A Framework for Software Component Reusability Analysis using Flexible Software Components Extraction

Sampath Korra, D.Vasumathi, A.Vinayababu

Abstract— With the adaptation of DevOps and agile methodologies for software developments, the demand from the consumers of the software applications are increasing to fulfil short time development phases. The application development community is facing the everlasting challenge of managing the bottleneck of timely delivery and quality of the software. The software quality being one of the most important factors to maintain the market reputation and the customer satisfaction, cannot be compromised. The software development companies have also tried adopting the strategies of deploying highly skilled development team in order to reduce the development life cycle span. Thus, several research attempts are made to identify the correct software reusable components for any projects. Nonetheless, the recent research outcomes are challenged by the development community with the arguments of reliability as different applications from different domain demands unique specifications and standards to be matched. Also, the components, which are to be reused for other specifications, must match the compliances of that specification. Considerably, identification of the reusable components is always a challenge and it is observed that the identification of the flexible components, for which matching the domain specification is easy, is also high complex process. Finally, finding the ideal coupling point of the identified modules are also a significant challenge. Thus, this research proposes a novel framework for identification of the flexible components for reuse in software development life cycle and identifies the unique coupling points of these components. Firstly, this work analyses the software components for features to be identified as flexible components using the deep extraction algorithm. The deep extraction algorithm demonstrates a novel characteristic for extracting feature based on the external and internal feature dependencies. The extracted features are also ranked based on the degree of independency and finally listed based on given ranks for reusability. Secondly, the extracted features are tested for outcome-based equivalence to ensure best reusability from the component library. Due to the multi – level equivalency analysis, this proposed framework provides higher accuracy and lower time complexity using the moderate number of extracted features for component analysis.

Key Words: Software Reusability, Component Feature, Deep Extraction, Equivalency Analysis, Component Integration

1. INTRODUCTION

In software engineering and programming building, reusability is the utilization of existing resources in some shape inside the product item advancement process, these advantages are items and results of the product improvement life cycle and incorporate code, programming segments, test suites, structures and documentation. The contrary idea of reusability is use, which adjusts existing resources as expected to meet explicit framework prerequisites. Since reuse infers the making of an independently kept up adaptation of the benefits, it is favoured over use [Fig – 1].

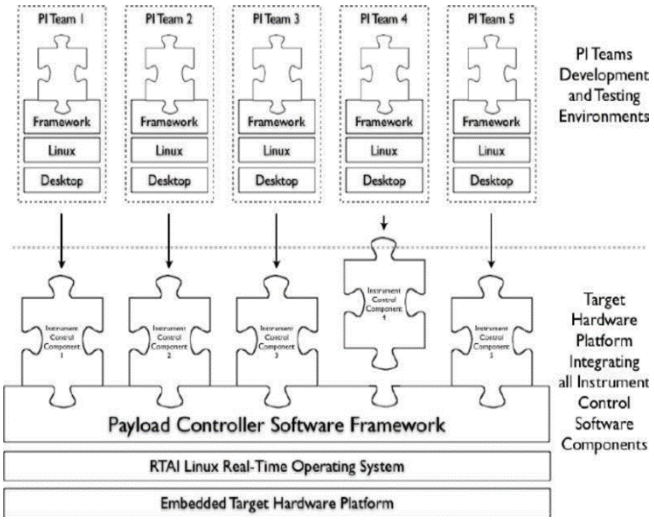

Fig. 1 Software Component Integration

Software reuse can reduce software improvement time and expenses. The significant focal points for software reuse are to observe increment in software profitability, abbreviate software improvement time, enhance software framework interoperability and create software with less individuals. Also, the effect of moving work forceseven more effectively from venture to extend, decrease software advancement and upkeep costs, create increasingly institutionalized software and finally, create better quality software and give an incredible upper hand [Fig – 2].

Revised Version Manuscript Received on July 10, 2019.

Sampath Korra, Research Scholar, Dept of CSE, JNTUK, JNT University, Kakinada, Andhra pradesh, India.
D.Vasumathi, Professor, Dept of CSE, JNTUHCE, JNT University, Kukatpally, Hyderabad, Telangana, India.
A.Vinayababu, Professor& Director, Dept of CSE, Stanley College of Engg& Tech for Women, Hyderabad.

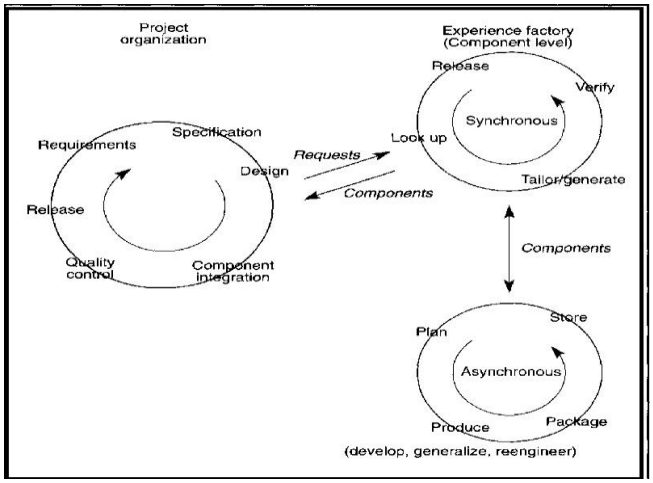

Fig. 2 Software Component Engineering

Henceforth, realizing the benefits of the software reusability, this work proposes the framework to identify the flexible software components and identify the unique coupling points for the identified flexible software components.

