.

6.3.9 Documentation

The ninth factor identified as the attribute of reusability is documentation of software. The results include the following opinions, 31% of population 'agree', 31% of population 'strongly agree', 38% of population opted for 'neither agree nor disagree', 'disagree' and 'strongly disagree'. The above mentioned empirical evidences point towards the importance of documentation as an attribute of reusability in the scenario studied in this research.

6.4 Key Findings and Implications

The contributions of this research can be divided into three types. These include review of literature, contributions related to the methodology and practical contribution.

6.4.1 Review of Literature

Two reviews of the literature have been conducted in this research study. These reviews include the followings:

- A review of reusability assessment approaches
- An analysis of object-oriented variability mechanisms

6.4.1.1 Review of Reusability Assessment Approaches

In this section, the key findings of the review and their implications/recommendations are presented.

- A majority of the approaches available in the literature are applicable to object oriented paradigm.

This majority shows a research trend in object-oriented software development. On the other hand, it is a call for researchers to explore other programming paradigms such as aspect-oriented and feature-oriented.

- A large number of reusability assessment approaches are meant for java based implementation.

The object-oriented researchers working in other object-oriented languages such as C#, python and .net, may replicate the available approaches for other object-oriented languages.

- There is a lack of validation of approaches available in the literature. The results of some of the studies have not been validated.

Researchers should not neglect the validation of newly proposed approaches. This validation acts as proof of the usefulness of the approach. It is a way to get the confidence of the potential user. Therefore, the newly proposed approaches should be validated.

- A majority of approaches are white box.

Open source is prevailing within the software development community. This shows the trend of component-based software development is towards using open source software components with an intention to manipulate source code.

6.4.1.2 Variability Implementation Mechanism Analysis

An analysis of object-oriented variability mechanisms has been conducted to update the body of knowledge regarding the variability mechanisms. The key findings and their implications are presented as follows.

- The variability implementation mechanisms; aspect-oriented and overloading are capable of handling most of the variability types.

The two variability implementation mechanisms i.e. aspect-oriented and overloading can handle most of the variability types of different scopes.

In our opinion, this work will be helpful for the software development community in identifying/selecting an appropriate variability handling mechanism.

6.4.2 Methodological Contribution

The methodological contributions and their implications are as follows.

- Mixed Method (Qualitative and Quantitative)

The qualitative method has been helpful in exploring the phenomenon. The results were used to develop the quantitative method and triangulation of results.

This study can be considered as an example in software engineering studies. In future, new phenomenon can be explored to generate hypotheses and to test them by employing the methodology used in this research.

- Adaptation in qualitative analysis

The adaptation (use of word cloud) of coding process may be used in other studies where textual data is analysed. In this research, the word cloud is used after the open coding to aid it by ensuring that none of the recurring word related to a concept is missed. However, this technique can be used to pilot the open coding process especially when large number of textual data is processed.

Word cloud can be more useful in analysis of transcripts of unstructured interviews. The adaptation of coding process may be used in the content analysis studies.

The examples of using word cloud can be seen on web / search engines, news sites. However, to our knowledge word cloud is used for the first time in academics / qualitative analysis process in this thesis as extension of [112].

6.4.3 Practical Contribution

The practical contributions of this study are presented in this section along with their implications and recommendations.

6.4.3.1 Challenges to OSS

The following challenges are identified in this study.

- Finding and evaluating OSS

This finding implies that there is a need of search engines to search (find) open source software.

The evaluation of OSS has different facets such as risk, legal aspects, functional compliance etc. One of the evaluation aspects (reusability) has been explored in this research. However, the others are still to be explored.

The searching (finding) of OSS can be improved by implanting a standard cataloguing system, like the one implemented for the books/ libraries.

- Lack of documentation

Documentation of OSS has a key role in understanding the OSS for its appropriate usage. The lack of documentation can be handled by providing documentation with the OSS and encouraging developers to contribute to documentation as well.

- Developers are reluctant to make their software OSS

This finding implies one way interaction of developers to the OSS community i.e. only using the OSS. The contribution by the community to the OSS is more important. So, developers using OSS to develop application should be encouraged to contribute to the OSS.

- Lack of information and awareness about intellectual property rights

There should be awareness among software developers about the intellectual property rights. This information could be disseminated by arranging seminars in organizations and universities. It will improve the situation and may result in increased contribution to the OSS.

- Lack of adherence to coding standards

Software developers should adhere to the coding standards. It will ease the job of others to reuse the software.

- Security of OSS

This finding points towards the need to invest more research efforts to investigate the security issues in OSS. A comprehensive knowledge on security of OSS may help to categorize the security threats and improve the security measures in OSS.

- Improper comments of reviewers about the OSS

This finding is a call to improve the standard of comments and reviews. This can be improved by providing structured reviews and comments rather than just providing a text box. A rating scheme may also be adopted. There is need to put research efforts in this area by including the social scientists to improve the comments and user feedback, which will be useful for the potential users.

6.4.3.2 *Current Reuse Practices*

The key findings and implications/recommendations about the reuse practices are as follows:

- There is form of reuse i.e. ‘knowledge reuse’; where the component itself is not used. However, it is translated into desired programming language.

Effort should be made to improve the ‘knowledge reuse’. It will lead to inter language / inter domain reuse.

- Demonstration (Demo) of OSS plays an important role in selection of software.

This finding implies that availability of demo version of the OSS provides help to potential user in making a decision. Therefore, the OSS developers should provide demo version to the customers.

- SPLs are not started from scratch. Therefore, OSS is helpful in starting a product line.

It implies that OSS has a potential to be a platform for providing support to start new product lines / products.

6.4.3.3 *Desirable Characteristics of OSS*

The key findings and implications/recommendations about the desirable characteristics of OSS are as follows:

- There are two perspectives in which the desirable characteristics of OSS can be seen (i) academic (ii) industry. In academic settings, novelty of ideas and functionality is more important. On the other hand, in commercial settings risk assessment is considered more important.

The OSS should be viewed either in the context of academics or industry. The reason is the criteria of evaluation differ in different settings.

- Maintenance support is a desirable characteristic of OSS

The concerns regarding the maintenance of the OSS should be addressed by the OSS community.

- Maintenance agreement is also desirable by the customer

There is need to improve the infrastructure support of OSS. This calls for the OSS research community to invest more efforts in identifying and resolving the infrastructure issues related to OSS.

- Maturity of OSS is also seen as a desirable characteristic of OSS

The maturity of OSS is represented by its usage history. So, a proper usage history of OSS should be maintained.

- The error handling mechanism provided by the OSS component is also considered while making a decision.

Efficient error handling mechanism should be incorporated in the OSS.

- Scalability of OSS is also an important characteristic.

This finding implies that scalability of software should be kept in view while developing the software. Scalable software is more likely to be used.

6.4.3.4 Suggestions

The followings are the key findings and implications/recommendations drawn on the basis of suggestions made by the respondents.

- There is need to develop techniques and tools for inter language reuse.

This suggestion is call for more research in developing tools for inter language reuse, which will result in enormous increase in reuse.

- Development of software agents to help the developers in reusing the OSS.

It is a call to invest research effort in the development of software agents to facilitate the software developers. Such agents will ease the software development especially in the context of OSS.

6.4.3.5 Factors Affecting Reusability

The recommendations/implications regarding the factors affecting reusability are as follows.

- The factors affecting reusability are flexibility, understand ability, maintainability, portability, scope-coverage, stability, usage history, variability and documentation.

The identified factors are useful for the software developers to develop reusable software. On the other hand, these could be used while evaluating an OSS.

- The surveys results show a high ranking for understand ability, flexibility, maintainability and usage history.

These factors have more importance amongst the others. So, while prioritizing the factors these should be considered first.

The following conclusions are made on the basis of statistical analysis at class level.

- The metrics LOC and NOM are positively correlated to understand ability. However, comments don't have a significant relationship with understand ability.
- During the evolutionary reusability analysis (experiment 3), it is observed that there is a disproportionate increase in number of comments and size of the software which resulted into an insignificant correlation.

There is a need to improve programming practices by including comments in codes. It will increase the understand ability of codes.

- It is concluded on the basis of the statistical analysis at class level that flexibility, understand ability, and maintainability have a strong positive correlation with reusability.

While assessing the reusability of a class these factors should be considered.

The following conclusions are made on the basis of statistical analysis at package level.

- At package level understand ability, flexibility, portability, variability and maintainability has strong positive correlation with reusability.

The finding related to the correlation of these attribute is helpful to assess the reusability of OSS and the newly added attribute i.e. variability since it has a key role in reuse intensive software environments. Its inclusion makes this approach useful in such software development environment.

- The effect of attributes on reusability can be seen in different versions of different packages.

The application of the proposed approach on two OSS and the results imply that the approach is helpful to assess the reusability of OSS.

6.5 Summary

This chapter highlights the key findings of this study which comes under the three main contributions; literature review, methodological contribution and practical contributions. The findings of comprehensive literature reviews conducted on the reusability assessment approaches, variability implementation mechanism and aspect-oriented implementation of software product lines are discussed.

The practical contributions include seven categories and their 39 dimensions. A comparison of these findings with the available contemporary studies is conducted in this chapter. The comparative results gave confidence in the findings. A synthesis of qualitative and quantitative results is presented to exhibit the triangulation. The key findings and their implications are also discussed in this chapter.

CHAPTER 7

CONCLUSIONS

Nature's music is never over; her silences are pauses, not conclusions.

(Mary Webb, 1881-1927)

7.1 Research Summary

The following research questions have been raised in this research:

RQ1 – How reuse of open source software has been practiced in reuse intensive software development (SPLE and CBSE)?

RQ2 – What are the factors affecting reusability of open source software in a reuse intensive software development?

RQ3 – How to measure the factors affecting the reusability?

RQ4 – What is the significance and correlation among the identified reusability attributes?

The first question which is about the practice of reusing OSS in reuse intensive software development is associated with the objective 'to explore the use of OSS in reuse intensive software development'. As the nature of the question and objective is exploratory, it needs to be addressed by employing an exploratory research method in answering the research question in order to achieve this objective. In this study, interview is used as method of research for this purpose. The study, being an exploratory one, resulted in ideas, hypotheses, issues, challenges and future directions. These findings include challenges in using OSS from the perspective of consumer of the software (software engineer), the current practice of reusing the OSS in reuse intensive software development, and the prospects of using OSS in reuse intensive software development with a focus on SPL. The role of OSS in promotion of software reuse is explored and the desirable characteristics of OSS are elaborated.

The second question is related to the factors which affect the reusability of OSS. Some of the previous studies have identified these factors for CBSE. However, in this study SPL has been brought into the picture along with CBSE. It is again an exploratory question due to the fact that discussion on the factors affecting the reusability of OSS, specifically code assets in the context of SPL, is not available in the literature. The interviews have helped to lay down the basis to answer this question. After identifying the factors, a survey was conducted to know the relative importance of the factors.

The third research question is about the measurement of factors. These measures are identified by a literature search. A set of well established metrics are identified. The metrics for two of the factors are proposed in this study due to their non existence in the literature. The newly added factor 'variability' in context of the implementation mechanisms has been analyzed and discussed in detail. This analysis has helped in reaching to the appropriate measures of variability.

The fourth question is related to the analysis and validation of the identified factors and measures. This part of the study was completed using statistical analysis and evolutionary reusability analysis. The results of this study have been presented in chapter 4 and 5. These results provide answers to the research questions raised in this study and show the achievement of the research objectives of this study.

7.2 Achievement of Research Objectives

In this section the achievement of objectives of this research is discussed. The first objective is:

To explore the use of OSS in reuse intensive software development.

This objective include some sub-objectives such as *identification of challenges in OSS, identification of current reuse practices, identification of practices related to the use of OSS in SPL, exploration of role of OSS in promotion of reuse, identification of desirable characteristics of OSS.*

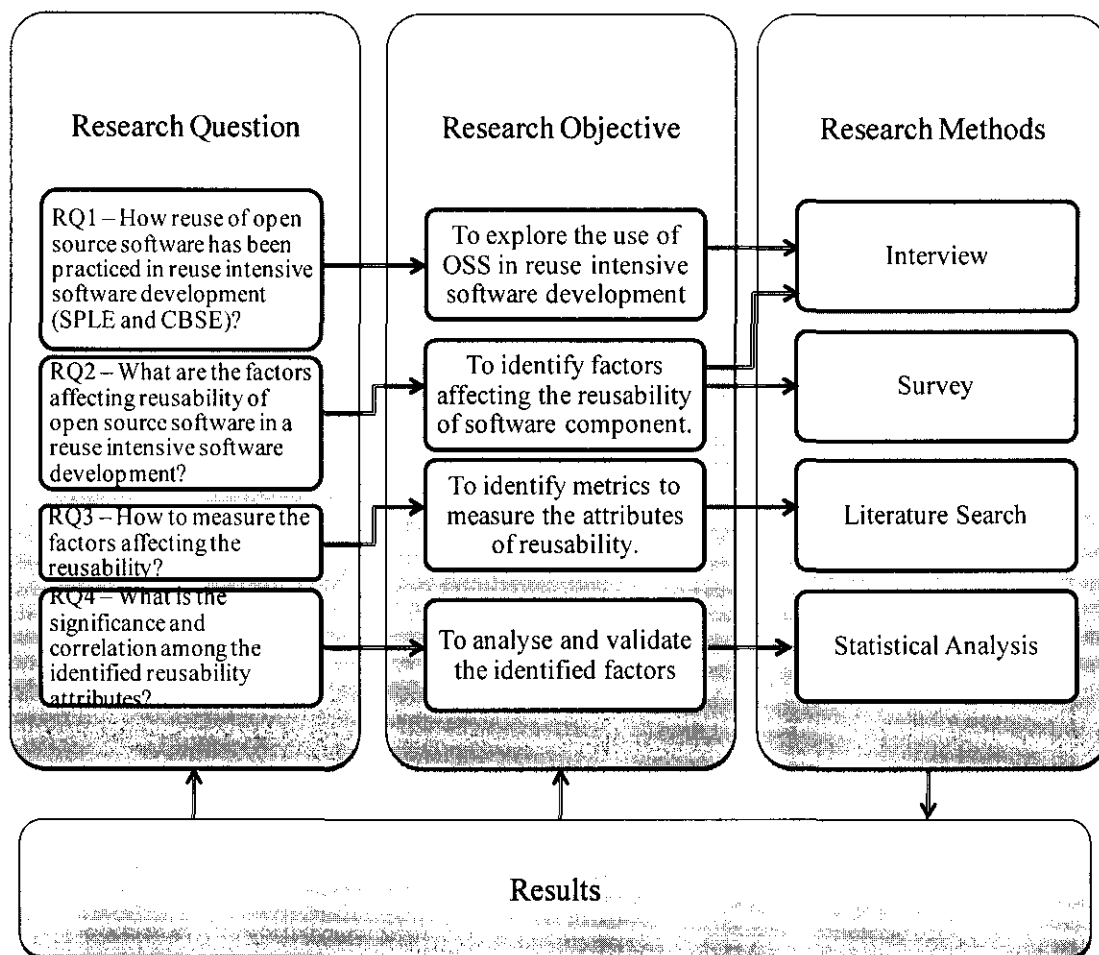


Figure 7.1 Summary of the research work

The nature of the objective and its relevant question is exploratory; keeping the exploratory research methods in view, literature search and interviews [107] are used. The literature contains information regarding the use of OSS in CBSD. However, the use of OSS in SPLs is recently proposed by the researchers. So, the opinions of the informants / respondents were collected through interviews. These respondents were selected carefully on the basis of their knowledge and expertise in this area. The analysis of the qualitative data gathered through interviews was performed using content analysis approach [112]. The details on the analysis and results are presented in chapter 4. The results are in the form of categories and sub-categories, which provide an insight into the different dimensions of the categories. The findings are contribution towards the body of knowledge and results in meeting the objective.

The second objective in this context is:

To identify factors affecting the reusability of software component.

A sum of nine factors has been identified using the qualitative method (interview). Prior to the interviews, a review of the literature was conducted to report the state of the art in reusability assessment. The results are presented in chapter 2. After identifying the factors, a survey was conducted to know their importance. This use of mixed research methods can be seen as ‘partially mixed sequential dominant status’ [92], where the methods are mixed in a sequence and the qualitative phase is dominant. The mixing serves the purpose of ‘triangulation’ [96], while the survey results help to increase the validity of findings regarding the factors of reusability. The second purpose served by the mixing is ‘development’ [96]; the results of qualitative method are used to develop the basis of the quantitative method.

The third objective is:

To identify metrics to measure the attributes of reusability.

The reusability attribute model (at class level and package level) has been proposed and presented in chapter 5. The factors identified in chapter 4 are considered as reusability attributes to form the model. The next step is the measurement of these attributes. So, to measure the attributes metrics were used, since our concern is the measurement of these attributes at code level. Suitable code metrics were identified from the literature. These metrics are well established and have been used in other studies as well.

The fourth objective is:

To analyse and validate the identified factors.

Once the metrics have been identified and associated to the attributes, an analysis was carried out to analyze and validate the attributes. The metrics were calculated (application of reusability attribute model) at class level on 103 classes, and at package level on 77 open source packages. The results are presented in chapter 5. The correlations were determined using Pearson’s correlation analysis. These analyses were performed at attribute level and at sub-attribute level. Following that, in order to demonstrate the application of the proposed assessment approach, it was applied on

two open source software which have multiple versions. So, an evolutionary analysis of reusability has been conducted to see the effects of attributes on reusability during the evolution of software.

A synthesis of the qualitative and quantitative results is presented in chapter 6. On the basis of the results, it can be safely said that the objectives of this study have been met in the context of this thesis. A summary of findings is presented in Figure 7.2.

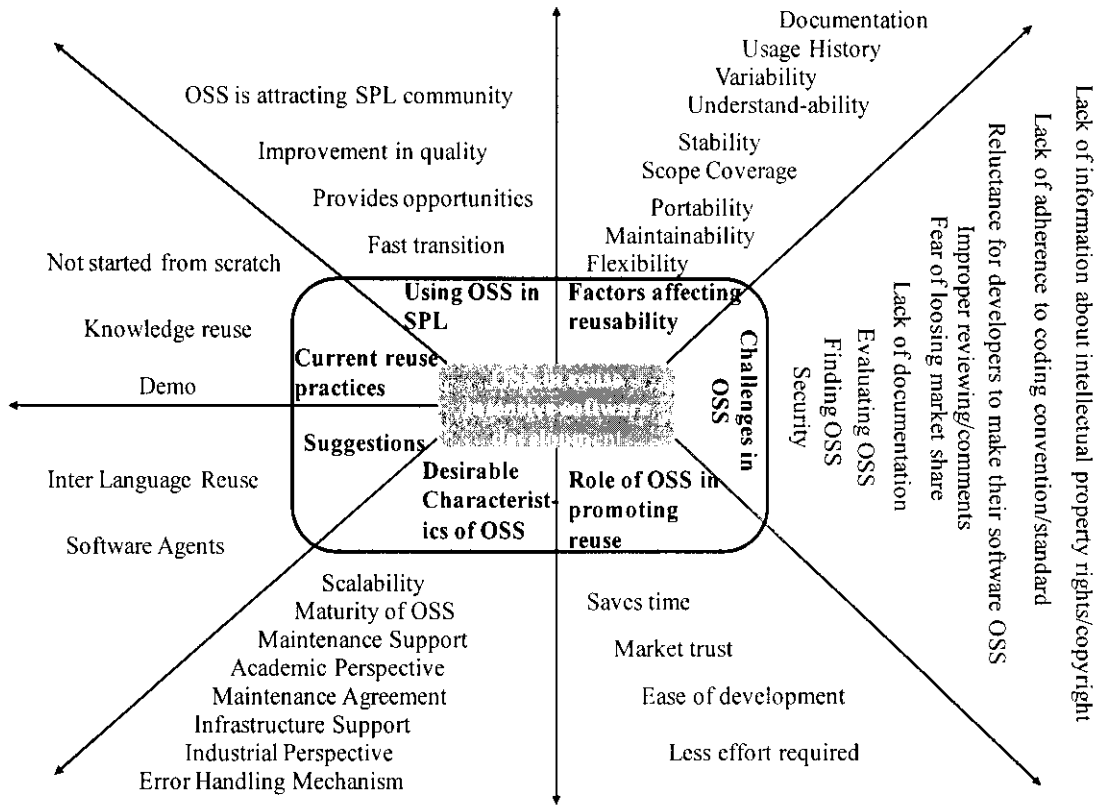


Figure 7.2 Summary of findings

The categories and dimensions presented in chapter four, as part of the qualitative findings, are compared with contemporary literature in the discussions chapter. These studies include [166], [35], [24], [51], and [52]; and the findings are extended by including the results of [165]. It is observed that none of the finding is contradicting the available literature. This fact validates the findings of this research. Furthermore, new categories and dimensions have been identified which will be helpful to expand the body of knowledge in this area. The implications of key findings of this study are presented in chapter six.

