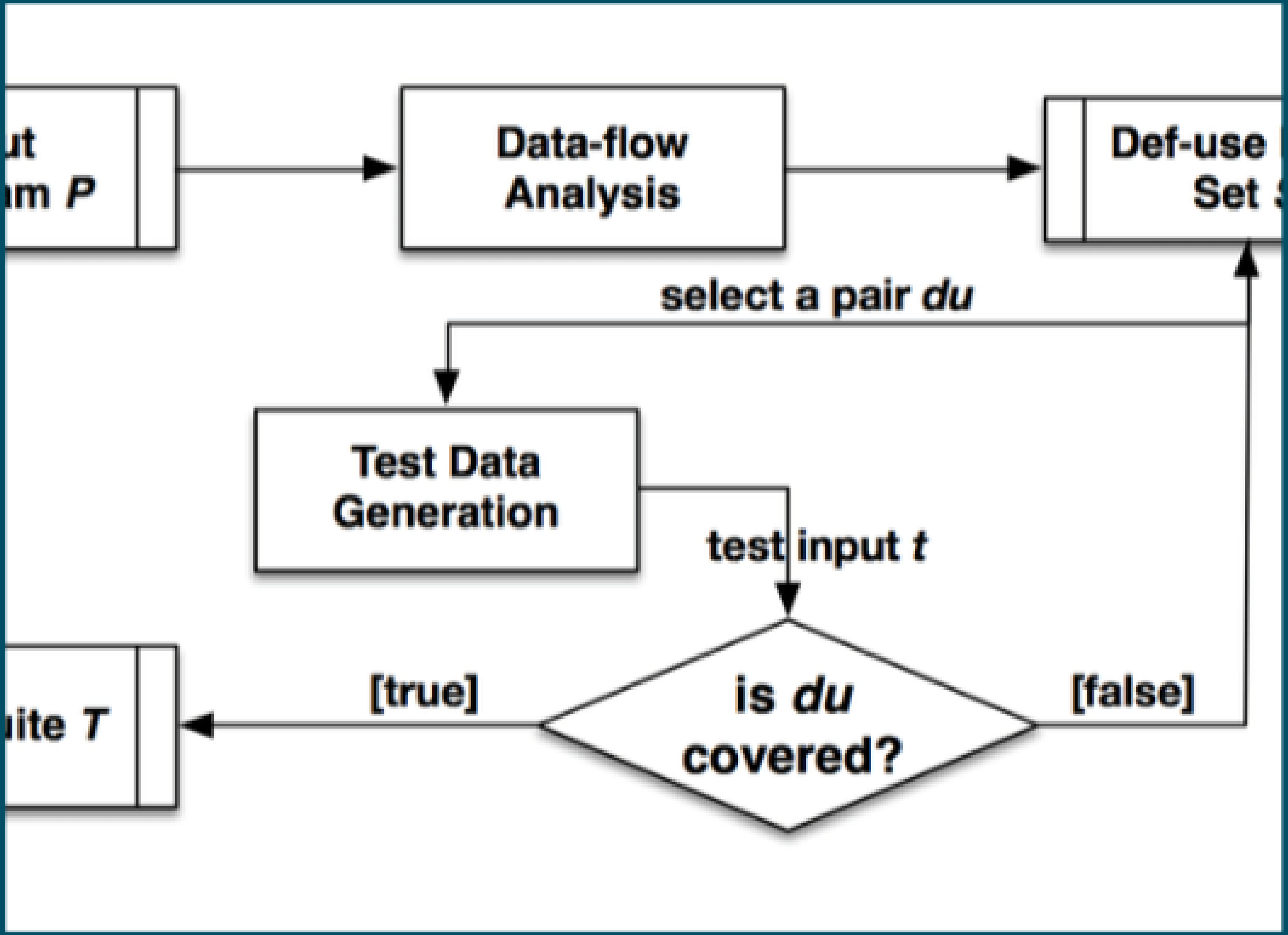
CMR Technical Campus

Software Testing Methodologies

Presented by :-
Shaik Sameer Hussain
CSE-D
Roll No: 227R1A05Q7

Definition and Significance

Data Flow Testing refers to a set of techniques that focus on the control flow of a program to examine the status and changes of data objects. Its significance lies in the early detection of semantic errors and anomalies relating to data management, crucial for software reliability and performance.