II. OUTCOME OF THE PARALLEL RESEARCH

The benefits from the component based reusable model for software development have intrigued the interest of many research attempts in the recent time. Also, the higher adaptability of component-based software development in all segments of domains makes this field of research highly popular. In this section of the work, the notable outcomes from the parallel searches are discussed.

A pattern in the advancement of inserted frameworks is the use of Component-Based Development as initially demonstrated by I. Crnkovic et al. [1]. This product building philosophy advances the improvement of frameworks through the synthesis of officially existing programming units called programming components. The business effectively utilizes different component models such as AUTOSAR [2], Rubus developed by K. Hänninen et al. [3] and IEC 61131 designed by K.-H. John et al. [4].

Notwithstanding, CBD is insufficient for implanted stages that join CPUs and GPUs. This is because of the absence of explicit help for GPUs. This general test has a few features. One of them alludes to the advancement of components with GPU abilities, which is intricate, tedious and blunder inclined. The component engineer exertion is expanded because of the way that nearby the usefulness, the component must contain settings and GPU-explicit condition data. In this way, typifying inside the components all the required data, results in equipment explicit components appropriate for specific GPU structures as it were as showcased by G. Campeanu et al. [5] and J. Carlson et al. [6].

The generic guidelines are initially developed by M. Chaudron et al. [7] as a component is constructed into a system. Also, the inserted and constant framework industry effectively received the CBD strategy through different component models. These component models pursue diverse collaboration styles reasonable for specific kind of uses as showcased by I. Crnkovic et al. [8]. GPUs were produced back in 90s and were utilized just in realistic based applications. In time, on account of the expansion of their calculation control and getting to be less demanding to program, GPUs were used in various sort of utilizations getting to be universally useful GPUs as highlighted by J.D. Owens et al. [9]. For instance, cryptography applications as presented by S.A. Manavski et al. [10] and Monte Carlo re-enactments as demonstrated by T. Preis et al. [11] have GPU-based arrangements.

The GPU is a preparing unit furnished with a parallel design that may utilize at once, a huge number of calculation strings through its different centers. Because of its execution display, GPU exceed expectations in executing information parallel applications. For instance, a reproduction of bio-sub-atomic frameworks may accomplish multiple times accelerate on GPU contrasted with the CPU as proven by J.E. Stone et al. [12] in his notable study.

To realise the benefits of the proposed system, it is highly recommended to carry out the comparative analysis with the other existing parallel research outcomes dealing with metric based analysis.

The work by Claudio Sant'Anna et al. [13] have demonstrated the use of software metric method for analysing the reusability level focusing on the aspect-oriented software. The limitations of this study can clearly be observed that the tested dataset and the proposed algorithm is majorly focuses on the modularity of the code, which influences the reusability to a greater extends. Thus, higher accuracy without any other influences. Further, this work was improved by H. Washizaki et al. [14] by introducing the reduction of feature analysis parameters. This work analyses the parameters and aims to reduce the number of features in the metric. Nonetheless, this work could not reduce a greater number of parameters and also increases the time complexity for the additional analysis.

Other parallel research attempts have also provided methodologies for analysing the interface-based reusability of the components. The work by M.A.S. Boxall et al. [15] demonstrates the interface component reusability analysis rather than the functional. It is natural to realize that as this study does not analyse and guarantee the software functional component reusability, thus the accuracy is criticized.

Yet another attempt by L. H. Etzkorn et al. [16] focuses on the legacy software applications for detecting the reusability degree. The legacy software applications are majorly based on batch of instructions and demands higher number of features for detecting the reusability factors as this must deal with the independency of the transactions. Thus, the time complexity of this method increases randomly.

The other attempts by K. K. Aggarwal et al. [17], S. D. Kim et al. [18], F. Dandashi et al. [19], A. Sharma et al. [20], P. Sandhu et al. [21], Marko Mijač et al. [22] have made several attempts to identify the optimal number of the features and optimal time complexity. Nonetheless, this trade-off was barely been resolved due to the absence of advanced equivalence analysis, which is proposed by this work.

The concept of cognitive reuse framework will lead to better product quality, reduces cost, time and effort. In the software development process, software reuse offers time, cost saving potentially and improves the quality in the construction of a new software system[23].

Developing reusable components are one of the main objectives of component-based software engineering. They play a crucial role in the field of application development and support. CBSE use certain architectural patterns and infrastructures of standard software to increase overall product quality for software development[24].

