An Integrated Method for Improving Remodularization in Software Systems using Probability Method

Bright Gee Varghese, Kumudha Raimond, Jeno Lovesum

Abstract*: Effective software system must advance to stay pertinent, however this procedure of development can cause the product design to rot and prompt essentially diminished efficiency and even dropped projects. Remodularization tasks can be performed to fix the structure of a software system and evacuate the disintegration brought about by programming advancement. Software remodularization comprises in rearranging software entities into modules to such an extent that sets of substances having a place with similar modules are more comparable than those having a place with various modules.However, re-modularizing systems automatically is challenging in order to enhance their sustainability. In this paper, we have introduced a procedure of automatic software remodularization that helps software maintainers to enhance the software modularization quality by assessing the coupling and attachment among programming components. For precision coupling measures, the proposed technology uses structural coupling measurements. The proposed methodology utilizes tallying of class' part capacities utilized by a given class as a basic coupling measure among classes. The interaction between class files measures structural connections between software elements (classes). In this paper, probability based remodularization (PBR) approach has been proposed to remodularize the software systems. The file ordering process is done by performing probability based approach and remodularization is done based on the dependency strength or connectivity among the files. The proposed technique is experimented on seven software systems. The efficiency is measured by utilizing Turbo Modularization Quality (MQ) that promotes edge weighing module dependence graph (MDG). It very well may be presumed that when comparing performance with the subsisting techniques, for instance, Bunch – GA (Genetic Algorithm), DAGC (Development of Genetic Clustering Algorithm) and Estimation of Distribution Algorithm (EDA), the proposed methodology has greater Turbo MQ value and lesser time complexity with Bunch-GA in the software systems assessed.

Index Terms*: Code Dependency, Dependency Matrix, Probability, Remodularization, Software System, Software System Maintenance, Turbo Modularization Quality.

I. INTRODUCTION

The structure of a software system primarily affects its practicality. To improve viability, programming frameworks are typically composed into subsystems utilizing the develops of bundles or modules.Notwithstanding, amid programming development the structure of the software system experiences consistent alterations, floating away from its unique plan, regularly decreasing its quality. In software engineering, programming upkeep in the created programming framework may require alterations to change the prerequisites of client. In such cases, support ends up fundamental. The product support process involves a circumstance of programming building execution that happen after the product has been conveyed to the client. The idea of programming support and development of frameworks was first proposed by Lehman [21], who did a few perceptions. One of the fundamental perceptions was that vast programming frameworks are never finished, proceed to develop and progressively complex some days. Large software systems are evolving difficult and complex to maintain due to the following problems:

- Constant changes: The software environment is therefore constantly changing and the software needs to be changed to function in the new environment.
- Increasing unpredictability: The structure of the program turns out to be progressively troublesome with steady change in code, therefore, some expectant advances must be taken to improve and streamline its structure.
- Large programming development: Software is a self-sufficient procedure. Programming traits, for example, size, time, and the quantity of blunders are practically steady for every framework discharge.
- Organizational dependability: The cost with which the product is created remains roughly consistent and is free of the assets given to the product improvement.
- Preservation of shared trait: During the season of program, added to it in each release, may be displayed.

To deal with those complexities of programming framework, one of the generally utilized systems called software remodularization which is an imperative segment in the software maintenance exercises. Software remodularization is a process to restructure and rebuild the existing software. Object oriented software modularization divides the software product into packages that contains several classes. Several kinds of package dependencies can be found in software systems.

Revised Manuscript Received on July 05, 2019.

Bright Gee Varghese R, Assistant Professor, Karunya Institute of Technology and Sciences, Coimbatore, India

Dr.KumudhaRaimond, Professor, Karunya Institute of Technology and Sciences, Coimbatore, India

Dr. JenoLovesum, Associate Professor, Presidency University, Bengaluru, India.