7.3 Contributions

The main contribution of this study is the results of exploration of the phenomenon of reusing OSS in reuse intensive software development and proposal for reusability attribute model. The model is outcome of two empirical methods (interview and survey). The proposed reusability attribute model is presented in chapter 5 at class level and package level. The factors of reusability are quantified using well established software metrics. However, metrics for two attributes namely variability and scope coverage are newly defined in this study due to their non existence in literature. The phenomenon of variability is extensively analyzed from the view point of implementation mechanisms. The proposed reusability attribute model is applied at the level of class and package in chapter 5. The results obtained by applying metrics are statistically analyzed to have a deep understanding about the relationship of attributes and reusability. Multiple versions of two open source software are analyzed to assess and observe their reusability during evolution. The results of these analyses are discussed under the light of earlier qualitative study and the studies available in literature.

Other contributions include; review of literature, methodological contribution and practical contributions. The reviews of literature consist of two major reviews; (i) review of reusability assessment approaches and (ii) analysis of variability implementation mechanism. On part of methodology the study has demonstrated the use of mixed methodology. The content analysis approach is adapted by using word cloud in open coding process. The practical contributions include seven categories and 39 dimensions.

7.4 Limitations

The approach presented in this thesis is meant to be used by the users i.e. (software engineers). The approach is applied on open source software to obtain the results. The findings are specific to the open source projects. The results are acquired by analyzing the source code. Therefore, results may not comply with the black-box reuse i.e. when the user has no access to the source code of the project.

The metrics used to assess the attributes are generic object oriented metrics. However, the data set used in the statistical experiment consists of projects implemented in java. Therefore, results using some other programming language may differ from the results of this study.

7.5 Future Directions

In this study, it is explored that how the reuse of OSS is practiced in reuse intensive software development. In future, studies may be carried out by including the development of software ecosystems and OSS. There is room to further research on the issues of variability, its implementation mechanisms, and variability of software artifacts other than source code.

The findings of the qualitative method which include seven categories and 39 dimensions are open for exploration and confirmation. All the categories and dimensions identified in this study can be seen as the future directions. The methodology followed in this thesis; to work on one of the identified dimension (factors affecting reusability and variability) can be used as basis in future work on the other dimensions.

REFERENCES

- [1] Fernández, M., *Models of Computation: An Introduction to Computability Theory*: Springer, 2009.
- [2] Simons, C. L., *et al.*, "35 years on: To what extent has software engineering design achieved its goals?," *IEE Proceedings: Software*, vol. 150, pp. 337-350, 2003.
- [3] IEEE, "Systems and Software Engineering Vocabulary ISO/IEC/IEEE 24765:2010," ed, 2010, pp. 1-418.
- [4] Krueger, C. W., "Software reuse," *ACM Computing Surveys*, vol. 24, pp. 131-183, 1992.
- [5] Prieto-Diaz, R., "Status report: software reusability," *IEEE Software*, vol. 10, pp. 61-66, 1993.
- [6] Frakes, W. and Terry, C., "Software reuse: metrics and models," *ACM Computing Surveys*, vol. 28, pp. 415-435, 1996.
- [7] Sametinger, J., *Software engineering with reusable components*: Springer-Verlag New York, Inc., 1997.
- [8] Frakes, W. B. and Kyo, K., "Software reuse research: status and future," *IEEE Transactions on Software Engineering*, vol. 31, pp. 529-536, 2005.
- [9] Frakes, W. B. and Succi, G., "An industrial study of reuse, quality, and productivity," *Journal of Systems and Software*, vol. 57, pp. 99-106, 2001.
- [10] Mohagheghi, P. and Conradi, R., "Quality, productivity and economic benefits of software reuse: a review of industrial studies," *Empirical Software Engineering*, vol. 12, pp. 471-516, 2007.
- [11] Niemi, T., *et al.*, "Server-Based Computing Solution Based on Open Source Software," *Information Systems Management*, vol. 26, pp. 77-86, 2009.
- [12] Stafford, J. (2006, 01 June). *Time to plan your company's escape from Microsoft*. Available: <http://searchenterpriselinux.techtarget.com/news/1163576/2006-Time-to-plan-your-companys-escape-from-Microsoft>

- [13] Kenwood, C. A., "A Business Case Study of Open Source Software," The MITRE Corporation 2001.
- [14] Krishnamurthy, S., "A Managerial Overview of Open Source Software," *Business Horizons*, vol. September-October 2003, 2003.
- [15] Linden, F. v. d., *et al.*, "Commodification of Industrial Software: A Case for Open Source," *IEEE Software*, vol. 26, pp. 77-83, 2009.
- [16] Wheeler, D. A. (2005, 01 June). *Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers!* Available: http://www.dwheeler.com/oss_fs_why.html
- [17] Howe, C., "Open Source Cracks The Code," Forrester Research 2000.
- [18] Ågerfalk, P. J., *et al.*, "Assessing the Role of Open Source Software in the European Secondary Software Sector: A Voice from Industry," presented at the First International Conference on Open Source Systems, Genova, 2005.
- [19] Hummel, O., *et al.*, "Code Conjurer: Pulling Reusable Software out of Thin Air," *IEEE Software*, vol. 25, pp. 45-52, 2008.
- [20] Wasserman, A., "How the Internet transformed the software industry," *Journal of Internet Services and Applications*, vol. 2, pp. 11-22, 2011.
- [21] Sommerville, I., *Software Engineering*, 8th ed.: Addison-Wesley, 2007.
- [22] Ågerfalk, P., *et al.*, "Open Source in Software Product Line: An Inevitable Trajectory," in *10th International Software Product Line Conference (SPLC '06)*, 2006.
- [23] Ahmed, F., *et al.*, "A Model of Open Source Software-Based Product Line Development," in *Computer Software and Applications, 2008. COMPSAC '08. 32nd Annual IEEE International*, 2008, pp. 1215 -1220.
- [24] Stol, K. J. and Babar, M. A., "Challenges in using open source software in product development: a review of the literature," presented at the Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development, Cape Town, South Africa, 2010.
- [25] Samadi, S., *et al.*, "Strategies for enabling software reuse within the Earth Science Community," in *Proceedings of IEEE International Geoscience and Remote Sensing Symposium, 2004. IGARSS '04.* , 2004, pp. 2196-2199 vol.3.

- [26] Linden, F., *et al.*, *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*: Springer-Verlag Berlin Heidelberg, 2007.
- [27] Boxall, M. A. S. and Araban, S., "Interface Metrics for Reusability Analysis of Components," presented at the Proceedings of the 2004 Australian Software Engineering Conference, 2004.
- [28] Gui, G. and Scott, P. D., "Ranking reusability of software components using coupling metrics," *Journal of Systems and Software*, vol. 80, pp. 1450-1459, 2007.
- [29] Gui, G. and Scott, P. D., "New Coupling and Cohesion Metrics for Evaluation of Software Component Reusability," in *The 9th International Conference for Young Computer Scientists, 2008. ICYCS 2008.* , 2008, pp. 1181-1186.
- [30] Yoonjung, C., *et al.*, "Practical S/W Component Quality Evaluation Model," in *10th International Conference on Advanced Communication Technology, 2008. ICACT 2008.* , 2008, pp. 259-264.
- [31] Sharma, A., *et al.*, "Reusability assessment for software components," *SIGSOFT Softw. Eng. Notes*, vol. 34, pp. 1-6, 2009.
- [32] Harman, M., "Why Source Code Analysis and Manipulation Will Always be Important," in *Source Code Analysis and Manipulation (SCAM), 2010 10th IEEE Working Conference on*, 2010, pp. 7-19.
- [33] Burgin, M., *et al.*, "Software technological roles, usability, and reusability," in *Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration, IRI 2004*, 2004, pp. 210-214.
- [34] Mellarkod, V., *et al.*, "A multi-level analysis of factors affecting software developers' intention to reuse software assets: An empirical investigation," *Information & Management*, vol. 44, pp. 613-625, 2007.
- [35] Höst, M., *et al.*, "Usage of Open Source in Commercial Software Product Development – Findings from a Focus Group Meeting," in *Product-Focused Software Process Improvement*. vol. 6759, D. Caivano, *et al.*, Eds., ed: Springer Berlin / Heidelberg, 2011, pp. 143-155.

- [36] Firesmith, D., "Common Concepts Underlying Safety, Security, and Survivability Engineering," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA CMU/SEI-2003-TN-033, 2003.
- [37] Brummermann, H., *et al.*, "Variability issues in the evolution of information system ecosystems," presented at the Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems, Namur, Belgium, 2011.
- [38] Chen, L. and Ali Babar, M., "A systematic review of evaluation of variability management approaches in software product lines," *Information and Software Technology*, vol. 53, pp. 344-362, 2011.
- [39] Gacek, C. and Anastasopoulos, M., "Implementing product line variabilities," *SIGSOFT Softw. Eng. Notes*, vol. 26, pp. 109-117, 2001.
- [40] Pohl, C., *et al.*, "Survey of existing implementation techniques with respect to their support for the practices currently in use at industrial partners," AMPLE Project deliverableD3.1, 2007.
- [41] Kim, S. D., *et al.*, "A theoretical foundation of variability in component-based development," *Information and Software Technology*, vol. 47, pp. 663-673, 2005.
- [42] Sharp, D. C., "Containing and facilitating change via object oriented tailoring techniques," in *First Software Product Line Conference*, Denver, Colorado, 2000.
- [43] Svahnberg, M., *et al.*, "A taxonomy of variability realization techniques: Research Articles," *Software: Practice and Experience*, vol. 35, pp. 705-754, 2005.
- [44] OSI. (2011, 31st May). *Open Source Definition* Available: <http://www.opensource.org/docs/osd>
- [45] Sohn, S. Y. and Mok, M. S., "A strategic analysis for successful open source software utilization based on a structural equation model," *Journal of Systems and Software*, vol. 81, pp. 1014-1024, 2008.
- [46] Joode, R. v. W. d., *et al.*, "Rethinking free, libre and open source software.," *Knowledge, Technology & Policy*, vol. 18, pp. 5-16, 2006.
- [47] Forge, S., "The rain forest and the rock garden: the economic impacts of open source software," *info*, vol. 8, pp. 12-31, 2006.

- [48] Varian, H. R. and Shapiro, C., "Linux adoption in the public sector: An economic analysis (Technical Report)," UC Berkeley2003.
- [49] Giera, J., "The Costs And Risks Of Open Source: Debunking The Myths," Forrester Research2004.
- [50] Webb, M. (2001, 01 June). *Going With Open Source Software: Is it the right choice for your organization?* Available: <http://www.techsoup.org/learningcenter/software/archives/page9905.cfm>
- [51] Spinellis, D., "Choosing and Using Open Source Components," *IEEE Software*, vol. 28, pp. 96-96, 2011.
- [52] Sojer, M. and Henkel, J., "Code Reuse in Open Source Software Development: Quantitative Evidence, Drivers, and Impediments," *Journal of the Association for Information Systems*, vol. 11, pp. 868-901, 2010.
- [53] Jones, C., *Applied Software Measurement*, 3rd ed.: McGraw-Hill, 2008.
- [54] Horstmann, C., *Big Java*, 4th ed.: John Wiley & Sons, Inc., 2010.
- [55] Heinemann, L., *et al.*, "On the Extent and Nature of Software Reuse in Open Source Java Projects," in *Top Productivity through Software Reuse*. vol. 6727, K. Schmid, Ed., ed: Springer Berlin / Heidelberg, 2011, pp. 207-222.
- [56] Greanier, T., *Java Foundations*: SYBEX Inc., 2004.
- [57] Niemeyer, P. and Knudsen, J., *Learning Java* 3rd ed.: O' Reilly Media, Inc., 2005.
- [58] Buyya, R., *et al.*, *Object Oriented Programming with Java* McGraw Hill, 2009.
- [59] Jain, P. M., *The Class of Java*: Pearson Education, 2011.
- [60] Clements, P. and Northrop, L., *Software product lines: practices and patterns*: Addison-Wesley Longman Publishing Co., Inc., 2001.
- [61] Pohl, K., *et al.*, *Software Product Line Engineering Foundations, Principles, and Techniques*: Springer-Verlag Berlin Heidelberg, 2005.
- [62] SEI. (2011, 31st July). *A Framework for Software Product Line Practice, Version 5.0*.
- [63] Bachman, F., *et al.*, "Volume II: Technical Concepts of Component-Based Software Engineering," Software Engineering Institute2000.

- [64] Capretz, L. F., *et al.*, "COTS-based software product line development," *International Journal of Web Information Systems*, vol. 4, pp. 165 - 180, 2008.
- [65] Brereton, P., *et al.*, "Lessons from applying the systematic literature review process within the software engineering domain," *Journal of Systems and Software*, vol. 80, pp. 571-583, 2007.
- [66] Gómez, O., *et al.*, "A Systematic Review Measurement in Software Engineering: State-of-the-Art in Measures," in *Software and Data Technologies*. vol. 10, ed: Springer Berlin Heidelberg, 2008, pp. 165-176.
- [67] Etzkorn, L. H., *et al.*, "Automated reusability quality analysis of OO legacy software," *Information and Software Technology*, vol. 43, pp. 295-308, 2001.
- [68] Dandashi, F., "A method for assessing the reusability of object-oriented code using a validated set of automated measurements," presented at the Proceedings of the 2002 ACM symposium on Applied computing, Madrid, Spain, 2002.
- [69] Washizaki, H., *et al.*, "A Metrics Suite for Measuring Reusability of Software Components," presented at the Proceedings of the 9th International Symposium on Software Metrics, 2003.
- [70] C. Sant'anna, A. G., C. Chavez, C. Lucena, and A. v. von Staa, "On the reuse and maintenance of aspect-oriented software: An assessment framework," presented at the Proceedings XVII Brazilian Symposium on Software Engineering, 2003.
- [71] Kim, S. D. and Park, J. H., "C-QM: A Practical Quality Model for Evaluating COTS Components," in *IASTED International Conference on Software Engineering*, Innsbruck, Austria, 2003, pp. 991-996.
- [72] Kalaimagal, S. and Srinivasan, R., "Q' FACTO 10-A commercial off-the-shelf component quality model proposal," *Journal of Software Engineering*, vol. 4, pp. 1-15, 2010.
- [73] Cho, E. S., *et al.*, "Component Metrics to Measure Component Quality," presented at the Proceedings of the Eighth Asia-Pacific Conference on Software Engineering 2001.

- [74] Rotaru, O. P. and Dobre, M., "Reusability metrics for software components," in *The 3rd ACS/IEEE International Conference on Computer Systems and Applications, 2005*, 2005, pp. 24-1.
- [75] Aggarwal, K. K., *et al.*, "Software reuse metrics for object-oriented systems," in *Third ACIS International Conference on Software Engineering Research, Management and Applications, 2005*, 2005, pp. 48-54.
- [76] Gui, G., "Component Reusability and Cohesion Measures in Object-Oriented Systems," in *Information and Communication Technologies, 2006. ICTTA '06. 2nd*, 2006, pp. 2878-2882.
- [77] Gui, G. and Scott, P. D., "Measuring Software Component Reusability by Coupling and Cohesion Metrics," *Journal of Computers*, vol. 4, pp. 797-805, 2009.
- [78] Noor, M., *et al.*, "A Collaborative Method for Reuse Potential Assessment in Reengineering-Based Product Line Adoption," in *Balancing Agility and Formalism in Software Engineering*, ed: Springer, 2008, pp. 69-83.
- [79] Bi, S., *et al.*, "A Measurement Model of Reusability for Evaluating Component," in *1st International Conference on Information Science and Engineering (ICISE), 2009* 2009, pp. 20-22.
- [80] Gill, N. S., "Reusability issues in component-based development," *SIGSOFT Softw. Eng. Notes*, vol. 28, pp. 1-5, 2003.
- [81] Münch, J., *et al.*, "A Framework for Measuring and Evaluating Program Source Code Quality," in *Product-Focused Software Process Improvement*, vol. 4589, ed: Springer Berlin / Heidelberg, 2007, pp. 284-299.
- [82] Sant'anna, C., *et al.*, "On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework," in *Proceedings XVII Brazilian Symposium on Software Engineering*, 2003, pp. 19-34.
- [83] Deelstra, S., *et al.*, "Variability assessment in software product families," *Information and Software Technology*, vol. 51, pp. 195-218, 2009.
- [84] Fenton, N. and Pfleeger, S., *Software Metrics: A Rigorous and Practical Approach*, 2nd ed.: PWS Publishing Co., 1997.
- [85] IEEE, "IEEE Standard for a Software Quality Metrics Methodology 1061-1998," ed, 1998.

- [86] Abreu, B. F., *et al.*, "Toward the design quality evaluation of object-oriented software systems.," in *Proceedings of the Fifth International Conference on Software Quality*, 1995, pp. 44-57.
- [87] Ritchie, J. and Lewis, J., *Qualitative Research Practice: A Guide for Social Science Students and Researchers.*: Sage Publications Ltd., 2005.
- [88] Guba, E. G. and Lincoln, Y. S., *Handbook of Qualitative Research*: Sage Publications Ltd., 1994.
- [89] Ceotty, M., *The Foundations of Social Research: Meaning and Perspective in the Research* Sage Publications Ltd., 1994.
- [90] Creswell, J. W. and Clark, V. L. P., *Designing and Conducting Mixed Methods Resaerch* 2nd ed.: Sage Publications, Inc., 2011.
- [91] Gray, D. E., *Doing Research in the Real World*, 2nd ed.: SAGE Publication Ltd., 2009.
- [92] Leech, N. and Onwuegbuzie, A., "A typology of mixed methods research designs," *Quality & Quantity*, vol. 43, pp. 265-275, 2009.
- [93] Tashakkori, A. and Teddlie, C., *Mixed Methodology Combining Qualitative and Quantitative Approaches*: SAGE Publications, Inc, 1998.
- [94] Tashakkori, A. and Teddlie, C., *Handbook of mixed methods in social and behavioral research*: Sage Thousand Oaks, 2003.
- [95] Creswell, J. W. and Clark, V. L. P., *Designing and Conducting Mixed Methods Research.*, 1st ed.: SAGE Publications, Inc, 2007.
- [96] Greene, J. C., *et al.*, "Toward a Conceptual Framework for Mixed-Method Evaluation Designs," *Educational Evaluation and Policy Analysis*, vol. 11, pp. 255-274, 1989.
- [97] Greene, J. C., *Mixed Methods in Social Inquiry* John Wiley and Sons, 2007.
- [98] Mingers, J., "Combining IS Research Methods: Towards a Pluralist Methodology," *Information Systems Research*, vol. 12, pp. 240-259, 2001.
- [99] Jackson, M., "The Name and Nature of Software Engineering," in *Advances in Software Engineering: Lipari Summer School 2007, Lipari Island, Italy, July 8-21, 2007, Revised Tutorial Lectures*, ed: Springer-Verlag, 2008, pp. 1-38.