Association rule learning is a standard based AI technique for finding fascinating relations between factors with regards to extensive databases. It is proposed to distinguish solid principles found in databases utilizing a few proportions of intriguing quality of the software[25].

Hence with the fundamental understanding of the problem in the existing research, this work formulates the problem in the next section.

III. PROBLEM FORMULATION

The software component reusability completely depends on the fact that, the initial developed software must be reusable and should be flexible in development nature. Also, the component to be compatible with the specification where it is intended to be coupled. Hence, these two problems are to be identified and should be represented in a convenient approach for determination of reusability.

Henceforth, in this section of the work, two lemmas are formed in order to realize the solvability of these two earlier mentioned problems.

Lemma – 1: Any software component with minimum dependencies, generic method signature and less cohesion is most suitable for reuse.

Where,

m and M denotes any single component and the set of components in the software respectively

d and D denotes any component dependency and the set of dependencies of the software respectively

a and A denotes any component characteristics and the set of characteristics of the software respectively

Proof: Firstly, any component in the software can be considered to be reused based on the feasibility and part of the complete software,

$$m_i \in M \qquad \text{(Eq. 1)}$$

And

$$M = \sum_{i=1}^{n} m_i \qquad \text{(Eq. 2)}$$

Also, for any component in the software may have some dependencies as

$$m_i \rightarrow d_i \qquad \text{(Eq. 3)}$$

Further, each dependency is also part ( ) of the complete software dependency set,

$$d_i \in D \qquad \text{(Eq. 4)}$$

It is natural to realise that, each dependency in the dependency set also will have other dependencies,

$$m_i \rightarrow d_i$$
$$m_j \rightarrow d_j \qquad \text{(Eq. 5)}$$

If the dependencies of each component do not further depend on any other dependencies, then the identified components are not dependent on each other and can be identified as potential flexible component for reuse.

If,

$$d_i \xrightarrow{\text{No Dependency}} d_j \qquad \text{(Eq. 6)}$$

Then,

$$m_i \xrightarrow{\text{No Dependency}} m_j \qquad \text{(Eq. 7)}$$

Hence, it is natural to understand that mi and mj are having no dependencies and can be considered as feasible components for reuse.

Further, considering the characteristics of any software and any component must be nearly equal in order to consider the component to use for majority of functionalities in the software and thus can be reused.

Hence, any component and the characteristics of the component must be extracted first,

$$\lambda(m_i) = a_i \qquad \text{(Eq. 8)}$$

Naturally, the component characteristics is part of the total software characteristics and can be understood as,

$$a_i \in [A] \qquad \text{(Eq. 9)}$$

Henceforth, in order to make the component highly flexible and reusable, the characteristics of the component and characteristics set of the total software must be same or nearly same,

$$a_i \subseteq [A] \qquad \text{(Eq. 10)}$$

Thus, it can be concluded that, any component having less dependencies are most likely to have no or less dependencies on other components in the software and also for the same component, the characteristics must be same as the total software characteristics, can be most suitable for reuse.

Lemma – 2: Any software must be build using multiple reusable components and the components can be integrated in any order in the software for making a most feasible software for component integration.

Where,

s and S denote any software component and the complete software component set respectively

R_1 and R_2 denotes any two-random order of use of the software components

Proof: Firstly, any software must be a collection (∑) of components and should be denoted as,

$$S = \sum_{i=1}^{n} s_i \qquad \text{(Eq. 11)}$$

Also, it is important to have no dependencies of the software components among each-others,

$$s_i \neq s_j \qquad \text{(Eq. 12)}$$

Further, the order of use of the software components can be denoted as,

$$E[S] \rightarrow R_1$$
$$\Pi[S] \rightarrow R_2 \qquad \text{(Eq. 13)}$$

The order of the use of the components must not differ in the outcome of the software as,

$$R_1 = R_2 \qquad \text{(Eq. 14)}$$

Thus, it can be concluded that, the software components order of use must be independent of the outcome of the software for making the complete software most feasible for component integration and can be considered as best-case scenario for component-based development.

Henceforth, with the deep analysis of the problem, this work presents the software component feature extraction algorithm in the next section.

IV. PROPOSED COMPONENT FEATURE EXTRACTION ALGORITHM

In the light of Lemma – 1 and Lemma – 2, it is natural to understand that the extraction of features such as component dependency, component characteristics and total software characteristics are a must to identify any component for reuse.

Hence, in this section of the work, the feature extraction algorithm is proposed.

Algorithm - I: Deep Feature Extraction for Software Components (**DFESC**)

Step - 1. Accept the complete software component set
Step - 2. For each component
  a. Analyse for System Calls
  b. If the system calls in system library
    i. Then list the system library as dependency
  c. Else list the external library as dependency
  d. Analyse for component signature
    i. List the inputs to the method
    ii. Identify the input transformation
    iii. List the output control flow for the component
Step - 3. End

The flow of the algorithm is also analysed graphically here [Fig – 3].