Intra-edge dependencies and inter-edge dependencies are two primary kinds ofdependencies. The intra-edges incorporate a wide range of interior dependencies between classes in a similar bundle or package, for example, invocation of function, composition and inheritance. The inter edge dependencies incorporate outside dependencies among classes which belong to different bundles. As elucidated of module dependency graph (MDG) in Fig. 1, the system consists of 2 modules with 3 cohesion factors such as (A-C, B-C, E-D) and 3 coupling factors such as (E-B, D-B, C-D). A large portion of the procedures rely upon the utilization of cohesion in every package and coupling among the packages in order to assess the quality of remodularization. The acceptable arrangements are those that expansion cohesion and lessen coupling. Cohesion of modules is, all in all, characterized by the quantity of intra-edges of a module and coupling as the quantity of between edges among the modules. Modules by their very natures ought to be profoundly strong much of the time. In programming dialects engineers need to import bundles or modules so as to have perceivability to the classes within them. This gives a characteristic form and parity that the module arrangement well. On the off chance that the modules were difficult to import, clients of those modules would either transform them, or solicitation that auxiliary variations are prepared. The objective is to endeavor to improve this organizing to make the framework simpler to keep up. Despite the reality that a big number of methodologies are sufficiently groundbreaking to provide remodularization provisions, there should be certain open problems in an attack to provide effective and entirely robotized remodularization.

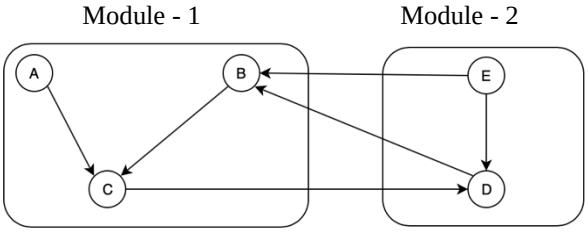


Fig.1 MDG incorporating two modules with 3 inter-edges and 3 intra-edges

II. LITERATURE SURVEY

In fact, a few researches tended to clustering issues so as not to improve existing modularizations, yet to locate the best deterioration of a framework as far as the modules. The first search based approach was employed by Mancoridis et al. [19] to deal with a problem of modularization with single objective approach. Their concept of distinguishing the modularization of a software system depends on the utilization of the heuristic hill-climbing search to augment cohesion and limit coupling. A similar method has been additionally utilized in Mitchell and Mancoridis [3], a tool which bolsters automatic system deterioration. Bunch performs the decomposition of the subsystem by partitioning the module dependency graph and relationships of the module dependence in a given source code. A fitness perform is employed for assessing the standard of the graph partition to find a balance between connectivity (i.e., dependence between two modules of different subsystems) and interconnectivity (i.e. dependency between modules in the same subsystem). Harman et al., [12] utilized a genetic algorithm to enhance the subsystem disintegration of a product framework. For maximizing the fitness function, a mix of value measurements, for example, coupling, cohesion, and multifaceted nature, is defined. So also, Seng et al., [16] contemplated the re-modularization process as a solitary target advancement issue utilizing genetic algorithm. The objective is to build up a strategy for item arranged frameworks that, beginning from existing subsystem deterioration, decides de-synthesis with better measurement esteems and less infringement of structure standards. A heuristic search-based approach, simulated annealing is proposed by Abdeen et al., [8] for reducing the dependencies among the modules or packages in a software system. Their optimisation technique is predicated on moving classes between the modules. Abdeen et al. suggested essential coupling and cohesion metrics arrangements to assess package arrangement in object-oriented software significant heritage. Numerous massive software systems which are object-oriented consisting of many classes that are dealt with into number of packages. The software modularisation components of such software systems cannot be contemplated as classes. Packages, in this case, not just class containers, yet they likewise assume the job of modules: a package ought to give well recognized services to the remainder of the product framework. The evaluation of package organization is therefore essential for the maintenance of software. Although many work has been carried out with the aim of achieving a single class quality and/or a quality of inter-class relationships, some works address some aspects of quality and relationship organization of packages. We believe there is a need for further investigations to evaluate aspects of package modularity. For multiple-objective optimization problems, Jaimes et al., [2] suggested a non-dominated genetic sorting methodology. Many objectives are the number of objectives i.e., greater than three. Non-dominated sorting genetic algorithm will optimize the objectives to the software re-modularization.

