Int J Syst Assur Eng Manag (June 2017) 8(2):393–407
DOI 10.1007/s13198-016-0442-2

ORIGINAL ARTICLE

Components integration-effect graph: a black box testing and test case generation technique for component-based software

Umesh Kumar Tiwari[1] · Santosh Kumar[2]

Received: 11 January 2015 / Revised: 8 March 2016 / Published online: 25 March 2016

Abstract Testing plays a vital role in the evolution and establishment of any quality product as well as any quality system. Testing is essential to prove the correctness (valid output when input is valid, and proper handling techniques when input is invalid) of the system and it is crucial to prove the compatibility of the system with the operating environment. In component-based software systems, various components interact with each other to access as well as provide required functionalities. In such complex systems testing is one of the most important activities. Since component-based software engineering relies on the concept of "use of pre-built and pre-tested components", our focus here, is on functional testing rather than structural testing. Functional testing emphasizes the behavioural attributes of the components when they interact with each other. Components interact through operands and parameters. In this paper we suggest functional testing strategy and test case generation technique for component-based software. When two components are integrated then they generate some specific effect. This strategy is named Integration-effect graph. This strategy is a black-box technique as it covers the input and output domains only. We have used the graph theory notations to show the integration and interaction among the components.

✉ Umesh Kumar Tiwari
umeshtiwari22@gmail.com

Santosh Kumar
amu.santosh@gmail.com

[1] Department of Computer Application, Graphic Era University, Dehradun, India

[2] Department of Computer Science, Graphic Era University, Dehradun, India

Keywords Black box · Component · Component-based software · Correctness · Functional testing · Pre-built · Pre-tested · Structural testing

1 Introduction

Component-based software development (CBSD) (Kozaczynski and Booch 1998), emphasizes the concept of reusability. Reusability offers software development team to focus on the new and innovative solutions of the problem rather than concentrating on coding (Pressmen 2001). A software component should be designed and implemented in such a way that it can be reused in similar as well as different contexts (Pressmen 2001). To provide and access services in a predefined architecture as well as in different contexts components have explicit and well defined interfaces (Senthil et al. 2007).

The goal of CBSD is to buy or develop such components which can be used in various systems and the same system can be deployed in many optimized ways, that is, development through reusability and development for reusability. A component- based software system can be developed by the composition of existing and newly developed components with defined interfaces (Shepperd 1998; Capretz 2005). Components are the basic building blocks of CBSD. Bennatan (Bennatan 1992) has categorized components on the basis of their reusability as off-the shelf components, adaptable components, and new components.

1.1 Off-the-shelf components

These are the pre-built, pre-tested components. They are either provided by the third party or developed by the development team as part of the previous software.

1.2 Adaptable components

These are the components which can accommodate with existing requirements, design, code, or test cases with minor or major modifications. These components are selected from the component repository and can be used by making minor or major modifications. On the basis of the degree of modifications, adaptable components are divided into two categories: *fully qualified components* and *partially qualified components.* Fully qualified components are the components which require no modification or a small degree of modification, and partially qualified components are the components which require major degree of modification.

1.3 New components

These components are developed by the development team itself from the scratch according the specifications and requirements for the particular project.

2 Component-based software testing

Whether it is traditional, object-oriented or component-based software, testing plays a vital role. According to studies and researches (Elberzhager et al. 2012; Tsai et al. 2003) inadequate testing leads to unreliable software products. When we use third party commercially off-the-shelf components, we have no access to the source code. This makes testing such components more difficult and challenging (Gao et al. 2003; Weyuker 1998). Testing is the set of activities that starts with the culmination of requirement elicitation. It can be planned in the early phases of development and conducted through out the lifecycle of the software development. In the context of CBS, testing begins at the individual component level and progresses towards outwards the integration of the entire CBS system (Pressmen 2001). Testing is the oldest and the most widely used technique for *verification* and *validation* (Myers 2004). In CBS, component *verification* can be defined as the set of activities to ensure that each individual component is implemented and deployed according to its intended functionality and each component plays its specified role. *Component validation* can be defined as the set of activities to ensure that the integrated components are compatible with the design architecture and are traceable to customer's requirement.

3 Black box testing and related work

To maintain the reliability of software, testing is must. It is a process done with the goal of finding all the possible errors in the software. Black box testing is a technique used to test the software based on its behaviour, rather testing its structure or the internal code. In black box testing, our focus is on testing the software on the basis of inputs provided and the outputs achieved. Here the internal code is treated as black box and the behaviour of the software is captured on similar basis.

Achieving highly reliable software is a difficult task, even when quality, pre-tested and trusted software components are combined (Tyagi and Sharma 2014). Several techniques have therefore emerged to analyze the reliability of component based applications. These techniques mainly fall under two categories (Tyagi and Sharma 2014):

System Level System level reliability is estimated for the application as a whole.

Component Level Component level reliability of the application is estimated on the basis of reliability of the individual component and their interaction mechanism.

In the literature (Ramamoorthy et al. 1976; Voas 1992; Voas and Miller 1992; Ntafos 1988; Ostrand and Balcer 1988; Voas and Miller 1995; Weyukar 1993), the following strategies of black box testing have been proposed and are in wide use. This testing only takes the external view of the software. This testing is categorized into four different types.

- Boundary value analysis
- Equivalence class partitioning
- Decision Table-Based testing
- Cause-effect graphing

3.1 Boundary value analysis

Boundary value analysis is a black box testing technique to ensure that the software works correctly at the boundary values. As inputs and outputs of the software tend to behave more abruptly at the boundary values when put to test, as compared to the values that are in the center or the limits of the conditions (Ramamoorthy et al. 1976; Voas 1992). One of the other reason of the boundary value analysis is that defects and errors are easier to be depicted and caught at the boundary lines than in the middle of the range of the test cases. It focuses on data at the "edges" of an equivalence class (Voas and Miller 1992).

3.2 Equivalence class partitioning

In Equivalence Class Partitioning (ECP) the input data of a software unit is divided into number of equivalence classes or partitions assuming that the test of a value of each class is equivalent to a test of other values. In ECP, input domain is partitioned as valid and invalid classes. The validity of each value of the valid class is equivalent; similarly each value of invalid class is equivalently invalid. Equivalence class partitioning is always implemented to the inputs of an already tested component, but in rare cases it may be implemented to the outputs also (Ntafos 1988; Ostrand and Balcer 1988).

3.3 Decision table-based testing

One problem with boundary value analysis and equivalence class portioning is that they are applied on individual conditions. Decision tables are used to evaluate and analyze the combinations of complex input conditions. Decision tables are typically split into four quadrants, Condition stub, Action stub, Condition Entries and Action Entries, as shown below in Table 1.

Each decision corresponds to a variable, relation or predicate whose possible values are listed among the condition alternatives. Each action is a procedure or operation to perform, and the entries specify whether (or in what order) the action is to be performed for the set of condition alternatives the entry corresponds to (Voas and Miller 1995; Weyukar 1993).

3.4 Cause-effect graphing

In cause effect-graphing, input conditions are identified as causes and outputs are denoted as effects. Then these causes and effects are analysed and transformed into a Boolean graph, linking the causes and effects.

4 Motivation

The problem with boundary value analysis, equivalence class partitioning, cause-effect graphing and decision table-based testing technique is that they are well suited for standalone programs or the software where interaction and integrations take place within the single component. These methods are effective for single component software, where interactions occur within a component. But as software system is divided into multiple components, the design as well as the behaviour of the software system changes. To accommodate these changes in design and behaviour, different types of components are developed in different contexts. Then these newly developed components as well as the existing components are integrated. None of these techniques are applicable in the scenarios where we have two or more than two components. To calculate the number of test cases in white box testing we have some techniques like Cyclomatic complexity in the context of individual components as well as for CBS (McCabe 1976; Tiwari and Kumar 2014). But in the context of black-box testing, we have no such technique that can calculate the number of test cases for component-based software.

In this paper, we propose a testing technique and a test case generation method for component-based software, where two or more than two components are integrated. We have compared this method with the Boundary value analysis method and shown that this method is quite suitable for CBS.

Table 1 Decision table

The four quadrants	
Condition Stub	Condition entries
Action Stub	Action entries

5 Components integration-effect graph

To represent the integration of components, at least two components are required. Figure 1 shows the base case of the proposed technique. Components are interacting with each other through request and response edges. Request and response edges correspond to interacting and returning parameters.

5.1 Flow graph notations

Two or more components can interact to access and provide services to each other (Gill and Balkishan 2008; Tiwari and Kumar 2014). To draw the flow graph diagram we use graph theory notations (Berge 1973). A component can make a request for some service to another component, which has been shown as *request edge*, and the component can respond through the r*esponse edge* as shown in Fig. 1.

