Research Article

A Multi-Objective Dragongfly Optimization for Requirement Traceability Establishment

Swathine.K[a], and Dr. N. Sumathi[b]

[a]
Research Scholar, Department of Computer Science, Sri Ramakrishna College of
Arts and Science, Coimbatore, Tamil Nadu, India.
[b]Professor and Head, Department of Information Technology, Sri Ramakrishna College of Arts and Science, Coimbatore, Tamil Nadu, India.

Article History: Received: 11 January 2021; Accepted: 27 February 2021; Published online: 5 April 2021

Abstract: Software traceability is a crucial component of various exact software development process and it is needed for various component certification and approval process in security system. With the tremendous growth of system, traceability is considered as a recent research topic. The traceability is a software development process that is indefinable. Various manufacturers struggle in predicting the appropriate traceability degree for their needs and produce the appropriate set of traceability links. The effort, cost, and discipline have to be maintained with tracking links with the faster development of software systems that are extremely higher. Also, it produces various advantages in practical realization; as it can be either ad-hoc or not properly defined traceability process, produces poor training or lack of effectual tool support. Moreover, the traceability process has to be determined as it can diminish the development effort and to enhance the development process. Generally, traceability research is based on empirical investigations for exploring newer investigational queries or to compute newer tracing methods. Here, this work concentrates on traceability, functional requirements, link establishment. It shows a better trade-off among the prevailing approaches.

Keywords: Software traceability, functional requirements, degree of traceability, software development, transparency

1. Introduction

In general, software systems are evolved to deal with the various changing requirements. Based on the lifetime of software, newer requirements can be added with prevailing requirements [1]. It can be either dropped or modified. The processes of designing the internal structure of software’s are extremely complex which can drastically degrades the performance of system and reduces the software quality. As an outcome, software systems are constantly needed for maintaining the quality and to reduce the complexity [2]. It is also needed for enhancing the internal structure of refactoring. The software refactoring process concentrates on enhancing the software quality by adopting the variations in internal structure of the software. This cannot be altered based on the external behavior of systems used.

The refactoring process of software is extensively utilized to reduce the flaws in software design and to enhance the maintainability, software quality, extensibility, and reusability [3]. The underlying concept of software refactoring is rearranging the software elements like classes, variables, and approaches to assist future extensions. Typically, the process of software maintaining is based on following process: initially, piece of code is needed for enhancing the code smell and it needs to be identified [4]. Subsequently, based on the code smells and its types, certain solutions are adopted to provide the better outcomes.

At present, there are various powerful refactoring tools that are designed for automating the software refactoring procedure, for instance,iPlasma or JDeodorant [5]. However, there are some mainstream software development processes that are equipped with various refactoring abilities (Microsoft visual studio, and eclipse). There are some diverse tools that makes refactoring easier and to reduce the error [6]. For the provided code smell which is automatically identified, some refactoring tools can recommend various refactoring outcomes. Moreover, decision based on these solution needs to be implemented which is left by the developers [7]. Decision needs to be selected for finest solutions that are challenging tasks. For facilitating the developers with appropriate selection and decision making, prevailing tools are applied for diverse approaches. The general way is to evaluate and to choose an alternative outcome based on the influence of code metrics. With the code metrics, some parameters need to be determined. This process needs the consideration towards certain parameters [8]. From this approach, the assumption needs to provide better solutions to provide finest source code metrics. Based on the pertinent code metrics, ranking and quantifying solutions are used to establish cohesion and coupling [9]. Generally, coupling is depicted as the inter-dependences of part where it specifies the internal dependencies of design. It is measured based on the instance variable field which utilized by the techniques inside among the classes.

These techniques eliminate the fact of distinguishing the refactoring solutions which shows drastic impact on over the traceability among the implementation and requirements (source code) [10]. Generally, software system is made of various artifacts at distinct abstraction levels. These artifacts are related with traceability [11].

Generally, traceability is depicted as, "the competency to follow and determine the requirement lifetime in both backward and forward directions [12]. Generally, traceability information is preserved over the traceability matrix ™ which has the intention to project the appropriate links among the lower-level and higher-level entities [13]. Traceability is considered to be more critical for software maintenance and evolution. Investigators have anticipated various approaches for ensuring completeness and correctness of traceability information [14]-[15]. These techniques concentrate on assessment, retrieval, and traceability link maintenance.

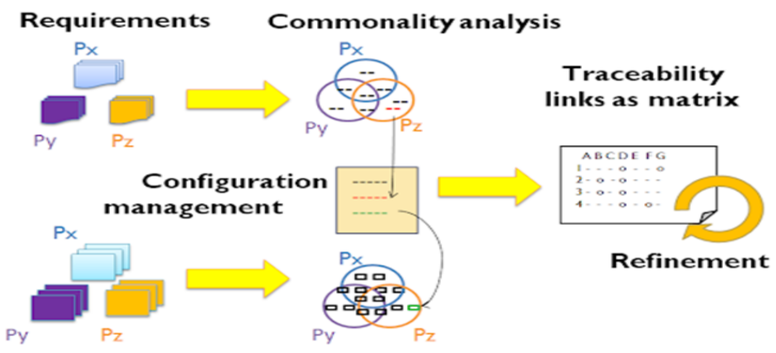


Fig 1: Generic view of traceability

The remainder of the work is organized as: Section 2 explains the background study of the work; section 3 is the methodology related with the traceability requirements of software design; section 4 is the numerical results and discussions; section 5 is the conclusion with future research directions.