The legitimate opinion is that great modularization should show strong cohesion and low coupling. [14, 16]. Cohesion and coupling measurements were estimated using various metrics yet that all will in general depend on linguistic parts of the source code(with a few exceptions). The proclaimed favorable circumstances of a modular architecture incorporates [17]: handle multifaceted nature of a large software system; plan and create various pieces of same system by various individuals; partial testing; fix defective system components with distinct components without interfacing; control imperfection engendering; or, the re-use of existing components in various contexts. Certain coupling measurements were found to be high indicators of vulnerability (e.g. [1, 10]), and it has been shown that a model including coupling metrics is a successful maintenance effort indicator [5]. The rule of high cohesion, low coupling, can be elucidated in different ways [3]. High cohesion, for instance, semantically implies that all parts of the module share a same motivation, named singularity and the similarity of purposes [18]. Low coupling would indicate that

components of separate modules (or to a lower degree) do not share this reason for current ones. Since computer systems are not efficient at managing semantics, different elucidations, simpler to gauge, are normally liked, for instance based functional dependencies — one part calls an element of another segment—(for example [14, 15, 20], or on information get to (for example [10], [8]), orco-changes in a form control software system (for example [4, 7, 9]). [11] distinguished and composed in excess of thirty coupling measurements. We assessed our methodology on seven programming frameworks. We assessed our methodology on seven programming frameworks. Our outcomes demonstrate that our methodology fundamentally beats, in normal, existing methodologies regarding improving the structure, decreasing the quantity of coupling among modules and expanding the quantity of attachment inside the modules.

III. PROBLEM FORMULATION

Obviously the previous works did not take into account the time convergence factor for re-modularization during the product maintenance period from the literature study that was performed. At the point when an extensive programming is being advanced joining the changes, it expends a substantial intermingling time amid figuring out. In addition, to complete the remodularization, it is obligatory to comprehend the conditions that exist among various modules. Likewise, the force of coupling among records of various modules and attachment among the documents inside every module of the whole framework are to be resolved. Static code investigation apparatuses can be utilized to decide the reliance network of the modules and the records can be gathered dependent on contiguousness and availability factor. For littler programming frameworks, this can be accomplished physically or utilizing traditional calculations. Be that as it may, it is hard to deal with huge and complex framework along these lines. An ideal methodology is required to take care of this issue with less time and computational overhead. In this way, in view of the above said realities, a methodology has been structured in this work based on probability and a novel remodularizing approach. Fig. 2 portrays the mechanism used in the proposed strategy to remodularize the software system.

a) Generate MDG

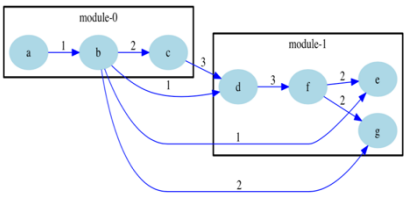


Fig. 2 Outline of Proposed Approach in Software Remodularization

The objective function is to boost the Turbo MQ that Mitchell suggested and utilized in BUNCH [13]. As characteristics of well-planned software systems, low coupling and high cohesion are regarded. On the off chance that μ_i is the quanti ty of edges within i[th] module and $\epsilon_{i,i}$ is the quantity of edges between modules, iand j, at that point the Turbo MQ can be resolved as

$$Turbo\ MQ = \sum_{i=1}^{k} MF_i \quad (1)$$

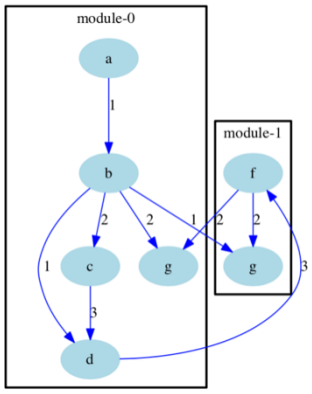
where MF_i is the factor for module i.

$$MF_i = \begin{cases} 0, & \mu_i = 0 \\ \dfrac{2\mu_i}{2\mu_i + \sum_{j=1, i\neq j}^{k} (\epsilon_{i,j} + \epsilon_{j,i})}, & otherwise \end{cases} \quad (2)$$

The purpose of the Turbo MQ is to improve the satisfactory design. i.e., restricting the coupling amid the modules and expanding the module cohesion. The greater the Turbo MQ, the closer the bundle accomplished is to an efficient program [13].