In coding, these interactions take place through some parameters or operands. The calling component passes some parameter(s) to the called component and in response

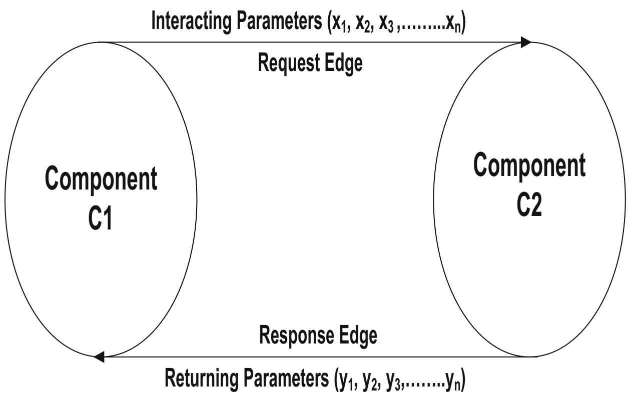

Interacting Parameters ($x_1, x_2, x_3, \ldots x_n$)
Request Edge
Component C1
Component C2
Response Edge
Returning Parameters ($y_1, y_2, y_3, \ldots y_n$)

Fig. 1 Interaction between two components

the called component will return some parameter(s) to the calling one. This scenario is shown in the Fig. 1.

In this paper, we suggest Components Integration-Effect Graph strategy for software where more than one component is involved to provide and access their services as per the specification and architecture document (Gill and Balkishan 2008). When two or more than two components are assembled they must share some sort of information to each other through well defined interfaces (Basili 2001; Chen 2011). In CBS, we have some third party components for which the code is not available. So it is difficult to apply rigorous testing, structural testing or exhaustive testing on such software. Black box strategy is quite suitable for such software.

5.2 Method

Usually on the basis of the architectural design of the software, we draw a control flow graph to show the interaction among the components, as shown in Fig. 1. But here in Integration-Effect graph, we introduced the effect of their integration via an interface. Integration-Effect graph based on the control flow graph is shown in Fig. 2.

With the help of this Integration-Effect graph we can draw an Integration-Effect Matrix as discussed in Table 2. This matrix contains two types of values:

(a) Integration values among components, and
(b) Effect values, generated due to the integration of different components.

Integration-Effect values among components can be computed as:

5.2.1 Effects of individual components ∧ effects generated due to the integration of components

Where, these Effects are the Boolean values, either 0 or 1.

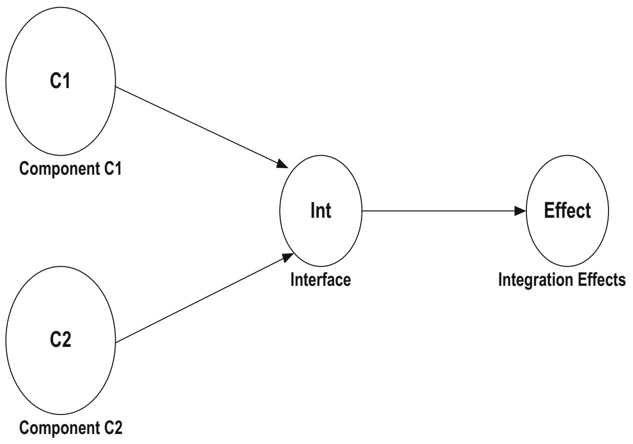


Fig. 2 Integration-effect graph for two components

Table 2 Integration-effect matrix for two components

Components	C1	Effect	C2	Effect
C1	1	Eff(C1): 0/1	0/1	Eff(C1 ∧ C2): 0/1
C2	0/1	Eff(C2 ∧ C1): 0/1	1	Eff(C2): 0/1

On the basis of these values and the above defined formula, we can draw an Integration-effect matrix. From the Integration-effect matrix, we can define the number of test cases in two steps:

Step1. Number of ‘1 s’ under ‘Effect’ column in each row will specify the number of test cases for corresponding component.
Step2. Calculate the total number of test cases achieved in step 1. This specifies the total number of test cases for the given software.

In this paper, we use four cases to analyse our proposed method. In case 1, the graph consists of two components. Case 2 involves three components. Case 3 has four components. Case 4 consists of five components.

5.2.2 Case 1: when we have two components

This case is shown in Fig. 1; here we have two components C1 and C2. These components can communicate through some interaction and returning parameters. This is shown as an edge from C1 to C2 and C2 to C1. To integrate C1 and C2, we need an interface. This interface will be compatible with C1 and C2.

5.2.3 Integration-effect graph

When we integrate these components, they will generate some effect. If the effect is in specified way, we say that components are working as per the requirement. But if the effect is not as per our intention, we need to test these components. We draw integration-effect graph to show the integration of components and the effects of these integrations as shown in Fig. 2.

We represent the component Integration-effect graph as:

The output of integration of component C1 and C2 is 1 (without any error) if, the effect of component C1 (denoted as Eff(C1)) is 1, effect of component C2 (Eff(C2)) is 1 and the integration effect of C1 and C2 is error free. To generate the true Integration-effect of component C1 and C2, we have to take into account the individual as well as combined effects generated due to component C1 and component C2 and the integrated effects of C1 and C2. That is,

$$\boldsymbol{Int}(\boldsymbol{C}1 \wedge \boldsymbol{C}2) = \boldsymbol{Eff}(\boldsymbol{C}1) \wedge \boldsymbol{Eff}(\boldsymbol{C}2) \wedge \boldsymbol{Eff}(\boldsymbol{C}1 \wedge \boldsymbol{C}2)$$

where, $\mathrm{Int}(C1 \wedge C2)$ represents integration of components C1 and C2, $\mathit{Eff}(C1)$ represents the effect generated due to component C1, $\mathit{Eff}(C2)$ represents the effect generated due to component C2, $\mathit{Eff}(C1 \wedge C2)$ represents the effect generated due to the integration of the components C1 and C2, $\wedge$ denotes the AND operation and $\vee$ denotes the OR operation.

If the effect is true that is without any error it is denoted as 1, and if it is negative that is, it having error, it is denoted as 0.

For $\mathit{Int}(C1 \wedge C2)$ to be true, $\mathit{Eff}(C1), \mathit{Eff}(C2),$ $\mathit{and}\, \mathit{Eff}(C1 \wedge C2)$ all must be true, that is, 1. That is, component C1, C2 and integration of C1 and C2 must be error free.

5.2.4 *Integration-effect matrix*

Integration-effect matrix is a row-column matrix. If two components are connected through an edge, then they are represented as 1, otherwise 0. Component C1 is connected with component C2, so the value in the matrix C1 to C2 is 1. Each component is connected to itself by the property of cohesion, so we have used 1 to show their connectivity. Component C1 is connected to itself; therefore C1 to C1 has value 1.

If the component C1 is error free, the value of Eff(C1) is 1, otherwise it is 0. If the component C2 is error free the value of Eff(C2) is 1, otherwise it is 0. If the integration of component C1 and C2 is error free the value of Eff(C1 $\wedge$ C2) is 1, otherwise it is 0.

Possible values of integration-effect matrix for two components is given in Table 2.

5.2.5 *Integration-effect matrix when no error in integration*

Table 3 shows the Integration-effect matrix for the case where all the integrated components are error free and their integration-effects are also error free.

Row C1 has all the values as 1 which represents that C1 is connected with every other component of the graph and the integration effect of C1 with every other component is error free.

Table 3 Integration-effect matrix for two components

Components	C1	Effect	C2	Effect
C1	1	1	1	1
C2	1	1	1	1

Table 4 Integration-effect matrix for two components

Components	C1	Effect	C2	Effect
C1	1	1	1	1
C2	1	0	1	1

Row C2 has all the values as 1 which represents that C2 is connected with every other component of the graph and the integration effect of C1 with every other component is error free.

Row C1 has all the values as 1 which represents that C1 is connected with every other component of the graph.

In row C2, the corresponding value of C1 is 1, means there is an interaction between C2 and C1, but their integration-effect value is 0, means there is an error in their interaction. And this integration needs retesting (Table 4).

