Software Testing
&
Software Maintenance

UNIT IV

 U4. 1

Learning Objectives

Software Testing:

- Testing process
- Design of test cases
- functional testing
- Structural testing
- Unit Testing
- Integration and System Testing
- Debugging
- Alpha & Beta Testing

 2 U4

Learning Objectives

Software Maintenance:

- Management of Maintenance
- Maintenance Process
- Maintenance Models
- Regression Testing
- Reverse Engineering
- Software Re-engineering
- Configuration Management
- Documentation.

 3 U4

Software Testing

process of executing a program with the intent of finding errors

 U4

Testing

- **Cat & Saint**
  - Reason to test
- **Audiologist**
  - Defective Test is more dangerous than defective product
- **Pest Control**
  - Revise Continuously
- **Policeman & Bridge**
  - Prevention is better than cure

 5 U4

Testing

Chinese Doctors

- Expert in surgery known in world
- Prescribe medicine to cure in early days known in city
- Give suggestion known in home
- Defect prevention & defect detection supplement each other

Pride in "test" will take care of "rest"

 6 U4

Software Testing

- Consumes at least half of the labor
- Process of testing software product
- Contribute to
  - Delivery of higher quality product
  - More satisfied users
  - Lower maintenance cost
  - More accurate and reliable results

 7 04

Error

- People make errors.
- Typographical error
- Misreading of a specification
- Misunderstanding of functionality of a module
- A good Synonym mistakes while coding we call these mistakes "bugs"
- is Mistake.
- When people make

 8 04

Defect

- Representation of an error
- DFD
- Hierarchy chart
- Source Code
- An error may lead to one or more faults

- Fault of Omissions
- If certain specifications have not been programmed

- Fault of Commission
- If certain program behavior have not been specified

 9 04

Failure, Test, Test Case

- Failure
- A particular fault may cause different failures, depending on how it has been exercised
- Test:
- A test is the act of exercising S/W with test cases

- Test Case:
- A test case has an identity and is associated with a program behaviour
- It has a set of inputs and a list of expected outputs

 10 U4

Test Case Template

Test Case ID:	
Section –I (Before Execution)	**Section –II (After Execution)**
Purpose:	Execution History:
Pre Condition: (If Any)	Result:
Inputs:	If Fails, any possible reason(Optional):
Expected Outputs:	Any Other observation:
Post Conditions:	Any Suggestions:
Written by:	Run By:
Date:	Date:

 U4

S/W Testing

- Testing is the process of demonstrating that errors are not present
- The purpose of testing is to show that a program performs its intended functions correctly
- Testing is the process of establishing confidence that a program does what it is supposed to do
- **Testing is the process of executing a program with the intent of finding errors**

 12 U4

Limitation of Testing

- Can we test the program completely?
  - Some text
  - Many managers
  - Test coverage analyzer
  - Salesperson
  - Some tester
- No matter how hard we try, how cleverly we plan, how much time we spend, and how many staff and computer we use, we still cannot do enough testing. We still miss bugs

 13 U4

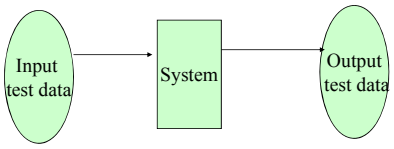
Reasons

- The domain of possible inputs is too large to test
- There are too many possible paths through the program to test
- The user interface issues are too complex to completely test

 14 U4

Functional Testing

- Testing is based on the functionality of the program
- Internal structure of the code is ignored (black box testing)



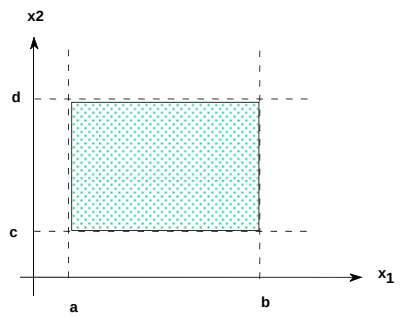
 15 U4

Boundary Value Analysis

- Test cases explore boundary condition have higher chances of detecting error
- Directly on, just above and just below the boundaries of input

 U4

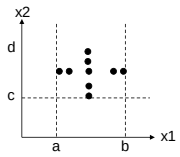
Simple Example (to be generalized)



 U4

Boundary Value Analysis cont..

- Consider function F with input variables x& y
  - a<= x<= b
  - c<=y<=d
- x & y bounded by two intervals [a,b] & [c,d]
- Basic idea is to use input variables
  - Minimum
  - Just above min
  - Just below max
  - max



 U4

Boundary Value Analysis cont..

Test cases

- <Xnom, Ymin>, <10,25>
- <Xnom, Ymin+>, <10,26>
- <Xnom, Ynom>, <10,30>
- <Xnom, Ymax>, <10,35>
- <Xnom, Ymax->, <10,34>
- <Xmin, Ynom>, <5,30>
- <Xmin+, Ynom>, <6,30>
- <Xnom, Ynom>, <10,30>
- <Xmax, Ynom>, <15,30>
- <Xmax+, Ynom> <14,30>

4n+1 test cases

Inputs
5<=x<=15
25<=y<=35

 19 U4

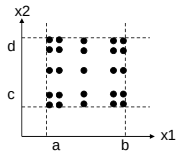
Robustness Testing

- Extension of boundary value analysis
- We see, what happens when values exceeds max and slightly less min
- **6n+1 test cases**

 20 U4

Worst Case Analysis

- We see what happens when more than one variable has an extreme values
- 5 power n test cases (min, min+, nom, max, max-)



 21 U4

Worst Case Analysis Example

<10,25>	<15,25>	<5,25>
<10,26>	<15,26>	<5,26>
<10,30>	**<15,30>**	**<5,30>**
<10,35>	<15,35>	<5,35>
<10,34>	<15,34>	<5,34>