- [100] Shaw, M., "What makes good research in software engineering?," *International Journal on Software Tools for Technology Transfer*, vol. 4, pp. 1-7, 2002.
- [101] Easterbrook, S., *et al.*, "Selecting Empirical Methods for Software Engineering Research," in *Guide to Advanced Empirical Software Engineering*, ed, 2008, pp. 285-311.
- [102] Lazaro, M. and Marcos, E., "An Approach to the Integration of Qualitative and Quantitative Research Methods in Software Engineering Research," in *2nd International Workshop on Philosophical Foundations of Information Systems Engineering (PHISE'06)*, 2006.
- [103] Wood, M., *et al.*, "Multi-method research: An empirical investigation of object-oriented technology," *Journal of Systems and Software*, vol. 48, pp. 13-26, 1999.
- [104] Mandić, V., *et al.*, "Towards Multi-Method Research Approach in Empirical Software Engineering," in *Product-Focused Software Process Improvement*, ed: Springer, 2009, pp. 96-110.
- [105] Robson, C., *Real World Research* 3rd ed.: Wiley-Blackwell, 2002.
- [106] Maxwell, J. A., *Qualitative Research Design: An interactive Approach* 2nd ed.: Sage Publications, Inc., 2005.
- [107] Saunders, M., *et al.*, *Research Methods for Business Studies* 5th ed.: Prentice Hall, 2009.
- [108] Orlikowski, W. J. and Robey, D., "Information Technology and the Structuring of Organizations," *Information Systems Research*, vol. 2, pp. 143-169, 1991.
- [109] Seaman, C. B., "Qualitative Methods in Empirical Studies of Software Engineering," *IEEE Transactions on Software Engineering*, vol. 25, pp. 557-572, 1999.
- [110] Punch, K. F., *Introduction to Research Methods in Education* Sage Publications Ltd., 2009.
- [111] Krippendorff, K., *Content analysis: an introduction to its methodology*, 2nd ed.: Sage, 2004.

- [112] Hsieh, H.-F. and Shannon, S. E., "Three Approaches to Qualitative Content Analysis," *Qualitative Health Research*, vol. 15, pp. 1277-1288, 2005.
- [113] Elo, S. and Kyngäs, H., "The qualitative content analysis process," *Journal of Advanced Nursing*, vol. 62, pp. 107-115, 2008.
- [114] Bongshin, L., *et al.*, "SparkClouds: Visualizing Trends in Tag Clouds," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, pp. 1182-1189, 2010.
- [115] atlas.ti. (2011, 06, June). *atlas.ti*. Available: www.atlasti.com
- [116] Pfleeger, S. L. and Kitchenham, B. A., "Principles of survey research: part 1: turning lemons into lemonade," *SIGSOFT Softw. Eng. Notes*, vol. 26, pp. 16-18, 2001.
- [117] Kasunic, M., "Designing an Effective Survey," SEI, CMU2005.
- [118] Gravetter, F. J. and Forzano, L. A. B., *Research Methods for the Behavioral Sciences*, 4th ed.: Cengage Learning, 2011.
- [119] Marsden, P. V. and Wright, J. D., *Handbook of survey research*, 2nd ed.: Emerald, 2010.
- [120] McCormack, B. and Hill, E., *Conducting a survey: the SPSS workbook*: International Thomson Business Press, 1997.
- [121] Peck, R., *et al.*, *Introduction to Statistics and Data Analysis*, 3rd ed.: Cengage Learning, 2011.
- [122] Ewy, R., *Stakeholder-Driven Strategic Planning in Education: A Practical Guide for Developing and Deploying Successful Long-Range Plans*: ASQ Quality Press, 2009.
- [123] Frost, *et al.*, "MSC Malaysia Supply - Demand Study for the ICT Industry," MSC Malaysia, Salangor Darul Ehsan, Malaysia2009.
- [124] Dowdy, S., *et al.*, *Statistics for Research*, 3rd ed.: John Wiley & Sons, Inc., 2004.
- [125] Jackson, S. L., *Research Methods and Statistics: A Critical Thinking Approach* 4th ed.: Wadsworth, Cengage Learning 2011.
- [126] Osborn, C. E., *Statistical Applications for Health Information Management*, 2nd ed.: Jones and Bartlett Publishers, Inc., 2006.

- [127] Brase, C. H. and Brase, C. P., *Understandable Statistics*, 8th ed.: Houghton Mifflin Company, 2006.
- [128] Triola, M. F., *Elementary Statistics*, 11th ed.: Addison Wesley 2010.
- [129] Newton, R. R. and Rudestam, K. E., *Your Statistical Consultant Answers to your data analysis questions*: Sage Inc., 1999.
- [130] Dictionary, O., "Concise Oxford Dictionary," 10 ed: Oxford University Press 2001.
- [131] Madhavji, N. H., *et al.*, *Software evolution and feedback: theory and practice*: John Wiley & Sons, Ltd., 2006.
- [132] Mubarak, A., *et al.*, "An evolutionary study of fan-in and fan-out metrics in OSS," in *Fourth International Conference on Research Challenges in Information Science (RCIS), 2010* 2010, pp. 473-482.
- [133] Machinery, V. (2010, July, 11). *JHawk* 5. Available: www.virtualmachinery.com/jhawkprod.htm
- [134] Basili, V. R., *et al.*, "The Goal Question Metric Approach," in *Encyclopedia of Software Engineering*, ed: Wiley, 1994.
- [135] Marshall, C. and Rossman, G. B., *Designing Qualitative Research* 5th ed.: Sage Publication Inc., 2011.
- [136] Basit, T. N., *Conducting Resaerch in Educational Contexts* Continuum International Publishing Group, 2010.
- [137] Johnson, B. and Christensen, L., *Educational Resaerch : Quantitative, Qualitative, and Mixed Approaches* 4th ed.: Sage Publication, Inc., 2011.
- [138] Sullivan, L. E., *The SAGE Glossary of the Social and Behavioral Sciences*: Sage Publications, Inc., 2009.
- [139] Salkind, N. J., *Encyclopedia of Research Design* Sage Publication, Inc., 2010.
- [140] McCune, S., *Practice Makes Perfect Statistics*: McGraw-Hill, 2010.
- [141] Onwuegbuzie, A. J. and Johnson, R. B., "The Validity Issues in Mixed Research " *RESEARCH IN THE SCHOOLS*, vol. 13, pp. 48-63, 2006.
- [142] Beck, K. (2011, *Confidence Intervals*. Available: <http://www.doh.wa.gov/healthyouth/technical/confidinterval.htm>

- [143] Kettemann, S., *et al.*, "Product line implementation technologies. Component technology view," Fraunhofer IESE 2003.
- [144] Elrad, T., *et al.*, "Discussing aspects of AOP," *Communications of the ACM*, vol. 44, pp. 33-38, 2001.
- [145] Kiczales, G., "Aspect-oriented programming," *ACM Computing Surveys*, vol. 28, p. 154, 1996.
- [146] Schmid, K. and John, I., "A customizable approach to full lifecycle variability management," *Science of Computer Programming*, vol. 53, pp. 259-284, 2004.
- [147] van der Linden, F., *et al.*, "Variability Issues in Software Product Lines," in *Software Product-Family Engineering*. vol. 2290, ed: Springer Berlin / Heidelberg, 2002, pp. 303-338.
- [148] Her, J. S., *et al.*, "A framework for evaluating reusability of core asset in product line engineering," *Information and Software Technology*, vol. 49, pp. 740-760, 2007.
- [149] Mujtaba, S., *et al.*, "Software Product Line Variability: A Systematic Mapping Study," in *15th Asia-Pacific Software Engineering Conference APSEC 08*, 2008.
- [150] Chidamber, S. R. and Kemerer, C. F., "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, pp. 476-493, 1994.
- [151] Li, W. and Henry, S., "Maintenance metrics for the object oriented paradigm," in *Proceedings of First International Software Metrics Symposium, 1993*, 1993, pp. 52-60.
- [152] Martin, R. C. and Martin, M., *Agile Principles, Patterns, and Practices in C#*: Prentice Hall, 2006.
- [153] Coleman, D., *et al.*, "Using Metrics to Evaluate Software System Maintainability " *Computer*, vol. 27, pp. 44-49, 1994.
- [154] Laired, L. M. and Brennan, M. C., *Software Measurement and Estimation : A Practical Approach*: John Wiley & Sons, Inc., 2006.
- [155] Lincke, R. and Lowe, W., "Compendium of Software Quality Standards and Metrics - Version 1.0," 2007.

- [156] Alves, T. L., *et al.*, "Deriving metric thresholds from benchmark data," in *IEEE International Conference on Software Maintenance (ICSM), 2010* 2010, pp. 1-10.
- [157] Ferreira, K. A. M., *et al.*, "Reference Values for Object-Oriented Software Metrics," in *XXIII Brazilian Symposium on Software Engineering, 2009. SBES '09, 2009*, pp. 62-72.
- [158] Ferreira, K. A. M., *et al.*, "Identifying thresholds for object-oriented software metrics," *Journal of Systems and Software*, vol. In Press, Corrected Proof, 2011.
- [159] Shatnawi, R., "A Quantitative Investigation of the Acceptable Risk Levels of Object-Oriented Metrics in Open-Source Systems," *IEEE Transactions on Software Engineering*, vol. 36, pp. 216-225, 2010.
- [160] Lanza, M. and Marinescu, R., *Object-Oriented Metrics in Practice Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*: Springer-Verlag Berlin Heidelberg 2006.
- [161] Coleman, D., *et al.*, "The application of software maintainability models in industrial software systems," *Journal of System and Software* vol. 29, pp. 3-16, 1995.
- [162] Martin, R. C., *Clean Code*: Pearson Education, Inc., 2009.
- [163] Goulão, M. and Brito, "Software Components Evaluation: an Overview," in *In Proceedings of the 5^ª Conferência da APSI, 2004*.
- [164] Goulão, M. and Abreu, F. B., "Software Components Evaluation: an Overview," in *5th Conferência da APSI, Lisbon, 2004*.
- [165] Schryen, G., "Is open source security a myth?," *Communications of the ACM*, vol. 54, pp. 130-140, 2011.
- [166] Stol, K. J., *et al.*, "A comparative study of challenges in integrating Open Source Software and Inner Source Software," *Information and Software Technology*, vol. In Press, Corrected Proof, 2011.
- [167] Haefliger, S., *et al.*, "Code Reuse in Open Source Software," *MANAGEMENT SCIENCE*, vol. 54, pp. 180-193, January 1, 2008 2008.
- [168] Georg von, K., *et al.*, "Knowledge Reuse in Open Source Software: An Exploratory Study of 15 Open Source Projects," in *System Sciences, 2005*.

HICSS '05. Proceedings of the 38th Annual Hawaii International Conference on, 2005, pp. 1-10.

List of Publications

- Fazal-e-Amin, A. K. Mahmood, and A. Oxley. (2011) A Mixed Method Study to Identify Factors Affecting Software Reusability in Reuse Intensive Development, IEEE National Postgraduate Conference (NPC) , 2011, Universiti Teknologi PETRONAS, Malaysia .
- Fazal-e-Amin, Mahmood A.K., Oxley A. (2011) Reusability Assessment of Open Source Components for Software Product Lines International Journal of New Computer Architectures and their Applications, vol. 1(3): 127-140.
- Fazal-e-Amin, Mahmood A.K., Oxley A. (2011) Using Open Source Components in Software Product Lines – an exploratory study, IEEE Conference on Open Systems 2011.
- Fazal-e-Amin, Mahmood A.K., Oxley A. (2011) Metrics Based Variability Assessment of Code Assets, in: J. M. Zain, et al. (Eds.), Software Engineering and Computer Systems, Springer Berlin Heidelberg. pp. 66-75.
- Fazal-e-Amin, Mahmood A.K., Oxley A. (2011) An analysis of object oriented variability implementation mechanisms. SIGSOFT Softw. Eng. Notes 36:1-4. DOI: 10.1145/1921532.1921538.
- Fazal-e-Amin, Mahmood A.K., Oxley A. (2011) Mechanisms for managing variability when implementing object oriented components, National Information Technology Symposium (NITS), King Saud University, KSA.
- Fazal-e-Amin, Mahmood A.K., Oxley A. (2011) A Review of Software Component Reusability Assessment Approaches. Research Journal of Information Technology 3:1-10.
- Fazal-e-Amin, Mahmood A.K., Oxley A. (2010) Proposal for evaluation of software reusability assessment approach employing a mixed method. SIGSOFT Softw. Eng. Notes 35:1-4. DOI: <http://doi.acm.org/10.1145/1838687.1838703>

- Fazal-e-Amin, Mahmood A.K., Oxley A. (2010) A Review on Aspect Oriented Implementation of Software Product Lines Components. *Information Technology Journal* 9:1262-1269.
- Fazal-e-Amin, Mahmood A.K., Oxley A. (2010) A proposed reusability attribute model for aspect oriented software product line components, *Information Technology (ITSim)*, 2010 International Symposium in. pp. 1138-1141.
- Fazal-e-Amin, Mahmood A.K. (2009) A Survey and Proposed Reusability Assessment Framework for Aspect Oriented Product Line Core Assets 2009 Student Conference on Research and Development (SCOReD 2009), UPM Serdang, Malaysia.

APPENDICES

A. Interview guide

The interview guide used for conducting the interviews is presented in this appendix. It contained the pre-planned questions responded by each interviewee and a set of terms and their definitions used in questions. There were a few sub questions which emerged during the interview those are not included in the following list. However, the crux is presented in the results and other parts of thesis.

Activity	Details	Estimated Time Required
Meeting and greeting	Researcher will introduce himself and greet the respondent.	02 minutes
Ice breaking sentences	A few casual sentences to break the ice and to smooth the conversation.	05 minutes
A brief introduction about the research project	Researcher will give a brief introduction of the research project.	05 minutes
Background of study	Researcher will present the background of study and motivations.	05 minutes
Question	<ol style="list-style-type: none"> 1. How do you see the role of OSS in promotion of reuse? 2. What are your views on OSS used and SPL? 3. How do you see these two fields? 	40-50 minutes

	<p>4. How do you see current practices in SE regarding reuse? (Specifically component reuse/code reuse and OSS).</p> <p>5. What are the characteristics of an OSS affects its reusability?</p> <p>6. What are the current challenges in SPL development?</p> <p>7. What are the challenges to OSS?</p> <p>8. What are your views on the key principles of SPLs and OSS?</p>	
Closing remarks and asking for future contact		02 minutes
Saying thanks		01 minute

- Note: Interview guide contains only the questions pre-planned to ask. Several other questions were asked during the interview (which were not pre-planned) to probe and to get understanding of the respondent's view point.

B. Code of Software used to calculate attribute values

The code which used to calculate the metrics is presented here in C++ based implementation.

```
#include <iostream.h>
#include <conio.h>

main()
{
    float cbo, lcom, comm, noc, nom,nom1, loc, CC, mi;
    cout <<"Enter Loc:";
    cin>> loc;
        float loc1=loc;
    if (loc <= 28)
        loc=0;
    else if (loc > 28 && loc<=70)
        loc=25;
        else if (loc > 70 && loc<=130)
        loc=50;
            else if (loc > 130 && loc<=195)
                loc=75;

    else
        loc=100;
        loc=loc/100;
    cout<<"Enter Nom:";
    cin>> nom;
    nom1=nom;
    if (nom <= 4)
        nom=0;
        else if (nom > 4 && nom<=7)
        nom=25;
            else if (nom > 7 && nom<=10)
                nom=50;
```



```

else if (nom > 10 && nom<=15)
    nom=75;

else
    nom=100;
    nom=nom/100;
float size= (0.5 * loc) + (0.5 * nom);
cout <<"Size is:" <<size;
cout <<endl<<"Enter CBO:";
cin>>cbo;
if ( cbo>9)
    cbo=1;
else
    cbo= (cbo/100 * 9);
    cout<< cbo<< endl;
    cout<<"Enter LCOM:";
    cin>>lcom;
if (lcom <= 0)
    lcom=0;
    else if (lcom > 0 && lcom<=19)
        lcom=50;

else
    lcom=100;
float coupling= cbo;
cout<< "coupling is" << coupling <<endl;
float cohesion= lcom/100;
cout <<"cohesion is " << cohesion<<endl;
float flex= (0.5 * coupling) + (0.5 * cohesion);
flex=1-flex; // because both coupling and cohesion(LCOM) are -ve for flex
cout <<"Flexibility=" << flex;
cout<<"Enter no. of Comments";
cin>>comm;
float comm1= (comm/loc1);
comm=1-comm1; // lack of comments

```

```

float under= (0.25 * coupling) +(0.25* cohesion) + (0.25 *comm)+ (0.25* size);
under=1-under; // 1- under because all hav -ve impact on under
cout<<endl<<"Understandability="<< under;
float dit;
cout<<"Enter DIT:";
cin>>dit;
    if (dit> 2)
        dit=1;
    else
        dit = (dit/100*2);

float port= 1-dit, tclass,tnom; // DIT has -ve impact on port
cout<<"Enter No. of Child";
cin>> noc;
cout<<endl<<"Enter Total No. of Classes:";
cin>> tclass;
cout<<endl<<"Enter total No. of Method";
cin>>tnom;
float sc =nom1/tnom;
float vari= (0.5 * noc/tclass) + (0.5 * nom1/tnom);
cout <<"Variability=" <<vari;
cout<<endl<<"Scope Coverage"<<sc;
cout<<endl<< "Enter MI:";
cin>> mi;
    if (mi <= 65)
        mi=0;
    else if (mi > 65 && mi<=85)
        mi=50;
    else
        mi=100;
        mi=mi/100;

cout<<"Enter CC:";
cin>> CC;

```

```

if (CC <= 1)
    CC=100;
    else if (CC > 1 && CC<=10)
        CC=75;
        else if (CC > 10 && CC<=20)
            CC=50;
            else if (CC > 20 && CC<=50)
                CC=25;

else
    CC=0;
    CC=CC/100; // to keep the maintainability value higher ("higher is
good") when CC is low its effect is +VE
float maintain= (0.5 * mi) + (0.5 * CC);
float r= (0.166* vari) + (0.166*sc) + (0.166*port) +(0.166*under) +
(0.166*maintain) + (0.166*flex);
cout <<endl<<"Flex: " << flex<<endl << "under:" <<under;
cout<<endl<<"Scope:" <<sc<<endl<< "Vari" <<vari;
cout<<endl<<"Maintain:" <<maintain<<endl<<"port:" <<port;
cout<< endl<<"Reusability=" << r;
cout<< "coupling is" << coupling <<endl;
cout <<"cohesion is " << cohesion<<endl<< "size:" << size;

}

```

C. Pseudo Code to calculate package level attribute values

Understand-ability = (Size * 0.50) + (0.5*Ratio of comments)

Ratio of comments = Comments / LOC

Size = (adjusted NOM *0.5) + (adjusted LOC * 0.5)

If (NOM<=NUMBER OF CLASSES/INTERFACES*4)

NOM_val = 0

Else If (NOM>NUMBER OF CLASSES/INTERFACES*4 && NOM<=NUMBER OF CLASSES/INTERFACES*7)

NOM_val=0.25

Else If (NOM>NUMBER OF CLASSES/INTERFACES*7 && NOM<=NUMBER OF CLASSES/INTERFACES*10)

NOM_val=0.5

Else If (NOM>NUMBER OF CLASSES/INTERFACES*10 && NOM<=NUMBER OF CLASSES/INTERFACES*15)

NOM_val=0.75

Else If (NOM>NUMBER OF CLASSES/INTERFACES*15)

NOM_val=1

If (LOC<=NUMBER OF CLASSES/ INTERFACES*28)

LOC_val=0

Else if (LOC>NUMBER OF CLASSES/ INTERFACES*28 && LOC<=NUMBER OF CLASSES/ INTERFACES*70)

LOC_val=0.25

Else if (LOC>NUMBER OF CLASSES/ INTERFACES*70 && LOC<=NUMBER OF CLASSES/ INTERFACES*130)