5.2.6 *Number of test cases through Integration-effect method (Table 5)*

$$\begin{aligned}&\text{Number of test cases where C1 is involved}\\&\quad = (\text{Number of 1s under ``Effect'' column in row C1}) - 1\\&\quad = 2 - 1\\&\quad = 1\end{aligned}$$

$$\begin{aligned}&\text{Number of test cases where C2 is involved}\\&\quad = (\text{Number of 1s under ``Effect'' column in row C2}) - 1\\&\quad = 2 - 1\\&\quad = 1\end{aligned}$$

$$\begin{aligned}&\text{Total number of test cases derived from this matrix}\\&\quad = \text{Number of test cases for C1} + \text{Number of Test cases for C2}\\&\quad = 1 + 1\\&\quad = 2.\end{aligned}$$

5.2.7 *Number of test cases through boundary value analysis method*

Assuming that ‘n’ is the number of components then, the minimum number of test cases are 4n + 1.

We have two components in this case, therefore n = 2.

$$\begin{aligned}\text{Number of test cases} &= 4n + 1\\ &= 4 * 2 + 1\\ &= 9\end{aligned}$$

Table 5 Integration-effect matrix for two components

Components	C1	Effect	C2	Effect
C1	1	$\boxed{1}$	1	$\boxed{1}$
C2	1	$\boxed{1}$	1	$\boxed{1}$

5.2.8 Case 2: when we have three components

This case is shown in Fig. 3, where we have three components C1, C2 and C3. These components can communicate to each other to provide and access services. Component C1 is communicating with component C2 and component C3. This is shown as an edge from C1 to C2 and C1 to C3. There is no interaction between components C2 and C3.

To integrate C1 and C2, we need an interface; similarly we need interface to integrate C1 and C3.

Now to show the effects of their integration we draw the Integration-Effect graph. When we integrate these components, they will generate some effect. If the effect is true it is denoted as 1, otherwise 0. This is shown in Fig. 4.

From the Fig. 4, we note that atmost two components interact at a time through some interface. So during integration testing we need to focus on atmost two components at a time.

We represent the component Integration-effect graph as:

The output of integration of component C1, C2 and C3 is 1 (i.e., true or without any error) if the effect of component C1 (Eff(C1)) is 1 (without any error), effect of component C2 (Eff(C2)) is 1 (without any error), effect of component C3 (Eff(C3)) is 1 (without any error), and the integration effect of C1, C2 and C3 is error free. To generate the true Integration-effect of component C1, C2 and C3, we have to take into account the individual as well as combined effects generated due to component C1, C2 and component C3.

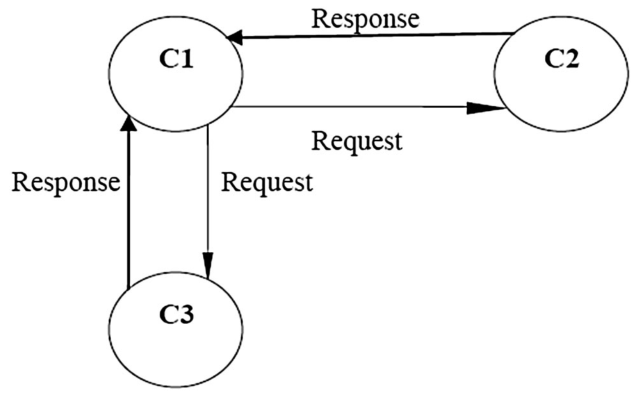


Fig. 3 Interaction among three components

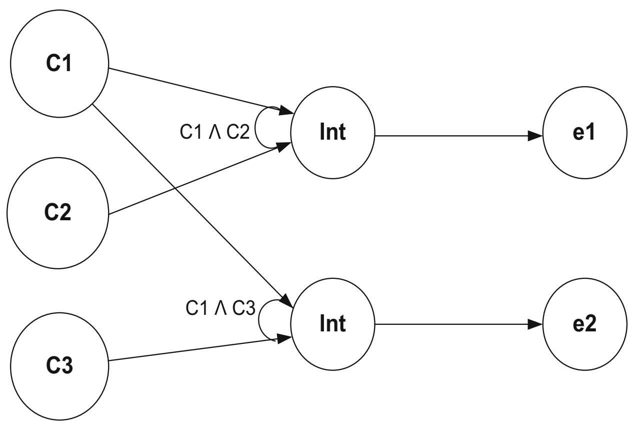


Fig. 4 Integration-effect graph for three components

$$\boldsymbol{Int}(\boldsymbol{C}1 \wedge \boldsymbol{C}2 \wedge \boldsymbol{C}3) = \boldsymbol{Eff}(\boldsymbol{C}1) \wedge \boldsymbol{Eff}(\boldsymbol{C}2) \wedge \boldsymbol{Eff}(\boldsymbol{C}3) \wedge \boldsymbol{Eff}(\boldsymbol{C}1 \wedge \boldsymbol{C}2) \wedge \boldsymbol{Eff}(\boldsymbol{C}2 \wedge \boldsymbol{C}3) \wedge \boldsymbol{Eff}(\boldsymbol{C}1 \wedge \boldsymbol{C}3)$$

where, $\mathrm{Int}(\mathrm{C1} \wedge \mathrm{C2} \wedge \mathrm{C3})$ represents integration of components C1, C2 and C3, $\mathit{Eff}(C1)$ represents the effect generated due to component C1, effect is 1 if C1 is error free and 0 if C1 have an error. $\mathit{Eff}(C2)$ represents the effect generated due to component C2, effect is 1 if C2 is error free and 0 if C2 have an error. $\mathit{Eff}(C3)$ represents the effect generated due to component C3, effect is 1 if C3 is error free and 0 if C3 is having error. $\mathit{Eff}(C1 \wedge C2)$ represents the effect generated due to the components C1 and C2. The integrated effect of C1and C2 is 1 if the integration is error free, otherwise the effect is 0. $\mathit{Eff}(C2 \wedge C3)$ represents the effect generated due to the components C2 and C3. The integrated effect of C2 and C3 is 1 if the integration is error free, otherwise the effect is 0. $\mathit{Eff}(C1 \wedge C3)$ represents the effect generated due to the components C1 and C3. The integrated effect of C1 and C3 is 1 if the integration is error free, otherwise the effect is 0. $\wedge$ denotes the AND operation and $\vee$ denotes the OR operation.

If the effect is positive that is without any error it is denoted as 1, and if it is negative that is, the having error, it is denoted as 0.

For $\mathrm{Int}(\mathrm{C1} \wedge \mathrm{C2} \wedge \mathrm{C3})$ to be true, Eff (C1), Eff(C2), Eff(C3), and $\mathrm{Eff}(\mathrm{C1} \wedge \mathrm{C2} \wedge \mathrm{C3})$ all must be true, that is, 1.

Table 6 Integration-effect matrix for three components

Components	C1	Effect	C2	Effect	C3	Effect
C1	1	Eff(C1): 0/1	1	Eff(C1 ∧ C2): 0/1	1	Eff(C1∧ C3): 0/1
C2	1	Eff(C2 ∧ C1): 0/1	1	Eff(C2): 0/1	0	Eff(C2 ∧ C3): 0/1
C3	1	Eff(C3 ∧ C1): 0/1	0	Eff(C3 ∧ C2): 0/1	1	Eff(C3): 0/1

Table 7 Integration-effect matrix for three components

Components	C1	Effect	C2	Effect	C3	Effect
C1	1	1	1	1	1	1
C2	1	1	1	1	0	0
C3	1	1	0	0	1	1

5.2.9 *Possible values of integration-effect matrix for three components (shown in Table 6)*

Table 7 shows the integration-effect matrix (Fig. 4):

5.2.10 *Number of test cases derived from the matrix*

$$\begin{aligned}&\text{Number of test cases where C1 is involved}\\&= (\text{Number of 1s under ``Effect'' column in row C1}) - 1\\&= 3 - 1\\&= 2\end{aligned}$$

$$\begin{aligned}&\text{Number of test cases where C2 is involved}\\&= (\text{Number of 1s under ``Effect'' column in row C2}) - 1\\&= 2 - 1\\&= 1\end{aligned}$$

$$\begin{aligned}&\text{Number of test cases where C3 is involved}\\&= (\text{Number of 1s under ``Effect'' column in row C3}) - 1\\&= 2 - 1\\&= 1\end{aligned}$$

$$\begin{aligned}&\text{Total number of test cases derived from this matrix}\\&= \text{Number of test cases for C1}\\&\quad + \text{Number of test cases for C2}\\&\quad + \text{Number of test cases for C3}\\&= 2 + 1 + 1\\&= 4.\end{aligned}$$

5.2.11 *Number of test cases through boundary value analysis method*

Assuming that ‘n’ is the number of components then, the minimum number of test cases are 4n + 1.