<6,25>	<14,25>
<6,26>	<14,26>
<6,30>	**<14,30>**
<6,35>	<14,35>
<6,34>	<14,34>

Inputs
5<=x<=15
25<=y<=35

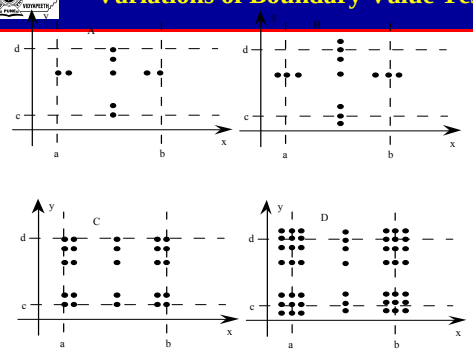
 U4

Robust Worst Case

- We see what happens when more than one variable has an extreme values
- (min-,min, min+, nom, max, max-, max+)

 23 U4

Variations of Boundary Value Testing



 24 U4

Limitations

- Variables to be truly independent
- Doesn't make sense for Boolean variables

 25 U4

Assignment

- Consider a simple program to classify a triangle. Its input is a triple of positive integer (x,y,z) and the data type for input parameters ensures that these will be integer greater than zero and less than equal to 200. The program output may be one of the following words:
- [Scalene; Isosceles; Equilateral; Not a triangle]
- Design the boundary value test cases and worst test case]

 26 U4

Assignment

- Previous date is a function of three variables : month, day and year. It returns the date of the day after the input date. The month, date and year variables have integer values subject to these conditions

C1 : 1 <= month <= 12

C2 :1 <= day <= 31

C3 : 1812 <= year <= 2012

Design the boundary value test cases and worst test case]

 27 U4

Equivalent Classes

- Important aspect
- form a partition of a set, where partition refers to a collection of mutually disjoint subsets where the union is the entire set
- If the same result is expected from two test cases, consider them equivalent

Equivalent Classes cont..

- A group of test cases forms an equivalent class if
  - They all test the same thing
  - If one test catches a bug, the other probably will too
  - If one test doesn't catch a bug, the others probably would not either
- Implication of testing
  - Completeness
  - Non redundancy

Equivalent Classes cont..

- Consider the triangle problem
- Output domain equivalence class test cases can be identified as follows
  - O1 = {<x,y,z>: Equilateral triangle with sides x,y,z}
  - O2 = {<x,y,z>: Isosceles triangle with sides x,y,z}
  - O3 = {<x,y,z>: Scalene triangle with sides x,y,z}
  - O4 = {<x,y,z>: Not a triangle with sides x,y,z}

Equivalent Classes cont..

Based on input domain

- I1 = {x,y,z : x=y=z}
- I2 = {x,y,z : x=y, x!=z}
- I3 = {x,y,z : x=z, x!=y}
- I4 = {x,y,z : y=z, x!=y}
- I5 = {x,y,z : x!=y!=z}
- I6 = {x,y,z : x>=y+z}
- I7 = {x,y,z : y>=x+z}
- I8 = {x,y,z : z>=x+y}
- I6' ={x,y,z : x=y+z}
- I6'' ={x,y,z : x>y+z}

Equivalent Classes cont..

Test Case	X	Y	Z	Expected Output
1	10	10	10	Equilateral
2	10	10	15	Isosceles
3	10	15	10	Isosceles
4	15	10	10	Isosceles
5	15	10	7	Scalene
6	25	10	10	Not a triangle
7	25	15	10	Not a triangle
8	10	25	10	Not a triangle
9	15	25	10	Not a triangle
10	10	10	25	Not a triangle
11	10	15	25	Not a triangle

Decision Table Based Testing

- Useful for describing situation in which a number of combinations of actions are taken under varying set of conditions
- four portions
  - Condition stub
  - Action stub
  - Condition entries
  - Action entries

Decision Table Based Testing

C1: x<=y+z?	F	T	T	T	T	T	T	T	T	T	T
C2: y<=x+z?	-	F	T	T	T	T	T	T	T	T	T
C3: z<=x+y?	-	-	F	T	T	T	T	T	T	T	T
C4 : x= y?	-	-	-	T	T	T	T	F	F	F	F
C5: x=z?	-	-	-	T	T	F	F	T	T	F	F
C6 : y = z?				T	F	T	F	T	F	T	F
A1: Not a triangle	*	*	*								
A2 : Scalene											*
A3: Isosceles							*		*	*	
A4 : Equilateral				*							
A5: Impossible					*	*		*			

 U4

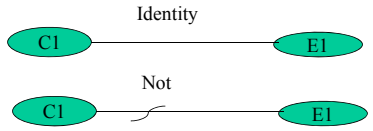
Cause Effect Graphing Technique

- addresses limitations of decision table where all inputs are treated separately although real world problem demand another approach.
- Boundary-value analysis & equivalence class partition also assume the independence of input
- a systematic method of generating test cases representing combination of conditions
- Cause-effect graphs capture relationships between specific combination of inputs (causes) and outputs (effects).
- Causes and Effects represented as nodes of a cause-effect graph – includes intermediate nodes to link cause & effects in forming logical expressions

 33 U4

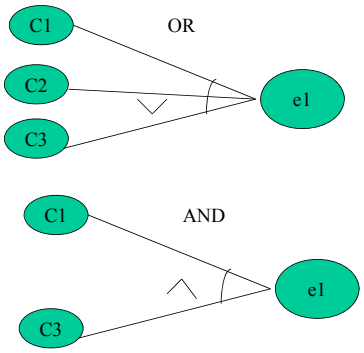
Cause Effect Graphing Technique

- a systematic method of generating test cases representing combination of conditions



 34 U4

Cause Effect Graphing Technique..



Cause Effect Graphing Technique cont..

The characters in column 1 must be an A or B. The character in column 2 must be a digit. In this situation, the file update is mad. If the the character in column 1 is incorrect, message x is issued. If the character in column 2 is not a digit, message y is issued

Cause Effect Graphing Technique cont..

