**B. Description of proposed algorithm:**

Finding the appropriate component is very hard in the large software repository, and it needs a proper specification for their correctness. Component specification helps understand the component's functionality and characteristics stored in the repository. Hence the proper specification encourages the correctness of the candidate component and fulfils the requirements of the system and customer. Therefore, the developer wants to adopt a specific approach that gives the unambiguous result because the proper component specification and selection affects the final requirement. A poor component specification increases the failure of the products. The formal approach is one of the approaches that provide the more accurate specification of the software component and also provide the acceptable/correct (unambiguous) result because it is based on mathematical proof. Finding the best component to compose and construct the application is essential task for developer. It requires the qualified or correct component from the repository. The developer does not guarantee that the find component is the correct or qualified component in the above steps. It may be an ambiguous component. Hence, the developer needs proper specification methods to specify component functionality and behaviour correctly. Formal methods is one of the approach that helps to know about the correct functionality and behaviour of the component because it is based on the mathematical approach. In this section, we have proposed an algorithm. With the help of a proposed algorithm, we have automated the selection and composition of the required components. The selection of appropriate components plays a central role in the overall process of CBD. A detailed algorithm description of a component selection and composition is given below.

Firstly, the user requirements (UserR) to develop a new software application arrive/get (come) to the developer. The software developer fetches/gathers/accrues (collects) this requirement and analyzes the feasibility of the component to the user requirement. In this step, the developer analyzes the functional and non-functional requirements and system constraints of the software application. After this, they identify and search the required component (CompR) from the reusable repository (SCR). This repository (SCR) is the collection of reusable components with an index. The developer searches the required component using keyword-based techniques and finds the target component. If the search is successful, then the component is found in the repository. Then, developer applies the first fit strategy to retrieve the desired components from the repository. If the search is not successful, then the component is not found in the repository. Then there may be arrive two possibilities. Either component (CompR) partially matches the user's requirement (UserR) or does not match the user's requirements. In the first case, the user requirement partially satisfied the component requirement of components available in the reusable repository (SC). Then, it needs some components modification to the existing component. For this the developer calls the modify\_requirements() function, in which they add some additional features to the existing components. Again developers design and implement this modified component. After modifying the component, this component is tested and verified by applying some testing mechanics and then it is again stored in the reusable repository (SCR). In the second case, the component does not match the user's requirements. In that case, the developer needs to design and implement this new component from scratch for which the developer calls the create\_CompR() function. In this, developer designs and implements the new component from scratch. After implementation, this new component is tested and verified by applying some testing mechanics and stored it into a reusable repository (SCR).

In the next step, developers write the formal specification of all selected components using the formal specification language CCS (calculus of communicating system) and create the model of each component. Then they write the requirements of each component using mu-calculus temporal logic. Then verify this component using a model checker tool called CWB-NC tool. After this, stores the set of components based on the formal specification in another reusable repository (SCF). Now, developer checks the user requirements match to the formal-based component specification. If the match is successful, then they apply the best fit strategy for selecting the component from the repository (SCF). After selection, composes all the components using prefix order and constructs the new system. After composition, tests this new system by applying some testing techniques. After testing, the developer evaluates and checks the system requirement with the user requirement. If the evaluation is successful, deployed this new software application to the user (this new software application is deployed to the user.).

If the above match is not successful. Then there may be arrive two possibilities. Either user requirements partially match the selected component requirement or do not match the user requirements. In the first case, the user requirement partially matches the selected component requirement. In this case, it needs a modification in the model of the component. For this the developer calls the adjust\_requirement() and modifies the model of this existing component. Again, designs the modified component's model and writes the requirement according to the specification. Finally, stores this modified component into the reusable repository (SCF). In the second case, if the user requirement does not match the requirement of any component. In that case, the developer needs to create a new component model. For which the developer calls new\_comp() and writes their formal specification and requirement of the new component using formal methods. Then verify this new component using a model checker tool (CWB-NC). After this, stores it into a set of the component reusable repository (SCF) based on formal specification. It can be used for building any software application in future. The overall workflow of our proposed methodology is shown in figure 6.

**Algorithm for Component Selection and Composition**

**Input:** UserR, CompR, SCR, SCF

//UserR: User Requirement, CompR: Set of component requirement can be specify a subset of requirements, SCR: Set of component in a reusable repository with index, SCF: Set of components based on formal specification.