We have 3 components in this case, therefore n = 3.

$$\begin{aligned}\text{Number of test cases} &= 4n + 1\\&= 4 * 3 + 1\\&= 13\end{aligned}$$

5.2.12 *Case 3: when we have four components*

This is the case when we have four components C1, C2, C3, and C4. This case is shown in Fig. 5. Component C1 is communicating with component C2, C3 and component C4. Component C2 is integrated with C1 and C4 and C3 in integrated with C1 and C4. And finally C4 is interacting with C1, C2, and C3. Every requesting component gets a response. This is shown as an edge from C1 to C2 and C2 to C1. Similarly an edge from C1 to C3 and C3 to C1, and so on. Here outgoing edges from the components are used to show the request by the component to other components and the incoming edges to the component are used to show the response of requests by some other components.

Integration of these components will produce some effect. As the previous cases when these components are integrated they will generate some effect. Figure 6 shows the Integration-effect graph of four components.

We represent the component Integration-effect graph as:

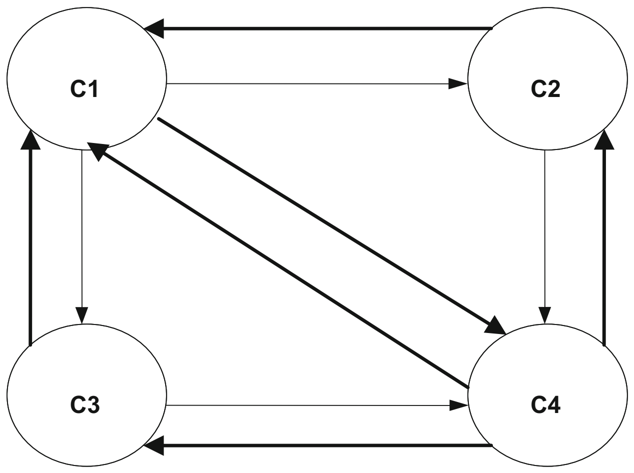


Fig. 5 Interaction among four components

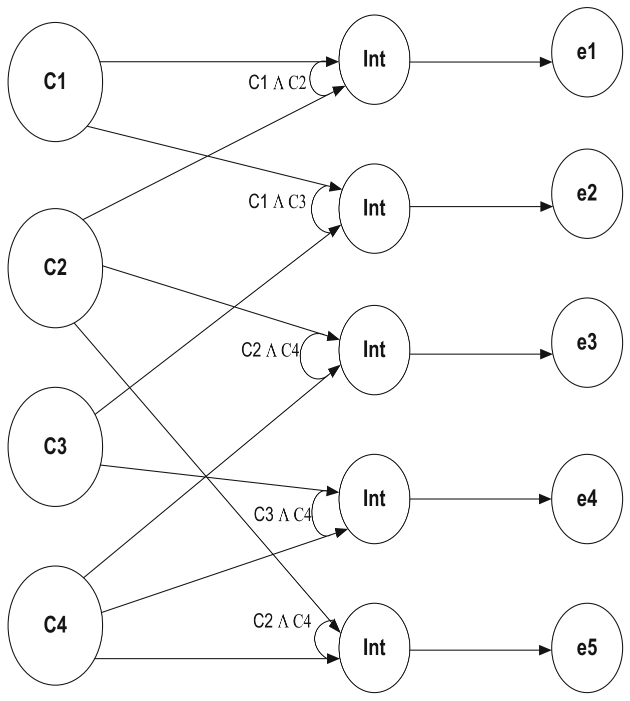


Fig. 6 Components integration-effect graph for four components

The output of integration of component C1, C2, C3 and C4 is 1 (i.e., true or without any error) if the effect of component C1 (Eff(C1)) is 1 (without any error), effect of component C2 (Eff(C2)) is 1 (without any error), effect of component C3 (Eff(C3)) is 1 (without any error), effect of component C4 (Eff(C4)) is 1 (without any error) and the integration effect of C1, C2, C3 and C4 is error free.
To generate the true Integration effect of component C1, C2, C3 and C4, we have to take into account the individual as well as combined effects generated due to component C1, C2, C3 and component C4.

$$\begin{aligned}\boldsymbol{Int}(\boldsymbol{C}1 \wedge \boldsymbol{C}2 \wedge \boldsymbol{C}3 \wedge \boldsymbol{C}4) &= \boldsymbol{Eff}(\boldsymbol{C}1) \wedge \boldsymbol{Eff}(\boldsymbol{C}2) \wedge \boldsymbol{Eff}(\boldsymbol{C}3) \wedge \boldsymbol{Eff}(\boldsymbol{C}4) \\ &\wedge \boldsymbol{Eff}(\boldsymbol{C}1 \wedge \boldsymbol{C}2) \wedge \boldsymbol{Eff}(\boldsymbol{C}1 \wedge \boldsymbol{C}3) \\ &\wedge \boldsymbol{Eff}(\boldsymbol{C}1 \wedge \boldsymbol{C}4) \wedge \boldsymbol{Eff}(\boldsymbol{C}2 \wedge \boldsymbol{C}1) \\ &\wedge \boldsymbol{Eff}(\boldsymbol{C}2 \wedge \boldsymbol{C}3) \wedge \boldsymbol{Eff}(\boldsymbol{C}2 \wedge \boldsymbol{C}4) \\ &\wedge \boldsymbol{Eff}(\boldsymbol{C}3 \wedge \boldsymbol{C}1) \wedge \boldsymbol{Eff}(\boldsymbol{C}3 \wedge \boldsymbol{C}2) \\ &\wedge \boldsymbol{Eff}(\boldsymbol{C}3 \wedge \boldsymbol{C}4) \wedge \boldsymbol{Eff}(\boldsymbol{C}4 \wedge \boldsymbol{C}1) \\ &\wedge \boldsymbol{Eff}(\boldsymbol{C}4 \wedge \boldsymbol{C}2) \wedge \boldsymbol{Eff}(\boldsymbol{C}4 \wedge \boldsymbol{C}3)\end{aligned}$$

where, $Int(C1 \wedge C2 \wedge C3 \wedge C4)$ represents integration of components C1, C2, C3 and C4, $Eff(C1)$ represents the effect generated due to component C1, effect is 1 if C1 is error free and 0 if C1 is having error. $Eff(C2)$ represents the effect generated due to component C2, effect is 1 if C2 is error free and 0 if C2 is having error. $Eff(C3)$ represents the effect generated due to component C3, effect is 1 if C3 is error free and 0 if C3 is having error. $Eff(C4)$ represents the effect generated due to component C4, effect is 1 if C4 is error free and 0 if C4 is having error. $Eff(C1 \wedge C2)$ represents the effect generated due to the components C1 and C2. The integrated effect of C1 and C2 is 1 if the integration is error free, otherwise the effect is 0.

5.2.13 *Possible values of integration-effect matrix for four components (shown in Table 8)*

Table 9 shows the integration-effect matrix (Fig. 6):
$Eff(C1 \wedge C3)$ represents the effect generated due to the components C1 and C3. The integrated effect of C1 and C3 is 1 if the integration is error free, otherwise the effect is 0. $Eff(C1 \wedge C4)$ represents the effect generated due to the components C1 and C4. The integrated effect of C1 and C4 is 1 if the integration is error free, otherwise the effect is 0. $Eff(C2 \wedge C1)$ represents the effect generated due to the components C2 and C1. The integrated effect of C2 and C1 is 1 if the integration is error free, otherwise the effect is 0. In the same manner values of the other components can be derived. ∧ denotes the AND operation and ∨ denotes the OR operation.

If the effect is positive that is without any error it is denoted as 1, and if it is negative that is, the having error, it is denoted as 0.

For $Int(C1 \wedge C2 \wedge C3 \wedge C4)$ to be true, $Eff(C1)$, $Eff(C2)$, $Eff(C3)$, $Eff(C4)$ *and* $Eff(C1 \wedge C2 \wedge C3 \wedge C4)$ all must be true, that is, 1.