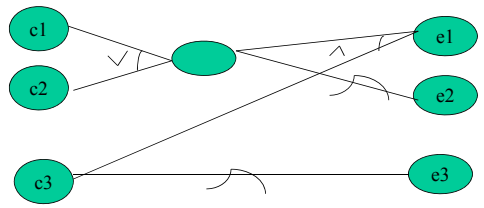
The causes are

- C1 : character in column1 is A
- C2 : character in column1 is B
- C3 :character in column 2 is digit

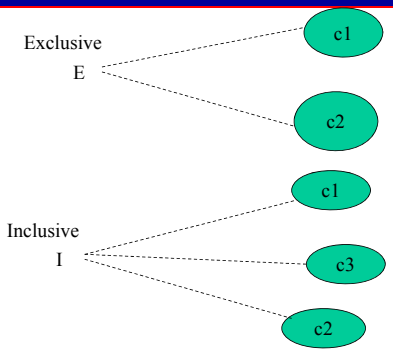
Effects are

- Update made
- Message x is issued
- Message y is issued

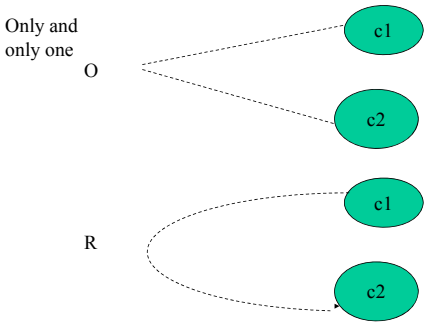
Cause Effect Graphing Technique cont..



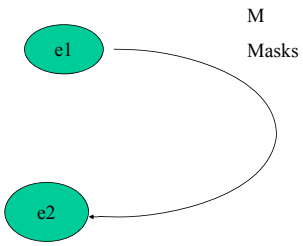
Cause Effect Graphing Technique cont..



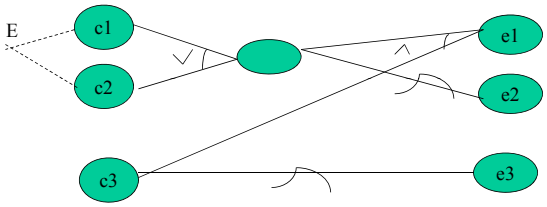
Cause Effect Graphing Technique cont..



Cause Effect Graphing Technique cont..



Cause Effect Graphing Technique cont..



Cause-Effect Testing Steps

Six steps

- Divide specification into ``workable " pieces
- Identify causes and effects
- Create Boolean graph that links causes to effects
- Annotate graph with constraints
- Create a limited-entry decision table from the graph
- Convert the columns in the decision table into testcases

Special value Testing

- Most intuitive & least uniform
- Tester uses domain knowledge experience with similar programs and information about soft spots to devise test cases
- Also called adhoc testing
- No guidelines are used other than to use "best engineering judgment"
- Depends on the ability of the tester

Combined Strategy

- Use all of the methods to generate a test suite with maximum coverage for the minimum cost
  - Start with cause-effect analysis
  - Always use boundary cases
  - Check equivalence classes to ensure they are all covered
  - Use error guessing for supplement test cases
- A good test suite can be augmented and re-run throughout the life of a project

Structural Testing

White box Testing

Possess complete knowledge about the internal structure of the source code

Structural Testing Vs. Functional Testing

- Compares test program behavior against the apparent intention of the source code.
  - Whereas functional testing compares test program behavior against a requirements specification.
- Examines how the program works, taking into account possible pitfalls in the structure and logic.
  - Functional testing examines what the program accomplishes, without regard to how it works internally.
- cannot find errors of omission
  - cannot find errors of commission
- doesn't ensure that you've met user expectations

 49 U4

Path Testing

- Based on selecting a set of test paths through the program
- Requires the complete knowledge of the program's structure and used by developers to unit test their own code
- It involves
  - Generating a set of paths that will cover every branch in the program
  - Finding a set of test cases that will execute every path in this set of program paths

 50 U4

Example

```
#!python
  def func(x):
  if x==0:
  return 3
  elif x==1:
  return 4
  elif x==2:
  return 5
  else:
  return 0
```

 51 U4

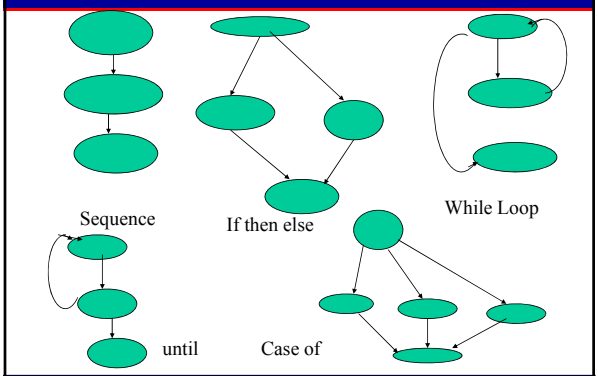
Test Case Template

Test Case ID:	
Section –I (Before Execution)	**Section –II (After Execution)**
Purpose:	Execution History:
Pre Condition: (If Any)	Result:
Inputs:	If Fails, any possible reason(Optional):
Expected Outputs:	Any Other observation:
Post Conditions:	Any Suggestions:
Written by:	Run By:
Date:	Date:

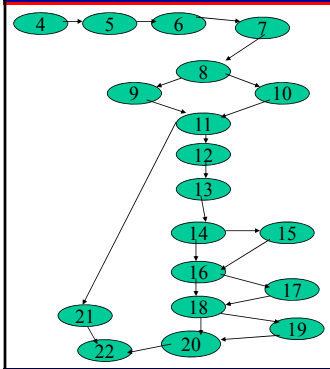
Path Testing cont..

- Program control flow is analyzed on a graphical representation, called a **Program graph**
- Program graph is a directed graph in which nodes are program statements and edges represent flow of control
  - If i and j are nodes in the program graph, there is an edge from node i to node j if the statement (fragment) corresponding to node j can be executed immediately after the statement (fragment) corresponding to node i

Path Testing cont..



Path Testing cont..



1. program triangle (input,output)
2. var: a, b, c :integer;
3. isatriangle: boolean;
4. begin
5. writeln("-----");
6. readln(a,b,c);
7. writeln("------");
8. if(a<b+c) and (b<a+c)and (c<a+b)
9. than isatriangle := true
10. else isatriangle := true;
11. If isatriangle ….