LOC_val=0.5

Else if (LOC>NUMBER OF CLASSES/ INTERFACES*130 && LOC<=NUMBER OF CLASSES/ INTERFACES*195)

```

LOC_val=0.75
Else if (LOC>NUMBER OF CLASSES/ INTERFACES*195)
LOC_val=1
Adjusted MI
IF(MI<=65,0,IF(AND(MI>65,MI<=85),0.5,IF(MI>85,1)))
If (MI<=65)
MI_val=0
Else if (MI>65 && MI<=85)
MI_val=0.5
Else if (MI>85)
MI_val=1
Adjusted Complexity
If (CYLOMATIC COMPLEXITY>0 && CYLOMATIC COMPLEXITY<=10)
COMP_val=0.75
Else If (CYLOMATIC COMPLEXITY>10 && CYLOMATIC COMPLEXITY<=20)
COMP_val=0.5
Else If (CYLOMATIC COMPLEXITY>20 && CYLOMATIC COMPLEXITY<=50)
COMP_val=0.25

Else if (CYLOMATIC COMPLEXITY>50)
COMP_val=0

```

D. Detailed Component Specifications

Address Book 2010									
	LOC	NOM	CBO	LCOM	Comments	DIT	NOC	MI	CC
Class 1	158	8	1	0.2	3	2	0	96.7	22
Class 2	17	1	0	0	0	1	0	99.03	8
Class 3	5	1	0	0	0	1	0	125.21	1
Class 4	188	12	1	0.5	0	2	0	97.83	40
Class 5	112	6	1	0.3	3	2	0	95.12	11
Class 6	163	10	1	0.18	4	2	0	99.72	17
Class 7	32	6	0	0.4	0	1	0	130.53	7
Class 8	357	16	6	0.09	7	2	0	93.8	52
Class 9	301	10	2	0.14	9	2	0	84.31	46

Airline Reservation System									
	LOC	NOM	CBO	LCOM	Comments	DIT	NOC	MI	CC
Class 1	7	1	1	0	0	2	0	136.12	1
Class 2	104	2	4	0.88	0	2	0	73.13	2
Class 3	104	2	4	0.88	0	2	0	73.06	2
Class 4	133	2	4	0	2	1	0	119.33	16
Class 5	122	2	4	0	2	1	0	119.33	16
Class 6	42	3	1	0.25	0	1	0	107.47	7
Class 7	112	2	10	0.51	0	2	0	69.69	2
Class 8	71	1	2	0	2	2	0	106.02	1
Class 9	21	2	1	0.78	2	1	0	163.16	2
Class 10	21	2	1	0.78	2	1	0	163.16	2
Class 11	22	3	3	0	0	2	0	132.16	4
Class 12	68	3	1	0	0	2	0	93.67	6
Class 13	34	3	1	0	0	2	0	115.27	5

Menu Builder									
	LOC	NOM	CBO	LCOM	Comments	DIT	NOC	MI	CC
Class 1	480	21	1	0.03	10	1	0	115.84	93
Class 2	63	4	2	0.22	8	2	0	142.58	5

Car Sales System									
	LOC	NOM	CBO	LCOM	Comments	DIT	NOC	MI	CC
Class 1	134	6	3	0.17	21	2	0	47.89	22
Class 2	37	4	1	0	6	2	0	68.06	5
Class 3	274	20	4	0.04	43	2	0	60.02	50
Class 4	7	1	0	0	1	2	0	120.31	1
Class 5	173	14	2	0.14	9	2	0	124.29	16
Class 6	7	1	0	0	1	2	0	120.31	1
Class 7	69	15	3	0.01	3	1	0	181.71	15

Class Browser									
	LOC	NOM	CBO	LCOM	Comments	DIT	NOC	MI	CC
Class 1	89	3	1	0.57	41	2	0	63.36	13
Class 2	4	1	0	0	0	1	0	136.59	1
Class 3	6	1	1	0	0	1	1	148.9	1
Class 4	98	13	0	5	20	1	0	163.76	18
Class 5	6	1	0	0	0	1	0	148.9	1
Class 6	3	0	0	0	0	1	0	171	0
Class 7	27	5	1	3	0	2	0	140.29	5
Class 8	4	1	0	0	0	1	0	138.42	1
Class 9	58	10	0	4	8	1	0	173.2	11

Library System									
	LOC	NOM	CBO	LCOM	Comments	DIT	NOC	MI	CC
Class 1	110	2	0	1.36	25	2	0	37.3	10
Class 2	17	7	0	0.03	0	2	0	131.09	7
Class 3	148	5	0	0.41	38	2	0	45.69	17
Class 4	11	4	0	0.11	0	2	0	132.64	4
Class 5	8	1	0	0	0	1	0	117.06	2
Class 6	8	1	0	0	0	1	0	117.06	2
Class 7	8	1	0	0	0	1	0	117.06	2
Class 8	149	5	0	0.42	36	2	0	44.12	18

Flight Reservation System									
	LOC	NOM	CBO	LCOM	Comments	DIT	NOC	MI	CC
Class 1	227	14	3	0.12	126	1	0	50.69	32
Class 2	228	10	3	0.03	132	1	0	102.09	32
Class 3	49	9	3	6	1	1	1	133.13	9
Class 4	51	9	3	0.12	1	1	0	131.82	9
Class 5	44	4	2	0.17	7	1	0	170.24	8

Java Chat Application									
	LOC	NOM	CBO	LCOM	Comments	DIT	NOC	MI	CC
Class 1	40	2	1	1	2	2	0	100.53	6
Class 2	78	2	3	1	3	1	0	132.35	10
Class 3	45	5	2	0.19	6	2	0	89.26	8
Class 4	11	1	3	3	0	1	0	118.83	1
Class 5	119	1	1	2	3	1	0	77.69	12
Class 6	28	2	2	1.33	3	2	0	111.99	5

Banking Application Component									
	LOC	NOM	CBO	LCOM	Comments	DIT	NOC	MI	CC
Class 1	51	8	1	0.11	1	1	0	129.16	10
Class 2	52	9	0	0.08	1	1	0	131.07	10
Class 3	5	1	0	0	1	1	0	184	1
Class 4	24	4	0	0.5	5	1	0	180.81	6
Class 5	65	20	4	0.07	19	1	0	130.83	32

Banking Application Component B									
	LOC	NOM	CBO	LCOM	Comments	DIT	NOC	MI	CC
Class 1	29	5	2	0.75	3	2	0	133.13	6
Class 2	28	4	1	0.11	1	1	0	124.82	5
Class 3	21	2	3	0.33	3	1	0	168.48	3
Class 4	5	1	0	0	2	1	0	134.63	1

Banking Application Component C									
	LOC	NOM	CBO	LCOM	Comments	DIT	NOC	MI	CC
Class 1	4	1	0	0	2	1	0	144.8	1
Class 2	4	2	2	0	3	1	0	178.8	2
Class 3	50	13	2	0.07	7	1	0	191.2	13
Class 4	50	10	2	0.11	16	1	0	172.4	11

Banking Application Component A									
	LOC	NOM	CBO	LCOM	Comments	DIT	NOC	MI	CC
Class 1	51	8	1	0.11	1	1	0	129.16	10
Class 2	52	9	0	0.08	1	1	0	131.07	10
Class 3	24	4	0	0.5	5	1	0	180.81	6

XML Genie									
	LOC	NOM	CBO	LCOM	Comments	DIT	NOC	MI	CC
Class 1	40	3	3	2	37	2	0	83.28	6
Class 2	128	12	5	0.45	0	2	0	102.86	37
Class 3	75	2	3	3	65	1	0	118.92	7
Class 4	351	6	5	0.04	288	1	0	33.89	37
Class 5	130	6	3	5	115	2	0	133.95	12
Class 6	11	3	1	2	0	1	0	134.28	3
Class 7	17	5	3	1	0	2	0	131.99	6

Word Processor									
	LOC	NOM	CBO	LCOM	Comments	DIT	NOC	MI	CC
Class 1	33	1	1	0	0	1	0	77.96	20
Class 2	15	1	1	4	0	2	0	98.8	1
Class 3	27	6	0	1	0	1	0	125.41	9
Class 4	365	12	3	0.1	69	2	0	63.34	50
Class 5	14	1	0	0	0	1	0	97.44	6
Class 6	4	1	0	0	0	1	0	132.5	1
Class 7	9	1	0	0	1	1	0	160.66	1
Class 8	3	1	0	0	0	1	0	135	1
Class 9	3	1	0	0	0	1	0	138.42	1
Class 10	8	1	0	0	1	1	0	162.97	1
Class 11	33	1	1	0	0	1	0	77.96	20

JAIM									
	LOC	NOM	CBO	LCOM	Comments	DIT	NOC	MI	CC
Class 1	231	17	1	15	190	1	0	96.43	33
Class 2	74	12	4	6	65	1	0	81.42	16
Class 3	36	8	2	3	30	1	0	123.2	8
Class 4	177	9	8	11	150	2	0	136.01	25
Class 5	53	5	4	7	40	2	0	105.42	8
Class 6	14	2	2	1	10	1	0	83.17	2
Class 7	56	5	4	6	41	2	0	77.52	6
Class 8	16	2	1	0	9	2	0	75.15	4
Class 9	12	3	1	0	10	2	0	78.42	3
Class 10	6	3	2	0	0	1	0	156.26	3

E. Detailed Package Specifications

Address Book									
	Abstractness	Classes	NOM	LOC	Comments	Fanout	CC	MI	Instability
Package1	0.11	21	70	1362	26	1	102	95.48	0.25

XML Genie									
	Abstractness	Classes	NOM	LOC	Comments	Fanout	CC	MI	Instability
Package1	0	7	37	454	136	1	43	76.9	1

Word Processor									
	Abstractness	Classes	NOM	LOC	Comments	Fanout	CC	MI	Instability
Package1	0.13	15	33	568	150	1	51	133.13	0.2

JAIM									
	Abstractness	Classes	NOM	LOC	Comments	Fanout	CC	MI	Instability
Package1	0.08	26	85	907	409	1	44	74.25	1

JFreeChart									
	Abstractness	Classes	NOM	LOC	Comments	Fanout	CC	MI	Instability
Package1	0	351	1596	25118	570	1	581	90.49	1

MicroTrade									
	Abstractness	Classes	NOM	LOC	Comments	Fanout	CC	MI	Instability
Package1	0.08	12	81	911	400	1	49	77.16	1
Package2	0.16	12	24	510	159	1	44	138.97	0.25
Package3	0	4	39	467	179	1	22	81.86	1
Package4	0	4	18	230	189	0	17	86.11	0

Inforama Document Automation System									
	Abstractness	Classes	NOM	LOC	Comments	Fanout	CC	MI	Instability
Package1	0	4	40	352	54	3	26	133.21	0.6
Package2	0.12	8	38	332	57	3	20	134.36	0.75
Package3	0.5	2	10	133	28	2	5	143.82	0.66
Package4	0	3	25	225	50	1	14	115.33	1
Package5	0	4	25	252	55	2	7	81.8	0.66
Package6	0	2	41	444	133	2	10	60.59	0.66
Package7	1	1	2	10	1	0	1	114.44	0
Package8	0	3	29	261	47	2	9	116.74	1
Package9	0.33	3	13	83	17	2	12	81.94	0.33
Package10	0	1	4	37	5	1	2	69.56	1
Package11	0	4	24	252	63	3	15	61.87	0.75
Package12	0.5	2	13	66	18	2	3	133.82	0.66
Package13	0.07	13	56	1187	305	4	96	47.8	0.57
Package14	0	1	11	310	63	1	16	87.19	1
Package15	1	1	5	47	13	0	1	126.28	0
Package16	0.12	17	98	920	389	4	55	110.71	0.66
Package17	1	1	4	8	2	0	1	107.99	0
Package18	0	4	21	157	25	2	5	78.47	0.66
Package19	1	1	2	9	1	0	1	116.58	0
Package20	0.22	9	62	580	140	6	22	113.68	0.54
Package21	0.5	12	123	677	167	5	21	143.54	0.62
Package22	0	3	31	272	44	3	11	154.6	0.75
Package23	0	2	10	125	35	0	2	74.77	0
Package24	0.27	11	179	1659	433	3	65	153.53	0.6
Package25	0	4	8	189	47	4	13	32.86	0.66
Package26	0	1	9	83	8	1	8	112.96	1
Package27	0	2	4	66	11	2	3	137.41	0.66
Package28	0	4	21	158	35	2	5	78.47	0.66
Package29	0.14	7	44	324	54	3	21	119.25	0.6
Package30	1	1	2	10	1	0	1	114.44	0
Package31	0.33	3	13	118	59	1	6	158.59	1
Package32	0.14	14	45	533	68	2	35	127.97	0.66
Package33	0	3	40	354	61	3	20	134.4	0.6
Package34	1	1	2	9	1	0	1	116.58	0
Package35	0	3	10	269	5	0	9	124.37	0
Package36	0.14	14	48	536	69	2	31	128.62	0.66
Package37	0	4	22	182	38	2	6	74.9	0.66
Package38	0.14	7	30	949	182	7	42	36.32	0.58
Package39	0	2	4	92	8	0	4	84.62	0
Package40	0	2	16	124	24	2	10	138.79	0.66

Package41	0	2	37	598	60	2	15	118.36	0.66
Package42	0	1	6	34	9	0	1	79.44	0
Package43	0.16	6	64	643	82	4	27	77.8	0.57
Package44	0.2	5	21	109	22	1	6	90.14	1
Package45	0	4	22	182	38	2	6	74.9	0.66
Package46	0.14	7	30	949	182	7	42	36.32	0.58

Jasmin Version-1.0									
	Abstractness	Classes	NOM	LOC	Comments	Fanout	CC	MI	Instability
Jasmin	0.18	11	82	2145	1004	3	73	59.91	1
java_cup. runtime	0.1	10	56	573	300	1	24	73.48	0
Jas	0.06	48	191	1708	157	1	99	135.62	0
java_cup	0.07	30	288	3830	1972	2	156	56.82	1

Jasmin Version-2.0									
	Abstractness	NOC	NOM	LOC	Comments	Fanout	CC	MI	Instability
Jasmin	0.17	12	105	2759	1363	3	77	68.63	1
java_cup. runtime	0.1	10	56	573	300	1	13	73.48	1
Jas	0.05	57	244	2177	198	1	145	135.67	1
java_cup	0.07	30	288	3830	1972	2	156	56.82	1

Jasmin Version-2.1									
	Abstractness	Classes	NOM	LOC	Comments	Fanout	CC	MI	Instability
Jasmin	0.15	13	125	3860	2022	3	85	95.21	1
java_cup. runtime	0.1	10	56	573	300	1	24	0	1
Jas	0.05	66	308	2937	249	1	231	132.32	1
java_cup	0.07	30	288	3861	1994	2	156	56.66	1

Jasmin Version-2.2									
	Abstractness	Classes	NOM	LOC	Comments	Fanout	CC	MI	Instability
Jasmin	0.17	12	130	3890	1964	3	87	85.99	1
java_cup. runtime	0.1	10	56	573	300	1	24	73.48	1
Jas	0.05	66	318	3101	273	1	252	131.87	1
java_cup	0.07	30	288	3878	1997	2	161	56.61	1

Jasmin Version-2.3									
	Abstractness	Classes	NOM	LOC	Comments	Fanout	CC	MI	Instability
Jasmin	0.17	12	130	3902	1961	3	87	76.1	1
java_cup. runtime	0.1	10	56	573	300	1	24	73.48	1
Jas	0.05	66	318	3102	273	1	252	131.58	1
java_cup	0.07	30	288	3878	1997	2	161	56.61	1

Jasmin Version-2.4									
	Abstractness	Classes	NOM	LOC	Comments	Fanout	CC	MI	Instability
Jasmin	0.17	12	131	3911	1963	3	88	62.99	1
java_cup. runtime	0.1	10	56	573	300	1	24	73.48	1
Jas	0.05	66	319	3105	273	1	252	151.15	1

F. Reusability Attribute Values

Address Book 2010							
Class	Flexibility	Understandability	Scope Coverage	Variability	Maintainability	Portability	Reusability
1	0.705	0.45	0.11	0.057	0.625	0.96	0.48
2	1	0.75	0.01	0.007	0.875	0.98	0.601
3	1	1	0.01	0.007	1	0.98	0.66
4	0.7	0.415	0.17	0.08	0.625	0.96	0.491
5	0.7	0.51	0.08	0.04	0.75	0.96	0.5
6	0.7	0.45	0.14	0.07	0.75	0.96	0.51
7	0.75	0.56	0.08	0.04	0.875	0.98	0.54
8	0.48	0.244	0.22	0.11	0.5	0.96	0.41
9	0.66	0.399	0.14	0.07	0.37	0.96	0.433

Airline Reservation System							
Class	Flexibility	Understandability	Scope Coverage	Variability	Maintainability	Portability	Reusability
1	0.95	0.72	0.03	0.017	1	0.96	0.61
2	0.57	0.47	0.071	0.035	0.625	0.96	0.45
3	0.57	0.47	0.07	0.03	0.625	0.96	0.45
4	0.82	0.57	0.07	0.03	0.75	0.98	0.53
5	0.82	0.6	0.07	0.03	0.75	0.98	0.54
6	0.705	0.57	0.1	0.053	0.875	0.98	0.54
7	0.25	0.31	0.07	0.03	0.625	0.96	0.37
8	0.91	0.64	0.03	0.017	1	0.96	0.59
9	0.7	0.62	0.07	0.035	0.87	0.98	0.54
10	0.7	0.62	0.07	0.035	0.87	0.98	0.54
11	0.86	0.68	0.107	0.05	0.87	0.96	0.58
12	0.95	0.69	0.107	0.05	0.875	0.96	0.6
13	0.95	0.69	0.107	0.053	0.875	0.96	0.6

Menu Builder							
Class	Flexibility	Understand ability	Scope Coverage	Vari ability	Maintain ability	Portability	Reusability
1	0.705	0.35	0.84	0.42	0.5	0.98	0.63
2	0.66	0.58	0.16	0.08	0.875	0.96	0.55

Car Sales System							
Class	Flexibility	Understand ability	Scope Coverage	Vari ability	Maintain ability	Portability	Reusability
1	0.61	0.47	0.09	0.049	0.125	0.96	0.38
2	0.955	0.73	0.06	0.032	0.625	0.96	0.56
3	0.57	0.32	0.32	0.16	0.125	0.96	0.41
4	1	0.78	0.01	0.008	1	0.96	0.62
5	0.66	0.4	0.22	0.11	0.75	0.96	0.51
6	1	0.78	0.01	0.008	1	0.96	0.62
7	0.615	0.44	0.24	0.12	0.75	0.98	0.52

Class Browser							
Class	Flexibility	Understand ability	Scope Coverage	Vari ability	Maintain ability	Portability	Reusability
1	0.705	0.65	0.08	0.042	0.25	0.96	0.44
2	1	0.75	0.02	0.014	1	0.98	0.58
3	0.955	0.72	0.02	0.069	1	0.98	0.61
4	0.75	0.51	0.37	0.18	0.75	0.98	0.59
5	1	0.75	0.02	0.014	1	0.98	0.62
6	1	0.75	0	0	1	0.98	0.61
7	0.7	0.57	0.14	0.07	0.875	0.96	0.55
8	1	0.75	0.025	0.01	1	0.98	0.62
9	0.75	0.56	0.28	0.14	0.75	0.98	0.57

Library System							
Class	Flexibility	Understand ability	Scope Coverage	Vari ability	Maintain ability	Portability	Reusability
1	0.75	0.61	0.076	0.038	0.375	0.96	0.46
2	0.75	0.59	0.26	0.13	0.875	0.96	0.59
3	0.75	0.56	0.19	0.09	0.25	0.96	0.46
4	0.75	0.625	0.15	0.076	0.875	0.96	0.57
5	1	0.75	0.03	0.019	0.875	0.98	0.6
6	1	0.75	0.03	0.019	0.875	0.98	0.6
7	1	0.75	0.03	0.019	0.875	0.98	0.6
8	0.75	0.56	0.19	0.096	0.25	0.96	0.46

Flight Reservation System							
Class	Flexibility	Understand ability	Scope Coverage	Vari ability	Maintain ability	Portability	Reusability
1	0.615	0.47	0.3	0.15	0.12	0.98	0.44
2	0.61	0.51	0.21	0.108	0.625	0.98	0.5
3	0.615	0.46	0.19	0.19	0.875	0.98	0.53
4	0.615	0.46	0.19	0.097	0.875	0.98	0.53
5	0.66	0.58	0.08	0.043	0.875	0.98	0.53