Table 9 Integration-effect matrix for four components

Components	C1	Effect	C2	Effect	C3	Effect	C4	Effect
C1	1	1	1	1	1	1	1	1
C2	1	1	1	1	0	0	1	1
C3	1	1	0	0	1	1	1	1
C4	1	1	1	1	1	1	1	1

Table 8 Integration-effect matrix for four components

Components	C1	Effect	C2	Effect	C3	Effect	C4	Effect
C1	1	Eff(C1): 0/1	1	Eff(C1 ∧ C2): 0/1	1	Eff(C1 ∧ C3): 0/1	0	Eff(C1 ∧ C4): 0/1
C2	0	Eff(C2 ∧ C1): 0/1	1	Eff(C2): 0/1	0	Eff(C2 ∧ C3): 0/1	1	Eff(C2 ∧ C4): 0/1
C3	0	Eff(C3 ∧ C1): 0/1	0	Eff(C3 ∧ C2): 0/1	1	Eff(C3): 0/1	1	Eff(C3 ∧ C4): 0/1
C4	1	Eff(C4 ∧ C1): 0/1	0	Eff(C4 ∧ C2): 0/1	0	Eff(C4 ∧ C3): 0/1	1	Eff(C4): 0/1

5.2.14 *Findings and the number of test cases derived from the matrix*

$$\begin{aligned}\text{Number of test cases where C1 is involved}\\ &= (\text{Number of 1s under ``Effect'' column in row C1}) - 1\\ &= 4 - 1\\ &= 3\end{aligned}$$

$$\begin{aligned}\text{Number of test cases where C2 is involved}\\ &= (\text{Number of 1s under ``Effect'' column in row C2}) - 1\\ &= 3 - 1\\ &= 2\end{aligned}$$

$$\begin{aligned}\text{Number of test cases where C3 is involved}\\ &= (\text{Number of 1s under ``Effect'' column in row C3}) - 1\\ &= 3 - 1\\ &= 2\end{aligned}$$

$$\begin{aligned}\text{Number of test cases where C4 is involved}\\ &= (\text{Number of 1s under ``Effect'' column in row C4}) - 1\\ &= 4 - 1\\ &= 3\end{aligned}$$

$$\begin{aligned}\text{Total number of test cases derived from this matrix}\\ &= \text{Number of test cases for C1}\\ &\quad + \text{Number of Test cases for C2}\\ &\quad + \text{Number of Test cases for C3}\\ &\quad + \text{Number of Test cases for C4}\\ &= 3 + 2 + 2 + 3\\ &= 10.\end{aligned}$$

5.2.15 *Number of test cases through boundary value analysis method*

Assuming that ‘n’ is the number of components then, the minimum number of test cases are 4n + 1.

We have 2 components in this case, therefore n = 2.

$$\begin{aligned}\text{Number of test cases} &= 4\text{n} + 1\\ &= 4 * 4 + 1\\ &= 17\end{aligned}$$

5.2.16 *Case 4: when we have five components*

This is the case where we have five components C1, C2, C3, C4 and C5. This case is shown in Fig. 7. Component C1 is communicating with component C2, C3 and component C4. Component C2 in integrated with C1, C3 and C4. Component C3 is communicating with C1, C2, C4 and C5. Component C4 is sharing information with C1, C2, C3 and C5. Component C5 is having interaction with C3 and C4. All the interacting components have request and response edges. This is shown as an edge from C1 to C2 and C2 to C1.

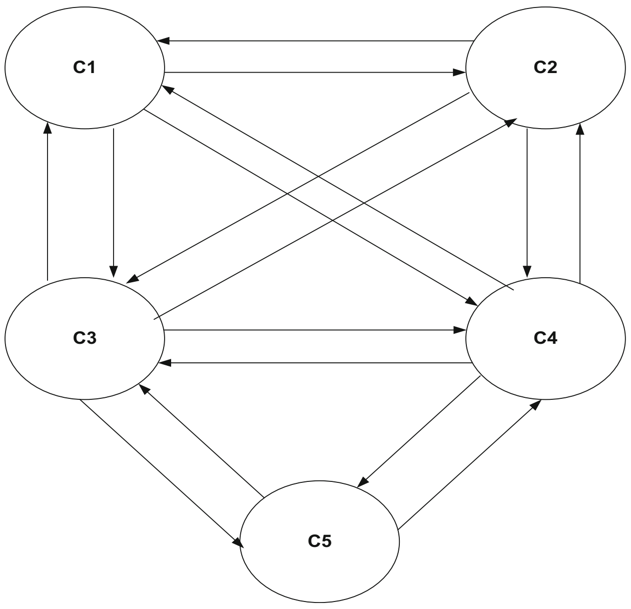


Fig. 7 Interaction between five components

Similarly edges from C1 to C3 and C3 to C1 are drawn to show the communication. To show the interaction between C3, C4 and C5 we have used edges between them. There is a request edge and response edge from C4 to C1and C1 to C4, and finally edges from C5 to C4 to show the interaction between them.

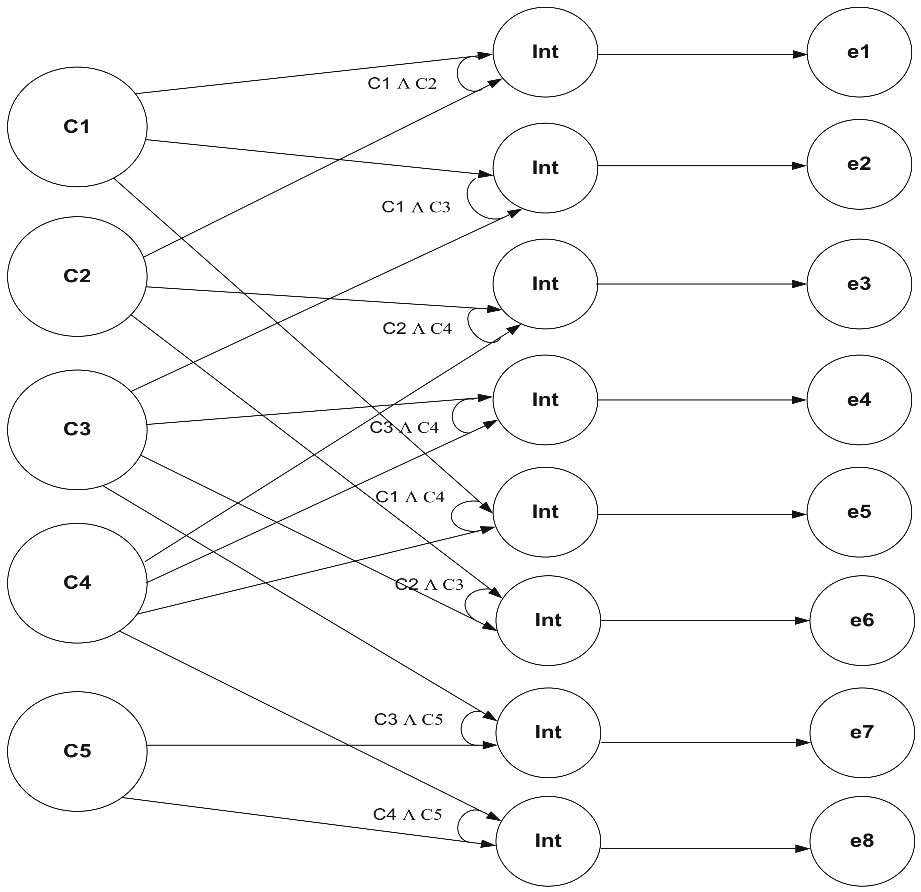
When we integrate these components, they will produce some effect. If the effect is in specified way we say that components are working as per the requirement. But if the effect is not as per our intention, we need to test these components. This is shown in Fig. 8.

From Fig. 8, it is noted that the output of integration of component C1, C2, C3, C4 and C5 is 1 (i.e., true or without any error) if the effect of component C1 (Eff(C1)) is 1 (without any error), effect of component C2 (Eff(C2)) is 1 (without any error), effect of component C3 (Eff(C3)) is 1 (without any error), effect of component C4 (Eff(C4)) is 1 (without any error), effect of component C5 (Eff(C5)) is 1 (without any error), and the integration effect of C1, C2, C3, C4 and C5 is error free.

To generate the true Integration effect of component C1, C2, C3, C4 and C5, we have to take into account the individual as well as combined effects generated due to component C1, C2, C3, C4 and component C5.