2. Related works

This section deals with the traceability factor related to the software development. It is not a trivial matter as it handles various activities that are essential for both creating and maintaining the traceability relationship. There are some methodologies that are specific with traceability that specifies how they create, maintain, and re-use it when it is not available generally [16]. The significant cause for the un-availability of higher variance in development process has to be analyzed with this process. Nonetheless, some common process is essential for specifying the artifacts that are associated with it. Some information has to be utilized to maintain the constant set of traceability relationship among the developers [17].

Author in [18], partitions the traceability production relationship for maintenance, registration, and perception. The author specifies the registration of establishing, creating, or installing traceability relations. These factors are utilized for inter-changeability. As well, traceability updation and maintenance and traceability association and links are utilized for synonymously.

Recently, there are diverse investigators who are more concentrated towards the techniques required for automated creation and identification of traceability association [19]. Enormous amount of methods are applied for information retrieval and text mining approaches to predict candidate relations. With the provisioning techniques, the manual intervention has to prune candidate relation that cannot be eliminated completely. It is so viable and easier to rely over the automated trace generation based on demand and required for processing [20]. However, it needs to substantially beneficial for these approaches for removal for constant re-confirmation of candidate relations. A substitution is to produce a quality set via these approaches and manual pruning and to concentrate on preserving them. This work concentrates on various strategies. Similarly, there is very lesser work that concentrates on automated traceability relationship maintenance [22]. The purpose of traceability is to maintain and to prevent decay that is related to various artifacts evolution. This refers to the maintenance of most confronting traceability aspects.

Author in [23] categorizes the traceability maintenance problem among the source and architectural elements that is provided as an initial set for establishing traceability links and provide both implementation and architecture which is evolved independently. The traceability links are updated with the addition of some newer links, the removal of prevailing links and variation is the links that are available already. This can ensure the architectural components over time for accurately connecting it with appropriate source code configuration items and vice verse. Devoid of any maintenance, the traceability relationship among the elements is lost and it specifies the false dependencies.

The step-by-step degradation of these relationship leads to decay traceability. This is eliminated by constant or on-demand traceability maintenance [24]. With this on-demand maintenance, it provides the theoretical advantages that are connected with updation based on the present model state which offers potentially lesser incremental updation step when compared to constant maintenance. Subsequently, the need for traceability update requires longer time after the model variation that cause maintenance need and it is harder to carry out than continuous maintenance. From the theoretical view point, both these options have advantages and disadvantages that highlight the need for empirical studies over this region. It is essential for accessing the traceability quality independently by establishing the relations. As an outcome, the results follow two strategies based on shared set of traceability facilitate the demand for tasks that impacts the change management and analysis [25]. However, the blending value continuous and some on-demand approaches are provided to maintain the area of research.

The above mentioned factors explain why this traceability maintenance as it is essential during the process of evolution and development of software system. Also, it discusses about the weakness and strength of prevailing methods towards various problem to put forwards the context of anticipated model.

3. Methodology

The need for this research work is to follow and to determine the requirement of life from its origin realization that give rise to requirement traceability which is generalized with various types of artifacts such as design model, code, and with test cases. The, traceability is regarding the connecting dots that is related to software artifacts to deal with both development process and software products.

This process is more tedious and error-prone those generates and maintain the manual information traceability, automated support is provided based on information retrieval. These techniques facilitate the traceability link recovering that relies over textual descriptions on target artifacts and query requirements. Various investigators have adopted the information retrieval techniques in automated tracing process. The empirical outcomes depict that this techniques show that these process have efficient level of comparison. The efficiency is evaluated based on metrics like RTM, color code, ratio analysis, and requirement missing.

The pre-requisite of measuring this objective relies over the optimization approach that pretends to given multi-objective solution. Therefore, this work concentrates on implementing the optimization approach that provides global results. The optimization approach used here is dragonfly optimization to evaluate the missing requirements, ratio analysis, color code, and RTM that provides the appropriate trace links. Hence, the higher performance metrics deals with the complete coverage of appropriate links with trace retrieval techniques with better accuracy. The information requirement is based on the tracing methods with the provided candidate links. The values are attributed with the tracing target size. The medium code is based on enormous artifacts and retrieval method returns the code artifacts. In contrary, certain information retrieval applications rely over the search of enormous retrieval with higher performance which is not so feasible.