Java Chat Application							
Class	Flexibility	Understand ability	Scope Coverage	Vari ability	Maintain ability	Portability	Reusability
1	0.705	0.58	0.15	0.076	0.875	0.96	0.55
2	0.615	0.5	0.15	0.07	0.875	0.98	0.53
3	0.66	0.55	0.38	0.19	0.875	0.96	0.6
4	0.615	0.55	0.07	0.038	1	0.98	0.54
5	0.705	0.54	0.07	0.038	0.5	0.98	0.47
6	0.66	0.58	0.15	0.076	0.875	0.96	0.54

Banking Application Component							
Class	Flexibility	Understand ability	Scope Coverage	Vari ability	Maintain ability	Portability	Reusability
1	0.7	0.51	0.19	0.09	0.875	0.98	0.55
2	0.75	0.53	0.21	0.1	0.875	0.98	0.57
3	1	0.8	0.02	0.01	1	0.98	0.63
4	0.75	0.67	0.09	0.047	0.875	0.98	0.56
5	0.57	0.45	0.47	0.23	0.625	0.98	0.55

Banking Application Component B							
Class	Flexibility	Understand ability	Scope Coverage	Vari ability	Maintain ability	Portability	Reusability
1	0.66	0.54	0.41	0.2	0.875	0.96	0.6
2	0.7	0.61	0.33	0.16	0.875	0.98	0.6
3	0.615	0.59	0.16	0.08	0.875	0.98	0.54
4	1	0.85	0.08	0.041	1	0.98	0.65

Banking Application Component C							
Class	Flexibility	Understand ability	Scope Coverage	Vari ability	Maintain ability	Portability	Reusability
1	1	0.875	0.03	0.019	1	0.98	0.64
2	0.91	0.89	0.07	0.03	0.875	0.98	0.62
3	0.66	0.49	0.5	0.25	0.75	0.98	0.6
4	0.66	0.56	0.38	0.19	0.75	0.98	0.58

Banking Application Component A							
Class	Flexibility	Understand ability	Scope Coverage	Vari ability	Maintain ability	Portability	Reusability
1	0.705	0.513	0.347	0.173	0.875	0.98	0.596
2	0.75	0.536	0.391	0.195	0.875	0.98	0.618
3	0.75	0.677	0.173	0.086	0.875	0.98	0.588

XML Genie							
Class	Flexibility	Understand ability	Scope Coverage	Vari ability	Maintain ability	Portability	Reusability
1	0.615	0.75	0.047	0.023	0.625	0.96	0.502
2	0.525	0.35	0.19	0.09	0.625	0.96	0.456
3	0.615	0.711	0.03	0.015	0.875	0.98	0.536
4	0.525	0.561	0.09	0.047	0.125	0.98	0.387
5	0.615	0.684	0.09	0.047	0.75	0.96	0.523
6	0.705	0.602	0.047	0.023	0.875	0.98	0.536
7	0.615	0.526	0.079	0.039	0.875	0.96	0.513

Word Processor							
Class	Flexibility	Understand ability	Scope Coverage	Vari ability	Maintain ability	Portability	Reusability
1	0.955	0.696	0.034	0.017	0.5	0.98	0.528
2	0.705	0.602	0.034	0.017	1	0.96	0.55
3	0.75	0.593	0.206	0.103	0.875	0.98	0.582
4	0.615	0.386	0.413	0.206	0.125	0.96	0.449
5	1	0.75	0.03	0.017	0.875	0.98	0.607
6	1	0.75	0.034	0.017	1	0.98	0.627
7	1	0.777	0.034	0.017	1	0.98	0.623
8	1	0.75	0.034	0.017	1	0.98	0.627
9	1	0.75	0.034	0.017	1	0.98	0.627
10	1	0.781	0.034	0.017	1	0.98	0.632
11	0.75	0.597	0.103	0.051	0.875	0.96	0.554

JAIM							
Class	Flexibility	Understand ability	Scope Coverage	Variability	Maintainability	Portability	Reusability
1	0.705	0.558	0.24	0.121	0.625	0.98	0.536
2	0.57	0.59	0.17	0.085	0.5	0.98	0.482
3	0.66	0.69	0.11	0.05	0.875	0.98	0.561
4	0.39	0.5	0.12	0.06	0.625	0.96	0.44
5	0.57	0.66	0.07	0.035	0.875	0.96	0.526
6	0.66	0.75	0.02	0.014	0.625	0.98	0.509
7	0.57	0.655	0.071	0.035	0.625	0.96	0.484
8	0.955	0.86	0.02	0.014	0.625	0.96	0.572
9	0.955	0.935	0.04	0.021	0.625	0.96	0.587
10	0.91	0.705	0.042	0.021	0.875	0.98	0.586

Address Book						
Package	Understand ability	Flexibility	Portability	Variability	Maintainability	Reusability
1	0.44	0.75	0.75	0.11	0.5	0.51

XML Genie						
Package	Understand ability	Flexibility	Portability	Variability	Maintainability	Reusability
1	0.52	0	0.75	0	0.375	0.32

Word Processor						
Package	Understand ability	Flexibility	Portability	Variability	Maintainability	Reusability
1	0.57	0.80	0.75	0.13	0.50	0.55

JAIM						
Package	Understand ability	Flexibility	Portability	Variability	Maintainability	Reusability
1	0.66	0.00	0.75	0.08	0.38	0.37

JFreeChart						
Package	Understand ability	Flexibility	Portability	Variability	Maintainability	Reusability
1	0.32	0.00	0.75	0.00	0.50	0.31

MicroTrade						
Package	Understand ability	Flexibility	Portability	Variability	Maintainability	Reusability
1	0.53	0.00	0.75	0.08	0.38	0.35
2	0.59	0.75	0.75	0.16	0.63	0.58
3	0.44	0.00	0.75	0.00	0.38	0.31
4	0.79	1.00	1.00	0.00	0.75	0.71

Inforama Document Automation System						
	Understand ability	Flexibility	Portability	Variability	Maintainability	Reusability
Package1	0.33	0.40	0.25	0.00	0.63	0.32
Package2	0.46	0.25	0.25	0.12	0.75	0.37
Package3	0.48	0.34	0.50	0.50	0.88	0.54
Package4	0.36	0.00	0.75	0.00	0.75	0.37
Package5	0.48	0.34	0.50	0.00	0.63	0.39
Package6	0.15	0.34	0.50	0.00	0.38	0.27
Package7	0.55	1.00	1.00	1.00	0.88	0.89
Pckage8	0.34	0.00	0.50	0.00	0.88	0.34
Package9	0.54	0.67	0.50	0.33	0.50	0.51
Package10	0.51	0.00	0.75	0.00	0.63	0.38
Package11	0.50	0.25	0.25	0.00	0.25	0.25
Package12	0.51	0.34	0.50	0.50	0.88	0.55
Package13	0.44	0.43	0.00	0.07	0.00	0.19
Package14	0.16	0.00	0.75	0.00	0.75	0.33
Package15	0.51	1.00	1.00	1.00	0.88	0.88
Package16	0.59	0.34	0.00	0.12	0.50	0.31
Package17	0.63	1.00	1.00	1.00	0.88	0.90

Package18	0.45	0.34	0.50	0.00	0.63	0.38
Package19	0.39	0.25	0.25	0.00	0.63	0.30
Package20	0.42	0.34	0.50	0.00	0.63	0.38
Package21	0.50	0.46	0.00	0.22	0.63	0.36
Package22	0.37	0.38	0.00	0.50	0.63	0.38
Package23	0.27	0.25	0.25	0.00	0.75	0.30
Package24	0.52	1.00	1.00	0.00	0.63	0.63
Package25	0.19	0.40	0.25	0.27	0.50	0.32
Package26	0.56	0.34	0.00	0.00	0.25	0.23
Package27	0.30	0.00	0.75	0.00	0.88	0.38
Package28	0.52	0.34	0.50	0.00	0.88	0.45
Package29	0.49	0.34	0.50	0.00	0.63	0.39
Package30	0.46	0.40	0.25	0.14	0.63	0.37
Package31	0.55	1.00	1.00	1.00	0.88	0.89
Package32	0.63	0.00	0.75	0.33	0.88	0.52
Package33	0.50	0.34	0.50	0.14	0.63	0.42
Package34	0.27	0.40	0.25	0.00	0.75	0.33
Package35	0.38	1.00	1.00	0.00	0.88	0.65
Package36	0.50	0.34	0.50	0.14	0.63	0.42
Package37	0.48	0.34	0.50	0.00	0.63	0.39
Package38	0.35	0.42	0.00	0.14	0.13	0.21
Package39	0.48	1.00	1.00	0.00	0.63	0.62
Package40	0.41	0.34	0.50	0.00	0.88	0.42
Package41	0.05	0.34	0.50	0.00	0.75	0.33
Package42	0.51	1.00	1.00	0.00	0.63	0.63
Package43	0.25	0.43	0.00	0.16	0.38	0.24
Package44	0.54	0.00	0.75	0.20	0.88	0.47
Package45	0.42	0.00	0.25	0.18	0.00	0.17
Package46	0.64	1.00	0.75	0.10	0.38	0.57

Jasmin Version-1.0						
	Understand ability	Flexibility	Portability	Variability	Maintainability	Reusability
Jasmin	0.42	0.00	0.25	0.18	0.00	0.17
java_cup. runtime	0.64	1.00	0.75	0.10	0.38	0.57
Jas	0.48	1.00	0.75	0.06	0.50	0.56
java_cup	0.51	0.00	0.50	0.07	0.00	0.22

Jasmin Version-2.0						
	Understand ability	Flexibility	Portability	Variability	Maintainability	Reusability
Jasmin	0.37	0.00	0.25	0.17	0.25	0.21
java_cup. runtime	0.64	0.00	0.75	0.10	0.50	0.40
Jas	0.42	0.00	0.75	0.05	0.50	0.34
java_cup	0.51	0.00	0.50	0.07	0.00	0.22

Jasmin Version-2.1						
	Understand ability	Flexibility	Portability	Variability	Maintainability	Reusability
Jasmin	0.39	0.00	0.25	0.15	0.50	0.26
java_cup. runtime	0.64	0.00	0.75	0.10	0.38	0.37
Jas	0.42	0.00	0.75	0.05	0.50	0.34
java_cup	0.51	0.00	0.50	0.07	0.00	0.22

Jasmin Version-2.2						
	Understand ability	Flexibility	Portability	Variability	Maintainability	Reusability
Jasmin	0.31	0.00	0.25	0.17	0.50	0.25
java_cup. runtime	0.64	0.00	0.75	0.10	0.38	0.37
Jas	0.42	0.00	0.75	0.05	0.50	0.34
java_cup	0.51	0.00	0.50	0.07	0.00	0.22

Jasmin Version-2.3						
	Understand ability	Flexibility	Portability	Variability	Maintainability	Reusability
Jasmin	0.31	0.00	0.25	0.17	0.25	0.20
java_cup. runtime	0.64	0.00	0.75	0.10	0.38	0.37
Jas	0.42	0.00	0.75	0.05	0.50	0.34
java_cup	0.51	0.00	0.50	0.07	0.00	0.22

Jasmin Version-2.4						
	Understand ability	Flexibility	Portability	Variability	Maintainability	Reusability
Jasmin	0.31	0.00	0.25	0.17	0.00	0.15
java_cup. runtime	0.64	0.00	0.75	0.10	0.38	0.37
Jas	0.42	0.00	0.75	0.05	0.50	0.34

G. Questionnaire

Please mark the appropriate box.

Strongly disagree (1), Disagree (2), Neither agree nor disagree (3), Agree (4), Strongly Agree (5)

Factors Affecting Software Reusability						
	Please indicate your software engineering experience (academic + industry)	years				
Sr		1	2	3	4	5
1	The software which is easy to understand is more likely to be reused					
2	The software which can be easily modified is more likely to be reused					
3	The software which can be easily transferred to other environment is more likely to be reused					
4	The component covering more features (providing more functions) is more likely to be reused					
5	The component which is easy to maintain is more likely to be reused					
6	Stable software components are more likely to be reused (Stability refers to error/bug free)					
7	The usage history of component influences the decision to reuse it					
8	A component with sufficient documentation is more likely to be reused					
9	The component which provides more variability is more likely to be reused					

Displaying 30 of 348 respondents

« Prev | Next » Jump To: Go »

Response Type:
Normal Response

Collector:
Web Link
(Web Link)
IP Address:
202 141 240 117

[Edit Response](#) [Delete](#)

Custom Value:
empty

1. Please indicate your software engineering experience (academic + industry)

Please choose number of years Years
7

2. The software which is easy to understand is more likely to be reused.

Agree

3. The software which can be easily modified is more likely to be reused.

Agree

4. The software which can be easily transferred to other environment is more likely to be reused.

Strongly Agree

5. The component covering more features (providing more functions) is more likely to be reused.

Neither Agree nor disagree

6. The component which is easy to maintain is more likely to be reused.

Agree

7. Stable software components are more likely to be reused. (stability refers to error/bug free).

Neither Agree nor disagree

8. The usage history of component influences the decision to reuse it.

Disagree

9. A component with sufficient documentation is more likely to be reused.

Neither Agree nor disagree

10. The component which provides more variability is more likely to be reused.

Agree

Displaying 19 of 348 respondents

« Prev Next » Jump To: Go »

Response Type:
Normal Response

Collector:
Web Link
(Web Link)
IP Address:
182.185.12.99

[Edit Response](#) [Delete](#)

Custom Value:
empty

1. Please indicate your software engineering experience (academic + industry)

Please choose number of years **Years**

2. The software which is easy to understand is more likely to be reused.

Strongly Agree

3. The software which can be easily modified is more likely to be reused.

Agree

4. The software which can be easily transferred to other environment is more likely to be reused.

Agree

5. The component covering more features (providing more functions) is more likely to be reused.

Disagree

6. The component which is easy to maintain is more likely to be reused.

Agree

7. Stable software components are more likely to be reused. (stability refers to error/bug free).

Agree

8. The usage history of component influences the decision to reuse it.

Agree

9. A component with sufficient documentation is more likely to be reused.

Strongly Agree

10. The component which provides more variability is more likely to be reused.

Neither Agree nor disagree

Displaying 345 of 348 respondents

« Prev | Next » Jump To: Go »

Response Type:
Normal Response

Collector:
Web Link
(Web Link)

[Edit Response](#)

[Delete](#)

Custom Value:
empty

IP Address:
60.52.36.26

1. Please indicate your software engineering experience (academic + industry)

Please choose number of years Years
 10

2. The software which is easy to understand is more likely to be reused.

Agree

3. The software which can be easily modified is more likely to be reused.

Strongly Agree

4. The software which can be easily transferred to other environment is more likely to be reused.

Agree

5. The component covering more features (providing more functions) is more likely to be reused.

Agree

6. The component which is easy to maintain is more likely to be reused.

Agree

7. Stable software components are more likely to be reused. (stability refers to error/bug free).

Strongly Agree

8. The usage history of component influences the decision to reuse it.

Strongly Agree

9. A component with sufficient documentation is more likely to be reused.

Neither Agree nor disagree

10. The component which provides more variability is more likely to be reused.

Agree

H. List of Sources

Source Accessed	Population
Emails	73
On paper	46
Facebook Groups	
MSC Malaysia Open Source Conference 2009	631
Malaysia Open Source Developers' Club	359
Open Source Competency Centre Malaysia (OSCC)	317
Open Source Alliance Cyberjaya	65
Kuching Open Source Community	50
MSC Malaysia Open Source Conference 2009	382
Malaysia Open Source Conference 2011	547
Malaysian Software Developers Network	30
Malaysian Open Source Community	207
Total	2707
Malaysian ICT SMEs contacted	58

I. Package Analysis (jasmin and pBeans)

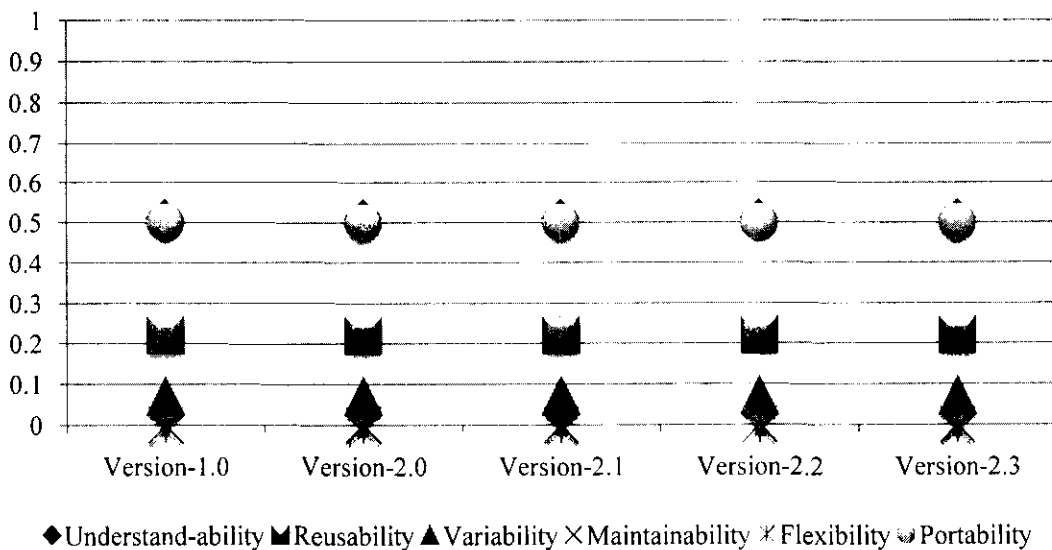
Analysis of Package-4 (java_cup)

The reusability and attributes values of package java_cup are presented in this section. This package is not part of the Jasmin software in the latest version.

Version wise values of reusability and its attributes (package-4)

java_cup	Versions					
	1.0	2.0	2.1	2.2	2.3	2.4
Understandability	0.51	0.51	0.51	0.51	0.51	-
Flexibility	0.00	0.00	0.00	0.00	0.00	-
Portability	0.50	0.50	0.50	0.50	0.50	-
Variability	0.07	0.07	0.07	0.07	0.07	-
Maintainability	0.00	0.00	0.00	0.00	0.00	-
Reusability	0.22	0.22	0.22	0.22	0.22	-

Reusability and Attributes Values Package -4 (java_cup)

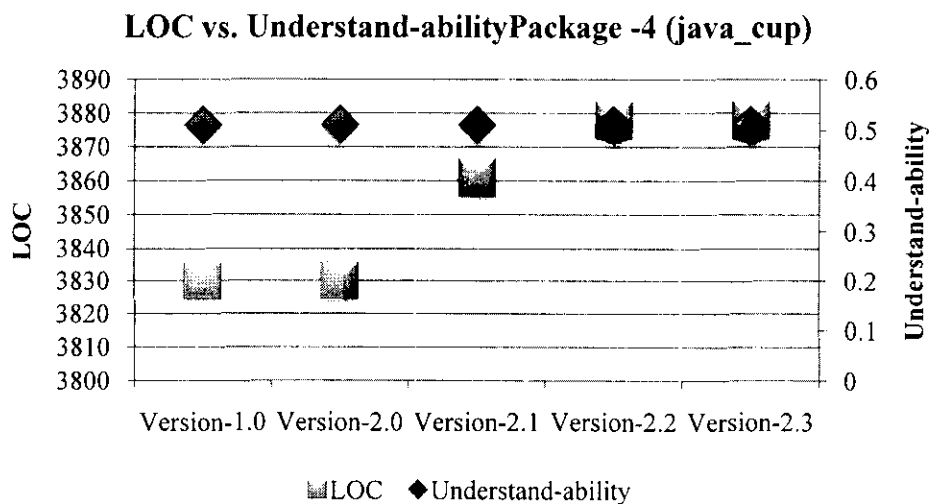


Reusability and its attribute values for packege-4

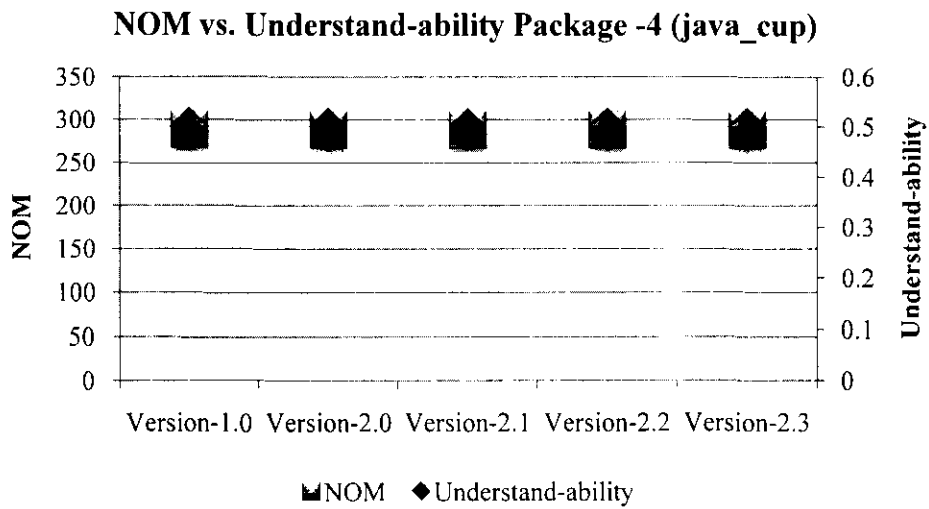
The values of NOM, LOC, comments and understandability are presented in previous section. There is no change in the value of NOM; the change in LOC and comments is insignificant. So, the value of understandability remains the same in all versions. It shows that during evolution this package has not been changed much.