We represent the component Integration-effect graph as:

Fig. 8 Components integration-effect graph for five components



$$\begin{aligned}&\boldsymbol{Int}(\boldsymbol{C}1 \wedge \boldsymbol{C}2 \wedge \boldsymbol{C}3 \wedge \boldsymbol{C}4 \wedge \boldsymbol{C}5)\\ &\quad = \boldsymbol{Eff}(\boldsymbol{C}1) \wedge \boldsymbol{Eff}(\boldsymbol{C}2) \wedge \boldsymbol{Eff}(\boldsymbol{C}3) \wedge \boldsymbol{Eff}(\boldsymbol{C}4)\\ &\qquad \wedge \boldsymbol{Eff}(\boldsymbol{C}5) \wedge \boldsymbol{Eff}(\boldsymbol{C}1 \wedge \boldsymbol{C}2) \wedge \boldsymbol{Eff}(\boldsymbol{C}1 \wedge \boldsymbol{C}3)\\ &\qquad \wedge \boldsymbol{Eff}(\boldsymbol{C}1 \wedge \boldsymbol{C}4) \wedge \boldsymbol{Eff}(\boldsymbol{C}1 \wedge \boldsymbol{C}5)\\ &\qquad \wedge \boldsymbol{Eff}(\boldsymbol{C}2 \wedge \boldsymbol{C}1) \wedge \boldsymbol{Eff}(\boldsymbol{C}2 \wedge \boldsymbol{C}3)\\ &\qquad \wedge \boldsymbol{Eff}(\boldsymbol{C}2 \wedge \boldsymbol{C}4) \wedge \boldsymbol{Eff}(\boldsymbol{C}2 \wedge \boldsymbol{C}5)\\ &\qquad \wedge \boldsymbol{Eff}(\boldsymbol{C}3 \wedge \boldsymbol{C}1) \wedge \boldsymbol{Eff}(\boldsymbol{C}3 \wedge \boldsymbol{C}2)\\ &\qquad \wedge \boldsymbol{Eff}(\boldsymbol{C}3 \wedge \boldsymbol{C}4) \wedge \boldsymbol{Eff}(\boldsymbol{C}3 \wedge \boldsymbol{C}5)\\ &\qquad \wedge \boldsymbol{Eff}(\boldsymbol{C}4 \wedge \boldsymbol{C}1) \wedge \boldsymbol{Eff}(\boldsymbol{C}4 \wedge \boldsymbol{C}2)\\ &\qquad \wedge \boldsymbol{Eff}(\boldsymbol{C}4 \wedge \boldsymbol{C}3) \wedge \boldsymbol{Eff}(\boldsymbol{C}4 \wedge \boldsymbol{C}5)\\ &\qquad \wedge \boldsymbol{Eff}(\boldsymbol{C}5 \wedge \boldsymbol{C}1) \wedge \boldsymbol{Eff}(\boldsymbol{C}5 \wedge \boldsymbol{C}2)\\ &\qquad \wedge \boldsymbol{Eff}(\boldsymbol{C}5 \wedge \boldsymbol{C}3) \wedge \boldsymbol{Eff}(\boldsymbol{C}5 \wedge \boldsymbol{C}4)\end{aligned}$$

where, $Int(C1 \wedge C2 \wedge C3 \wedge C4 \wedge C5)$ represents Integration of components C1, C2, C3, C4 and C5, $Eff(C1)$ represents the effect generated due to component C1, effect is 1 if C1 is error free and 0 if C1 is having error. $Eff(C2)$ represents the effect generated due to component C2, effect is 1 if C2 is error free and 0 if C2 is having error. $Eff(C1 \wedge C3)$ represents the effect generated due to the components C1 and C3. The integrated effect of C1 and C3 is 1 if the integration is error free, otherwise the effect is 0. $Eff(C1 \Lambda C4)$ represents the effect generated due to the components C1 and C4. The integrated effect of C1 and C4 is 1 if the integration is error free, otherwise the effect is 0. $Eff(C1 \wedge C5)$ represents the effect generated due to the components C1 and C5. The integrated effect of C1 and C5 is 1 if the integration is error free, otherwise the effect is 0. In the same manner other components value can be derived. $\wedge$ denotes the AND operation and $\vee$ denotes the OR operation.

If the effect is positive that is without any error it is denoted as 1, and if it is negative that is, the having error, it is denoted as 0.

For $Int(C1 \wedge C2 \wedge C3 \wedge C4 \wedge C5)$ to be true,$Eff(C1)$, $Eff(C2)$, $Eff(C3), f(C4)$, $Eff(C5)\, and\, Eff(C1 \wedge C2 \wedge C3 \wedge C4 \wedge C5)$ all must be true, that is, 1. $Eff(C3)$ represents the effect generated due to component C3, effect is 1 if C3 is error free and 0 if C3 is having error. $Eff(C4)$ represents the effect generated due to component C4, effect is 1 if C4 is error free and 0 if C4 is having error. $Eff(C5)$ represents the effect generated due to component C5, effect is 1 if C5 is error free and 0 if C5 is having error. $Eff(C1 \wedge C2)$ represents the effect generated due to the components C1 and C2. The integrated effect of C1and C2 is 1 if the integration is error free, otherwise the effect is 0.

Table 10 Integration-effect matrix for five components

Components	C1	Effect	C2	Effect	C3	Effect	C4	Effect	C5	Effect
C1	1	Eff(C1): 0/1	1	Eff(C1 ∧ C2): 0/1	1	Eff(C1 ∧ C3): 0/1	0	Eff(C1 ∧ C4): 0/1	0	Eff(C1∧ C5): 0/1
C2	0	Eff(C2 ∧ C1): 0/1	1	Eff(C2): 0/1	0	Eff(C2 ∧ C3): 0/1	1	Eff(C2 ∧ C4): 0/1	0	Eff(C2 ∧ C5): 0/1
C3	0	Eff(C3 ∧ C1): 0/1	0	Eff(C3 ∧ C2): 0/1	1	Eff(C3): 0/1	1	Eff(C3 ∧ C4): 0/1	1	Eff(C3 ∧ C5): 0/1
C4	1	Eff(C4 ∧ C1): 0/1	0	Eff(C4 ∧ C2): 0/1	0	Eff(C4 ∧ C3): 0/1	1	Eff(C4): 0/1	0	Eff(C4 ∧ C5): 0/1
C5	0	Eff(C5 ∧ C1): 0/1	0	Eff(C5 ∧ C2): 0/1	0	Eff(C5 ∧ C3): 0/1	0	Eff(C5 ∧ C4): 0/1	1	Eff(C5 ∧ C5): 0/1

Table 11 Integration-effect matrix for five components

Components	C1	Effect	C2	Effect	C3	Effect	C4	Effect	C5	Effect
C1	1	1	1	1	1	1	1	1	0	0
C2	1	1	1	1	1	1	1	1	0	0
C3	1	1	1	1	1	1	1	1	1	1
C4	1	1	1	1	1	1	1	1	1	1
C5	0	0	0	0	1	1	1	1	1	1

5.2.17 *Possible values of integration-effect matrix for five components (shown in Table 10)*

Table 11 shows the integration-effect matrix (Fig. 8):

5.2.18 *Findings and the number of test cases derived from the matrix*

$$\begin{aligned}&\text{Number of test cases where C1 is involved}\\&= (\text{Number of 1s under ``Effect'' column in row C1}) - 1\\&= 4-1\\&= 3\end{aligned}$$

$$\begin{aligned}&\text{Number of test cases where C2 is involved}\\&= (\text{Number of 1s under ``Effect'' column in row C2}) - 1\\&= 4-1\\&= 3\end{aligned}$$

$$\begin{aligned}&\text{Number of test cases where C3 is involved}\\&= (\text{Number of 1s under ``Effect'' column in row C3}) - 1\\&= 5-1\\&= 4\end{aligned}$$

$$\begin{aligned}&\text{Number of test cases where C4 is involved}\\&= (\text{Number of 1s under ``Effect'' column in row C4}) - 1\\&= 5-1\\&= 4\end{aligned}$$

$$\begin{aligned}&\text{Number of test cases where C5 is involved}\\&= (\text{Number of 1s under ``Effect'' column in row C5}) - 1\\&= 3-1\\&= 2\end{aligned}$$

$$\begin{aligned}&\text{Total number of test cases derived from this matrix}\\&= \text{Number of test cases for C1}\\&\quad + \text{Number of test cases for C2}\\&\quad + \text{Number of test cases for C3}\\&\quad + \text{Number of test cases for C4}\\&= 3+3+4+4+2\\&= 16.\end{aligned}$$

5.2.19 *Number of test cases through boundary value analysis method*

Assuming that ‘n’ is the number of components then, the minimum number of test cases are 4n + 1.

We have 2 components in this case, therefore n = 2.

$$\begin{aligned}\text{Number of test cases} &= 4n+1\\&= 4*5+1\\&= 21\end{aligned}$$

6 Case study

To analyze the proposed technique, we are taking the case study of *Request Redirection Model for JSP* (Fig. 9). In this model the client's request reaches an initial JSP page.

The page may start generating some results but at some point it may want to dynamically include the contents of some other page. These contents may be static but may also be dynamically generated by some other JSP page, servlet class, or some legacy mechanism like ASP.

Here we have five components interacting with each other to fulfill the client's request. If the client's request fulfils successfully without any error then we can generate test cases using the proposed technique.