b) Extract Dependency Matrix from MDG

	a	b	c	d	e	f	g
a	0	1	0	0	0	0	0
b	0	0	2	1	1	0	2
c	0	0	0	3	0	0	0
d	0	0	0	0	0	3	0
e	0	0	0	0	0	0	0
f	0	0	0	0	2	0	2
g	0	0	0	0	0	0	0



c) Remodularized Software System using PBR

IV. PROPOSED APPROACH

The proposed framework is shown in Fig.3. The proposed approach has two phases. The primary phase establishes traversal of software system traversal by agents utilizing connectivity-based probability to observe the arrangement of files to be visited. In the subsequent phase, files are restructured using the results and dependency matrix of every agent to achieve a restructuring framework.

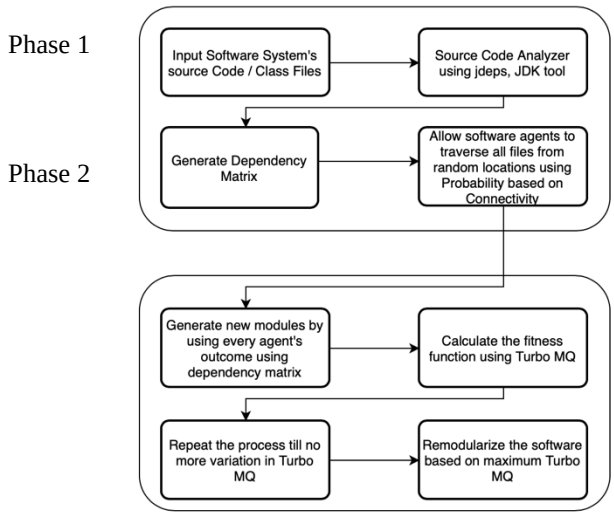


Fig. 3 Architectural diagram of the Proposed Approach

A. Proposed Algorithm

The following steps are provided for the proposed approach. The order wherein files are to be visited utilizing random based probability is specified in steps 4 to 10. The procedure of remodularization utilizing the ordered files and dependency weightage is specified in steps 11 to 15.

Step 1. Produce MDG by parsing the software system to be remodularized utilizing source code analysis tool.

Step 2. Create dependency matrix from MDG.

Step 3. Deploy software agents in random locations or files.

Step 4. Follow the steps for an agent to maneuver from current file to next file.

a. Locate all directly connected files from the present position of agent.

b. Compute the travel cost (TC) to see each neighbouring file as follows.

$$TC_{ij} = \frac{DW_{ij} * DT}{\sum_{j=1}^{M} DW_{ij}} \qquad (3)$$

where TC_{ij} is the actual cost to move from i^{th} file to j^{th} file

DW_{ij} is the weighted dependency from i^{th} file to j^{th} file.

DT is dependent type (DT is 1, if file j calls file i and DT is 0.5, if file i is called by file j

Fig. 4 shows the calculation of travel cost from file, F1 to its adjacent files.

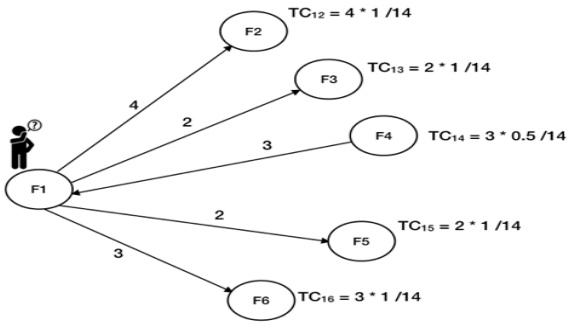


Fig. 4 Travel Cost from F1 to its adjacent files

a. The probability of moving from i^{th} file to j^{th} file, denoted by ρ_{ij} can be calculated as follows.

$$\rho_{ij} = \frac{1 / TC_{ij}}{\sum_{j=1}^{M} 1 / TC_{ij}} \qquad (4)$$

where j = 1…J…M

Fig. 5 indicates the probability of moving from file, i to its neighbouring files.