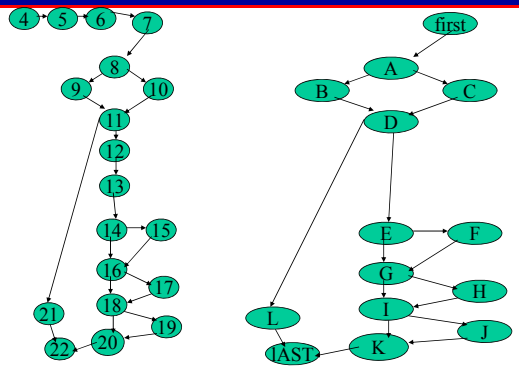
 U4

DD Path

1. Decision to Decision path
2. Sequence of statements that begins with the outway of a decision statement and ends with the inway of the next decision statement

 U4

Path Testing cont..



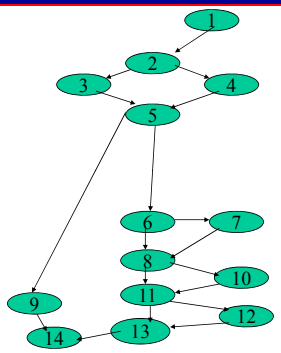
 U4

Cyclomatic Complexity

1. To find the number of independent paths through a program
2. Independent path is defined in terms of independent paths

 59 U4

Path Testing cont..



1. 1,2,3,5,9,14
2. 1,2,4,5,9,14
3. 1,2,4,5,6,8,11,13,14
4. 1,2,4,5,6,7,8,11,13,14
5. 1,2,4,5,6,8,10,11,13,14
6. 1,2,4,5,6,8,11,12,13,14

 U4

Cyclomatic Complexity

- McCabe's cyclomatic metric of a graph G with n vertices, e edges and connected component is

  V(G) = e –n+2P

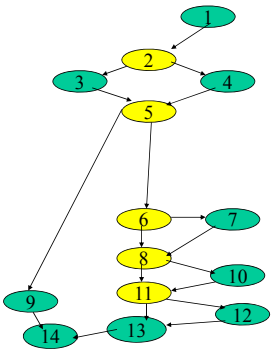
  V(G) = pi+1

pi is the number of predicate nodes contained in the flow graph

- Cyclomatic complexity is equal to the number of regions of the flow graph

 60 U4

Path Testing cont..



V (G) = 18-14+(2*1) =6
pi = 5
V(G) = 5+1 =6

Total regions are 6

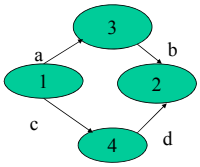
 U4

Graph Matrices

- A graph matrix is a tabular representation of flow graph
- Can be used to find cyclomatic complexity
- Each row having more than one entry shows the predicate node
- Empty column shows the beginning node
- Empty row tells the end node

 U4

Path Testing cont..



	1	2	3	4
1			a	c
2				
3		b		
4		d		

	1	2	3	4
1			1	1
2				
3		1		
4		1		

2-1 =1

1-1 =0
1-1 =0
1+1 = 2

 U4

Path Testing cont..

	1	2	3	4
1			a	c
2				
3		b		
4		d		

- Squaring a matrix a new matrix expresses a relation between the pair
- There are two paths ab and cd from node 1 to node 2

	1	2	3	4
1		ab+cd		
2				
3				
4				

 U4

Data Flow Testing

- Focus on the points at which variables receives values and the point at which these values are used
- Variables can be created, killed used
- d: defined, created, initialized, etc
- k: undefined released
- u: used for something
- c: used in a calculation
- p:used in a predicate

 66 U4

Data Flow Testing cont..

- Unhide few define/reference anomalies
- A variable is defined but not used/referenced
- A variable is used but never defined
- A variable is defined twice before it is used

 66 U4

Data Flow Testing cont..

•Def Use technique will establish relations between variable definition and its uses.

An example:

read (x)	//definition of variable x,
write (x)	//output usage of x (referenced),
if (x=1) then x := 7	//predicate usage of x, definition of x,
a[i] := x	//definition of a[i], usage of i and x,
x := x+1	//calculus usage (right side) and definition of x (left side)

 U4

"Fault" Detection with Def Fault" Detection with Def Use

```
Program p(input, output)
Var x, y, z, a, b, c : integer
begin
read (c);
x := 7;
y :=x + a;
b := x + y + a
c := y + 2*x + z;
return (c)
```

Variable Def/Use chain technique

Variable du-chain

- **x duuu**
- **y duu**
- **z u**
- **a uu**
- **b d**
- **c ddu**

 66 U4

Data Flow Testing cont..

- **Usage without definition (a, z)**
- **Sequence of definitions without usage between (c)**

Variable du-chain

•x	duuu
•y	duu
•z	u
•a	uu
•b	d
•c	ddu

General static analysis based on Def/usage chain

1)u... variable initial value is missing

2) ...dd.. consecutive definitions, the first is useless

3) ...d last definition is useless

 67 U4

Mutation Testing

- *Mutation testing* injects faults into code to determine optimal test inputs.
  - Similar to fault seeding
  - Determine properties of test cases
- For example, a constant might be incremented by one, decremented by one, or replaced by zero, yielding one of three mutants.

 70 U4

Mutation Testing cont..

- *Multiple copies of the program are made, and each copy is altered*
- *Alter copy is called a mutant*
- *Mutant detected by a test case is termed "killed"*
- *Objective is to find a set of test cases that are able to kill the group of mutants*

 71 U4

Mutation Testing cont..