When we integrate these components, they will produce some effect. If the effect is in specified way we say that components are working as per the requirement. But if the effect is not as per our intention, we need to test these components. This is shown in Fig. 10.

From Fig. 10, it is noted that the output of integration of component C1, C2, C3, C4 and C5 is 1 (i.e., true or without any error) if the effect of component C1 (Eff(C1)) is 1 (without any error), effect of component C2.

(Eff(C2)) is 1 (without any error), effect of component C3 (Eff(C3)) is 1 (without any error), effect of component C4 (Eff(C4)) is 1 (without any error), effect of component C5 (Eff(C5)) is 1 (without any error), and the integration effect of C1, C2, C3, C4 and C5 is error free.

To generate the true Integration effect of component C1, C2, C3, C4 and C5, we have to take into account the individual as well as combined effects generated due to component C1, C2, C3, C4 and component C5.

$$\begin{aligned} \boldsymbol{Int}(\boldsymbol{C}1 \wedge \boldsymbol{C}2 \wedge \boldsymbol{C}3 \wedge \boldsymbol{C}4 \wedge \boldsymbol{C}5) &= \boldsymbol{Eff}(\boldsymbol{C}1) \wedge \boldsymbol{Eff}(\boldsymbol{C}2) \wedge \boldsymbol{Eff}(\boldsymbol{C}3) \wedge \boldsymbol{Eff}(\boldsymbol{C}4) \\ &\quad \wedge \boldsymbol{Eff}(\boldsymbol{C}5) \wedge \boldsymbol{Eff}(\boldsymbol{C}1 \wedge \boldsymbol{C}2) \wedge \boldsymbol{Eff}(\boldsymbol{C}1 \wedge \boldsymbol{C}3) \\ &\quad \wedge \boldsymbol{Eff}(\boldsymbol{C}1 \wedge \boldsymbol{C}4) \wedge \boldsymbol{Eff}(\boldsymbol{C}1 \wedge \boldsymbol{C}5) \\ &\quad \wedge \boldsymbol{Eff}(\boldsymbol{C}2 \wedge \boldsymbol{C}1) \wedge \boldsymbol{Eff}(\boldsymbol{C}2 \wedge \boldsymbol{C}3) \\ &\quad \wedge \boldsymbol{Eff}(\boldsymbol{C}2 \wedge \boldsymbol{C}4) \wedge \boldsymbol{Eff}(\boldsymbol{C}2 \wedge \boldsymbol{C}5) \\ &\quad \wedge \boldsymbol{Eff}(\boldsymbol{C}3 \wedge \boldsymbol{C}1) \wedge \boldsymbol{Eff}(\boldsymbol{C}3 \wedge \boldsymbol{C}2) \\ &\quad \wedge \boldsymbol{Eff}(\boldsymbol{C}3 \wedge \boldsymbol{C}4) \wedge \boldsymbol{Eff}(\boldsymbol{C}3 \wedge \boldsymbol{C}5) \\ &\quad \wedge \boldsymbol{Eff}(\boldsymbol{C}4 \wedge \boldsymbol{C}1) \wedge \boldsymbol{Eff}(\boldsymbol{C}4 \wedge \boldsymbol{C}2) \\ &\quad \wedge \boldsymbol{Eff}(\boldsymbol{C}4 \wedge \boldsymbol{C}3) \wedge \boldsymbol{Eff}(\boldsymbol{C}4 \wedge \boldsymbol{C}5) \\ &\quad \wedge \boldsymbol{Eff}(\boldsymbol{C}5 \wedge \boldsymbol{C}1) \wedge \boldsymbol{Eff}(\boldsymbol{C}5 \wedge \boldsymbol{C}2) \\ &\quad \wedge \boldsymbol{Eff}(\boldsymbol{C}5 \wedge \boldsymbol{C}3) \wedge \boldsymbol{Eff}(\boldsymbol{C}5 \wedge \boldsymbol{C}4) \end{aligned}$$

where, $Int(C1 \wedge C2 \wedge C3 \wedge C4 \wedge C5)$ represents Integration of components C1, C2, C3, C4 and C5, $Eff(C1)$ represents the effect generated due to component C1, effect is 1 if C1 is error free and 0 if C1 is having error. $Eff(C2)$ represents the effect generated due to component C2, effect is 1 if C2 is error free and 0 if C2 is having error. $Eff(C3)$ represents the effect generated due to component C3, effect is 1 if C3 is error free and 0 if C3 is having error. $Eff(C4)$ represents the effect generated due to component C4, effect is 1 if C4 is error free and 0 if C4 is having error. $Eff(C5)$ represents the effect generated due to component C5, effect is 1 if C5 is error free and 0 if C5 is having error. $Eff(C1 \wedge C2)$ represents the effect generated due to the components C1 and C2. The integrated effect of C1and C2 is 1 if the integration is error free, otherwise the effect is 0. $Eff(C1 \wedge C3)$ represents the effect generated due to the components C2 and C3. The integrated effect of C2 and C3 is 1 if the integration is error free, otherwise the effect is 0. $Eff(C1 \wedge C4)$ represents the effect generated due to the

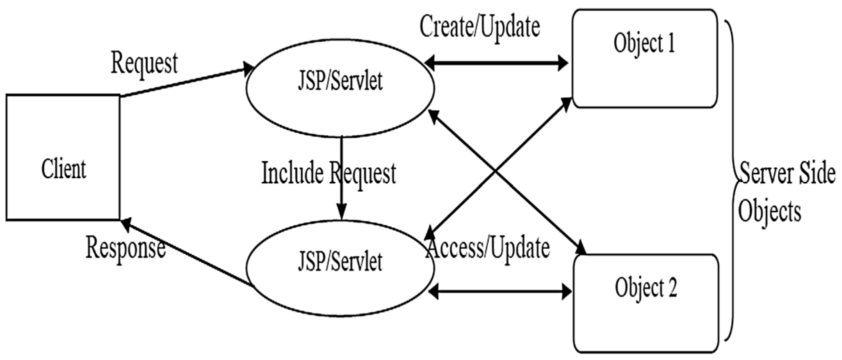


Fig. 9 Request Redirection Model for JSP

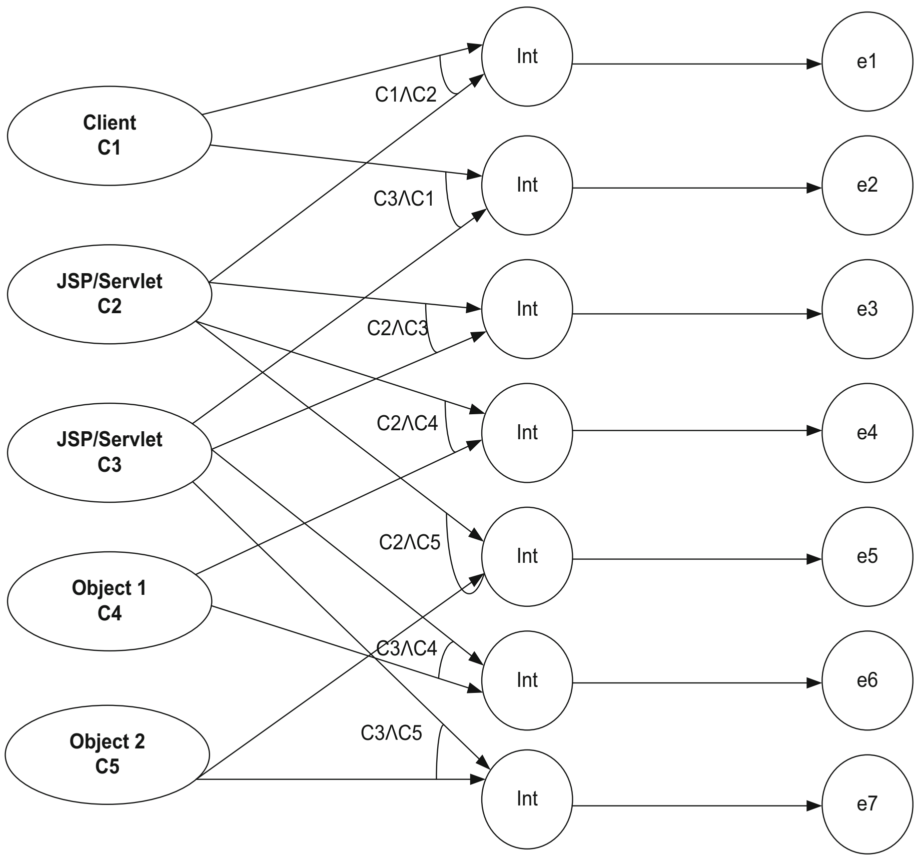


Fig. 10 Integration-effect graph for Request Redirection Model for JSP

components C1 and C3. The integrated effect of C1 and C3 is 1 if the integration is error free, otherwise the effect is 0. $Eff(C1 \wedge C5)$ represents the effect generated due to the components C1 and C4. The integrated effect of C1 and C4 is 1 if the integration is error free, otherwise the effect is 0. $Eff(C2 \wedge C1)$ represents the effect generated due to the components C2 and C1. The integrated effect of C2 and C1 is 1 if the integration is error free, otherwise the effect is 0. In the same manner other components value can be derived. ∧ denotes the AND operation and ∨ denotes the OR operation.