**Output:** Select and compose the components

1. Begin

2. sel\_comp<− 0

3. f\_sel\_comp<− 0

4. composition\_comp<− 0

5. while SC != {} & f1<− 1 do

//Given a keyword for user requirement to search the target component

6. for i <− 0 to n

7. for j <− 0 to m //m is the number of components available in the reusable repository with index

8. if (UserRi = CompRj)

// Apply the first fit strategy for selecting the subset of the component and provide a rank to the component.

9. sel\_comp<− CompRj

10. f1<− 0

11. break

12. return (sel\_comp)

13. else if (UserRi = CompRj)

// If user requirement partially satisfy the component requirement then call modify/improve requirement function of the component.

14. call modify\_requirement (UserR, CompR, solution1)

15. return (CompR’)

16. Repeats step 8 to 12 till find the desired component

17. else if (UserR != CompRj)

18. return (No solution)

19. else

// call create new component from scratch according to user requirement

20. call create\_CompR (CompR)

21. end if

22. end if

23. end if

24. end while

//Retrieve the best solution according to rank of the component

25. while SF != {} & f2<− 1 do

//Given a keyword for user requirement to search the target component

26. fori <− 0 to n

27. for j <− 0 to m //m is the number of selected components available in the formal specification based repository

28. if (UserRi = sel\_compj)

// Apply the best fit strategy for selecting the component and composed it one by one.

29. f\_sel\_comp<− sel\_compj

30. Composition\_comp<− Composition\_comp ⸧compf\_sel\_comp

31. f2<− 0

32. break

33. return (Composition\_comp)

34. else if (UserRi = sel\_compj)

// If user requirement partially satisfy the component requirement then call adjust/improve requirement function of the component.

35. call adjust\_requirement (UserR, sel\_comp, solution2)

36. return (sel\_comp’)

37. Repeats step 28 to 33 till find the desired component

38. else if (UserR != sel\_compj)

39. return (No solution)

40. else

// call create new component from scratch according to user requirement

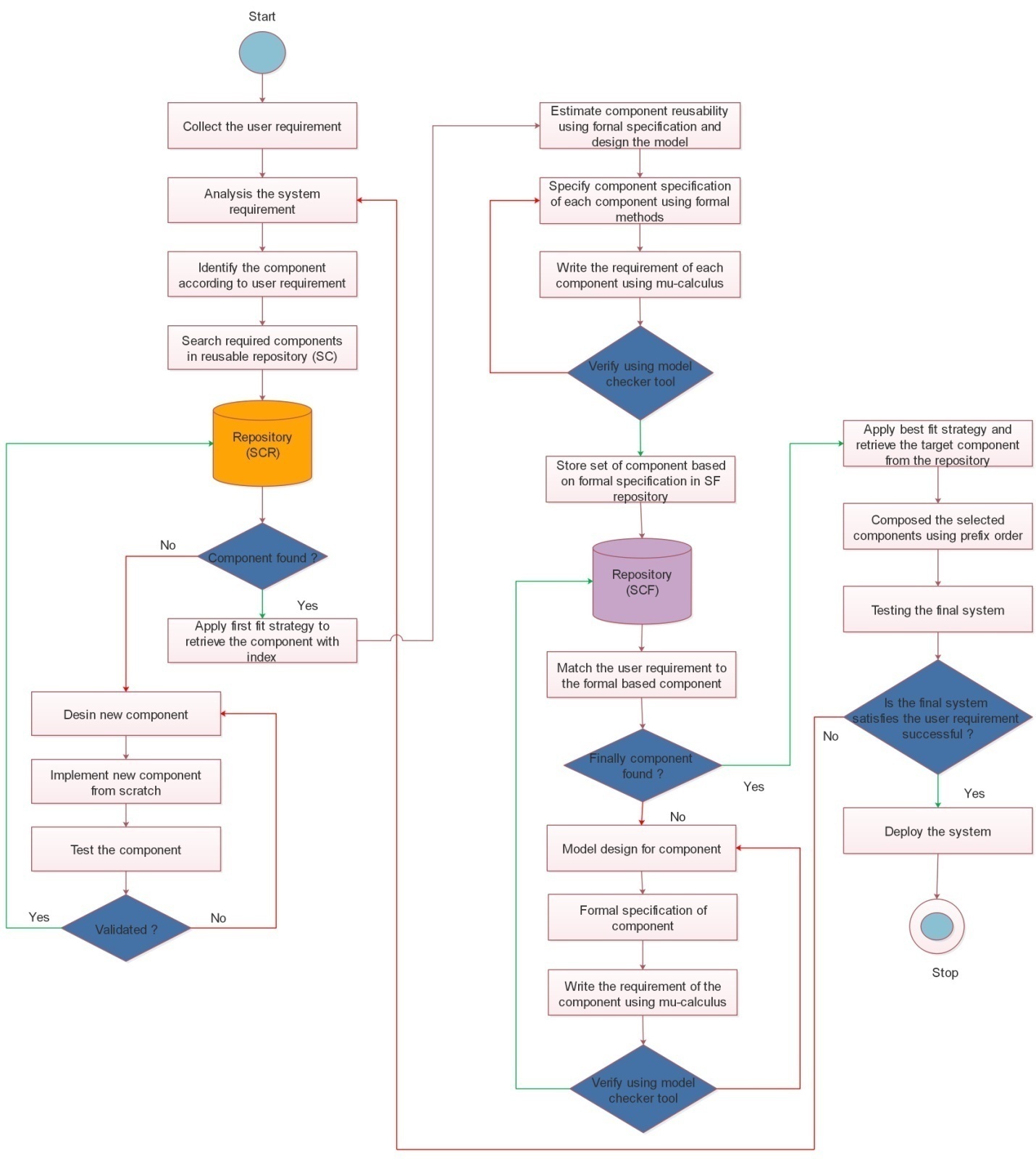
41. call new\_comp (sel\_comp)