- Total mutant = killed+ live+ equivalent
- Score with a test suite T and mutants M is

  #killed/(#total-#equivalent) * 100%

 72 U4

Levels of Testing

- Unit testing
- Integration testing
- System testing

 73 U4

Unit testing

- Reasons to support unit testing
  - Easy to locate the bug
  - Exhaustive testing upto some extent
  - Interaction of multiple errors can be avoid
- Requires overhead code for driver and stub called as scaffolding
- Generate scaffolding automatically by means of test harness

 74 U4

Integration Testing

- Target is interface
- Different integration strategy
  - Top down
  - Bottom up
  - Sandwich
- Require regression testing
  - Helps to ensure that changes should not introduce additional errors

 75 U4

System Testing

- According to Petschenik guidelines for choosing test cases during system testing
  - Testing the system capabilities is more important than testing components
  - Testing the usual is more important than testing the exotic
  - In case of modifications; test old capabilities rather than new ones
- Attributes evaluate during system testing
  - Usable
  - Secure
  - Compatible
  - Dependable
  - documented

 76 U4

Debugging

- Activity of locating and correcting errors
- Characteristics of bug
  - The symptom and cause nay be geographical remote
  - The symptom may disappear when another error is corrected
  - Symptom may be caused by no errors
  - Symptom caused by human erroe difficul to trace

 77 U4

Debugging cont..

- The symptom may be result of timing problem rather than process problem
- May be difficult to reproduce the accurately input conditions
- The symptom may be intermittent

 78 U4

Debugging Techniques

- Core dumps
- Traces
- Print statements
- Debugging programs

 79 U4

Debugging Approaches

- Trial and error
- Backtracking
- Insert watch points
- Induction & deduction

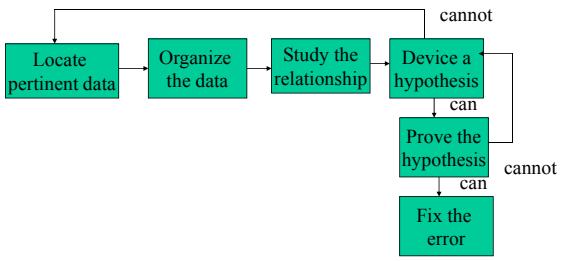
 80 U4

Induction Approach

- Formulation of a single working hypothesis based on the
- Data
- analysis of existing data
- specially collected data to prove or disapprove the hypothesis

 81 U4

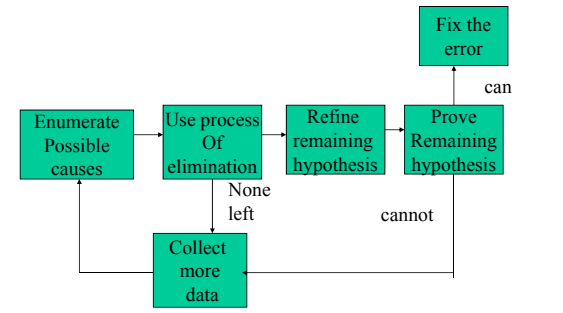
Induction Approach..



Deduction Approach

- Begins by enumerating all causes or hypotheses which seem possible.
- One by one, particular causes are ruled out until a single one remains for validation

Deduction Approach..



Testing Tools..

- One way to improve the quality and quantity of testing is to make the process as pleasant as possible for tester
- Different types of tools
  - Static analyzers
  - Code inspectors
  - Standard enforcers
  - Coverage analyzers (execution verifiers)
  - Output comparators
  - Test file/ data generators
  - Test harnesses
  - Test archiving systems

 85 U4

Software Maintenance

Any Work Done to Change the Software After it is in Operation

 U4. 86

Learning Objectives

Software Maintenance:

- Maintenance Types
- Management of Maintenance
- Maintenance Process
- **Maintenance Models**
- Estimation of Maintenance Cost
- Regression Testing
- **Reverse Engineering**
- **Software Re-engineering**
- **Configuration Management**
- Documentation.

 87 U4

What is Software Maintenance

- Includes
  - error correction
  - enhancement of capabilities
  - deletion of obsolete capabilities
  - optimization
- Preserve the value of software over time

 U4

Categories of Maintenance

Corrective maintenance
- Modification initiated by defects in the software

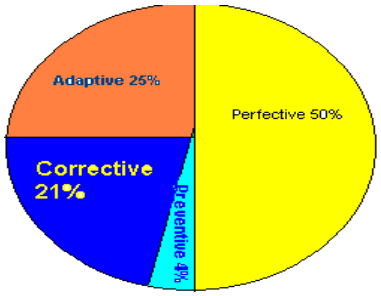
Adaptive maintenance
- Modifying software to match changes in the ever changing environment

Perfective maintenance
- Improving processing efficiency or performance or restructuring the software

Other types of maintenance
- **Preventive maintenance**
  - ✓ **Code restructuring, code optimization, and documentation updating**

 U4

Distribution of Maintenance Efforts



 U4

Distribution of Maintenance Effort

1	Emergency debugging	12.4%
2	Routing Debugging	9.3%
3	Data environment adaptation	17.3%
4	Changes in hardware and OS	6.2%
5	Enhancements for users	41.8%
6	Documentation Improvement	5.5%
7	Code efficiency improvement	4.0%
8	others	3.5%

Suggested by Lientz and Swanson

Kinds of Maintenance Requests

1	New reports	40%
2	Add data in existing reports	27.1%
3	Reformed Reports	10%
4	Condense Reports	5.6%
5	Consolidate Reports	6.4%
6	Others	10.1%

Problems During Maintenance

- Program written by another person
- Unclear understanding
- Difficulty in reading program listing
- High staff turnover in IT industry
- Communication gap between user and maintenance team
- Non maintainable software
- ***Translate to a huge maintenance expenditure 40% to 70%***

Potential Solution to Maintenance Problem

- Budget & effort Reallocation
- Complete replacement of the system
- Maintenance of existing system

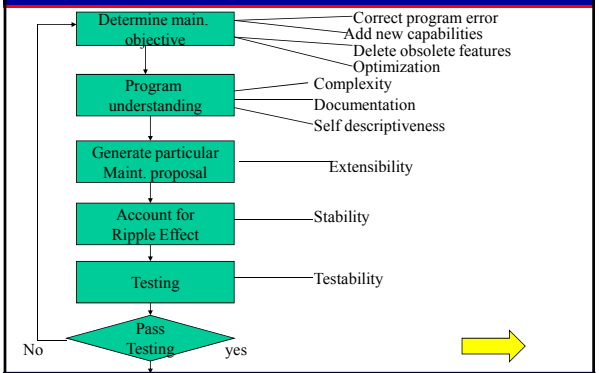
 U4

Maintenance Process

- Establish maintenance objective
- Maintenance personnel understand what they are to modify
- Modify the program to satisfy the maintenance objective
- Ensure that the modification doesn't affect other portion of the program
- Test the program
- Program maintainability & program understandability are parallel concept

 U4

Maintenance Process..