6.1 Possible values of integration-effect matrix (Request Redirection Model for JSP) for five components (shown in Table 12)

Table 13 shows the tntegration-effect matrix (Request Redirection Model for JSP) for Fig. 10.

Table 12 Integration-effect graph for Request Redirection Model for JSP

Components	C1	Effect	C2	Effect	C3	Effect	C4	Effect	C5	Effect
C1	1	Eff(C1): 0/1	1	Eff(C1 ∧ C2): 0/1	1	Eff(C1 ∧ C3): 0/1	0	Eff(C1 ∧ C4): 0/1	0	Eff(C1∧ C5): 0/1
C2	0	Eff(C2 ∧ C1): 0/1	1	Eff(C2): 0/1	0	Eff(C2 ∧ C3): 0/1	1	Eff(C2 ∧ C4): 0/1	0	Eff(C2∧ C5): 0/1
C3	0	Eff(C3 ∧ C1): 0/1	0	Eff(C3 ∧ C2): 0/0	1	Eff(C3): 0/1	1	Eff(C3 ∧ C4): 0/1	1	Eff(C3∧ C5): 0/1
C4	1	Eff(C4 ∧ C1): 0/1	0	Eff(C4 ∧ C2): 0/1	0	Eff(C4 ∧ C3): 0/1	1	Eff(C4): 0/1	0	Eff(C4∧ C5): 0/1
C5	0	Eff(C5 ∧ C1): 0/1	0	Eff(C5 ∧ C2): 0/1	0	Eff(C5 ∧ C3): 0/1	0	Eff(C5 ∧ C4): 0/1	1	Eff(C5): 0/1

Table 13 Integration-effect graph for Request Redirection Model for JSP

Components	C1	Effect	C2	Effect	C3	Effect	C4	Effect	C5	Effect
C1	1	1	1	1	0	0	0	0	0	0
C2	0	0	1	1	1	1	1	1	1	1
C3	1	1	0	0	1	1	1	1	1	1
C4	0	0	1	1	1	1	1	1	0	0
C5	1	1	1	1	1	1	0	0	1	1

6.2 Findings and the number of test cases derived from the matrix

$$\begin{aligned}&\text{Number of test cases where C1 is involved}\\&\quad= (\text{Number of 1s under ``Effect'' column in row C1}) - 1\\&\quad= 2 - 1\\&\quad= 1\end{aligned}$$

$$\begin{aligned}&\text{Number of test cases where C2 is involved}\\&\quad= (\text{Number of 1s under ``Effect'' column in row C2}) - 1\\&\quad= 4 - 1\\&\quad= 3\end{aligned}$$

$$\begin{aligned}&\text{Number of test cases where C3 is involved}\\&\quad= (\text{Number of 1s under ``Effect'' column in row C3}) - 1\\&\quad= 4 - 1\\&\quad= 3\end{aligned}$$

$$\begin{aligned}&\text{Number of test cases where C4 is involved}\\&\quad= (\text{Number of 1s under ``Effect'' column in row C4}) - 1\\&\quad= 3 - 1\\&\quad= 2\end{aligned}$$

$$\begin{aligned}&\text{Number of test cases where C5 is involved}\\&\quad= (\text{Number of 1s under ``Effect'' column in row C5}) - 1\\&\quad= 4 - 1\\&\quad= 3\end{aligned}$$

$$\begin{aligned}&\text{Total number of test cases derived from this matrix}\\&\quad= \text{Number of test cases for C1}\\&\qquad+ \text{Number of test cases for C2}\\&\qquad+ \text{Number of test cases for C3}\\&\qquad+ \text{Number of test cases for C4}\\&\qquad+ \text{Number of test cases for C5}\\&\quad= 1 + 3 + 3 + 2 + 3\\&\quad= 12.\end{aligned}$$

6.3 Number of test cases through boundary value analysis method

Assuming that ‘n’ is the number of components then, the minimum number of test cases are 4n + 1.

We have two components in this case, therefore n = 2.

$$\begin{aligned}\text{Number of test cases} &= 4n + 1\\&= 4 * 5 + 1\\&= 21\end{aligned}$$

7 Conclusions

Boundary value analysis, Equivalence Class Partitioning, Cause Effect graph and Decision table method, all are intra- module methods that are applicable within the module or component only. The proposed Integration-Effect matrix is an inter-module method that is applicable to interaction among components.

Component-based software development emphasizes the reusability of existing components available in the repository, but it also supports the innovative solutions of the problems. This proposed strategy is helpful to test and record the effects of such components whose code is accessible, and those third party components for which code is not available. This strategy includes only the interaction and integration attributes of the components. In this paper we have not included the individual testing techniques of the components as we are trying to develop method to analyze the interaction among components only. We get a greater degree of predictability in terms of costs, effort, quality and risk if we can predict the testability of the software properly and early.

References

Basili VR (2001) Cots-based systems top 10 list. IEEE Comput 34(5):91–93

Bennatan EM (1992) Software project management: a practitioner’s approach. McGraw-Hill Ryerson, Limited, UK

Berge C (1973) Graphs and hypergraphs. North-Holland, Amsterdam

Capretz L (2005) Y: a new component-based software life cycle model. J Comput Sci 1(1):76–78

Chen J (2011) Complexity metrics for component-based software systems. Int J Digit Content Technol Its Appl 5(3):235–244

Elberzhager F, Rosbach A, Münch J, Eschbach R (2012) Reducing testing effort: a systematic mapping study on existing approaches. Inf Softw Technol 54(10):1092–1106

Gao JZ, Tsao HS, Wu Y (2003) Testing and quality assurance for component-based software. Artech House, Boston

Gill NS, Balkishan (2008) Dependency and interaction oriented complexity metrics of component based systems. ACM SIGSOFT Softw Eng Notes 33(2). doi:10.1145/1350802.1350810

Kozaczynski W, Booch G (1998) Component-based software engineering. IEEE Softw 15(5):34–36

McCabe T (1976) A complexity measure. IEEE Trans Softw Eng 2(8):308–320

Myers GJ (2004) The art of software testing, 2nd edn. Wiley, New York

Ntafos SC (1988) A comparison of some structural testing strategies. IEEE Trans Softw Eng 14(6):868–874

Ostrand TJ, Balcer MJ (1988) The category-partition method for specifying and generating functional tests. Commun ACM 31(6):676–686

Pressmen SR (2001) Software engineering: a practitioners Approach, 5th edn. McGraw Hill, International

Ramamoorthy C, Ho S, Chen W (1976) On the automated generation of program test data. IEEE Trans Softw Eng 2(4):293–300

Senthil R, Kushwaha DS, Mishra AK (2007) An improved component model for component based software engineering. ACM SIGSOFT Softw Eng Notes 32(4). doi:10.1145/1281421.1281431

Shepperd M (1998) A critique of cyclomatic complexity as software metric. Softw Eng J 3(2):30–36

Tiwari U, Kumar S (2014) Cyclomatic complexity for component based software. ACM SIGSOFT Softw Eng Notes 39(1):1–6

Tsai WT, Saimi A, Yu L, Paul R (2003) Scenario-based object-oriented testing framework. In: Proceedings of the third international conference on quality software (QSIC'03), IEEE, 2003, pp 410–417

Tyagi K, Sharma A (2014) An adaptive neuro fuzzy model for estimating the reliability of component-based software systems. Appl Comput Inform Elsevier 10:38–51

Voas JM (1992) A dynamic testing complexity metric. Software Qual J 1(2):101–114

Voas JM, Miller KW (1992) The revealing power of a test case. J Softw Test Verif Reliab 2(1):25–42

Voas JM, Miller KW (1995) Software testability: the new verification. IEEE Softw 12(3):17–28

Weyukar EJ (1993) More experience with data flow testing. IEEE Trans Softw Eng 19(9):912–919

Weyuker EJ (1998) Testing component-based software: a cautionary tale. IEEE Softw 15(5):54–59