Understanding Dataflow Testing

Control Flow Graphs

Control flow graphs depict the sequence of statements in a program. They consist of nodes (statements) and links (control flow). Dataflow testing examines data flow within these graphs, identifying how data values change and interact.

Data Flow Operations

Key operations on data objects include:

d (define) - create or initialize a value

k (kill) - overwrite a value

u (use) - access a value

c (calculate) - transform a value

p (predicate) - evaluate a value in a conditional

Control Flow Graphs and Data Flow Analysis

Control Flow Graphs (CFG) provide a visual representation of the flow of control in a program, with nodes depicting statements and edges representing control flow paths. These graphs are essential in Data Flow Analysis, helping to trace data object lifecycle from definition to use, thereby identifying crucial points for testing.

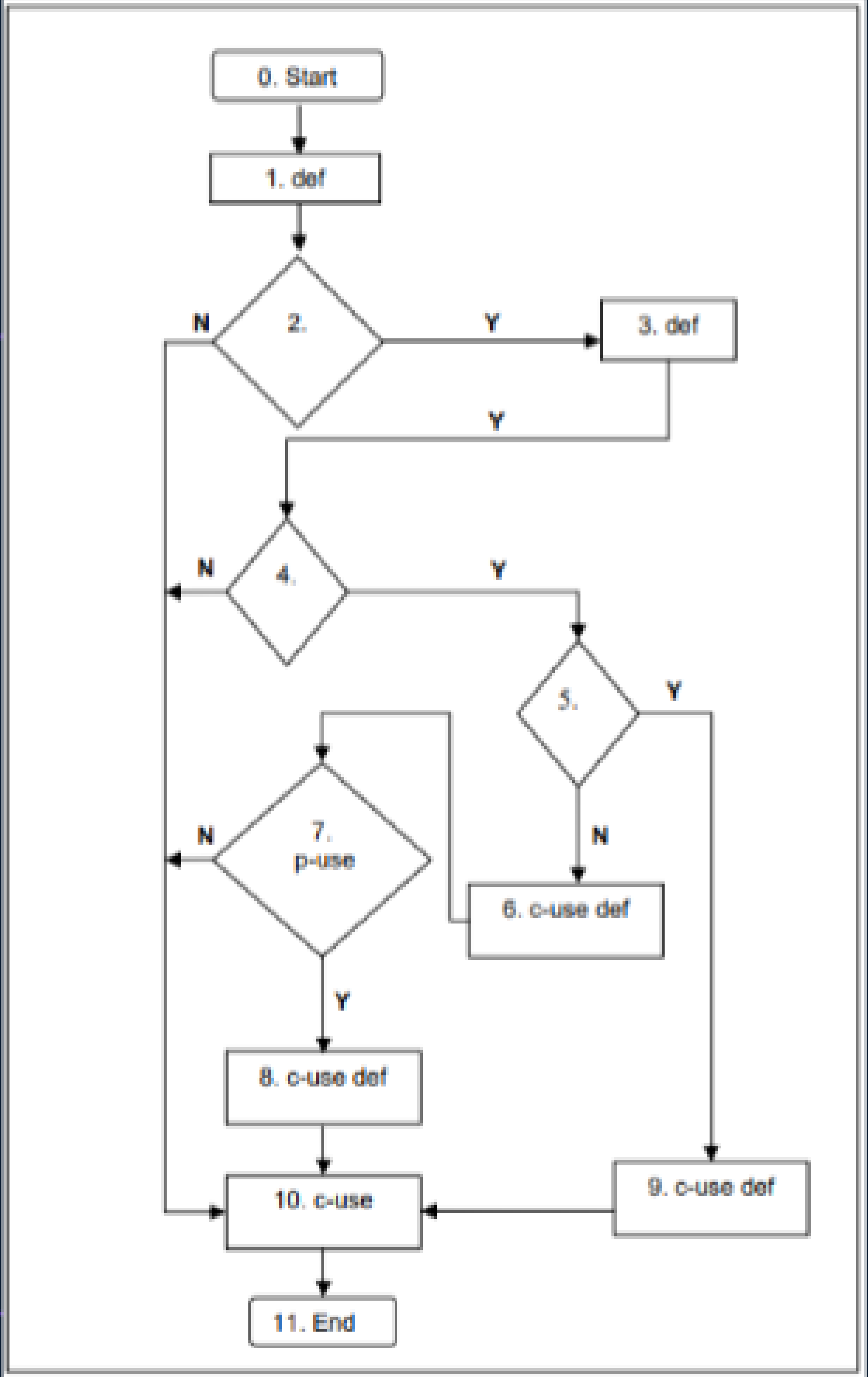
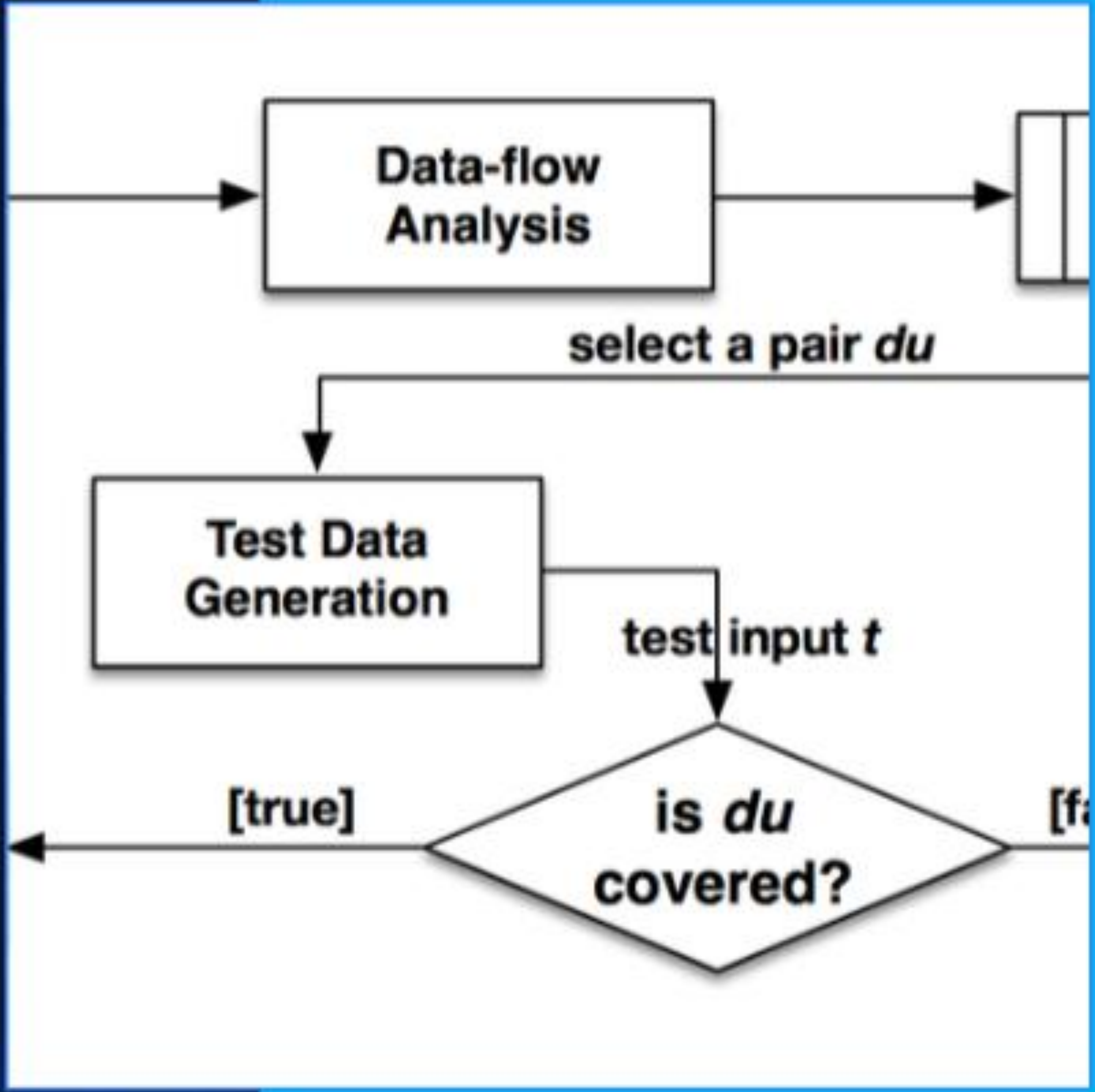