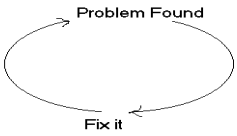


 U4

Maintenance Models

Quick Fix Model

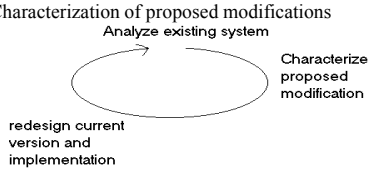
- Fire fighting approach
- Ripple effects



 97 U4

Iterative Enhancement Model

- Well suited for maintenance
- Three stage cycle
  - Analysis
  - Characterization of proposed modifications
  -
- As



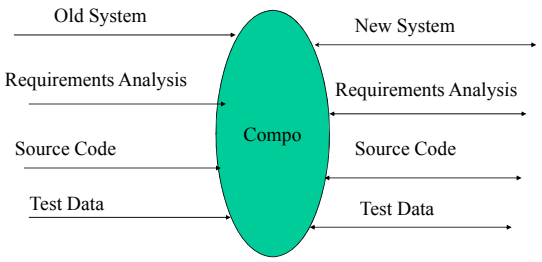
 98 U4

Reuse Oriented Model

- Viewed as an activity involving the reuse of existing program components
- Has four main steps
  - Identification of the parts of the old system that are candidates for reuse
  - Understanding these system parts
  - Modification of the old system parts appropriate to the new requirements
  - Integration of the modified parts into the new system

 99 U4

Reuse Model



 U4

Boehm's Model

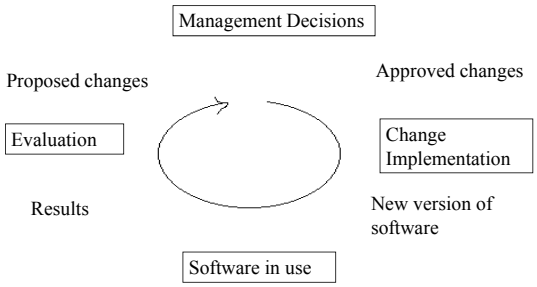
- Based on economic model & principles
- Improve productivity & helps in understanding the process

According to Boehm model

- It is the stage where management decisions are made that drives the process
- A set of approved changes is determined by applying a particular strategy and cost benefit evaluation
- Budget accompanied with approved changes determine the extent and type of resources expanded
- Manager's decision are based on the balancing of objectives against the constraints

 10 U4

Maintenance Process

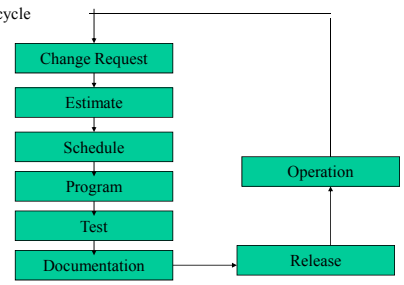


 U4

Taute Maintenance Model

Developed by B.J. Taut in 1983

Has eight phases un cycle



Estimation of Maintenance Cost

- Belady and Lehman Model
- Boehm Model

Belady and Lehman Model

Basic Equation

M = P+K e^(c-d)

M: Total Efforts expended

P: Productivity efforts

K: An empirical determined Constant

c: Complexity measure due to lack of good design and documentation.

d: Degree to which maintenance team is familiar with the software

Belady and Lehman Model

- The development efforts for a software is 500 PM. The empirically determined constant (K) is 0.3. The complexity of the code is quite high and is equal to 8. Calculate the total effort expended (M) if

(i) Maintenance team has good level of understanding of the project (d = 0.9)

(ii) Maintenance team has poor understanding of project (d= 0.1)

M = P+K e^(c-d)

Boehm Model

- Part of COCOMO Model
- Annual Change Traffic (ACT)
- The fraction of a software product's source instructions which undergo change during a year either through addition, deletion or modification

ACT = [KLOC(added) +KLOC (deleted)] / KLOC(total)

Annual Maintenance Effort(AME)

AME = ACT*SDE

Boehm Model

- Annual change traffic(ACT) for a software system is 15% per year. The development effort is 600 PMs. Computer an estimate for Annual Maintenance effort(AME). If life time of the project is 10 years, what is the total effort of the project?

AME = ACT*SDE

Boehm Model

A software project has development effort of 500 PMs. It is assumed that 10% code will be modified per year. Some of the cost multipliers are given as

RELY: high = 1.15

DATA: high = 1.08

ACAP: high = 0.86

AEXP: very high =0.82

LEXP: high = 0.95

Calculate the Annual Maintenance Effort (AME).

AME = ACT*SDE*EAF

 U4

Regression Testing

- Process of retesting the modified parts of the software and ensuring that no new errors have been introduced into previously tested code
- Purposes
  - Increase confidence
  - Locate errors
  - Preserve the quality and reliability
  - Ensure the software continued operation

 U4

Development Testing vs. Regression Testing

SN	Development Testing	Regression Testing
1	Create Test Suite & Test plan	Use of existing test suite & Test plan
2	Test all software component	retest affected components
3	Budget gives time for testing	Budget often does not give time for regression testing
4	Perform testing just once	Perform testing many times
5	Perform under the pressure of release date	Performed in crisis situations under grater time constraints

 U4

Regression Test Selection

Fragment A		Fragment B	
S1	Y=(x-1)*(x+1)	S1'	Y=(x-1)*(x+1)
S2	If(y=0)	S2'	If(y=0)
S3	Return(error)	S3'	Return(error)
S4	else	S4'	else
S5	Return(1/y)	S5'	return(1/(y-3)