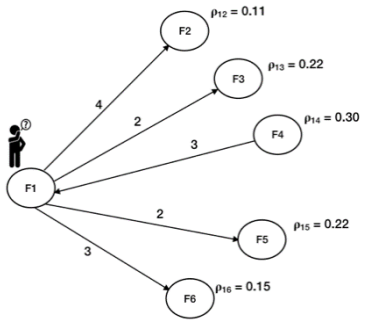


Fig. 5 Probability to move from F1 to its adjacent files

Step 5. Find cumulative probabilities.

Step 6. Create an arbitrary incentive somewhere in the range of 0.0 and 1.0.

Step 7. Find the closest lower vary and higher range from cumulative probabilities to the random value produced above.

Step 8. Find the lower range index which provides the subsequent file to visit.

Step 9. Continue from steps 3 to 8 until the agent visits all files of the software system.

Step 10. Process the outcome of an agent.

Step 11. Access the first file from the agent and mark it as currentFile.

Step 12. Create and mark a fresh module as the present module and add currentFile to the present module.

Step 13. Access next file from the agent.

Step 14. If currentFile has dependency with nextFile, then add nextFile into present module.

a. Change nextFile as currentFile and access next file from agent

b. Continue from step 14.

Step 15. If currentFile does not have dependency with nextFile, then

a. If more file is available in ant, then

i. Change nextFile as currentFile and access next file from agent.

ii. If the currentFile relies on nextFile, generate a fresh module and label it as the present module.AddcurrentFile and nextFile to present module. Continue from step 14(a).

iii. If currentFile has no nextFile dependency, attach nextFile to leftOverList.

b. If more file is available in ant, then change nextFile as currentFile and access next file from agent. Continue from step 15.a.ii.

c. If there is no more file to visit in the agent, then

i. By utilizing the dependency matrix, check the reliance of each file in leftOverList with the newly created modules.

ii. Insert file in the maximum dependence module.

Step 16. Check the cohesion of every module. If any module has cohesion value less than or equal to one, add the files of that module into leftOverList and continue from Step 15 c.

Step 17. Calculate the quality of remodularizing outcome produced by every agent using Turbo MQ value based on the equations (1) and (2).

Step 18. Rehash from step 3 till no more variety in maximum Turbo MQ value

V. RESULT AND DISCUSSION

The proposed approach is assessed on different software systems as shown in Table 1.

Table1. Benchmark data used for experimenting with the proposed approach

Sl No	Software Systems	No. of Files
1	compiler	13
2	nos	16
3	boxer	18
4	ispell	24
5	cia	38
6	ApacheAnt	195
7	EclipseJDT	432

Table 2 shows the comparison of results obtained by various approaches such as Bunch – GA [13], DAGC [22] and EDA [23]. For the software systems compiler and Boxer, our approach and the existing approaches give same Turbo MQ. As the number of files and number of connections increases in the evaluated software systems, the proposed approach outperforms the existing approaches shown in Table 2. Fig.6 portrays the improved aftereffects of the proposed methodology when contrasted with Bunch - GA, DAGC and EDA.

Table 2. Comparison of results obtained by various approaches such as Bunch - GA, DAGC and EDA with the proposed approach.

	Bunch-GA		DAGC		EDA		PBR	
Software Systems	**No. of clusters**	**Turbo MQ**	**No. of clusters**	**Turbo MQ**	**No. of clusters**	**Turbo MQ**	**No. of clusters**	**Turbo MQ**
compiler	4	**1.506**	4	**1.506**	4	**1.506**	4	**1.506**
boxer	7	**3.101**	7	**3.101**	7	**3.101**	7	**3.1**
ispell	7	2.177	8	1.997	6	2.19	6	**2.21**
cia	14	2.706	19	1.833	12	2.787	6	**4.1**
nos	5	1.636	5	1.606	5	1.636	4	**1.665**