Figure 6.2: Annotated Control Flow Diagram for Variable 'Bill'

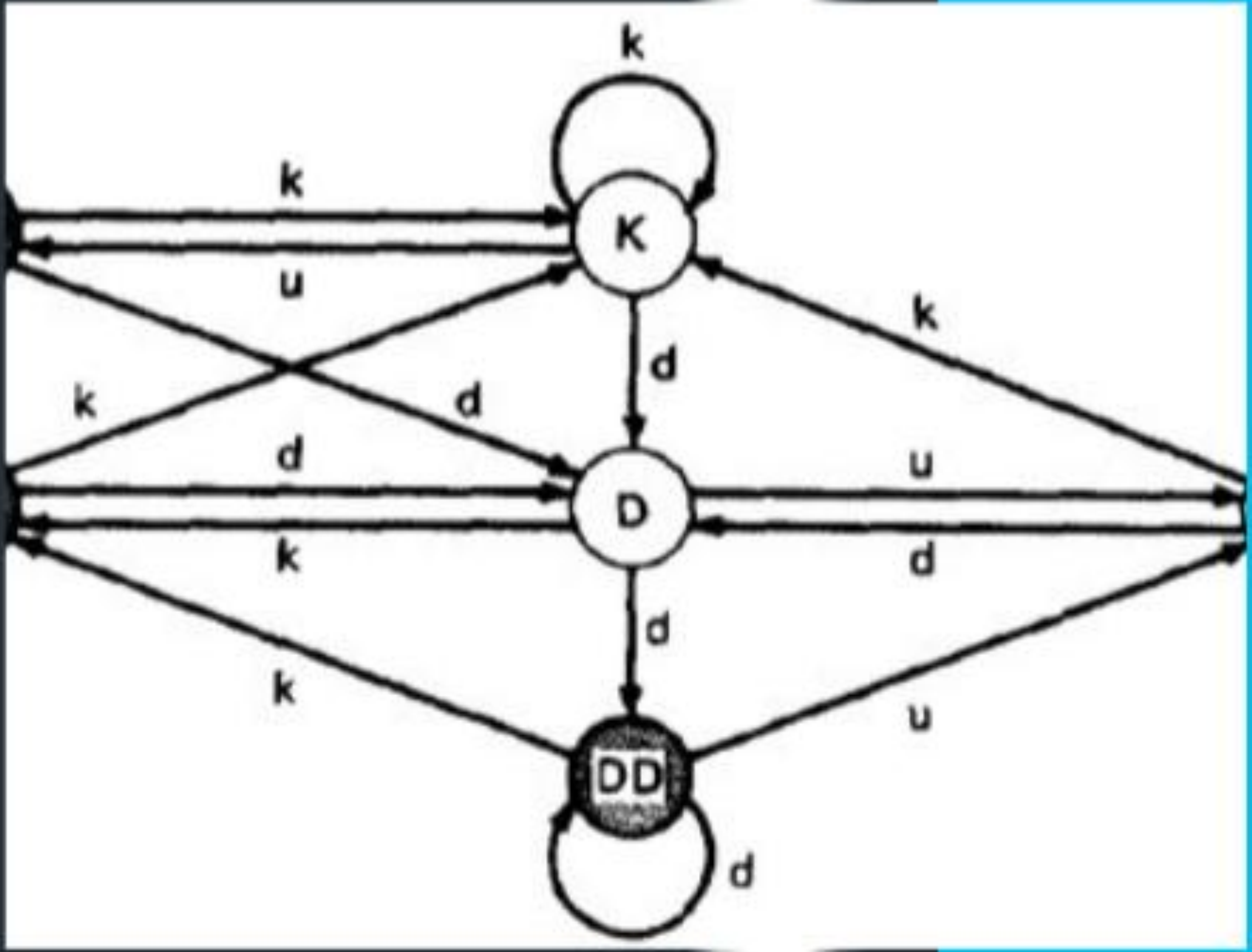
Data Flow Graphs



The data flow graph is a graph consisting of nodes and directed links i.e. links with arrows on them. We will be doing data flow testing but would not be using data flow graphs as such. We will use an ordinary annotated control flow graph.
Data objects can be created, killed and/or used. They can be used in two different ways in a calculation or as part of a control flow predicate.

d – Defined, created, initialized, etc.
k – killed, undefined, released.
u – used for something.
c – used in a calculation.
p – used in a predicate.

Data Flow Anomalies



An anomaly is the one denoted by a two-character sequence of action. The various possible two-letter combinations possible for d, k and u are as listed under. Some of these are suspicious, some are bugs and some are okay

dd (Define-Define): Harmless but suspicious; redefining a variable without using it in between raises questions about its necessity.

dk (Define-Kill): Likely a bug; defining a variable and immediately killing it wastes resources or indicates redundant code.

Data Flow Anomalies

1 **dd**

Data is defined twice before being used.

2 **dk**

Data is defined but then overwritten before being used.

3 **du**

Data is defined and used without being calculated or overwritten.