The baseline cause of trace retrieval approaches provides lower precision with the length of query requirements. The trace query can be a single sentence or numerous sentences (use case or user story) written with natural language. When compared to other query formats, longer queries are used for irrelevant link access with retrieval of reduced precision. The section given below discusses about the Dragonfly method for handling the multi-objective constraints related to requirement traceability.

a. Multi-objective Dragonfly for traceability requirements

Dragonfly algorithm is designed by Mirjalili which is a meta-heuristic optimization approach which is based on swarm intelligence designed based on dragonfly's behavior. There are two essential optimization stages known as exploitation and exploration. This model is either static or dynamic for searching food and eliminating the enemies. The intelligence emerges from feeding and migration. The separation concept avoids static collusion with the neighbors 1) alignment specifies speed where the agents are matched with the neighbors; 2) cohesion concept based on tendency of the individuals towards the centre. There are three preliminary nature of DA: avoiding enemy and moving towards food source. The behavior of DA is mathematically expressed as:

$$S_i = -\sum_{j=1}^{N} X - X_j \tag{1}$$

$$A_i = \frac{\sum_{j=1}^{N} V_j}{N} \tag{2}$$

$$C_i = \frac{\sum_{j=1}^{N} X_j}{N} - X \tag{3}$$

$$F_i = X^+ - X \tag{4}$$

$$E_i = X^- + X \tag{5}$$

Here, $'X'$ is instantaneous individual position, X_j is instantaneous position of j^{th} individual, $'N'$ is number of neighborhood individuals, V_j is speed of neighborhood individuals, X^+ and X^- is food source location and enemy source respectively. The position and step vector is updated based on expression given below:

$$\nabla X_{t+1} = (sS_i + aA_i + cC_i + fF_i + eE_i) + w\nabla X_t \tag{6}$$

$$X_{t+1} = X_t + \nabla X_{t+1} \tag{7}$$

Here, $'s^{'}, 'a', 'c'$ is separation, alignment, cohesion coefficients respectively. Similarly, $'f^{'}, 'e', 'w^{'}, 't'$ are food factor, enemy factor, inertia coefficient, and iteration number respectively. In dynamic swarm, DF pretends to align flight. Thus, the alignment coefficient is higher and cohesion coefficient is lower in exploration process, in exploitation process. The alignment coefficient is lower and co-efficient cohesion is high. For enhancing the randomness, the probabilistic behavior is based on neighborhood solution. Accordingly, the dragonflies are position is expressed as below:

$$X_{t+1} = X_t + Levy(d) * X_t \tag{8}$$

$$Levy(x) = 0.01 * \frac{r_1 * \sigma}{|r_2|^{\frac{1}{\beta}}} \tag{9}$$

$$\tau(x) = (x-1)! \tag{10}$$

Here, $'d'$ is size of position vector, $'r1'$ and $'r2'$ are random numbers in range [0,1] and β is constant value. The improved DA is expressed as below:

$$X_{t+1} = X_t + h * rand\ () \tag{11}$$

$$h = \sqrt{\frac{T}{N}} \tag{12}$$

$$N = 100 * T \tag{13}$$

$$P_g = \frac{1}{h\sqrt{2\pi}} \exp\left(-\frac{(dimension - agents)^2}{2h^2}\right) \tag{14}$$

Here, $'T'$ is the motion time period (Sec), $'T'$ value is 0.01.

```
Algorithm:
1.  Initialize DA for population and step vectors, archive size, number of segments
2.  While condition is not satisfied
3.  Compute objective values
4.  Predict parent optimal conditions over non-dominated solutions
5.  While condition is fulfilled
6.  Run for new solution with $D = number\ of\ attributes - sum\ of\ missing\ attribute\ values$
7.  If ($D == N$) the
8.  Good requirements
9.  Else
10. Mission requirements // ($D! = N$)
11. End if
12. End while
13. While solutions are not segmented
14. Update solution segment
15. End while
16. Choose food and enemy source
17. If there are neighbours for DA
18. Update position and velocity
19. If there are least DA in search space
20. Terminate
21. Else
22. Move to neighbourhood
23. End if
24. Else
25. Move to search space
26. End if
27. End while
```