Test Cases		
TN	**Input**	**Exe His**
T1	X=1	S1,S2, S3
T2	X=-1	S1,S2, S3
T3	X=2	S1,S2,S5
t4	X=0	S1,S2,S5

Selective Retest Techniques

- Differ from retest all
- Partition test suite into reusable, retestable & obsolete test cases
- Obsolete test cases may be Specification-obsolete or coverage-obsolete

Selective Retest Techniques

- Coverage techniques
- Minimization Techniques
- Safe Techniques
- Categories on which techniques can be compared
- Inclusive ness
- Precision
- Efficiency
- generality

Reverse Engineering

- Process followed in order to understand different unknown hidden information about a software system

 U4

Scope and Tasks

Areas where reverse engineering is applicable:

- Program comprehension
- Redocumentation and / or document generation
- Recovery of design approach and design details at any level of abstraction
- Identifying reusable components
- Identifying the components that need restructuring
- Recovering business rules
- Understanding high-level system

 U4

Scope and Tasks cont..

- Encompasses a wide array of tasks related to understanding and modifying software system
- Mapping between application and program domain
- Mapping between concrete and abstract levels
- Rediscovering high-level structures
- Finding missing links program syntax and semantics
- To extract reusable components

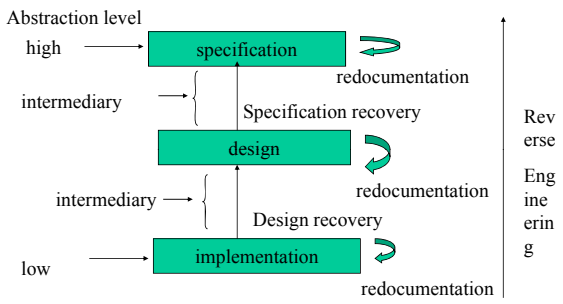
 U4

Levels of Reverse Engineering

- Detect low-level implementation constructs and replace them with their high level counterparts
- ▪ Redocumentation
- ✓ Alternative views
- ✓ Improve current documentation
- ✓ Generate documentation for new modified program
- ▪ Design Recovery

 U4

Levels of Reverse Engineering..



 U4

Reverse Engineering Tools

- Four basic components of a reverse engineering tool
  - ▪ Restructurer
  - ▪ Cross referencer
  - ▪ Static analyzer (identify anomalous constructs)
  - ▪ Text editor

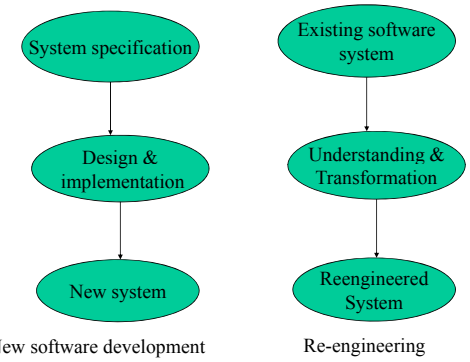
 U4

Software Re-engineering

- Concerned with taking existing legacy systems and re-implementing them to make them more maintainable
- Cost depends on
  - Extent of work
  - Quality of software
  - tool support
  - Extent of data conversion
  - Availability of expert staff

 U4

Maintenance Process



 U4

Suggestion for Reengineering

- Study code well
- Concentrate on control flow
- Heavily comment internal code
- Create cross reference
- Build symbol table
- Use own variables, constants and declaration
- Keep detailed maintenance documents
- Use modern design techniques

 U4

Source Code Translation

- Reasons
  - Hardware platform update
  - Staff skill shortage
  - Organization policy changes
- Translation can be done manually or using automation tool

 U4

Program Restructuring

- Involves transforming a system from one representational form to another without a change in the semantics or functionality
- Different types of restructuring techniques
  - Control flow driven restructuring
  - Efficiency driven restructuring
  - Adaption-driven restructing

 U4

Configuration Management

- Discipline for managing the evolution of computer products, both during the initial stages of development, and through to maintenance & final product retirement.
- Consists of a set of activities developed to manage change throughout the software life cycle.

 U4

Configuration Management..

SCM addresses

- What constitutes the software product at any point of time
- What changes have been made to the software product

 U4

Configuration Management Activities

- Identification of the component
- Change control
- Auditing the changes
- Accounting-recording and documenting all the activities

 U4

Elements of SCM

- Configuration Identification
- Configuration Control
  - Revision/variation
- **Configuration Audit**
- Configuration Status Accounting & Reporting

 U4

Versions

Version control is used to

- Keep track of SCI revisions
- To manage different versions of SCIs
- To ensure repeatability & ability to reproduce any version of the software at any time

 U4

Change Control Process

- Defining process changes to a known baseline
- requests for changes submitted to change control authority (CCA)
- CCA approve change request
- Revision is revalidated by the SQA team
- Changes are handed over to SCM team and is incorporated as a new version

 U4

Documentation

- Written record of facts about a software system recorded with the intent to convey purpose, content and clarity
- Different categories of software documents
  - User Documentation
  - System Documentation

 U4

Unit IV Learning's

Software Testing:

- Testing process
- Design of test cases
- functional testing
- Boundary value analysis
- Equivalence class testing
- Decision table testing
- Cause effect graphing
- Structural testing
- Path Testing
- Data flow and mutation testing
- Unit Testing
- Integration and System Testing
- Debugging
- Alpha & Beta Testing
- Testing Tools & Standards.

 U4

Unit IV Learning's..

Software Maintenance:

- Management of Maintenance
- Maintenance Process
- Maintenance Models
- Regression Testing
- Reverse Engineering
- Software Re-engineering
- Configuration Management
- Documentation.

 U4

Objective Questions

Q1. Will exhaustive testing guarantee that the program is 100% correct?

Q2. For a function of n variables robustness testing of boundary value analysis yields:

(i) 4n+1 (ii) 6n+1 (iii) 4n+3 (iv) none of above

Q3. Define Test Suit.

Q4. Define Scafffolding.

Q5. What are the Configuration Management Activities?