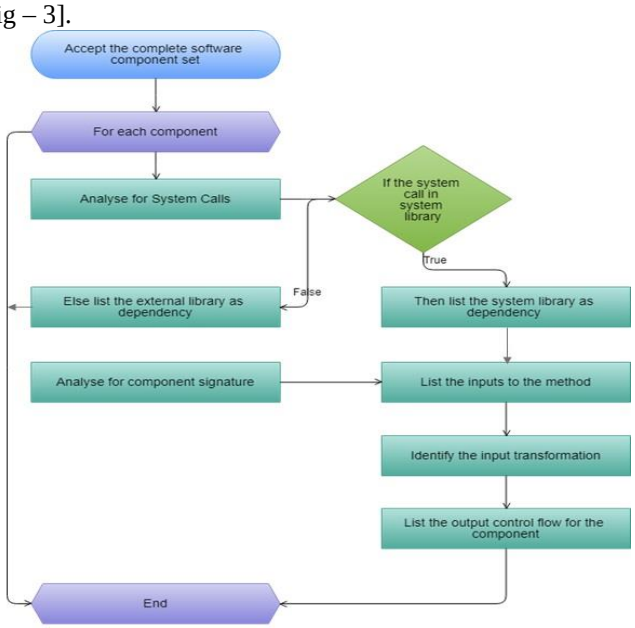

Fig. 3 The Flow Analysis of the DFESC algorithm

Henceforth, the proposed feature extraction algorithm is deployed for software component analysis and the results are elaborated in further sections of this work.

Once the features of the software components are extracted and considered to be reused, this work also proposes a reusability feasibility analysis metric in the next section of the work.

V. PROPOSED SOFTWARE REUSABILITY FEASIBILITY ANALYSIS METRIC

After the successful extraction of the software component features, which is the prime objective for finding the reusability of any component, the component must be analysed for feasibility for reuse.

Hence, this work proposes a novel metric for feasibility analysis for reusability. The parameters for metric are elaborated here [Table – 1].

TABLE I
REUSABILITY FEASIBILITY ANALYSIS

Parameter Name	Parameter Description	Expected Value
Software ID	The numeric identifier of the software during analysis	Numeric
Component ID	The numeric identifier of the component during analysis	Numeric
Number of Library Dependencies	The number of system calls related to core system library	High
Number of Component Dependencies	The number of dependencies on other components	Low
Number of External Dependencies	The number of system calls related to external system library	Low
Input Parameters	The characterisation of the input parameters with data types	Primitive Data Types
Degree of transformation	The change in the input parameters • High: Change in the data type • Medium: Change in variable name • Low: Change in value	Low
Output Parameters	The characterisation of the output parameters with data types	Primitive Data Types
Component Mutuality	Dependency of the other components on the analysing component	High

Development Approach	The software development programming approach: • Object Oriented • Procedural • Scripting • Batch	Any
Degree of reusability	Based on the expected value range	High

The proposed reusability feasibility analysis metric, the degree of the reusability for any software can be determined.

Once the software components are identified as reusable components, the next challenge is to analyses the possibilities for integration. Henceforth, in the next section of this work, the proposed integration feasibility analysis algorithm is discussed.

VI. INTEGRATION FEASIBILITY ANALYSIS ALGORITHM

The problem of integration in software component-based development and design is the major bottleneck in improving the software component-based reusability. As demonstrated in the Lemma – 2, finding the accurate integration points, increase the adoptability of the software and makes the software more feasible for integration.

Thus, analysing the software for integration feasibility is the second most prime objective of any research and this section of the work presents the novel algorithm for integration feasibility analysis.

Algorithm - II: Component Integration Feasibility Analysis (**CIFA**)

Step - 1. Accept the complete software component set
Step - 2. For each component
  a. Apply unit testing method to identify the component outcome
  b. Apply functional testing method to detect the system outcome
  c. Apply randomization to change the order
  d. For each random order
    i. Match the final system outcome
    ii. If the final system outcome is same
    iii. Then
      1. Mark the order as feasible
  e. If at least one random order is true
    i. Then
      1. Mark the software as integration feasible
Step - 3. End

The proposed algorithm is visualized graphically here [Fig – 4].