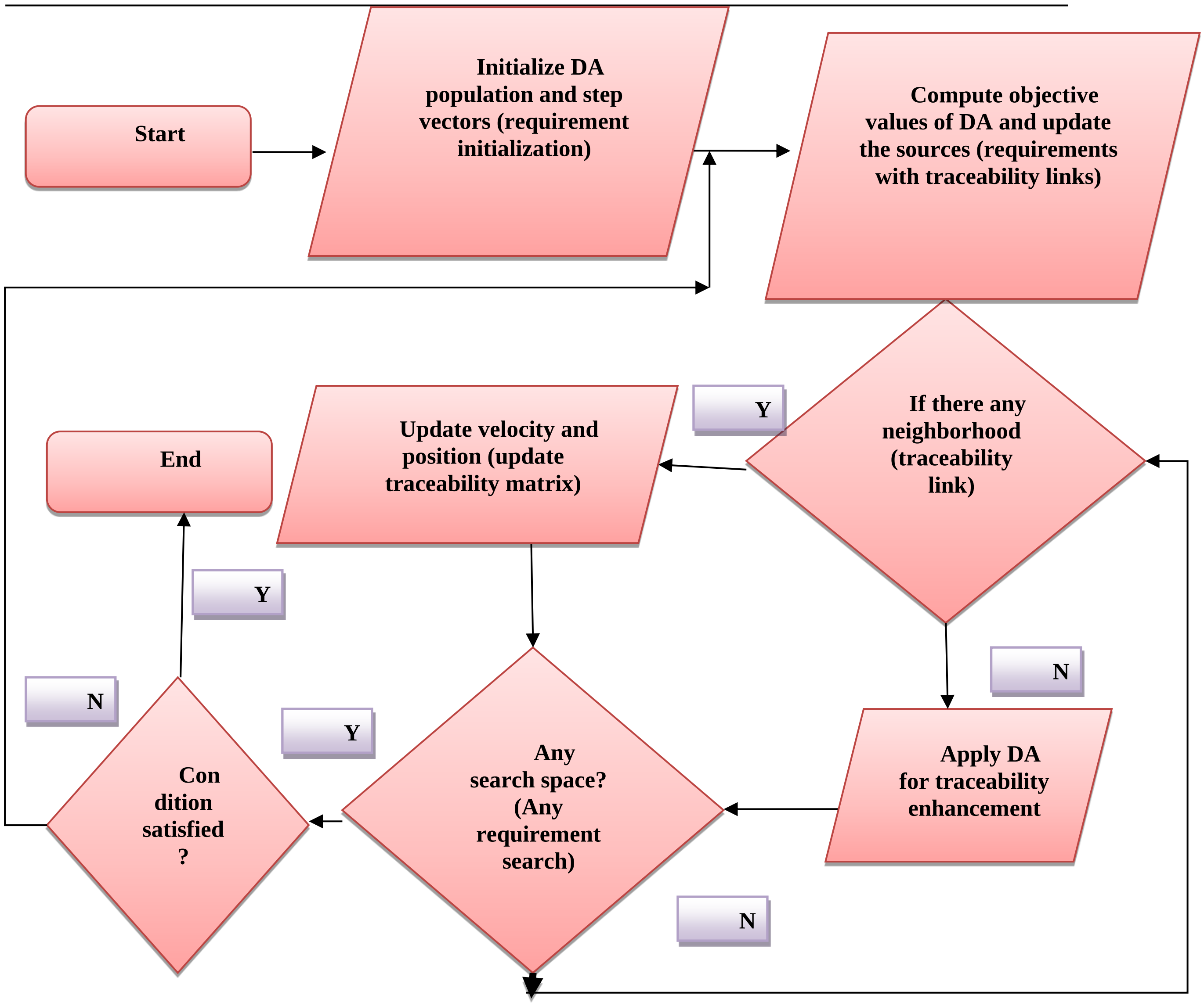

Start
Initialize DA population and step vectors (requirement initialization)
Compute objective values of DA and update the sources (requirements with traceability links)
If there any neighborhood (traceability link)
Y
N
Update velocity and position (update traceability matrix)
Apply DA for traceability enhancement
Any search space? (Any requirement search)
Y
N
Con dition satisfied ?
Y
N
End

Fig 2: Flow diagram of proposed model

b. Requirement analysis

The requirement analysis is performed by recording the user's feedback list which is chosen from the application observation. There are seven different functional requirements. They are:

1) Software has the ability to add, edit, and delete certain accounts.
2) Software has the ability to implement the plan.
3) Software has the ability to show the recapitulation of every account.
4) Software has the ability to explore data into file.
5) Software has the ability to show and add account.
6) Software has the ability to show and add categories.
7) Software needs to set active and non-active PIN and set the application settings.

The requirement modeling is performed with object-oriented and UML concepts. Also, activity diagram also included for use case scenario. The relationship among the classes in every class is shown in robustness diagram. After performing the complete development documentation, the testing is performed based on template, filling the template, checking the content of traceability matrix. The performance is performed with forwards traceability over functional requirements and to validate every requirement based on every development phase. The format for the complete traceability matrix is provided as below in Table I:

Table I: Traceability matrix

Functional requirements	Use case	Activity diagram	Architectural diagram	Sequence diagram	Database Table	User Interface	Test case	Verified
---	---	---	---	---	---	---	---	---

The test application is based on use case with accurate traceability matrix. It is performed with following phases:
1) Model a standard template for matrix content. This can fulfills the logical and consistent and assists in decision making.
2) The template is modeled with data from requirement catalog and documents.
3) Cross refer the matrix requirement to other model and solution component are produced as the modeling progresses and developed.
4) The references insert to test data into requirements traceability matrix to validate the requirements.

High-level requirements

HLR 01
Menu selection; control and shortcut key; loading menu selection

HLR 02
Address book for store contacts

HLR 03
System should be provided with explanation; item on screen based on requirements

Lower-level requirements

LLR 01	
Name	Store contact information
Summary	Address book should maintain some details
Description	1. Command terminal 2. address book 3. comment address and blank fields 4. save entry

LLR 02	
Use case name	Access system
Summary	User accesses help system
Description	Help key

After performing the set of phases, it is noted that the use of traceability matrix is performed over the application with DA optimizer. The outcomes attained from these searches for all the requirements are validated on every software development artifacts which commences from requirement analysis, interface, traceability link until the test case is established.