Version wise values of understandability and its attributes (package-4)

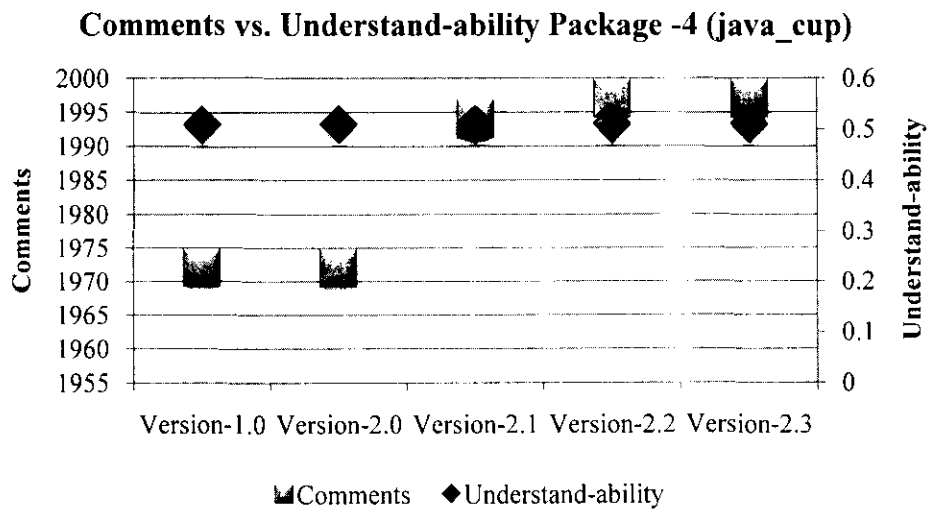
java_cup	Versions					
	1.0	2.0	2.1	2.2	2.3	2.4
LOC	3830	3830	3861	3878	3878	-
NOM	288	288	288	288	288	-
Comments	1972	1972	1994	1997	1997	-
Understandability	0.51	0.51	0.51	0.51	0.51	-



Graph plot of values of LOC and understandability package-4



Graph plot of values of NOM and understandability package-4



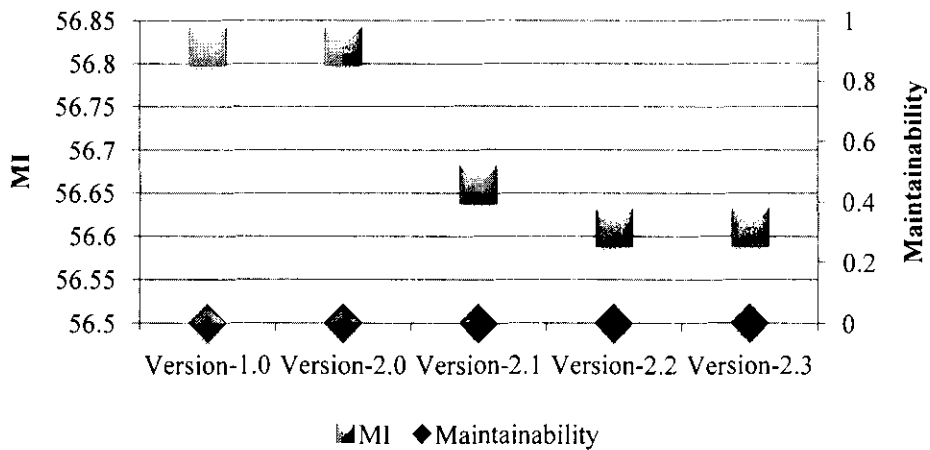
Graph plot of values of comments and understandability package-4

The values of CC and MI for the package java_cup remain the same in all the versions. Therefore, no change is observed in the values of maintainability. The values of CC, MI and maintainability are presented in this section.

Version wise values of maintainability and its attributes (package-4)

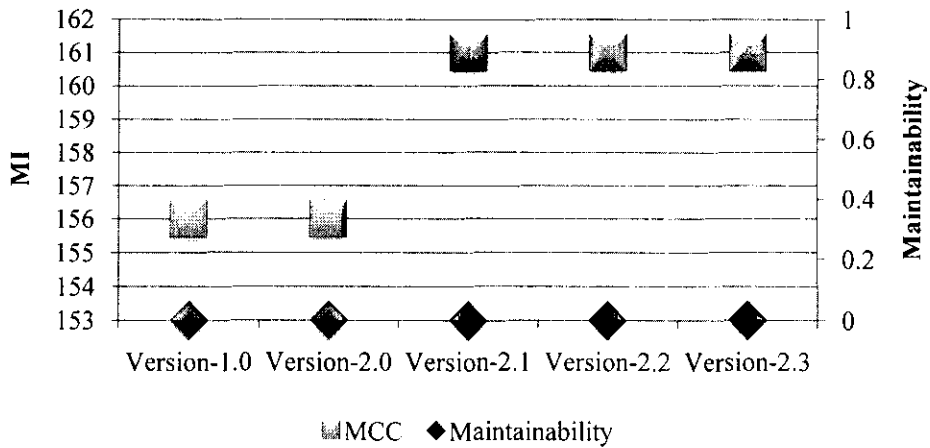
java_cup		Versions					
		1.0	2.0	2.1	2.2	2.3	2.4
MI		56.82	56.82	56.66	56.61	56.61	-
CC		156	156	156	161	161	-
Maintainability		0	0	0	0	0	-

MI vs. Maintainability Package -4 (java_cup)



Graph plot of values of MI and maintainability package-4

MCC vs. Maintainability Package -4 (java_cup)



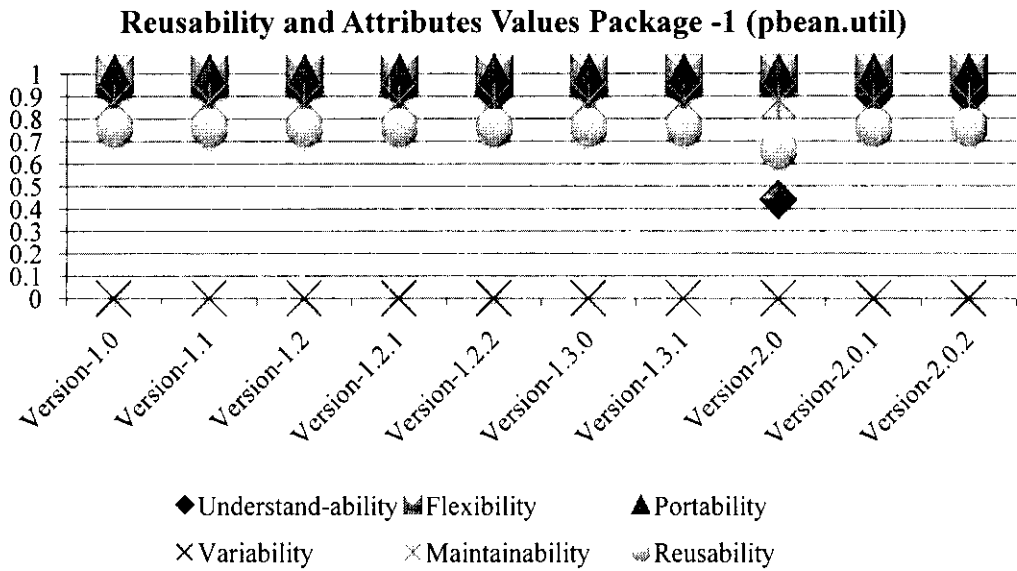
Graph plot of values of CC and maintainability package-4

Analysis of Package-1 (pbean.util)

The package pbean.utilil shows no change in the value of reusability and its attributes in this section. The only exception is version 2.0 where the value of understandability decreased from 0.94 to 0.44 which caused a decrease in the value of reusability. It shows that during the evolution of this software, pbean.utilil package has not been changed significantly.

Version wise values of reusability and its attributes (package-1)

pbean.util	Versions									
	1.0	1.1	1.2	1.2.1	1.2.2	1.3.0	1.3.1	2.0	2.0.1	2.0.2
Understandability	0.94	0.94	0.94	0.94	0.91	0.94	0.94	0.44	0.90	0.90
Flexibility	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Portability	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Variability	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Maintainability	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88
Reusability	0.76	0.76	0.76	0.76	0.76	0.76	0.76	0.66	0.76	0.76

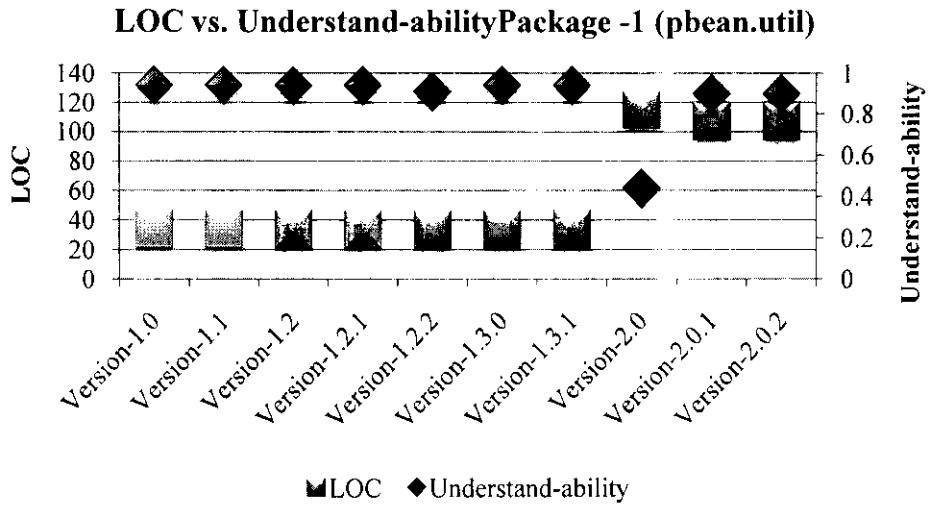


Reusability and its attribute values for package-1

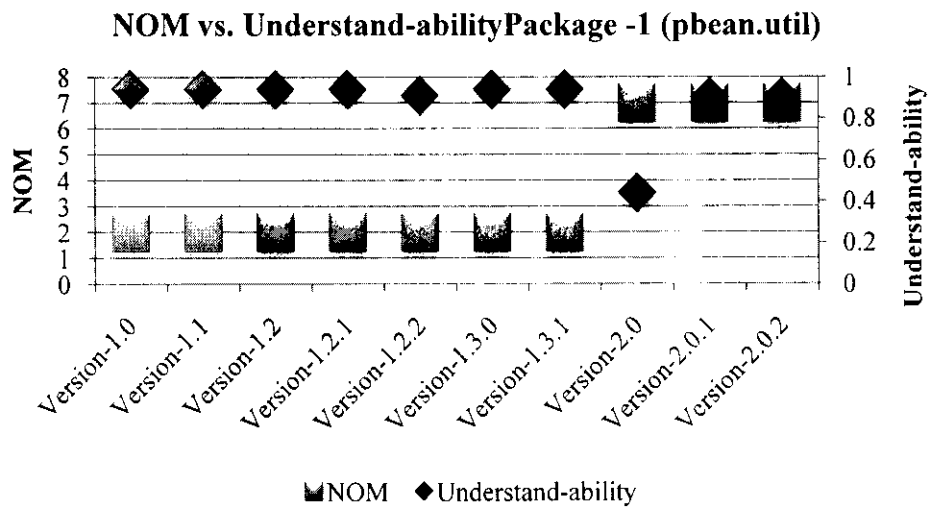
The values of LOC, NOM and comments remain the same for the versions 1.0 to 1.3.1. In version 2.0, the understandability value is decreased to 0.44; it is because of the lack of comments. The value of understandability is increased in version 2.0.1 up to 0.90 and remains the same for version 2.0.2. The values are presented in this section. The values show that the package has undergone major changes in version 2.0.

Version wise values of understandability and its attributes (package-1)

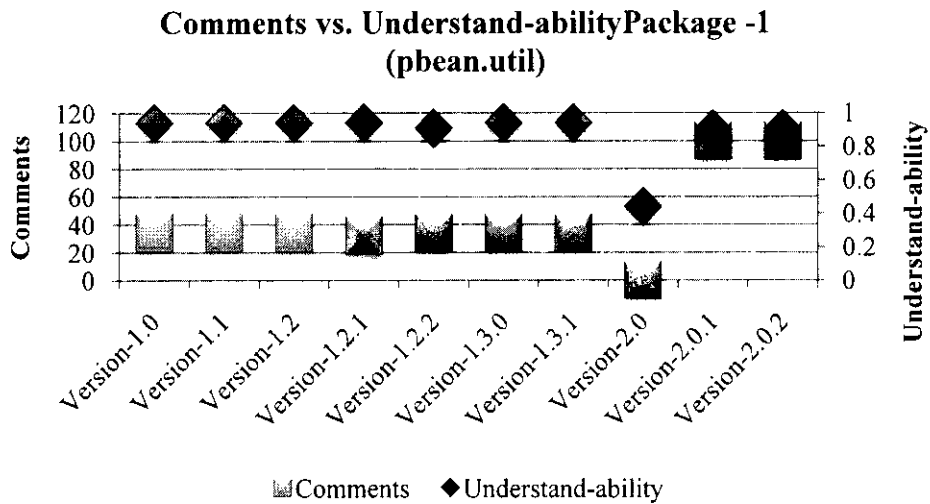
pbean.util	Versions									
	1.0	1.1	1.2	1.2.1	1.2.2	1.3.0	1.3.1	2.0	2.0.1	2.0.2
LOC	34	34	34	34	34	34	34	115	107	107
NOM	2	2	2	2	2	2	2	7	7	7
Comments	34	34	34	32	34	34	34	0	100	100
Understandability	0.94	0.94	0.94	0.94	0.91	0.94	0.94	0.44	0.90	0.90



Graph plot of values of LOC and understandability package-1



Graph plot of values of NOM and understandability package-1



Graph plot of values of comments and understandability package-1

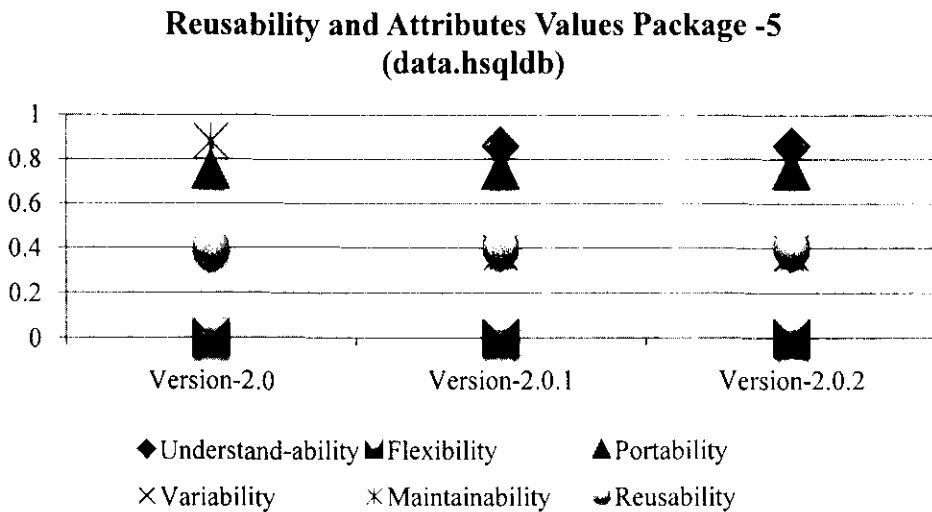
Analysis of Package-5 (data.hsldb)

The data.hsldb package is included in the pBeans software in version 2.0. Its reusability and attribute values are presented in this section. The reusability remains the same in all the three versions. The value of understandability has increased from 0.38 to 0.86 in version 2.0.1.

Version wise values of reusability and its attributes (package-5)

data.hsldb	Versions		
	2.0	2.0.1	2.0.2
Understandability	0.38	0.86	0.86
Flexibility	0.00	0.00	0.00
Portability	0.75	0.75	0.75
Variability	0.00	0.00	0.00
Maintainability	0.88	0.38	0.38
Reusability	0.40	0.40	0.40

The value of maintainability has decreased from 0.88 to 0.38 in version 2.0.1. This increase in understandability has compensated the decrease in maintainability. Therefore, no effect on reusability is observed.



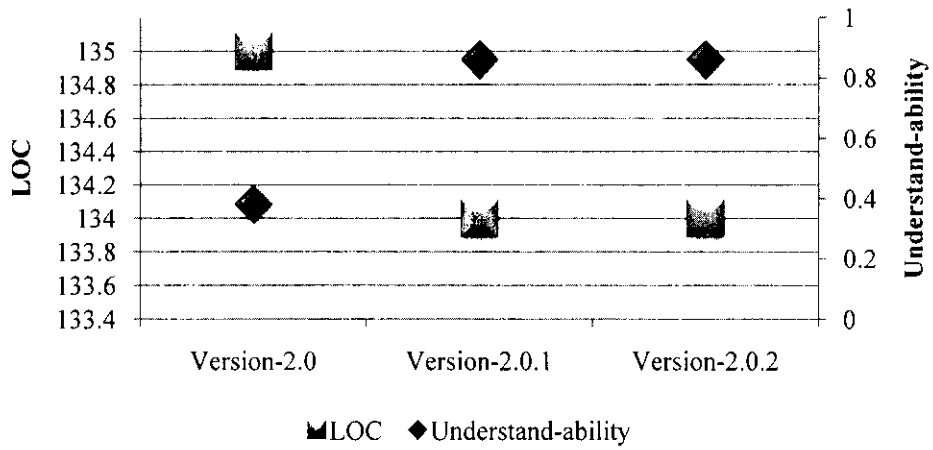
Reusability and its attribute values for package-5

The values of LOC, NOM, comments and understandability are presented in this section. An increase in the value of understandability can be seen in version 2.0.1. The increase is due to the increase in the number of comments in version 2.0.1. The value of understandability remains the same for the latest version.