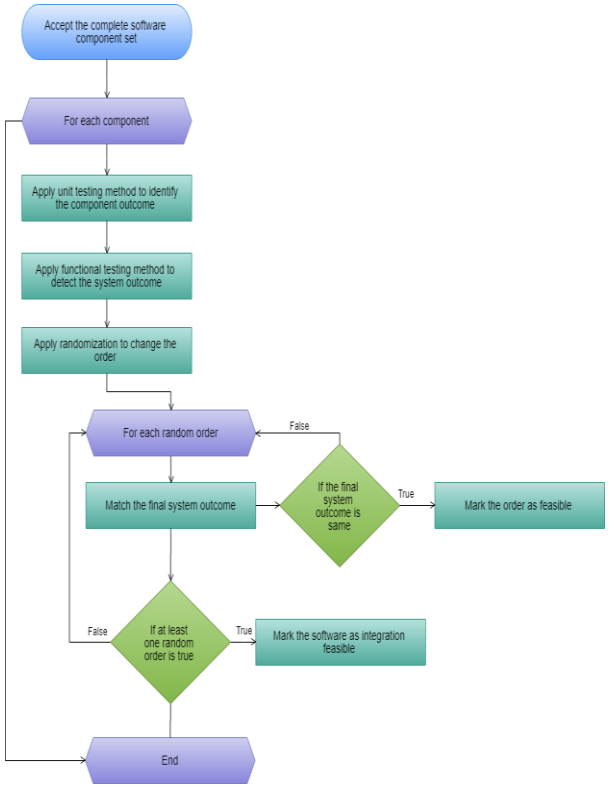

Fig. 4 The Flow Analysis of the CIFA algorithm

The proposed algorithm defines the software component integration. Software segments are parts of a framework or application. Segments are some methods for breaking the intricacy of software into sensible parts, which are accessed by this algorithm. Every segment shrouds the unpredictability of its usage behind an interface. Segments can be swapped in and out like the tradable parts of a machine. This lessens the intricacy of software advancement, upkeep, activities and bolster and enables a similar code to be reused in numerous spots.

Hence, it is natural to realize that the characteristics discussed of the proposed algorithm will certainly demonstrate the benefits during the results analysis.

Hence after, the results obtained from the proposed algorithms are analysed in the next section.

VII. RESULTS AND DISCUSSION

The results obtained from the proposed algorithms are highly satisfactory and are discussed.

The results are discussed in few sub-sections as feature extraction, identification of the dependency, Component integration feasibility and finally the time complexity of the proposed algorithms.

A. Feature Extraction Analysis

Firstly, the proposed algorithm is deployed on two different enterprise grade software applications for extraction of the features. The results are furnished here [Table – 2].

TABLE II
FEATURE EXTRACTION ANALYSIS

Software ID	Component ID	Number of Library Dependencies	Number of Component Dependencies	Number of External Dependencies	Input Parameters	Degree of transformation	Output Parameters
W-1	Comp - 1	12	0	5	7	Low	3
W-1	Comp - 2	5	0	5	4	High	4
W-1	Comp - 3	8	2	4	6	Low	2
W-1	Comp - 4	5	1	4	4	Medium	4
W-1	Comp - 5	9	3	6	5	Medium	4
W-2	Comp - 6	4	0	5	5	Medium	2
W-2	Comp - 7	6	1	5	6	Medium	4
W-2	Comp - 8	5	0	5	5	Low	4
W-2	Comp - 9	8	3	6	5	Low	4
W-2	Comp – 10	8	1	5	4	Low	3

At the point when the information to a calculation is too extensive to possibly be handled and it is suspected to be excess, at that point it tends to be changed into a decreased arrangement of highlights, additionally named a component vector. Deciding a subset of the underlying highlights is called include determination. The chose highlights are relied upon to contain the applicable data from the info information, with the goal that the ideal undertaking can be performed by utilizing this diminished portrayal rather than the total introductory information.

The results and analysed visually here [Fig – 5].

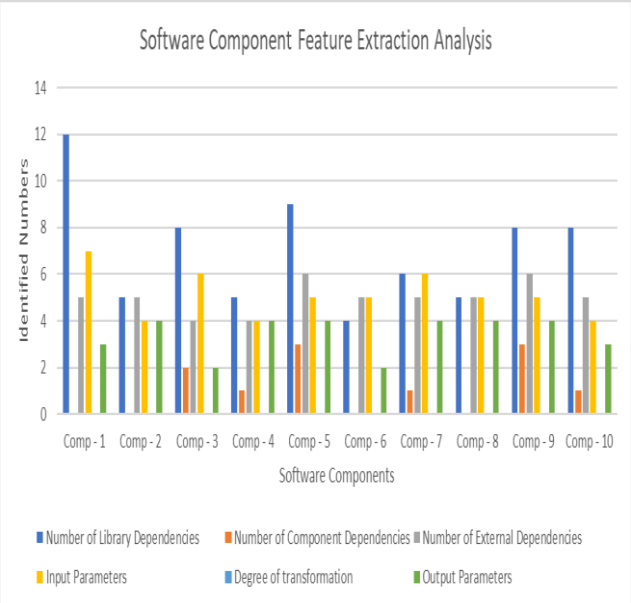

Fig. 5Software Component Feature Extraction Analysis

Referring to the proposed metric [Table – 1] and the expected value ranges, further the reusability analysis is carried out.

B. Component Dependency Analysis

Secondly, the component dependency analysis is carried out [Table – 3].

TABLE III
COMPONENT DEPENDENCY ANALYSIS

Software ID	Component ID	Number of Library Dependencies	Number of Component Dependencies
W-1	Comp - 1	0	4
W-1	Comp - 2	0	7
W-1	Comp - 3	2	4
W-1	Comp - 4	1	7
W-1	Comp - 5	3	9
W-2	Comp - 6	0	4
W-2	Comp - 7	1	6
W-2	Comp - 8	0	9
W-2	Comp - 9	3	8
W-2	Comp - 10	1	6

The results and analysed visually here [Fig – 6].