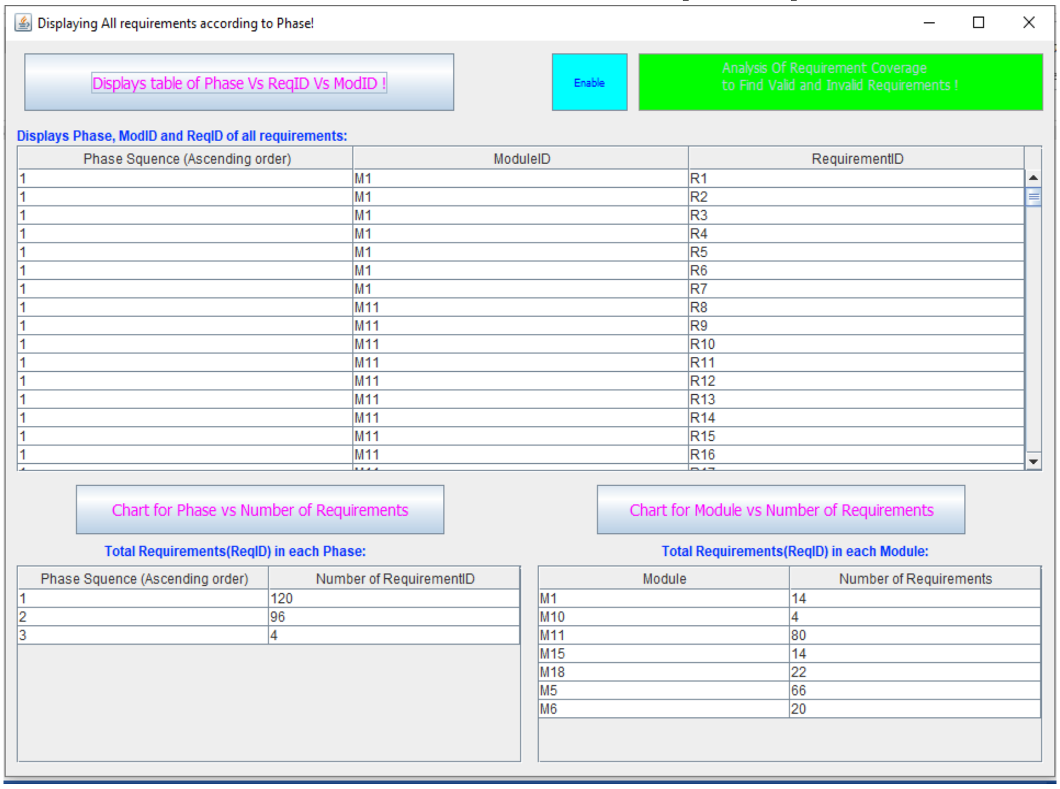
4. Numerical results

This section discusses about the simulation setup for carrying out this investigation. Here, Eclipse is utilized for simulation environment with Intel core I5 processor, 64 bit OS with 8GB RAM which is used for execution.

Some requirement information isbasedon features that are listed as below. 1) Cost 2) Time 3) Ease of Use 4) Ambiguous 5) Credible scores 6) Localization 7) Stakeholders rating 8) Compliance 9) Usability and benefits 10) Risk 11) Success rate 12) Integration 13) Maintainability 14) Security 15) Stability 16) Performance 17) Productivity. The target of this traceability is based on the solution to get rid of various inconsistencies. It is work-in-process. The requirement definition is based on quality, life time requirement. Similarly, traceability assists in re-engineering and reverse engineering. The traceability link along with priority provides better support for decision making. The decision is made based on gathering all the system requirements. It uses 18 attributes for computing weighted average and performs various priority requirements. Based on the priority requirements, optimizing the prioritized requirements are performed based on removal of invalid requirements. It is expressed based on following steps: 1) requirements are consistent, valid, and well-bounded. Therefore, it is ready for traceability. The DA optimization is applied for measuring the requirements with traceability matrix (RTM 1). It is expressed as:

Hashtable<key, value>DA technique for requirment<Module, Requirement>

The advantages of this model are: eliminating duplicates, eliminating null values/key values during plotting matrix. RTM with color code RTM 2 is termed as enhanced RTM and name complexity is overcome. Also, RTM with color code and the overcome of drawback of name complex and space.



Displaying All requirements according to Phase!
Displays table of Phase Vs ReqID Vs ModID !
Enable
Analysis Of Requirement Coverage to Find Valid and Invalid Requirements !
Chart for Phase vs Number of Requirements
Chart for Module vs Number of Requirements

Displays Phase, ModID and ReqID of all requirements:

Phase Squence (Ascending order)	ModuleID	RequirementID
1	M1	R1
1	M1	R2
1	M1	R3
1	M1	R4
1	M1	R5
1	M1	R6
1	M1	R7
1	M11	R8
1	M11	R9
1	M11	R10
1	M11	R11
1	M11	R12
1	M11	R13
1	M11	R14
1	M11	R15
1	M11	R16

Total Requirements(ReqID) in each Phase:

Phase Squence (Ascending order)	Number of RequirementID
1	120
2	96
3	4

Total Requirements(ReqID) in each Module:

Module	Number of Requirements
M1	14
M10	4
M11	80
M15	14
M18	22
M5	66
M6	20

Fig 3: Requirements Categorization

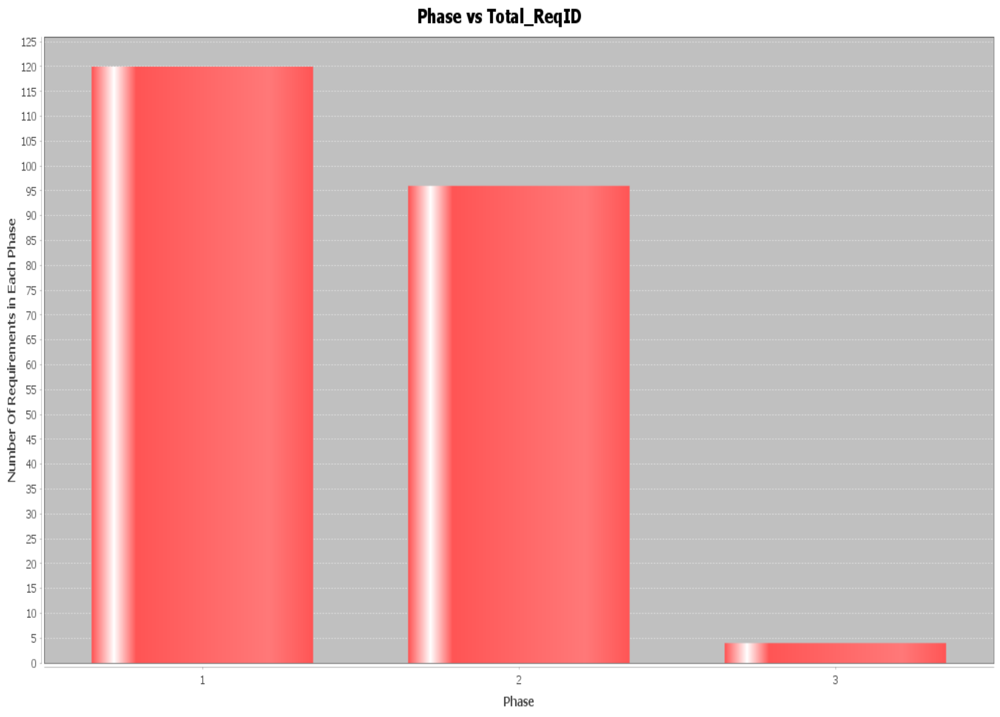


Fig4: Number of requirements in each phase

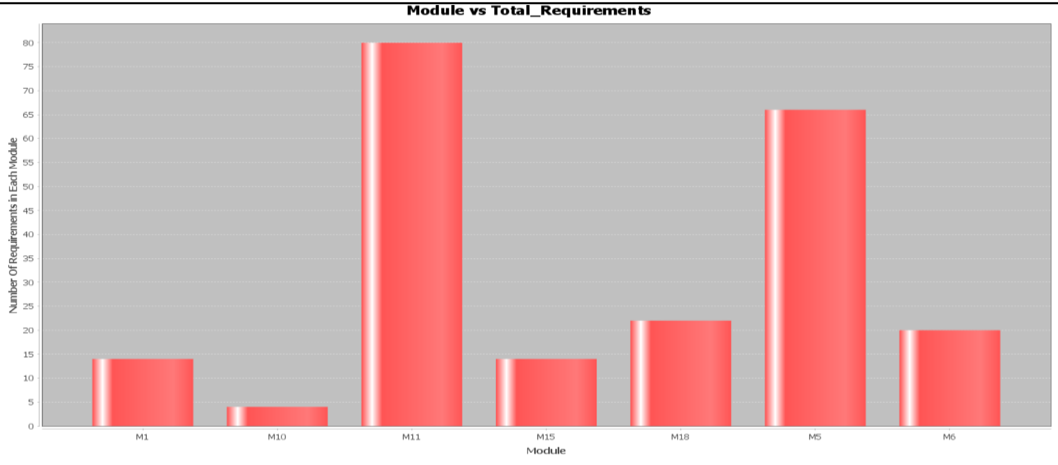


Fig 5: Module Vs Total Requirements

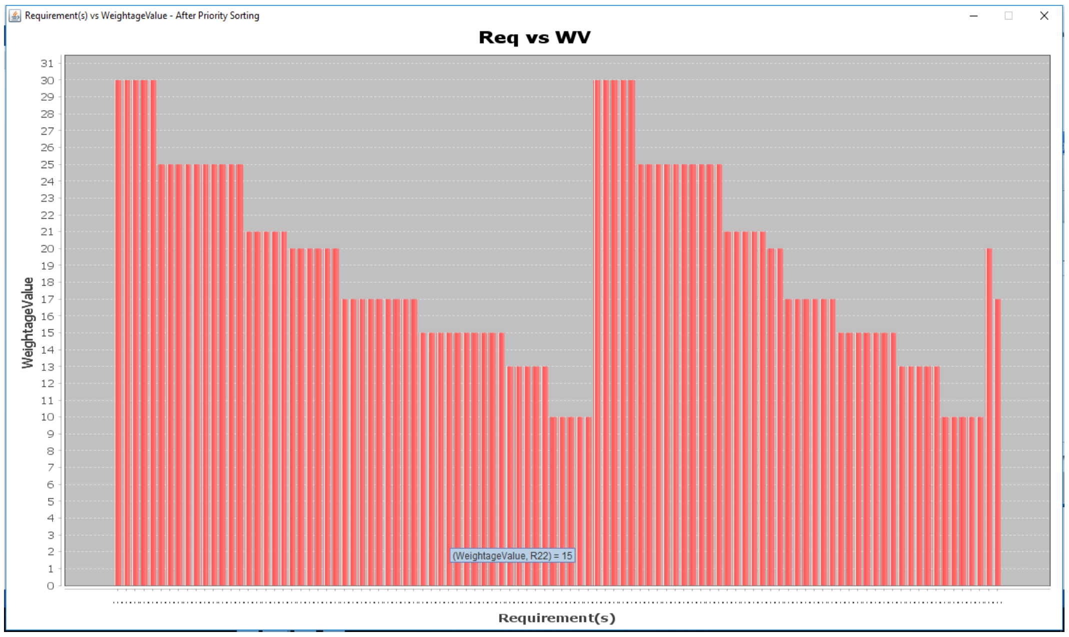
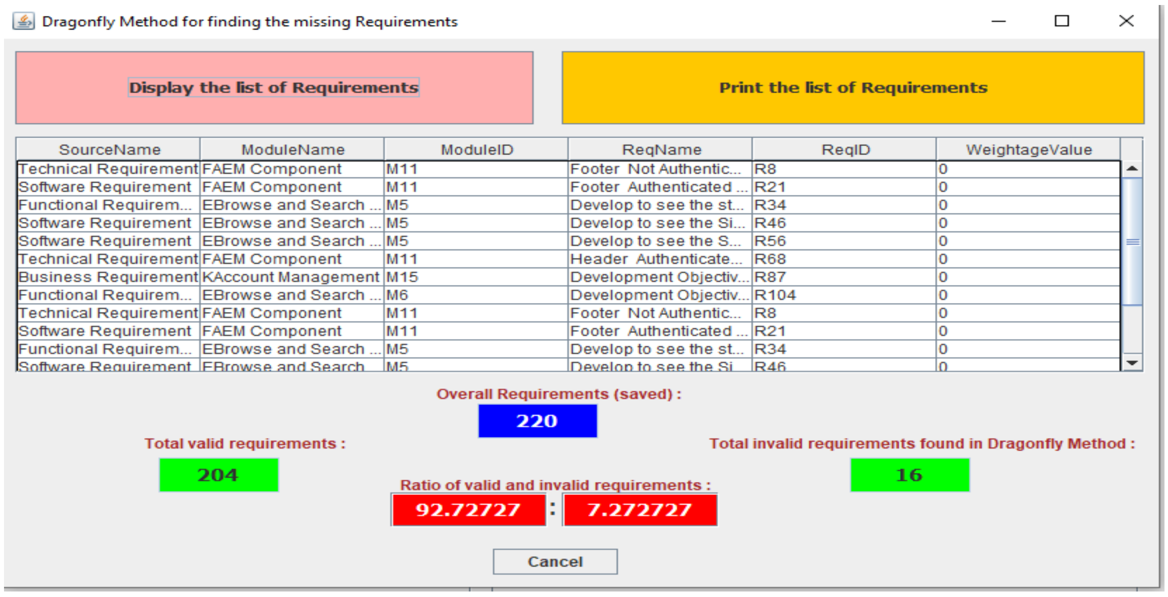


Fig 6:Requirements Vs weighted value (after optimization)


Dragonfly Method for finding the missing Requirements
Display the list of Requirements
Print the list of Requirements
Cancel

SourceName	ModuleName	ModuleID	ReqName	ReqID	WeightageValue
Technical Requirement	FAEM Component	M11	Footer Not Authentic...	R8	0
Software Requirement	FAEM Component	M11	Footer Authenticated ...	R21	0
Functional Requirem...	EBrowse and Search ...	M5	Develop to see the st...	R34	0
Software Requirement	EBrowse and Search ...	M5	Develop to see the Si...	R46	0
Software Requirement	EBrowse and Search ...	M5	Develop to see the S...	R56	0
Technical Requirement	FAEM Component	M11	Header Authenticate...	R68	0
Business Requirement	KAccount Management	M15	Development Objectiv...	R87	0
Functional Requirem...	EBrowse and Search ...	M6	Development Objectiv...	R104	0
Technical Requirement	FAEM Component	M11	Footer Not Authentic...	R8	0
Software Requirement	FAEM Component	M11	Footer Authenticated ...	R21	0
Functional Requirem...	EBrowse and Search ...	M5	Develop to see the st...	R34	0
Software Requirement	EBrowse and Search	M5	Develop to see the Si	R46	0

Overall Requirements (saved) : 220

Total valid requirements : 204

Total invalid requirements found in Dragonfly Method : 16

Ratio of valid and invalid requirements : 92.72727 : 7.272727

Fig 7:Dragonfly for missing requirements

Fig 3 depicts the requirement categorization based on total requirements (ReqID) for every phase and total requirements (ReqID) for each module. The phase sequences are provided in ascending order. Fig 5 depicts the graphical representation of module Vs Total requirements. Here, modules like $\mathrm{M}_1, \mathrm{M}_{10}, \mathrm{M}_{11}, \mathrm{M}_{15}, \mathrm{M}_{18}, \mathrm{M}_5, \mathrm{M}_6$ are considered with number of requirements for each module. Fig 6 depicts the number of requirements in each phase. Fig 6 depicts the dragonfly for missing requirements where the overall requirements are 220, total valid requirements are 204, total invalid requirements found in dragonfly method is 16 and the ratio among the valid and invalid requirements are 92.72727:7.272727.