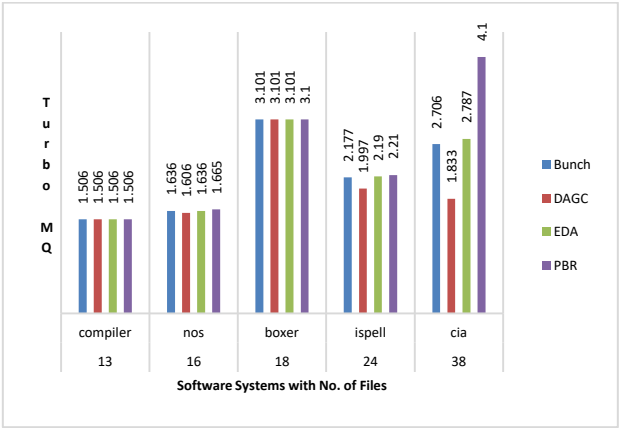


Fig. 6 Graph representation for software system vs Turbo MQ

This approach gives better solution in the software systems ApacheAnt and EclipseJDT with more than 150 files which is shown in Table 3. For the software systems, ApacheAnt and EclipseJDT, the proposed approach yields more number of clusters. The number of clusters is not directly proportional to Turbo MQ. On the off chance that there is less cohesion with progressively number of modules, it influences the Turbo MQ esteem contrarily based on equations (1) and (2). Fig.7 depicts the improved eventual outcomes of the proposed strategy when compared with Bunch - GA.

Table 3 Comparison of test results with Bunch - GA by using software systems with more than 150 files.

Software Systems	Bunch - GA		PBR	
	No. of clusters	Turbo MQ	No. of clusters	Turbo MQ
ApacheAnt	8	6.08	24	**15.79**
EclipseJDT	15	17.4	62	**35.85**

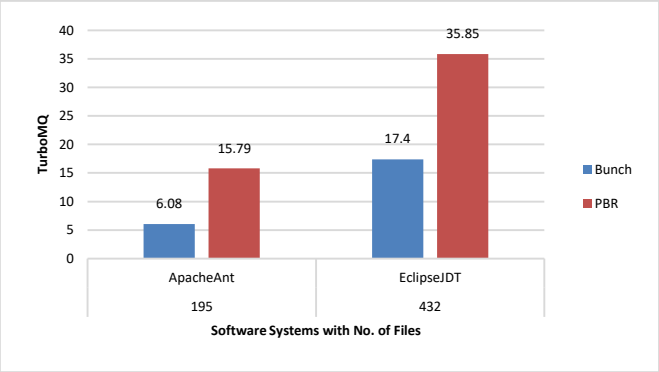


Fig. 7 Comparison of BUNCH - GA with software of 150 files

For software applications such as ApacheAnt and EclipseJDT, the amount of Turbo MQ assessments is checked. The proposed approach is shown to provide better Turbo MQ with fewer Turbo MQ assessments, which is shown Fig. 8. It clearly shows that our approach has better time convergence comparing with Bunch - GA.

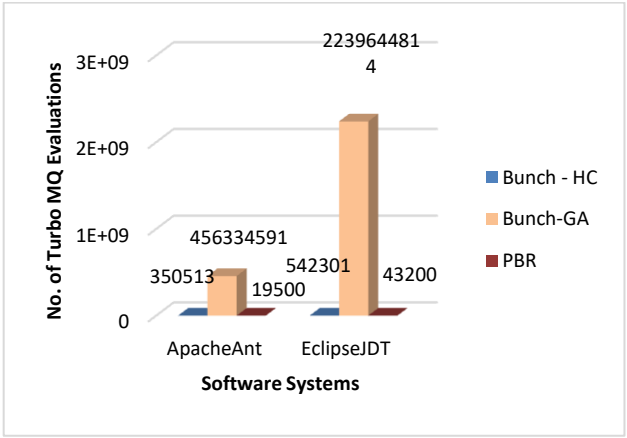


Fig. 8 Performance analysis of Turbo MQ value of proposed approach with software systems, ApacheAnt and EclipseIDT

VI. CONCLUSION AND FUTURE WORK

Modularizing a product framework profits to sort out the advancement in an increasingly viable way, coordinate alterations easily, complete testing and investigating effectively and productively, and to lead upkeep work without adversely influencing the working of the product. It is basic to keep up high attachment and less coupling which are the fundamental standards of modularization. Subsequently in this work we have proposed and shown the execution of programmed remodularization of programming frameworks utilizing likelihood technique. The proposed methodology is assessed utilizing different programming frameworks and the outcomes acquired are turned out to be increasingly productive when contrasted and the current methodologies like Bunch - GA, DAGC and EDA. It has been seen that the proposed methodology is reasonable for little and extensive programming frameworks and produced better Turbo MQ values, which is the software package quality parameter and lesser time complexity with Bunch-GA. For the future work, semantic connections and history of programming upkeep will be utilized in figuring of association quality that can prompt better nature of the improvement procedure while protecting the first structure up to greatest.