4 **uu**

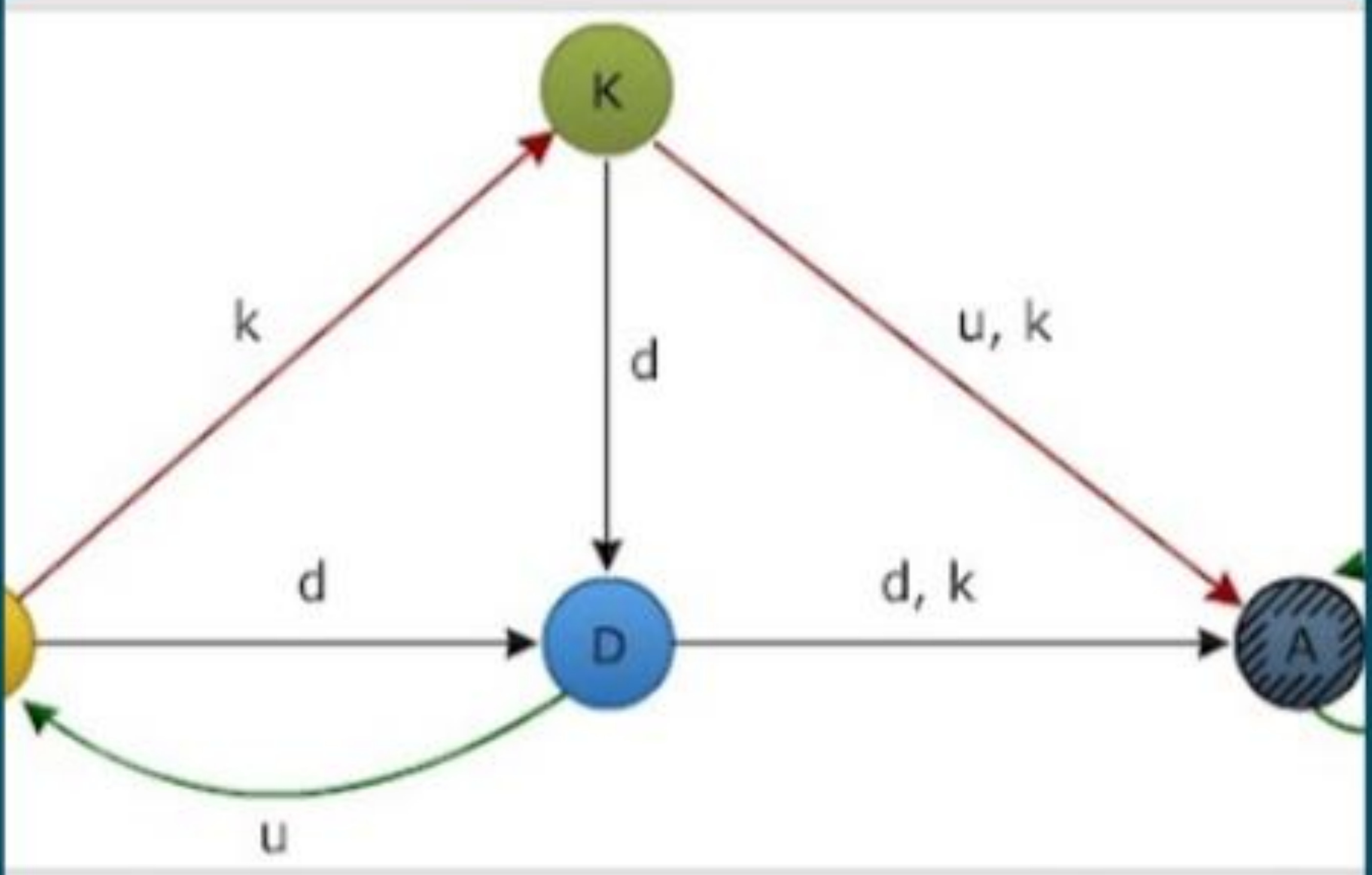
Data is used before being defined.

Types of Anomalies:

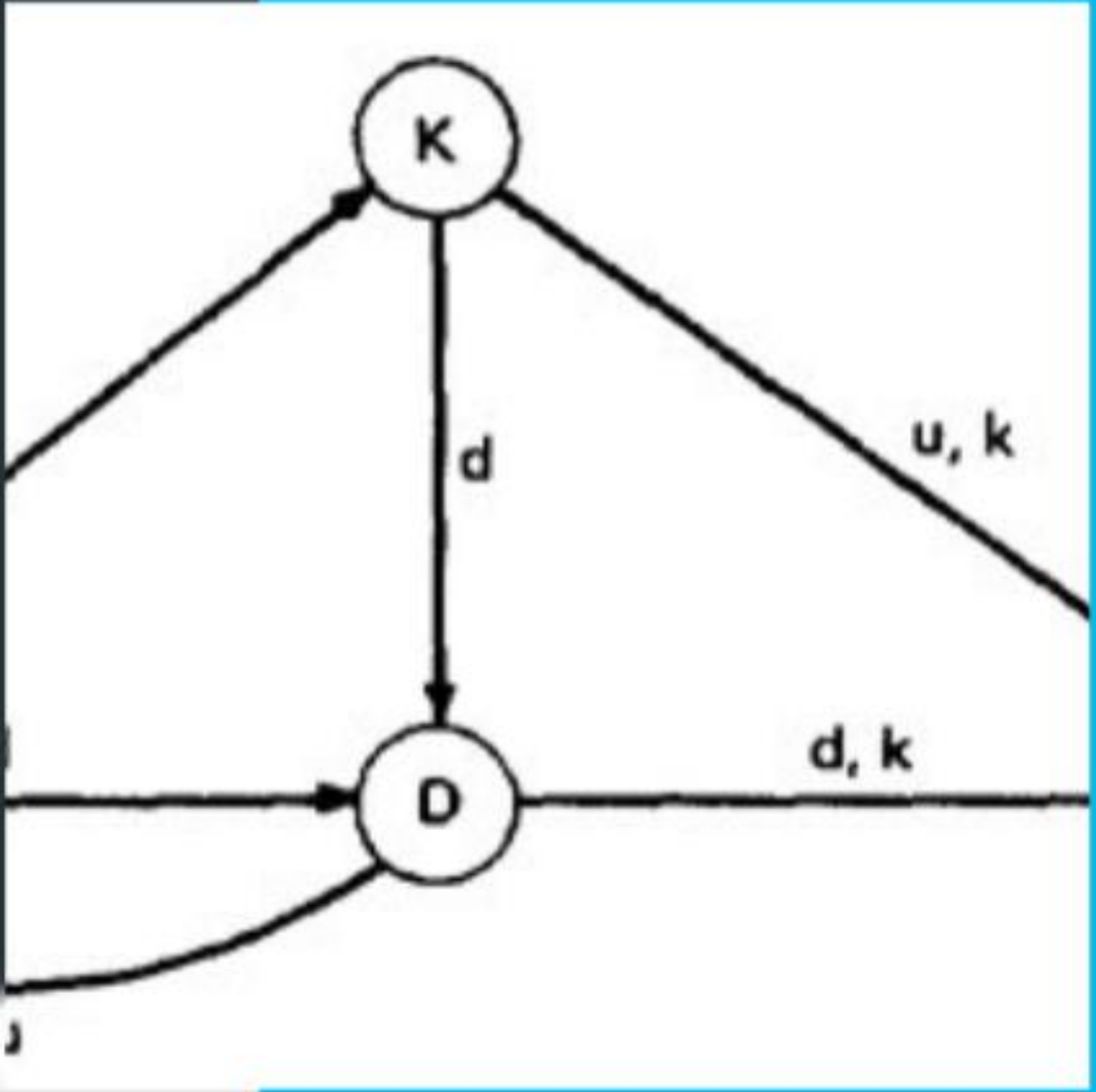
du (Define-Use): Normal and expected; a variable is used after being defined.

kd (Kill-Define): Normal; a variable is killed and then redefined, often seen in reassignment.

kk (Kill-Kill): Harmless but possibly buggy; re-killing an already killed variable may indicate redundant operations.



