**Introduction**

Software testing is an activity to check whether the actual results match the expected results and to ensure that the software system is defect free. It involves execution of a software component or system component to evaluate one or more properties of interest.

Software testing also helps to identify errors, gaps or missing requirements in contrary to the actual requirements. It can be either done manually or using automated tools. Some prefer saying Software testing as a white box and black box testing.

An early start to testing reduces the cost and time to rework and produce error-free software that is delivered to the client. However in Software Development Life Cycle, testing can be started from the Requirements Gathering phase and continued till the deployment of the software. It also depends on the development model that is being used. For example, in the Waterfall model, formal testing is conducted in the testing phase; but in the incremental model, testing is performed at the end of every increment/iteration and the whole application is tested at the end.

**Flow graph:**

Flow Graph is defined as a function in a program that can be represented as a control flow graph and the nodes in the flow graph are defined as program statements while the directed edges are the flow of control. A Flow Graph consists of nodes and edges. The two nodes in the Flow Graph can be either unconnected or connected by an edge in either direction.

While tracing a path from a source to a sink a back edge is an edge that leads back to a node that has already been visited. The Flow Graph contains one source node and one sink.   
A source node is the node that has no incoming edges while a sink node is the node with no outgoing edges. A program's function may contain more than one sink node, but this graph can be converted into a graph with only one sink. There are some languages that allow more than one source. This construct is very rare and not used in Structured Programming.

**Black Box Testing**

Black-box testing is a method of software testing that examines the functionality of an application based on the specifications. It is also known as Specifications based testing. Independent Testing Team usually performs this type of testing during the software testing life cycle.

This method of test can be applied to each and every level of software testing such as unit, integration, system and acceptance testing.

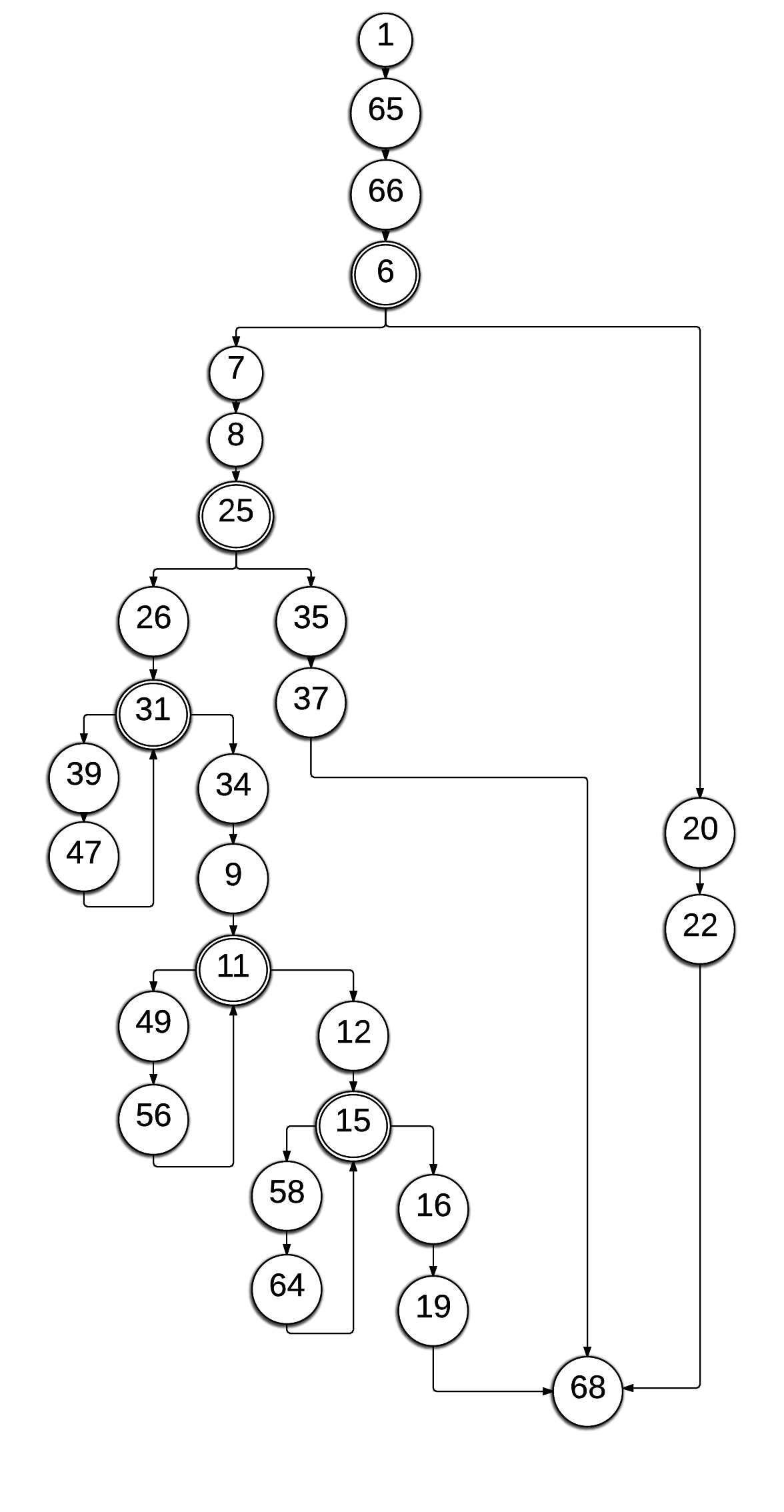
This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see. This method attempts to find errors in the following categories:

* Incorrect or missing functions
* Interface errors
* Errors in data structures or external database access
* Behavior or performance errors
* Initialization and termination errors

**Source Code**

1. **public class** AES {
2. **byte**[] **skey** = **new byte**[1000];
3. String **skeyString**;
4. **static byte**[] *raw*;
5. String **inputMessage**,**encryptedData**,**decryptedMessage**;
6. **public** AES() {
7. **try** {
8. generateSymmetricKey();
9. **inputMessage**=JOptionPane.showInputDialog(**null**,**"Enter message to  
   encrypt"**);
10. **byte**[] ibyte = **inputMessage**.getBytes();
11. **byte**[] ebyte=*encrypt*(*raw*, ibyte);
12. String encryptedData = **new** String(ebyte);
13. System.***out***.println(**"Encrypted message "**+encryptedData);
14. JOptionPane.showMessageDialog(**null**,**"Encrypted  
     Data"**+**"\n"**+encryptedData);
15. **byte**[] dbyte= *decrypt*(*raw*,ebyte);
16. String decryptedMessage = **new** String(dbyte
17. System.***out***.println(**"Decrypted message** +decryptedMessage
18. JOptionPane.showMessageDialog(**null**,**"Decrypted Data  
     "**+**"\n"**+decryptedMessage);
19. }
20. **catch**(Exception e) {
21. System.***out***.println(e);
22. }
23. }
24. *//Symmetric key generation*
25. **void** generateSymmetricKey() {
26. **try** {
27. Random r = **new** Random();
28. **int** num = r.nextInt(10000);
29. String knum = String.*valueOf*(num);
30. **byte**[] knumb = knum.getBytes();
31. **skey**=*getRawKey*(knumb);
32. **skeyString** = **new** String(**skey**);
33. System.***out***.println(**"AES Symmetric key = "**+**skeyString**);
34. }
35. **catch**(Exception e) {
36. System.***out***.println(e);
37. }
38. }
39. **private static byte**[] getRawKey(**byte**[] seed) **throws** Exception {
40. KeyGenerator kgen = KeyGenerator.*getInstance*(**"AES"**);
41. SecureRandom sr = SecureRandom.*getInstance*(**"SHA1PRNG"**);
42. sr.setSeed(seed);
43. kgen.init(128, sr);
44. SecretKey skey = kgen.generateKey();
45. *raw* = skey.getEncoded();
46. **return** *raw*;
47. }
48. *//encrypt portion*
49. **private static byte**[] encrypt(**byte**[] raw, **byte**[] clear) **throws** Exception
50. {
51. SecretKeySpec skeySpec = **new** SecretKeySpec(raw, **"AES"**);
52. Cipher cipher = Cipher.*getInstance*(**"AES"**);
53. cipher.init(Cipher.***ENCRYPT\_MODE***, skeySpec);
54. **byte**[] encrypted = cipher.doFinal(clear);
55. **return** encrypted;
56. }
57. *//decrypt portion*
58. **private static byte**[] decrypt(**byte**[] raw, **byte**[] encrypted) **throws** Exception {
59. SecretKeySpec skeySpec = **new** SecretKeySpec(raw, **"AES"**);
60. Cipher cipher = Cipher.*getInstance*(**"AES"**);
61. cipher.init(Cipher.***DECRYPT\_MODE***, skeySpec);
62. **byte**[] decrypted = cipher.doFinal(encrypted);
63. **return** decrypted;
64. }
65. **public static void** main(String args[]) {
66. AES aes = **new** AES();
67. }
68. }

**Flow graph**



**Calculating cyclometic complexity of the registration module:**

No. of Nodes=27

No. of Edges=31

Cyclometic complexity, V(G)=E-N+2 = 31-27+2=4+2=6

**Basis set of independent paths:**

Path 1: 1-65-66-6-7-8-25-26-31-34-9-11-12-15-16-19-68

Path 2: 1-65-66-6-7-8-25-26-31-39-47-31-34-9-11-12-15-16-19-68

Path 3: 1-65-66-6-7-8-25-26-31-34-9-11-49-56-11-12-15-16-19-68

Path 4: 1-65-66-6-7-8-25-26-31-34-9-11-12-15-58-64-16-19-68

Path 5: 1-65-66-6-7-8-25-35-37-68

Path 6: 1-65-66-6-20-22-68

**Conclusion:**

Software testing is an art. Most of the testing methods and practices are not very different from years ago. It is nowhere near maturity, although there are many tools and techniques available to use. Good testing also requires a tester's creativity, experience and intuition, together with proper techniques.

Testing is more than just debugging. Testing is not only used to locate defects and correct them. It is also used in validation, verification process, and reliability measurement.

Testing is expensive. Automation is a good way to cut down cost and time. Testing efficiency and effectiveness is the criteria for coverage-based testing techniques.