REFERENCES

1. [1] A. B. Binkley and S. R. Schach, "Validation of the coupling dependency metric as a predictor of run-time failures and maintenance measures", in ICSE '98: Proceedings of the 20th international conference on Software engineering. Washington, DC, USA: IEEE Computer Society, 1998, pp. 452–455.
2. [2] A. L. Jaimes, C. A. CoelloCoello, and J. E. U. Barri-entos, "Online objective reduction to deal with many-objective problems", in Proceedings of the 5th Inter-national Conference on Evolutionary Multi criterion Optimization, 2009, pp. 423–437.
3. [3] B. S. Mitchell and S. Mancoridis, "On the automatic modularization of software systems using the bunch tool", IEEE Transactions on Software Engineering, 2006, pp. 193–208.
4. [4] D. Beyer and A. Noack, "Clustering software artifacts based on frequent common changes", in IWPC '05: Proceedings of the 13th International Workshop on Program Comprehension, Washington, DC, USA: IEEE Computer Society, 2005, pp. 259–268.
5. [5] F. B. Abreu and M. Goulao, 'Coupling and cohesion as modularization drivers: Are we being over-persuaded?' in CSMR '01: Proceedings of the Fifth European Conference on Software Maintenance and Reengineering, Washington, DC, USA: IEEE Computer Society, 2001, pp. 47–57.
6. [6] G. A. Hall, W. Tao, and J. C. Munson, "Measurement and validation of module coupling attributes", Software Quality Control, Vol. 13, No. 3, 2005, pp. 281–296.
7. [7] H. Gall, M. Jazayeri, and J. Krajewski, "Cvs release history data for detecting logical couplings", inIWPSE '03: Proceedings of the 6thInternational Workshop on Principles of Software Evolution.IEEEComputer Society, 2003, pp. 13–23.
8. [8] H.Abdeen, S. Ducasse, H. Sahraoui, and I. Alloui, "Automatic package coupling and cycle minimization", in Proceedings of the 16th Working conference on Reverse Engineering, IEEE, 2009, pp. 103–112.
9. [9] J. Davey and E. Burd, "Evaluating the suitability of data clustering for software remodularization", in WCRE '00: Proceedings of the Seventh Working Conference on Reverse Engineering (WCRE'00), Washington, DC, USA: IEEE Computer Society, 2000, p. 268.
10. [10] L. Briand, P. Devanbu, and W. Melo, "An Investigation into Coupling Measures for C++", in ICSE '97: Proceedings of the 19th international conference on Software engineering, New York, NY, USA: ACM, 1997, pp. 412–421.
11. [11] L. C. Briand, J. W. Daly, and Jürgen Wüst, "A Unified Framework for Cohesion Measurement in Object-Oriented Systems", Empirical Software Engineering: An International Journal, Vol. 3 No. 1, 1998, pp.65–117.
12. [12] M. Harman, R. M. Hierons, and M. Proctor, "A new representation and crossover operator for search based optimization of software modularization", in Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann Publishers Inc., 2002, pp. 1351-1358.
13. [13] Brian S. Mitchell and Spiros Mancoridis, "On the evaluation of the Bunch search-based software modularization algorithm", Soft Computing, Vol. 12 No. 1, 2008, pp.77–93.
14. [14] N. Anquetil and T. Lethbridge, "Comparative study of clustering algorithms and abstract representations for software remodularization", IEEE Proceedings - Software, Vol. 150 No. 3, 2003, pp. 185–201.
15. [15] N. Anquetil and T. Lethbridge, "Experiments with Clustering as a Software Remodularization Method", in Proceedings of Working Conference on Reverse Engineering