Version wise values of understandability and its attributes (package-5)

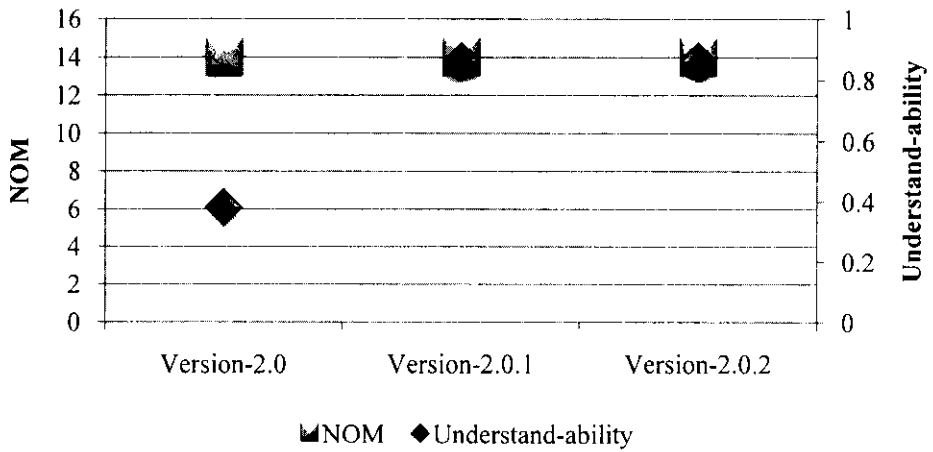
data.hsqldb		Versions		
		2.0	2.0.1	2.0.2
LOC		135	134	134
NOM		14	14	14
Comments		0	130	130
Understandability		0.38	0.86	0.86

LOC vs. Understand-abilityPackage -5 (data.hsldb)



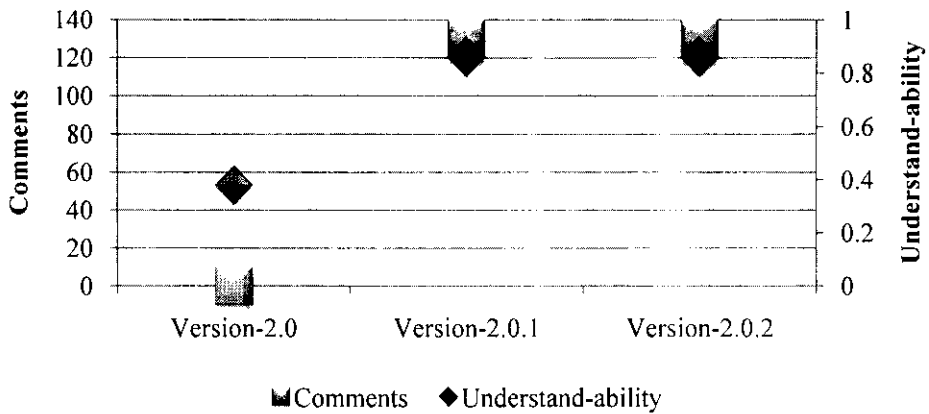
Graph plot of values of LOC and understandability package-5

NOM vs. Understand-abilityPackage -5 (data.hsldb)



Graph plot of values of NOM and understandability package-5

**Comments vs. Understand-abilityPackage -5
(data.hsqldb)**

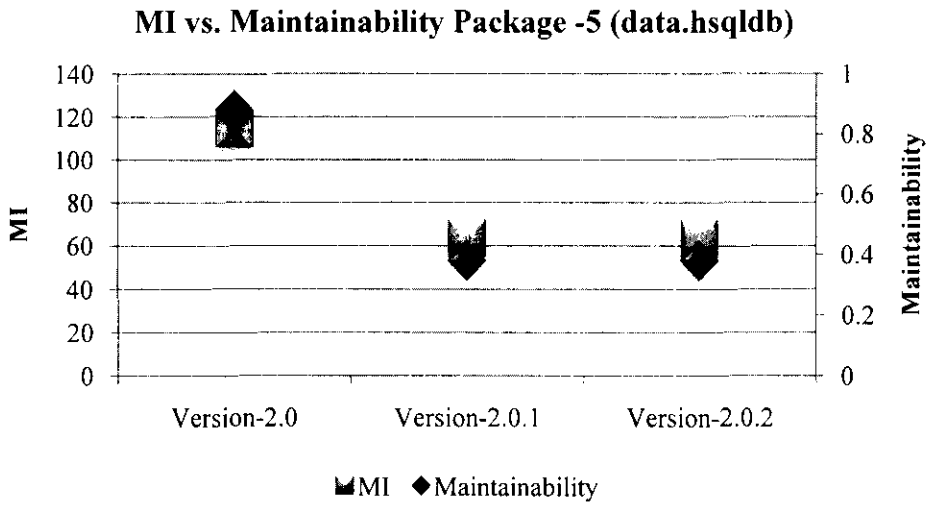


Graph plot of values of comments and understandability package-5

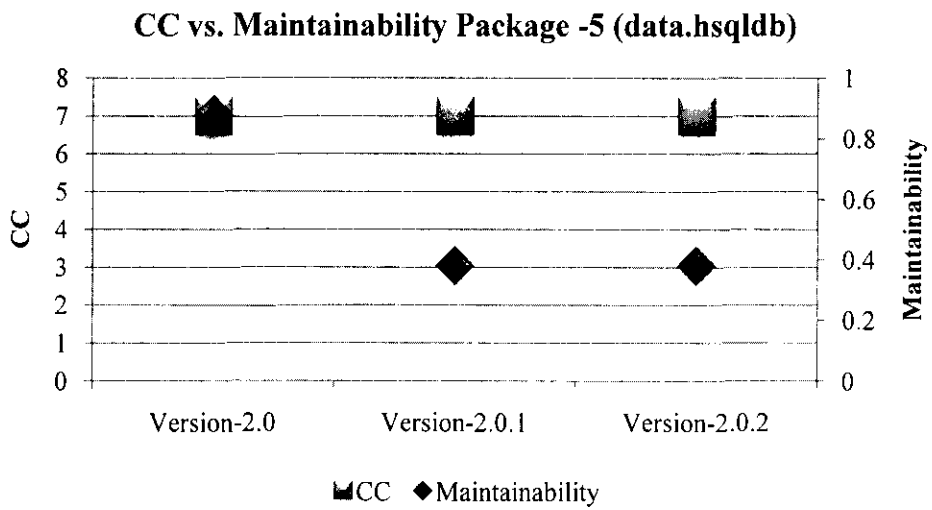
The values of MI, complexity and maintainability are presented in this section. The value of maintainability decreased significantly in version 2.0.1. This decrease in the value of maintainability is due to the decrease in the value of MI in version 2.0.1 from 114.55 to 63.33.

Version wise values of maintainability and its attributes (package-5)

data.hsqldb	Versions		
	2.0	2.0.1	2.0.2
MI	114.55	63.53	63.53
CC	7	7	7
Maintainability	0.88	0.38	0.38

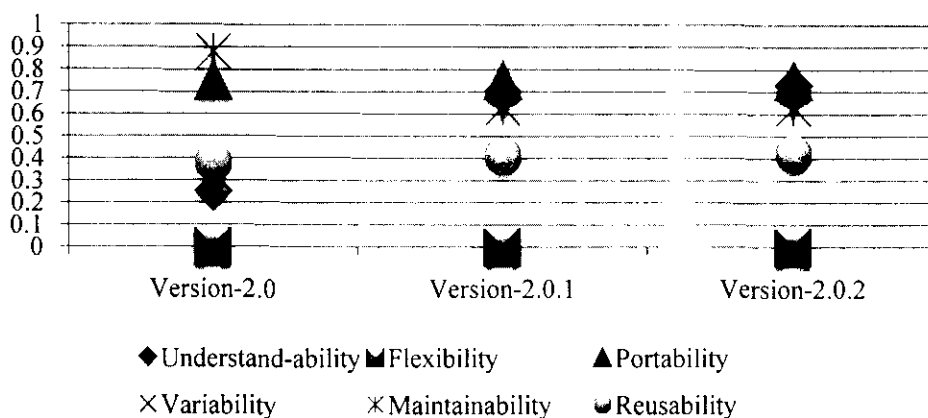


Graph plot of values of MI and maintainability package-5



Graph plot of values of CC and maintainability package-6

Reusability and Attributes Values Package -6 (data.sqlserver)



Reusability and its attribute values for package-6

Analysis of Package-6 (data.sqlserver)

The package data.sqlserver is part of the pBeans software in the version 2.0. The values of reusability and attributes are presented in this section.

Version wise values of reusability and its attributes (package-6)

data.sqlserver	Versions		
	2.0	2.0.1	2.0.2
Understandability	0.25	0.70	0.73
Flexibility	0.00	0.00	0.00
Portability	0.75	0.75	0.75
Variability	0.00	0.00	0.00
Maintainability	0.88	0.63	0.63
Reusability	0.38	0.41	0.42

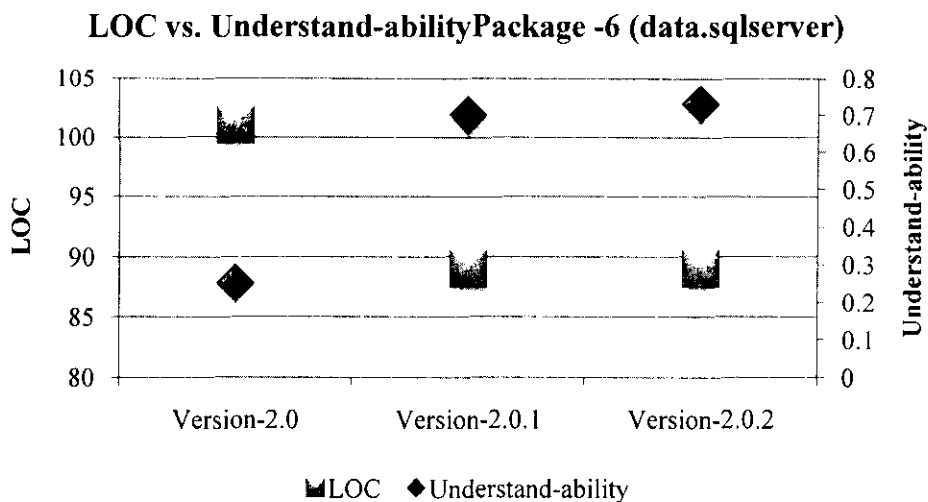
An increase in the value of reusability is observed in the versions 2.0.1 and 2.0.2. This increase in the value of reusability is due to the increase in the value of understandability from 0.25 to 0.70, and subsequently to 0.73 in versions 2.0.1 and

2.0.2 respectively. There is a decrease in the value of maintainability from 0.88 to 0.63. Although, the maintainability value is decreased, however this decrease is compensated by the value of understandability.

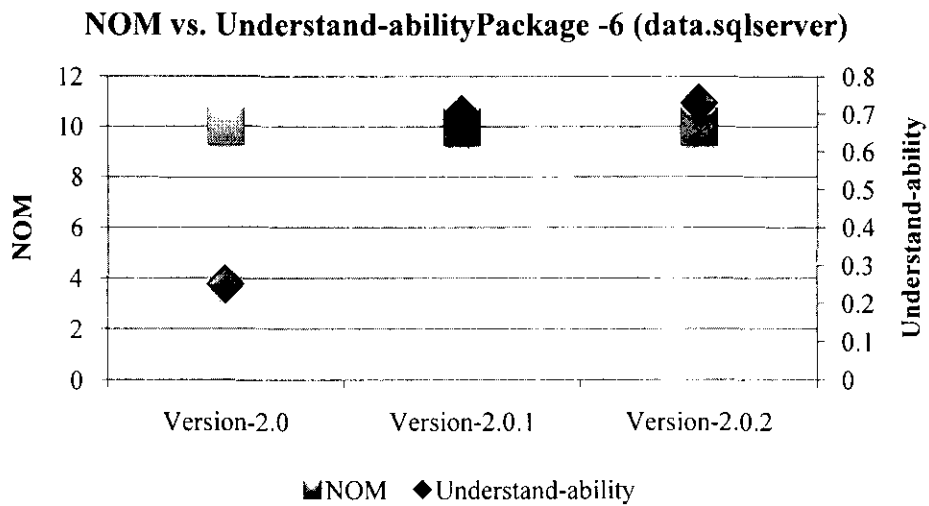
Version wise values of understandability and its attributes (package-6)

data.sqlserver		Versions		
		2.0	2.0.1	2.0.2
LOC		101	89	89
NOM		10	10	10
Comments		0	80	85
Understandability		0.25	0.70	0.73

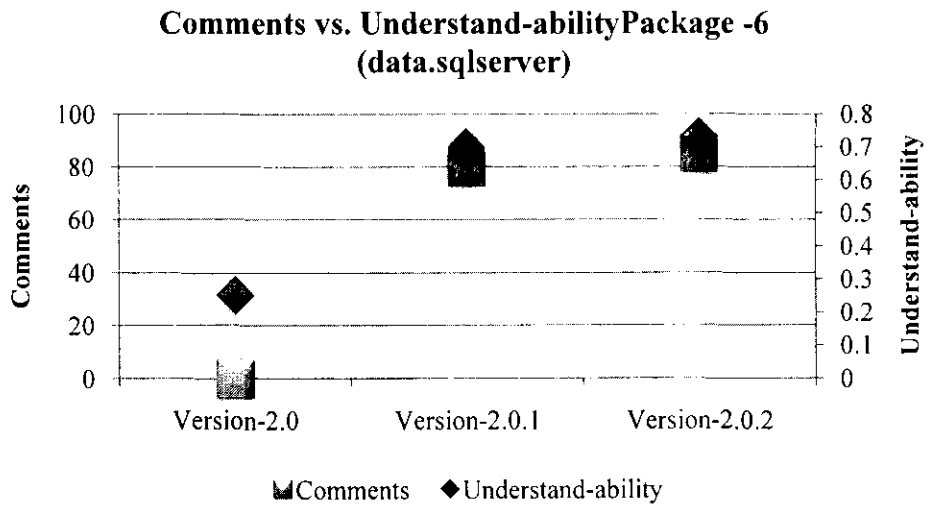
The values of LOC, NOM, comments and understandability are presented in this section. The value of understandability is increased in the versions 2.0.1 and 2.0.2. This increase in the value of understandability is due to the decrease in the LOC and number of comments.



Graph plot of values of LOC and understandability package-6



Graph plot of values of NOM and understandability package-6



Graph plot of values of comments and understandability package-6

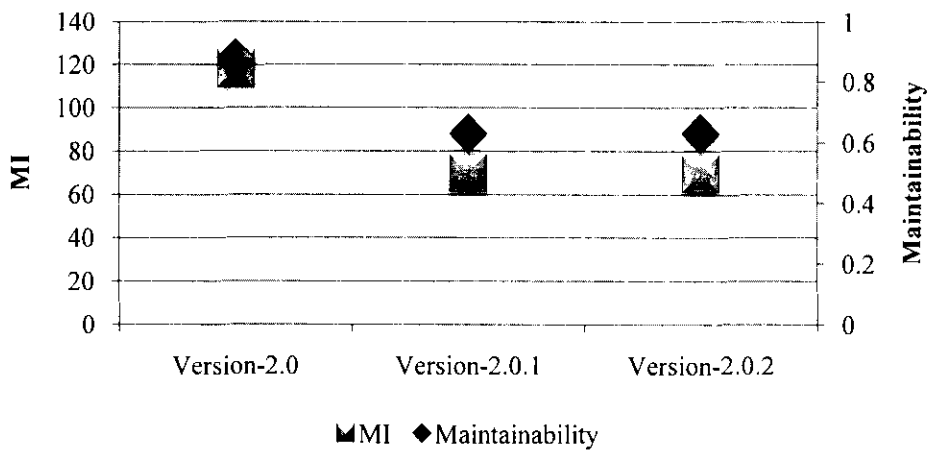
The values of MI, complexity and maintainability are presented in this section. A decrease in the value of maintainability can be observed in version 2.0.1. The MI value in version 2.0 is 118.36 which decreased to 69.86 in version 2.0.1. This decrease in the value of maintainability is due to the decrease in the value of MI. The value of maintainability is directly related to MI.

Version wise values of maintainability and its attributes (package-6)

data.sqlserver

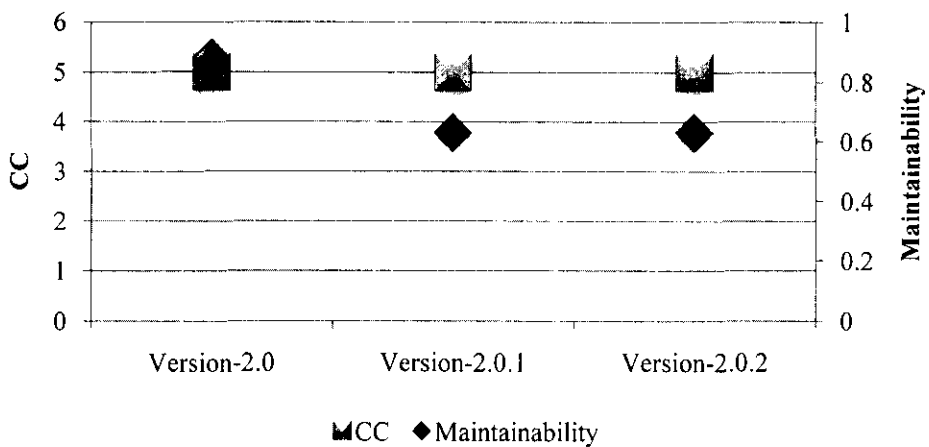
	Versions		
	2.0	2.0.1	2.0.2
MI	118.36	69.86	69.86
CC	5	5	5
Maintainability	0.88	0.63	0.63

MI vs. Maintainability Package -6 (data.sqlserver)



Graph plot of values of MI and maintainability package-6

CC vs. Maintainability Package -6 (data.sqlserver)



Graph plot of values of CC and maintainability package-7

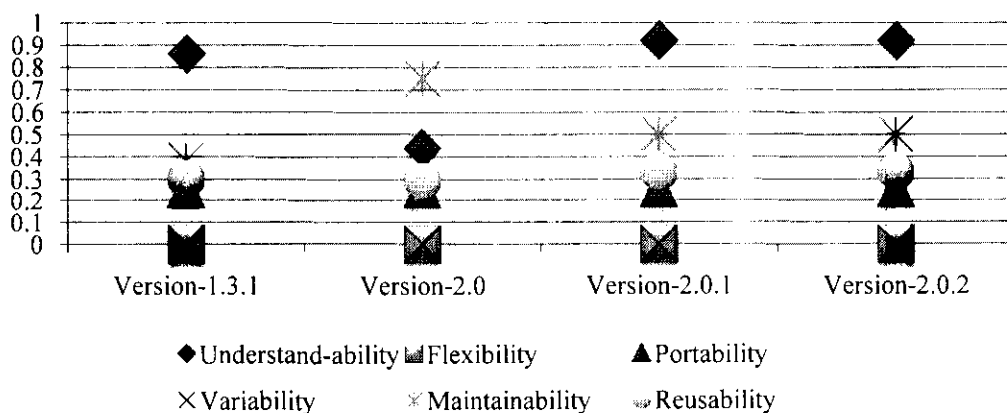
Analysis of Package-7 (pbeans.servlet)

The package pbeans.servlet is part of the pBeans software for the versions 1.3.1 and onwards. The reusability and attribute values are presented in this section. A slight decrease in the value of reusability can be observed in the version 2.0. This is due to the increase in maintainability and decrease in the value of understandability. The value of reusability is increased to 0.33 in version 2.0.1 and 2.0.2. This increase is due to the increase in the values of maintainability and understandability.