42. end if

43. end if

44. end if

45. end while



C. **Experimental evaluation:**

In this section, we have shown how formal methods help select the correct component from the repository. For example, a user sends a query like “A java language-based component for the railway reservation system containing login page having/with username and password functionality. It contains a date page having/with functionality with calendar, cities name page with the state name. User can see the train numbers, class of train functionality, and enters the number of person in this system. It contains the functionality to calculate their age and also calculate fare according to distance and number of person. It also displays train ability functionality according to class and date, and displays the payment page with options like debit card, credit card, cash, page. It also contains final booking page functionality after payment”. After understanding all the necessary requirements, we have searched and selected the component based on keyword searching (capturing some keywords) and using formal approach. According to this requirement, we have shown the results in table 1 and 2. In table 1, we have to retrieve components without applying the formal method only using keyword-based searching, and in table 2, we have to retrieve components applying the formal method technique. Table 2 shows the more correct result in comparison to table 1. There are no ambiguous components presented in table 2. Hence, the formal approach helps solve the component selection issue and their composition. In figure 17, we have shown the comparative result for the selection of required components.

Table 1: Selection of the required component without using formal approach

|  |  |  |  |
| --- | --- | --- | --- |
| **Component index** | **Component Name** | **Language** | **Functionality** |
| 12 | Payment | Java | Payment page displaying options like credit card, debit card, UPI |
| 6 | Class | Java | Display different class |
| 46 | Login | Java | Login using username and password with captcha protection |
| 22 | City name | Java | Dropdown displaying all cities name with state |
| 10 | Fare calculator | Java | Fare calculation according to distance and train class |
| 1 | Train number | Java | Display unique id |
| 17 | Date picker | Java | Display calendar to pick dates |
| 25 | City name | Java | Dropdown displaying all cities |
| 13 | Payment | Java | Payment page displaying options like credit card and debit card only |
| 54 | Train display | Java | display list of trains availability according to class and date |
| 8 | Book submission | Java | Display booking confirmation |
| 49 | Login | Java | Login using username and password |
| 3 | Calculator | Java | Calculate button |
| 76 | Book submission | Java | Display booking confirmation after payment |
| 16 | Date picker | Java | Display dates with dd/mm/yy format |
| 45 | No. of person | Java | Display number of persons |
| 88 | Fare calculator | Java | Fare calculation according to distance, train class and number of persons |
| 2 | DOB | Java | Calculate age through calendar |
| 11 | Payment | Java | Payment page displaying options like credit card, debit card and also using cash |
| 74 | Train display | Java | Display trains availability according to class and quota |

Table 2: Selection of the required component using formal approach

|  |  |  |  |
| --- | --- | --- | --- |
| **Component index** | **Component Name** | **Language** | **Functionality** |
| 6 | Class | Java | Display different class |
| 22 | City name | Java | Dropdown displaying all cities name with state |
| 1 | Train number | Java | Display unique id |
| 17 | Date picker | Java | Display calendar to pick dates |
| 54 | Train display | Java | display list of trains availability according to class and date |
| 49 | Login | Java | Login using username and password |
| 76 | Book submission | Java | Display booking confirmation after payment |
| 45 | No. of person | Java | Display number of persons |
| 88 | Fare calculator | Java | Fare calculation according to distance, train class and number of persons |
| 2 | DOB | Java | Calculate age through calendar |
| 11 | Payment | Java | Payment page displaying options like credit card, debit card and also using cash |