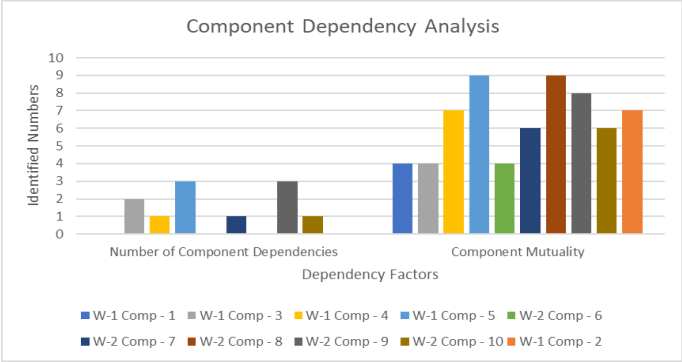

Fig. 6Software Component Dependency Analysis

The instruments, libraries, and structures we use to manufacture our web applications today are definitely unique in relation to the ones we utilized only a couple of brief years prior. In a couple of brief a very long time from now, the vast majority of these advances will have changed significantly once more. However, a large number of us make these a focal, inseparable piece of our applications. the designers import, use, and acquire from the kind of-the-month structures as though they're all going to be near and unaltered until the end of time. The outside dependencies of the software code are expected to be very low as in case of component – 6, under testing. This calculation extricates the outside conditions. Outside conditions represent an incredible danger to the long-haul soundness and practicality of any application.

C. Component Dependency Analysis

Third, the software component's architectural and design specifications are also analysed in order to establish the thought of the applicability of the proposed algorithms for all programming languages [Table – 4].

TABLE IV
DEVELOPMENT APPROACH ANALYSIS

Software ID	Component ID	Development Approach	Degree of reusability
W-1	Comp - 1	Batch	High
W-1	Comp - 2	Procedural	Low
W-1	Comp - 3	Object Oriented	Low
W-1	Comp - 4	Scripting	High
W-1	Comp - 5	Scripting	Medium
W-2	Comp - 6	Batch	Low
W-2	Comp - 7	Procedural	High
W-2	Comp - 8	Scripting	Medium
W-2	Comp - 9	Object Oriented	High
W-2	Comp - 10	Procedural	High

Automated software improvement is an applied structure for undertaking software designing tasks. There are various light-footed software improvement systems e.g. Precious stone Methods, Dynamic Systems Development Model, and Scrum. Most light-footed techniques endeavour to limit chance by creating software in short time boxes, called emphases, which normally last one to about a month. Every cycle resembles a small-scale software venture of its own and incorporates every one of the assignments important to discharge the little augmentation of new usefulness: arranging, prerequisites examination, plan, coding, testing, and documentation. While emphasis may not add enough usefulness to warrant discharging the item, a nimble software venture means to be fit for discharging new software toward the finish of each cycle. Toward the finish of every emphasis, the group reconsiders venture needs. Coordinated strategies underscore constant correspondence, ideally up close and personal, overwritten records. Most spry groups are situated in a warm up area and incorporate every one of the general population important to complete the software. At the very least, this incorporates software engineers and the general population who characterize the item, for example, item chiefs, business examiners, or genuine clients. The warm up area may likewise incorporate analysers, interface planners, specialized journalists, and the executives. Dexterous strategies likewise underline working software as the essential proportion of advancement. Joined with the inclination for up close and personal correspondence, nimble strategies create next to no composed documentation in respect to different techniques.

The results and analysed visually here [Fig – 7] & [Fig – 8].

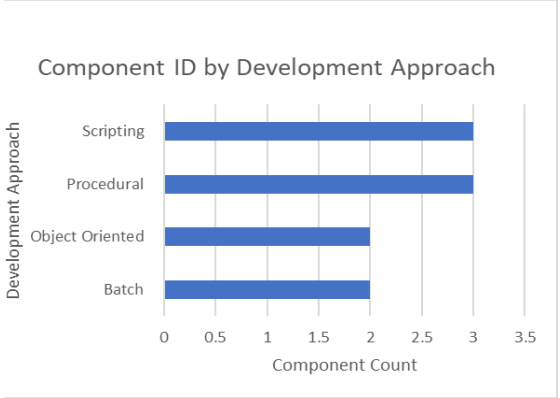

Fig. 7 Software Component Development Approach Analysis

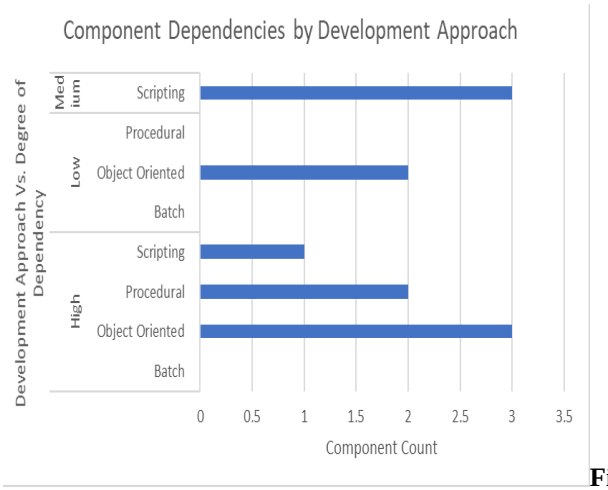

Fig. 8 Software Component Development Approach &Degree of Reusability Analysis

Hence, it is natural to realise that the degree of reuse completely depends on the design methodology rather ran development methodology.