5. Conclusion

This work anticipated a model for suggesting the traceability refactoring solutions with respect to the traceability requirements. This work attempts in facilitating the developers for offering a refactoring solutions from various features that are recommend by the requirement tool. This work leverages the development metrics for accessing the randomness degree of various methods and classes over the traceability matrix. Subsequently, this work recommends the solution for enhancing the traceability entropy and to establish code design with dragonfly optimization. Here, this work uses traceability metrics to evaluate the finest refactorization that can be performed and to maintain well-established traceability links.

As future research directions, the process is to validate the traceability of various refactoring that is not included in this research, specifically those who involves in partitioning of this approach. Subsequently, the effort of preserving and assessing the effect can be enhanced with source code design metrics that relies over the traceability. This is not measured with the initial tries and do not re-evaluate the traceability information. As an outcome, there is no proper approach to access how traceability impacts the code design metrics. At last, this work fails to devise a tool that can effectively and automatically carry out recommendation tasks.

References

HongyuKuang, Patrick Mäder, Can method data dependencies support the assessment of traceability between requirements and source code? Journal of Software: Evolution and Process 27, 11 (2015), 838–866.

Marco D’Ambros, Evaluating defect prediction approaches: a benchmark and an extensive comparison. Empirical Software Engineering 17, 4-5, 531–577, 2012.

Patrick Mäder, A visual language for modeling and executing traceability queries. Software and System Modeling 12, 3 (2013), 537–553.

Patrick Mäder,“Do developers benefit from requirements traceability when evolving and maintaining a software system? Empirical Software Engineering, 2015.

Patrick Mäder, Empirical studies in software and systems traceability. Empirical Software Engineering 22, 3 (2017), 963–966, 2017.

Dongsun Kim, Where should we fix this bug? a two-phase recommendation model. Software Engineering, IEEE Transactions on 39, 11 (2013), 1597–1610, 2013

JinGuo, Mona Rahimi, Jane 2016. Cold-start software analytics. In Proceedings of the 13th International Conference on Mining Software Repositories, MSR 2016,

R. Padmayathi, “New Frameworks for Automated Test and Retest (ATRT) Test Case Requirement Traceability Matrix,” International Journal for Innovative Research in Applied Science and Engineering, vol 1, no 2, pp 44-50, 2017.

J.Yoo, Code Assessment Plan and Requirement Traceability Matrix, Idaho Falls: Idaho National Lab, 2016.

N. Niu, Faceted navigation for software exploration. In ICPC, pages 193–196, ingston, Canada, June 2011.

E. van Veenedal, “Standard Glossary of Terms Used in Software Testing Version 2.3,” International Software Testing Qualification Board, Brussels, p 49, 2014.

Jane Cleland-Huang, Utilizing supporting evidence to improve dynamic requirements traceability. In Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on. IEEE, 135–144, 2005.

Daniel Port, Text mining support for software requirements: Traceability assurance. In System Sciences (HICSS), 44th Hawaii International Conference on. IEEE, 1–11, 2011.

N. Ali, Y.-G. Gu´eh´eneuc, and G. Antoniol. Trustrace: mining software repositories to improve the accuracy of requirement traceability links. IEEE Transactions on Software Engineering, 39(5):725–741, May 2013.

A. Ghabi and A. Egyed. Exploiting traceability uncertainty among artifacts and code. Journal of Systems and Software, 108:178–192, October 2015.

M. Borg, P. Runeson, Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. Empirical Software Engineering, 19(6):1565–1616, December 2014.

T. Dasgupta, M. Grechanik, E. Moritz, B. Dit, and D. Poshyvanyk. Enhancing software traceability by automatically expanding corpora with relevant documentation. In ICSM, pages 320–329, Eindhoven, The Netherlands, September 2013.

A. Egyed, F. Graf, and P. Gr¨unbacher. Effort and quality of recovering requirements-to-code traces: two exploratory experiments. In RE, pages 211–230, Sydney, Australia, September-October 2010.

A. Mahmoud and N. Niu. On the role of semantics in automated requirements tracing. Requirements Engineering, 20(3):281–300, September 2015.

A. Mahmoud, N. Niu, and S. Xu. A semantic relatedness approach for traceability link recovery. In ICPC, pages 183–192, Passau, Germany, June 2012.

A. Meneely, B. Smith, and L. Williams. iTrust electronic health care system case study. In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, Software and Systems Traceability, pages 425–438. Springer, 2012.

A. N. Meyer, T. Fritz, G. C. Murphy, and T. Zimmermann. Software developers' perceptions of productivity. In FSE, pages 19–29, Hong Kong, China, November 2014.

H. Sultanov, J. H. Hayes, and W.-K. Kong. Application of swarm techniques to requirements tracing. Requirements Engineering, 16(3):209–226, September 2011.

W. Wang, N. Niu, H. Liu, and Y. Wu. Tagging in assisted tracing. In SST, pages 8–14, Florence, Italy, May 2015.

R. E. Walpole, R. H. Myers, S. L. Myers, and K. E. Ye. Probability and Statistics for Engineers and Scientists. Pearson, 2011