(WCRE'99), 1999, pp. 235–255.
16. [16] O. Seng, M. Bauer, M. Biehl, and G. Pache, "Search-based improvement of subsystem decompositions", in Proceedings of the 7th annual conference on Genetic and evolutionary computation (GECCO'05), ACM Press, Washington DC, USA, 2005, pp. 1045–1051.
17. [17] P. Bhatia and Y. Singh, "Quantification Criteria for Optimization of Modules in OO Design", in Proceedings of the International Conference on Software Engineering Research and Practice & Conference on Programming Languages and Compilers, SERP 2006, Vol. 2. CSREA Press, 2006, pp. 972–979.
18. [18] R. Sindhgatta and K. Pooloth, "Identifying software decompositions by applying transaction clustering on source code", in COMPSAC '07 : Proceedings of the 31st Annual International Computer Software and Applications Conference, Washington, DC, USA: IEEE Computer Society, 2007, pp. 317–326.
19. [19] S.Mancoridis, B. S. Mitchell, C. Rorres, Y. F. Chen, and E. R. Gansner, "Using automatic clustering to produce high-level system organizations of source code", in Proceedings of the International Workshop on Program Comprehension, 1998, pp. 45–55.
20. [20] W. Li and S. Henry, "Object oriented metrics that predict maintainability", Journal of System Software, Vol. 23, No. 2, 1993, pp. 111–122.
21. [21] M.M. Lehman, "Programs, life cycles, and laws of software evolution", Proceedings of the IEEE, Vol. 68, No. 9, 1980, pp. 1060 - 1076.
22. [22] Saeed Parsa, Omid Bushehrian, "A Framework to Investigate and Evaluate Genetic Clustering Algorithms for Automatic Modularization of Software Systems", in International Conference on Computational Science, 2004, pp. 699-702.
23. [23] MahjoubehTajgardan, Habib Izadkhah, Shahriar Lotfi, "Software Systems Clustering Using Estimation of Distribution Approach" in Journal of Applied Computer Science Methods, 2016, pp. 99–113.

AUTHORS PROFILE

Mr. Bright Gee Varghese, Assistant Professor in Department of Computer Science and Engineering, Karunya Institute of Technology and Sciences. He completed his Bachelor of Engineering from ManonmaniumSundarnar University, Tamil Nadu in 2003. Later, continued his Master of Engineering in Computer Science and Engineering and completed it in 2011. Currently pursuing PhD in Computer Science and Engineering. His area of research is Software Engineering. He is passionate in learning cutting edge technologies and mobile application development. He has 13 years of experience in college as well as university level. He has completed industry recognized certification, Oracle Certified Professional Java Programmer. His subject expertise involves C, C++, Java SE, Java EE and Android.

Dr.KumudhaRaimond is research oriented who earned her Ph.D. from Indian Institute of Technology (IIT) Madras, India. Currently she is working as Professor in the Department of Computer Science and Engineering, Karunya Institute of Technology and Sciences, Coimbatore. Her research focus is on the development of efficient models using hybrid intelligent techniques for various applications in the areas of biometrics, biomedical, bioinformatics, compression, watermarking, etc. Her further areas of interest are Big Data Analytics, Satellite Image Processing, Watermarking, Wireless Sensor Networks, and Compression and Image Retrieval. She has a good number of research publications in peer reviewed national and international journals, proceedings of international conferences and book chapters to her credit. Besides having 20 years of teaching experience, she also has 3 years of MNC experience at John F. Welch Technology Research Centre, a Research wing of General Electric (GE), Bangalore. She worked as an Energy System Analyst in the Remote Monitoring and Diagnostic Lab of GE, was involved in analysing and predicting the status of remote machines such as steam turbines and locomotives.

Dr. JenoLovesum is currently working as Associate Professor in CSE department at Presidency University,Bangalore. She Completed her Bachelor of Engineering from ManonmaniumSundaranar University in the year 2001 and later continued her Master of Engineering at Annamalai University, Chidambaram and completed it in the year 2005. She pursued her Ph.D in at Anna University, Chennai and completed her Ph.D in the year 2015. . Her research focus is on Cloud Computing and Software Engineering. In international meetings and publications, she has published a number of articles. Teaching has been an important part of her career and she has teaching experience of 15 years at college and university level. Her subject expertise involves software Engineering and software testing. Currently she is continuing her research in the area of cloud based IoT applications.