D. Time Complexity Analysis

Finally, the time complexity analysis is carried out for the testing components [Table – 5].

TABLE V
TIME COMPLEXITY ANALYSIS

Software ID	Component ID	Analysis Time (sec)
W-1	Comp - 1	15.4929
W-1	Comp - 2	12.1642
W-1	Comp - 3	17.8832
W-1	Comp - 4	12.8145
W-1	Comp - 5	19.8786
W-2	Comp - 6	16.9762
W-2	Comp - 7	14.4435
W-2	Comp - 8	10.855
W-2	Comp - 9	18.0534
W-2	Comp - 10	19.6548

The proposed framework produces the outcomes in measured time complexity between 10 to 20 sec.

The results and analysed visually here [Fig – 9].

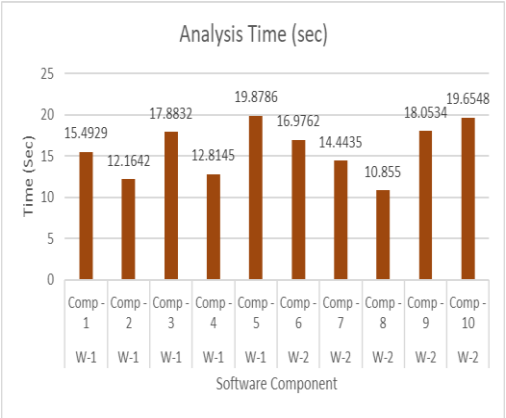

Fig. 9 Time Complexity Analysis

Once in a while, there are more than one approach to take care of an issue. We have to figure out how to look at the execution changed calculations and pick the best one to tackle a specific issue. While breaking down a calculation, we generally consider time unpredictability and space multifaceted nature. Time unpredictability of a calculation measures the measure of time taken by a calculation to keep running as an element of the length of the info. Essentially, Space multifaceted nature of a calculation evaluates the measure of room or memory taken by a calculation to keep running as a component of the length of the information.

Henceforth, it is natural to realize that the proposed algorithms produce highly accurate and satisfactory results with constant time complexity.

In the next section of this work, the comparative analysis is carried out.

VIII. COMPARATIVE ANALYSIS

In order to build the believe that the proposed methodology is better than the other parallel research outcomes, in this section of the work the comparative analysis is furnished. The methodologies are compared based on number of metric parameters, spread of technology analysis, time complexity and finally the reusability detection measures [Table – 6].

TABLE VI
COMPARATIVE ANALYSIS

Methodology	Number of Metric Parameters	Analysis of Development Technologies	Reusability Detection *	Time Complexity +
Sant'Anna C. et al. [13]	12	Object Oriented	Moderate	Low
Washizaki H. et al. [14]	11	Object Oriented	Moderate	High
Boxall M.A.S. et al. [15]	8	Object Oriented	Low	Moderate
Etzkorn L.H. et al. [16]	13	Object Oriented	Moderate	Low
Aggarwal K.K. et al. [17]	5	Object Oriented	Low	High
Kim S.D. et al. [18]	8	Object Oriented	Low	High
Dandashi F. et al. [19]	8	Object Oriented	Low	High
Sharma A. et al. [20]	9	Object Oriented	Low	Low
Sandhu P.S. et al. [21]	10	Object Oriented	Moderate	Moderate
Marko Mijač et al. [22]	10	Object Oriented	High	Moderate
Proposed DFESE & CIFA Algorithm	12	Object Oriented Procedural Scripting Batch	High	Low

* Low: 60% to 65%, Moderate: 65% to 70% and High: 70% and Higher

+ * Low: O(n), Moderate: O (n * m) and High: O (m * n log(n)) and Higher

Hence, it is natural to realise that the proposed methodology has outperformed the other parallel research outcomes.

Further, the work presents the final conclusion of this search in the next section.

CONCLUSION

The increase in demand for the higher quality software with reduced development time have increased the use of software component reusability in application development domains. The major challenge of identification of reusable components is to identify and match the complete specification of the selected component. Thus, this work proposes a novel deep feature extraction algorithm to analyse and extract the features of the software components and proposes a novel metric for analysing the feasibility for reuse of any selected software component. Also, yet another outcome of the proposed research is the component integration feasibility analysis algorithm for any software using equivalency theory. The final outcome of the work is an automated framework to identify the reusable components and recommend a integration possibilities for any given software under any domain for any development methodologies for making the software development life cycle a much accurate and timely field of work.