Unforgiving Data Flow Anomaly State Graph



Types of Anomalies

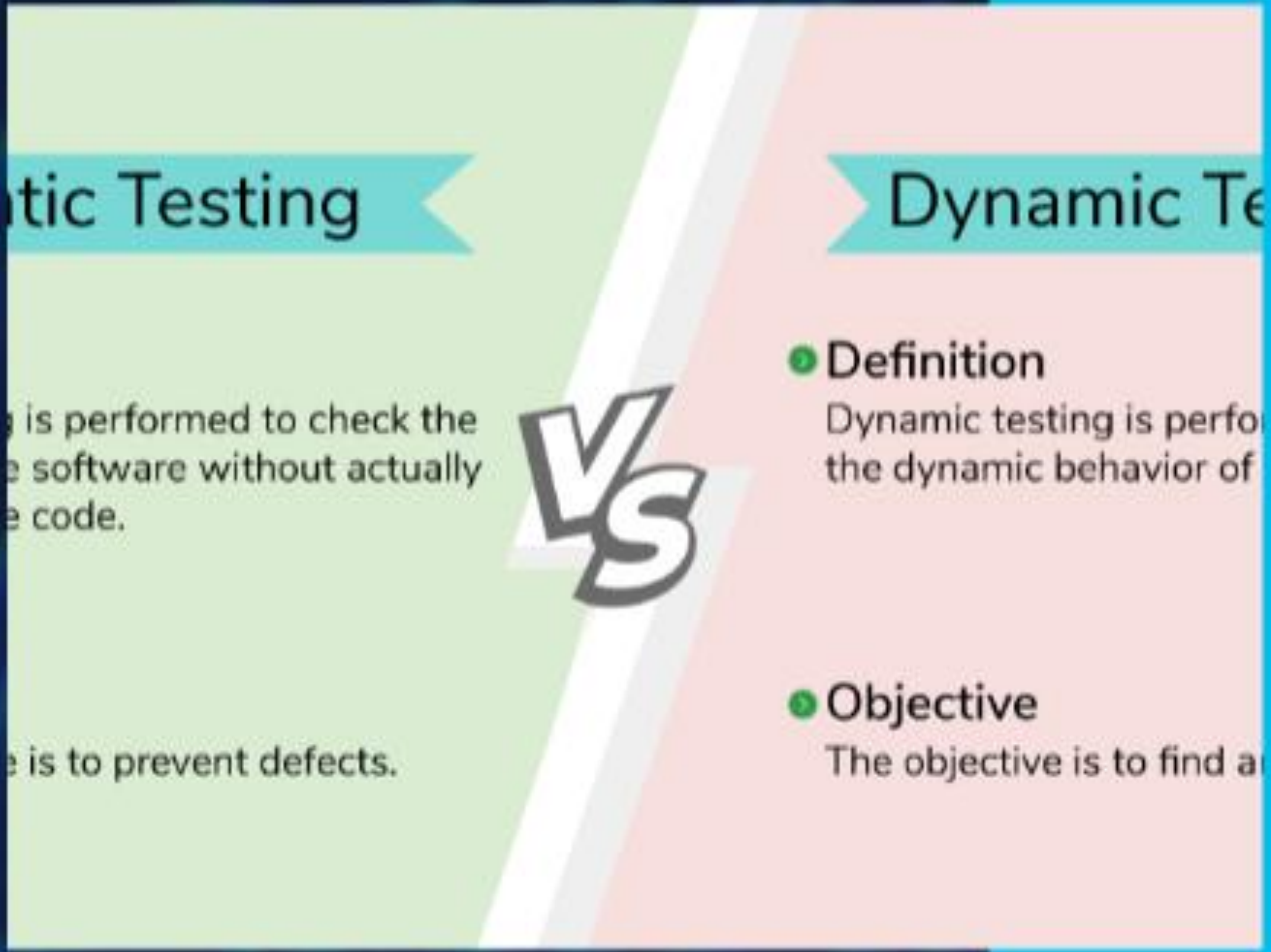
ku (Kill-Use): A bug; using a variable after it is killed leads to undefined behavior.

ud (Use-Define): Usually not a bug; reassignment to variables is common in most programming languages.

uk (Use-Kill): Normal; using a variable and then killing it is expected behavior.

uu (Use-Use): Normal; repeatedly using a variable without redefining or killing it is common.

Static versus Dynamic Anomaly Detection



tic Testing
Dynamic Te
VS

is performed to check the
software without actually
code.

is to prevent defects.

Definition
Dynamic testing is perfo
the dynamic behavior of

Objective
The objective is to find a

Static Analysis: Checks the source code for errors without executing it (e.g., detecting syntax errors).

Dynamic Analysis: Identifies issues during program execution (e.g., detecting division by zero).

If a data flow anomaly can be caught through static analysis, it becomes the job of the language processor, not testing.

Applications of Dataflow Testing

Bug Reduction

Dataflow testing can identify and eliminate data-related errors early in the development cycle, reducing the likelihood of bugs reaching production.

Enhanced Software Quality

By ensuring data integrity and correctness, dataflow testing contributes to building more robust and reliable software.

Key Takeaways & Conclusion

1

Data Integrity

Dataflow testing plays a crucial role in ensuring data integrity and correctness.

2

Bug Detection

It's a powerful technique for detecting data-related anomalies and preventing bugs.

3

Structured Testing

Dataflow testing is an essential component of structured software testing methodologies.