Q6. How does preventive maintenance differ from adaptive maintenance?

Q7. List four types of systems tests

 U4

Short Questions

Q1. Write short note on
- Reverse engineering
- Reengineering
- Integration Testing
- Maintainability
- Boehm's maintenance model
- Regression Testing
- Reverse Engineering
- Configuration Management

Q2. Discuss the importance of path testing during white box testing.

Q3. What is the difference between system testing and performance testing?

 U4

Short Questions..

Q4. Differentiate White Box Testing vs. Black Box Testing

Q5. Differentiate Top Down vs. Bottom up integration strategy

Q6. Differentiate Induction vs Deduction debugging Approach

Q7. Write a short note on.
Testing tool
Regression Testing

Q8. Write short note on any three
- Debugging approach
- Taute Maintenance Model

 U4

Short Questions..

Q9. Differentiate the following
- Top-down vs. Bottom up integration testing
- Black-box testing vs. White-box testing.
- New software development vs. Re-engineering

 U4

Long Questions

Q1. Consider a program that computes grade of students. The grading is done as:

Marks obtained Grade

80-100 A

60-79 B

50-59 C

40-49 D

0-39 E

Generate test cases using robust testing and decision table based testing.

Long Questions..

Q2. Consider the program for the determination of previous date in a calendar. Its input is a triple of day, month and year with the following range

1<= month <=12

1<= day <=31

1801<= year <= 2009

The possible outputs would be previous date or invalid date. Design the boundary value , robust and worst test cases for the program.

Long Questions..

Consider the following program segment.

```
void sort{ int b[],int n} {
int i,j;
for(i=0;i<n;i++)
for(j=i+1;j<n;j++)
if(b[i]>b[j])
{
temp=b[i];
b[i] =b[j];
b[j] =temp;
}
}
```

Long Questions..

- Draw the control flow graph and DD path graph for this program segment.

  Determine the cyclomatic complexity for this program using all the methods. (Show the intermediate steps in your computation. Writing only the final result is not sufficient).

Q5. Describe various maintenance cost estimation models.

Q6. Write a short note on Boledy and Lehman model for the calculation of maintenance effort.

Q7. What are various debugging approaches? Discuss them with the help of Examples

Q8. Consider program for determination of date in a calendar. Its input is a triple of day, month And year with following range 1≤month≤12 1≤day≤31 1900≤year≤2005. The possible Outputs would be Net date or invalid input date. Design boundary value, robust and worst test cases for this program.

Long Questions..

Q9. Describe equivalence class testing method.

Q10. Expain usefulness of decision table during testing.

Q11. Draw flow graph for a program of largest of three numbers. Find out all independent paths that will guarantee that all statements in the flow graph have been tested.

Q12. Discuss suggestions that may be useful for modification of the legacy code.

Q13. What are configuration management activities? Explain.

Q14. Explain the steps of software maintenance with help of diagram.

Q15. What are selective retest techniques? How are they different from "retest-all"technique?

Q16. What is reverse engineering? Write in short the various levels of reverse engineering

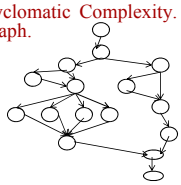
Research Problems

Q1. Design a decision table based on the below problem.

Students can receive the grades: A, B, C, D, or F. When looking at course work the standard grading methodology: *90-100 A, 80-89 B, 70-79 C, 60-69 D, Below 60 F* is used. However a student's grade can be changed according to the following rules: those whose attendance is less than 75% will be reduced one letter grade, those whose attendance is less than 60% will reduced two letter grades, those who miss more than 50% of classes will fail regardless of their percentage grade in the course work. Additionally, any student who is an athlete and receives a grade lower than a "C" or whose attendance is less than 75% will be refered to the Director of Student Athlete Development.

Research Problems..

Q2. Desccribe Cyclomatic Complexity. Calculate the V(G) for the given flow graph.



Q3. In a computer store, the computer item can have a quantity between -500 to +500. What are the equivalence classes?

Q4. Draw decision table for the following statement No charges are reimbursed to the patient until the deductible has been met. After the deductible has been met, reimburse 50% for Doctor's Office visits or 80% for Hospital visits

Research Problems..

Q5. What will be the cyclomatic complexity of given program

```
#!python
def func(x):
if x==0:
return 3
elif x==1:
return 4
elif x==2:
return 5
else:
return 0
```


References

1. K. K. Aggarwal & Yogesh Singh, "Software Engineering", 2nd Ed., New Age International, 2005.
2. R. S. Pressman, "Software Engineering – A practitioner's approach", 5th Ed., McGraw Hill Int. Ed., 2001.
3. Pankaj Jalote, "An Integrated Approach to Software Engineering", Narosa, 3rd Ed., 2005.
4. Stephen R. Schach, "Classical & Object Oriented Software Engineering", IRWIN, 1996.
5. James Peter, W. Pedrycz, "Software Engineering: An Engineering Approach", John Wiley & Sons.
6. Sommerville, "Software Engineering", Addison Wesley,8th Ed., 2009.

References..

7. Frank Tsui and Orlando Karan, "Essentials of Software Engineering", Joes and Bartlett, 2nd Ed., 2010.
8. Kassem A. Saleh, "Software Engineering", Cengage Learning, 2009.
9. Rajib Mall, "Fundamrntal of Software Engineering", PHI, 3rd Ed., 2009.
10. Carlo Ghizzi , Mehdi Jazayeri and Dino Mandrioli, " Fundamental of Software Engineering", PHI, 2nd Ed.,2003.
11. Carol L. Hoover, Mel Rosso-Llopart and Gil Taran, "Evaluating Project Decision Case Studies in Software Engineering", Pearson, 2010.

References..

14.http://www.softpanorama.org/SE/software_life_cycle_models.shtml

15.http://en.wikipedia.org/wiki/Systems_Development_Life_Cycle

16. http://www.levela.com/software_home.htm

17. www.ifpug.com

18. www.softwaremetrics.com

19. www.qualityworld.com

20. www.sei.cmu.edu

21. www.spr.com