REFERENCES

1. Crnkovic, M. Larsson, Building Reliable Component-Based Software Systems, Norwood, MA, USA:Artech House, Inc., 2002.
2. AUTOSAR — Technical Overview, [online] Available: http://www.autosar.org.
3. K. Hänninen, J. Mäki-Turja, M. Nolin, M. Lindberg, J. Lundbäck, K.-L. Lundbäck, "The Rubus component model for resource constrained real-time systems", Industrial Embedded Systems 2008. SIES 2008. International Symposium on, pp. 177-183, 2008.
4. K.-H. John, M. Tiegelkamp, IEC 61131–3: programming industrial automation systems: concepts and programming languages requirements for programming systems decision-making aids, Springer Science & Business Media, 2010.
5. G. Campeanu, J. Carlson, S. Sentilles, "A GPU-aware component model extension for heterogeneous embedded systems", The Tenth International Conference on Software Engineering Advances, 2015.
6. G. Campeanu, J. Carlson, S. Sentilles, S. Mubeen, "Extending the Rubus component model with GPU-aware components", Component-Based Software Engineering (CBSE) 2016 19th International ACM SIGSOFT Symposium on, pp. 59-68, 2016.
7. M. Chaudron, I. Crnkovic, H. van Vliet, "Component-based software engineering" in Software Engineering: Principles and Practice, Wiley, 2008.
8. I.Crnkovic, S. Sentilles, A. Vulgarakis, M.R. Chaudron, "A classification framework for software component models", IEEE Transactions on Software Engineering, vol. 37, no. 5, pp. 593-615, 2011.
9. J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone, J.C. Phillips, "GPU computing", Proceedings of the IEEE, vol. 96, 2008.
10. S.A. Manavski, "CUDA compatible GPU as an efficient hardware accelerator for AES cryptography", Signal Processing and Communications IEEE International Conference, pp. 65-68, 2007.
11. T. Preis, P. Virnau, W. Paul, J.J. Schneider, "GPU accelerated Monte Carlo simulation of the 2D and 3D Ising model", Journal of Computational Physics, vol. 228, no. 12, pp. 4468-4477, 2009.

12. J.E. Stone, J.C. Phillips, P.L. Freddolino, D.J. Hardy, L.G. Trabuco, K. Schulten, "Accelerating molecular modeling applications with graphics processors", Journal of computational chemistry, 2007.
13. Claudio Sant'Anna, Alessandro F. Garcia, Christina Chavez, Carlos Lucena, and Arndt Staa, "On the reuse and maintenance of aspectoriented software: An assessment framework" in Proceedings of Brazilian Symposium on Software Engineering, 2003.
14. H. Washizaki, H. Yamamoto, and Y. Fukazawa, "A Metrics Suite for Measuring Reusability of Software Components" in 9th International Software Metrics Symposium Proceedings, Sydney, Australia, 2003, pp. 211–223.
15. M.A.S. Boxall and S. Araban, "Interface metrics for reusability analysis of components" in Australian SE Conference Proceedings., Melbourne, Australia, 2004, pp. 40–51.
16. L. H. Etzkorn, W. E. Hughes, and C. G. Davis, "Automated reusability quality analysis of OO legacy software" Inf. Softw. Technol., vol. 43, no. 5, pp. 295–308, 2001.
17. K. K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, "Software reuse metrics for objectoriented systems" in Third ACIS International Conference on Software Engineering Research, Management and Applications, 2005, Mount Pleasant, Michigan, USA, 2005, pp. 48–54.
18. S. D. Kim and S. H. Chang, "A Systematic Method to Identify Software Components" in 11th Asia-Pacific Software Engineering Conference, 2004., Busan, Korea, 2004, pp. 538–545.
19. F. Dandashi, "A method for assessing the reusability of object-oriented code using a validated set of automated measurements", 2002.
20. A. Sharma, P. S. Grover, and R. Kumar, "Reusability assessment for software components", ACM SIGSOFT Softw. Eng. Notes, vol. 34, no. 2, p. 1, Feb. 2009.
21. P. Sandhu and H. Singh, "Automatic Reusability Appraisal of Software Components using Neurofuzzy Approach", Int. J. Inf. Technol., vol. 1, no. 8, pp. 2407–2413, 2007.
22. Reusability Metrics of Software Components: Survey, Marko Mijač, Zlatko Stapic, Central European Conference on Information and Intelligent Systems, 2015.
23. S. Korra, D. Vasumathi and A. Vinayababu, "An Approach for Cognitive Software Reuse Framework," 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 2018, pp. 1-6.
24. Korra, Sampath, D. Vasumathi, & A. Vinaybabu. "A Novel Approach for Building Adaptive Components using Top-Down Analysis." International Journal of Engineering & Technology [Online], 7.4.19 (2018): 1036-1040.
25. Sampath Korra*, D. Vasumathi and A. Vinaybabu, "A Novel approach to component classifications and adaptation using Apriori algorithm", Recent Patents on Computer Science (2019) 12: 1.