Version wise values of reusability and its attributes (package-7)

pbeans.servlet		Versions			
	1.3.1	2.0	2.0.1	2.0.2	
Understandability	0.86	0.44	0.92	0.92	
Flexibility	0.00	0.00	0.00	0.00	
Portability	0.25	0.25	0.25	0.25	
Variability	0.00	0.00	0.00	0.00	
Maintainability	0.38	0.75	0.50	0.50	
Reusability	0.30	0.29	0.33	0.33	

**Reusability and Attributes Values Package -7
(pbeans.servlet)**



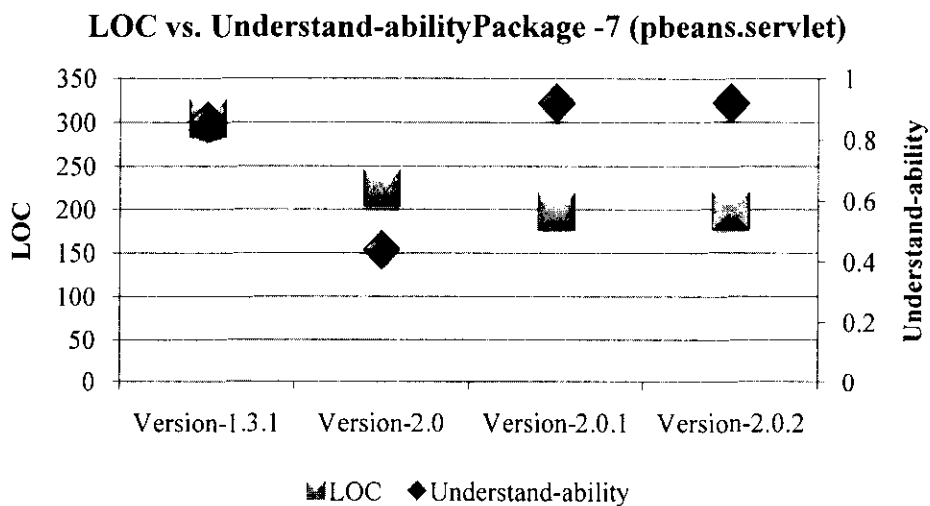
Reusability and its attribute values for package-7

The values of LOC, NOM, comments and understandability are presented in this section. The value of understandability is decreased from 0.86 to 0.44 in version 2.0.

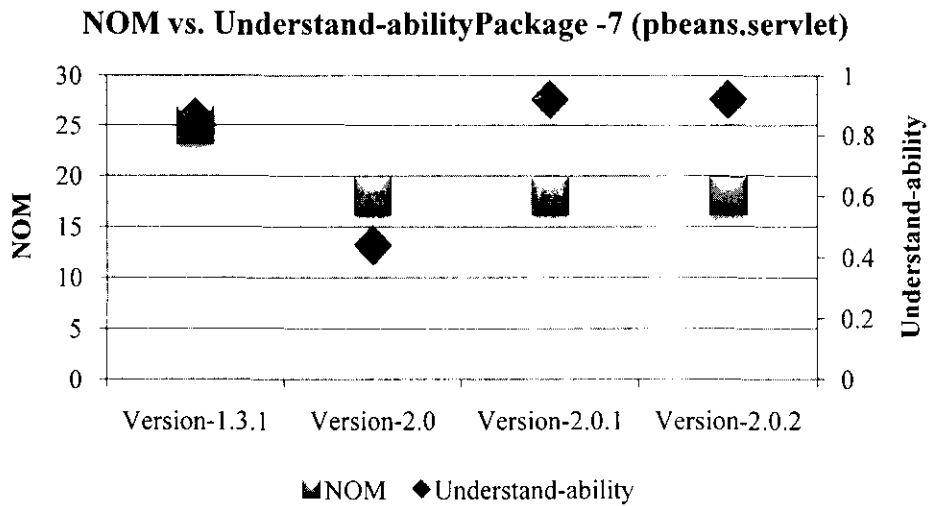
An increase is observed in the version 2.0.1. This increase is due to the decrease in LOC and increase in the number of comments.

Version wise values of understandability and its attributes (package-7)

pbeans.servlet	Versions			
	1.3.1	2.0	2.0.1	2.0.2
LOC	305	225	198	198
NOM	25	18	18	18
Comments	298	3	190	190
Understandability	0.86	0.44	0.92	0.92

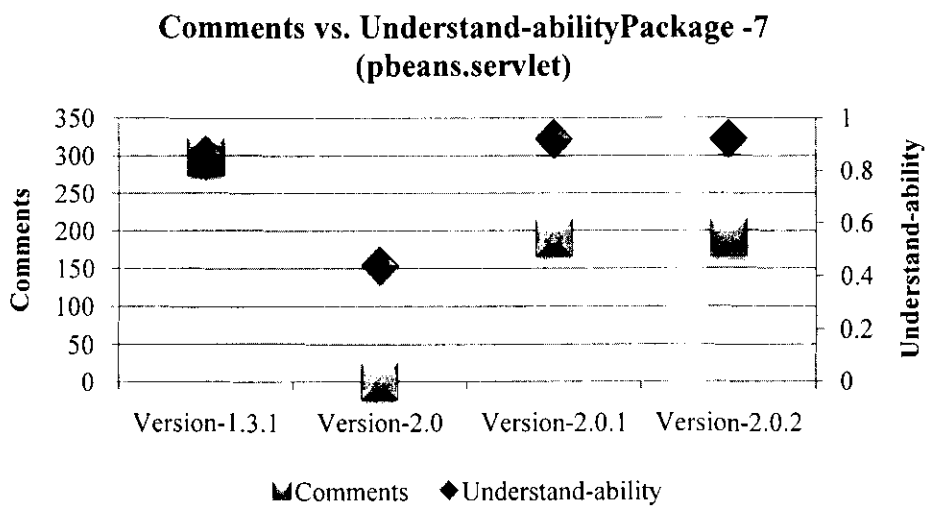


Graph plot of values of LOC and understandability package-7



Graph plot of values of NOM and understandability package-7

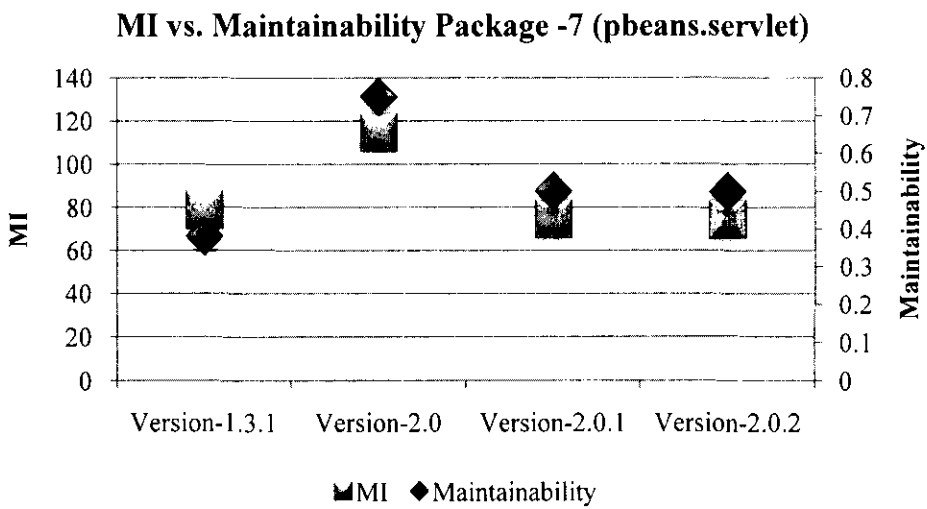
The values of MI, complexity and maintainability are presented in this section. An increase in the maintainability value can be observed in version 2.0. This increase is due to the increase in the value of MI and decrease in the value of complexity. In version 2.0.1, the value of maintainability is decreased to 0.50. This decrease in maintainability value is due to the decrease in the value of MI.



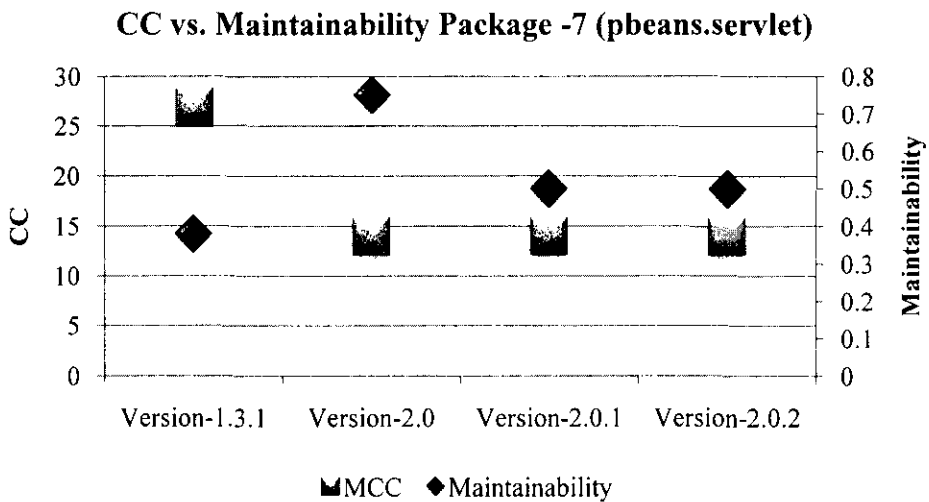
Graph plot of values of comments and understandability package-7

Version wise values of maintainability and its attributes (package-7)

pbeans.servlet				
	Versions			
	1.3.1	2.0	2.0.1	2.0.2
MI	78.91	114.72	74.62	74.62
CC	27	14	14	14
Maintainability	0.38	0.75	0.50	0.50



Graph plot of values of MI and maintainability package-7



Graph plot of values of CC and maintainability package-7

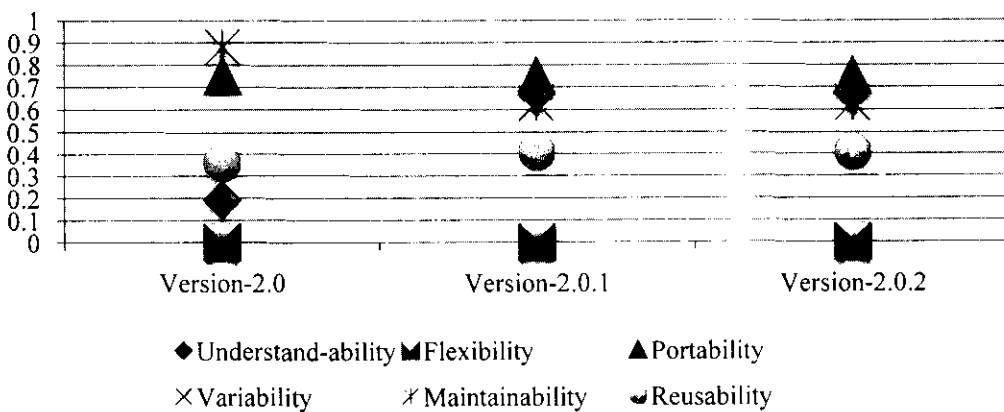
Analysis of Package-8 (data.postgresql)

The package data.postgresql is included in the version 2.0 and onwards. The reusability value is increased from 0.36 to 0.41. This increase in the value of reusability is due to the increase in the value of understandability. On the other hand maintainability value is decreased in version 2.0.1. The decrease in the value of maintainability is overshadowed by the increase in the value of understandability. Therefore, an increase in value of reusability is observed.

Version wise values of reusability and its attributes (package-8)

data.postgresql		Versions		
	2.0	2.0.1	2.0.2	
Understandability	0.19	0.68	0.68	
Flexibility	0.00	0.00	0.00	
Portability	0.75	0.75	0.75	
Variability	0.00	0.00	0.00	
Maintainability	0.88	0.63	0.63	
Reusability	0.36	0.41	0.41	

**Reusability and Attributes Values Package -8
(data.postgresql)**

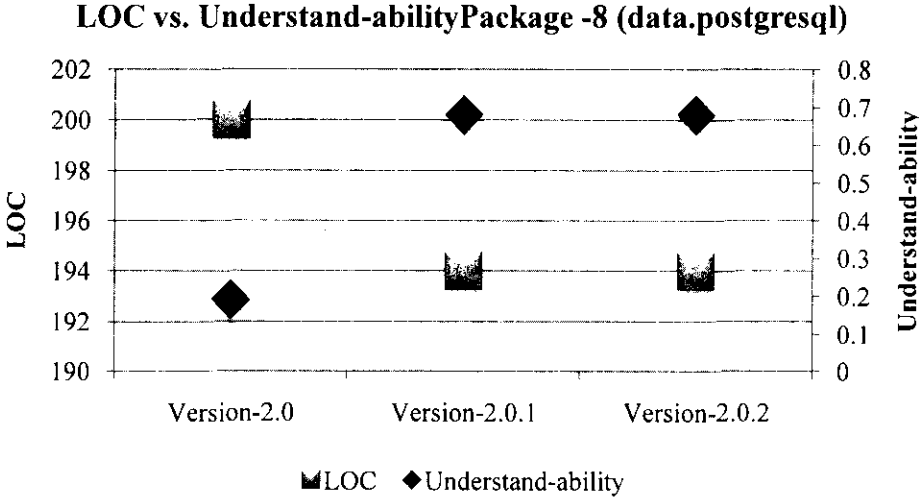


Reusability and its attribute values for package-8

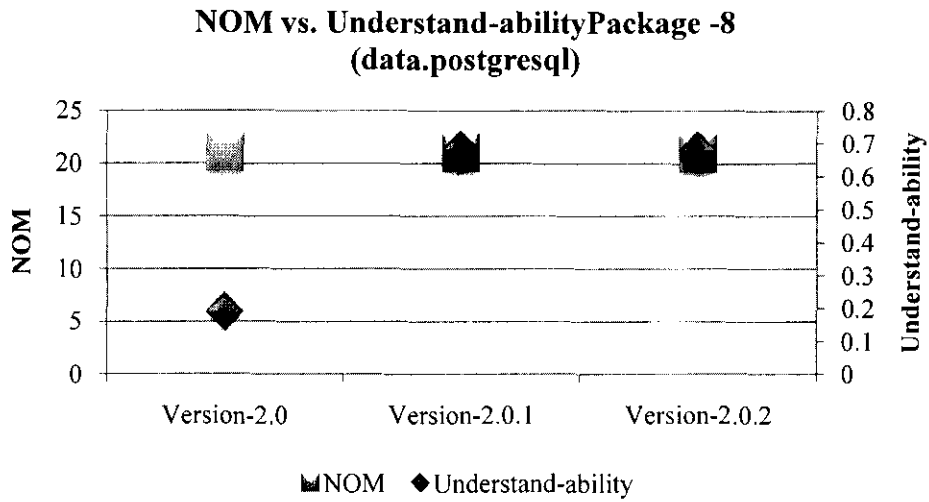
The values of understandability, LOC, NOM and comments are presented in this section. A significant increase in understandability value can be observed in version 2.0.1. This increase in the value of understandability is due to the decrease in LOC and significant increase in the number of comments.

Version wise values of understandability and its attributes (package-8)

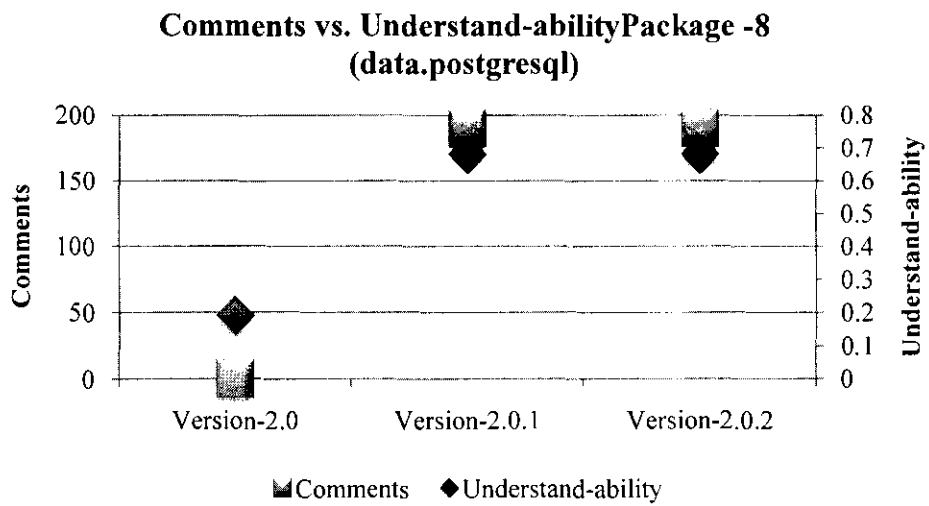
data.postgresql		Versions		
	2.0	2.0.1	2.0.2	
LOC	200	194	194	
NOM	21	21	21	
Comments	0	190	190	
Understandability	0.19	0.68	0.68	



Graph plot of values of LOC and understandability package-8



Graph plot of values of NOM and understandability package-8



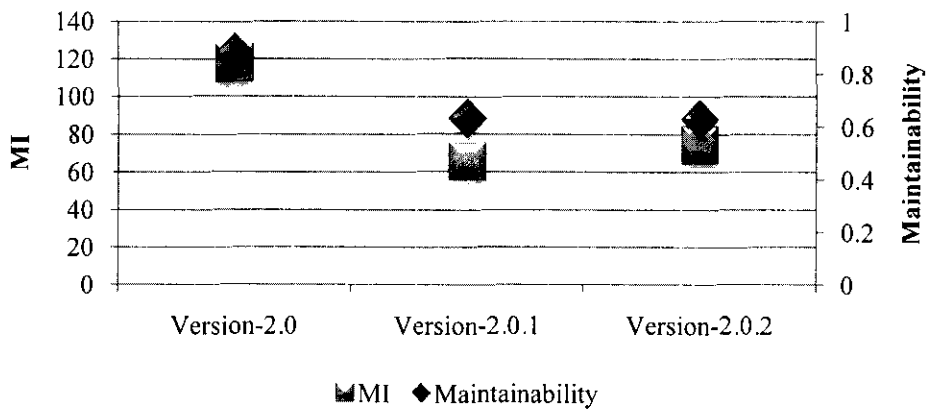
Graph plot of values of comments and understandability package-8

The values of MI, complexity and maintainability are presented in this section. A decrease in the value of maintainability can be observed in the version 2.0.1. This decrease is due to the decrease in the value of MI in version 2.0.1.

Version wise values of maintainability and its attributes (package-8)

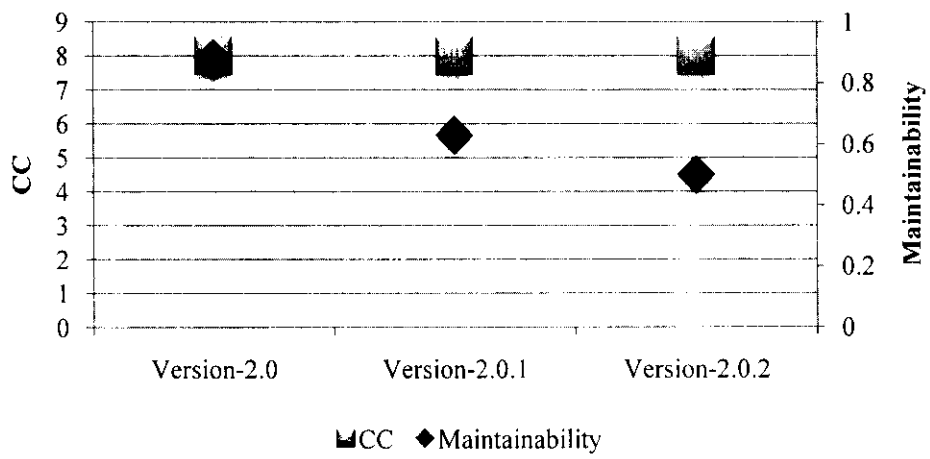
data.postgresql		Versions		
		2.0	2.0.1	2.0.2
MI	117.92	65.53	65.53	
CC	8	8	8	
Maintainability	0.88	0.63	0.63	

MI vs. Maintainability Package - 8 (data.postgresql)



Graph plot of values of MI and maintainability package-8

CC vs. Maintainability Package -8 (data.postgresql)



Graph plot of values of CC and maintainability package-8

J. Critical Values of the Pearson Correlation Coefficient r

n	$\alpha = .05$	$\alpha = .01$
4	0.950	0.990
5	0.878	0.959
6	0.811	0.917
7	0.754	0.875
8	0.707	0.834
9	0.666	0.798
10	0.632	0.765
15	0.514	0.641
20	0.444	0.561
25	0.396	0.505
30	0.361	0.463
35	0.334	0.430
40	0.312	0.403
45	0.294	0.380
50	0.279	0.361
60	0.254	0.330
70	0.235	0.306
80	0.220	0.286
90	0.207	0.269
100	0.196	0.256