13. TESTING STRATEGIES

13.1 STRATEGIC APPROACH TO S/W TESTING : planned in advance & conducted systematically, characteristics of testing

- s/w team should conduct FTR, many errors uncovered before testing
- begins at component level, move outward toward integration of entire computer-based system
- different testing techniques at different points
- conducted by developer as well as independent test group
- debugging accommodated in testing strategy

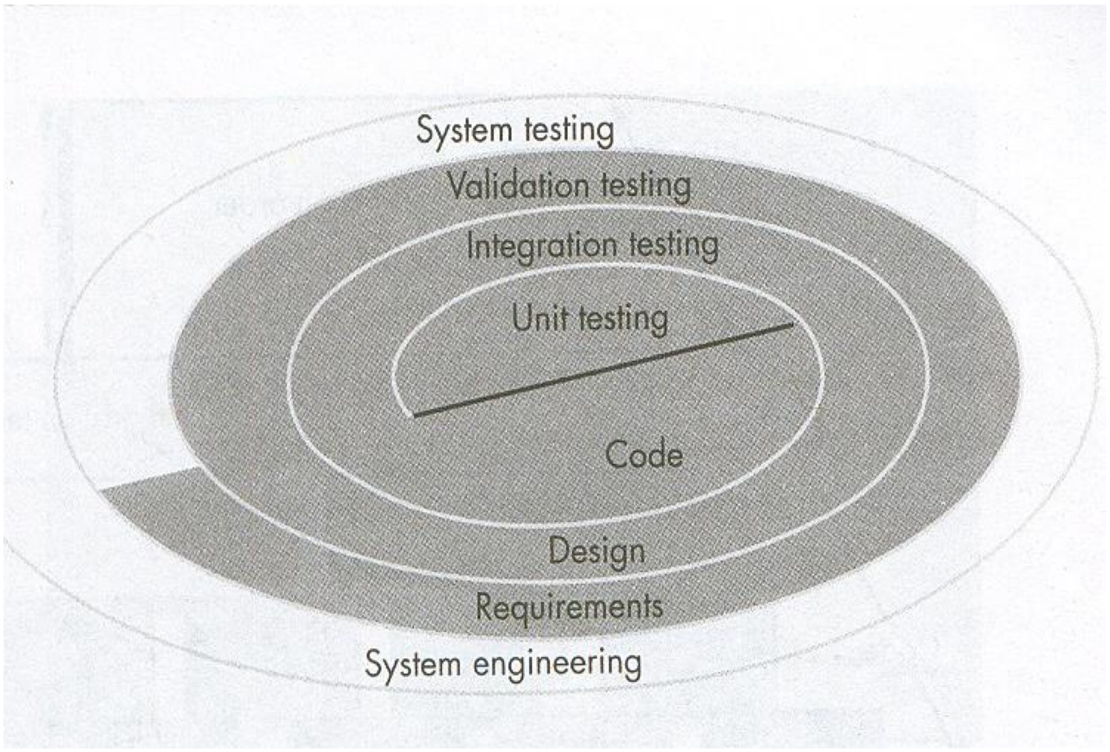
Accommodate low-level testing to check source code implemented correctly & high level testing – major system functions acc to requirements

<u>13.1.1 Verification & Validation</u> : veri refers to activities that ensure that s/w correctly implements a specific function, vali refers to activities that ensure that s/w built acc to customer requirements

Encompass FTR, Quality & configuration audits, performance monitoring, simulation, feasibility studies, diff testing etc, quality can be assessed thro testing but quality can not be tested

<u>13.1.2 Organizing for S/w Testing</u> : developers like to show product as error free, works according to requirements, complete on schedule & within budget, avoid thorough testing, analysis & design are constructive whereas testing is a destructive task, s/w developer must test individual units even perform integration testing, independent test group for testing after s/w archi completes, developer & group work together, group must be part of s/w team

<u>13.1.3 S/w Testing Strategy for Conventional S/W Architecture</u> : testing a spiral activity, initially system engineering defines s/w role, next s/w requirement analysis establishes info domain, function, behavior, performance, constraints & validation criteria, next design & then coding



Than begin unit testing, each unit tested, next integration testing focus on design & construction of archi, validation testing where requirements validated, finally system testing, s/w & other elements tested as a whole

In context of s/w engineering, testing is a series of 4 steps, initially unit-testing focus on units, exercise specific paths, ensure complete coverage & max error detection, integrated component tested by integration testing, testing for verification & program construction, focus on i/p & o/p, after s/w integrated high order testing- validation testing for functional, behavioral & performance requirements conformance, s/w combined with other system elements, system testing for overall system function & performance

<u>13.1.4 A S/W Testing Strategy for OO Architectures</u> : testing broadened to include error discovery techniques ex FTR, completeness consistency of objects assessed as constructed, classes are integrated into OO archi egression testing performed to uncover errors in communication & collaboration b/w classes, lastly system as a whole

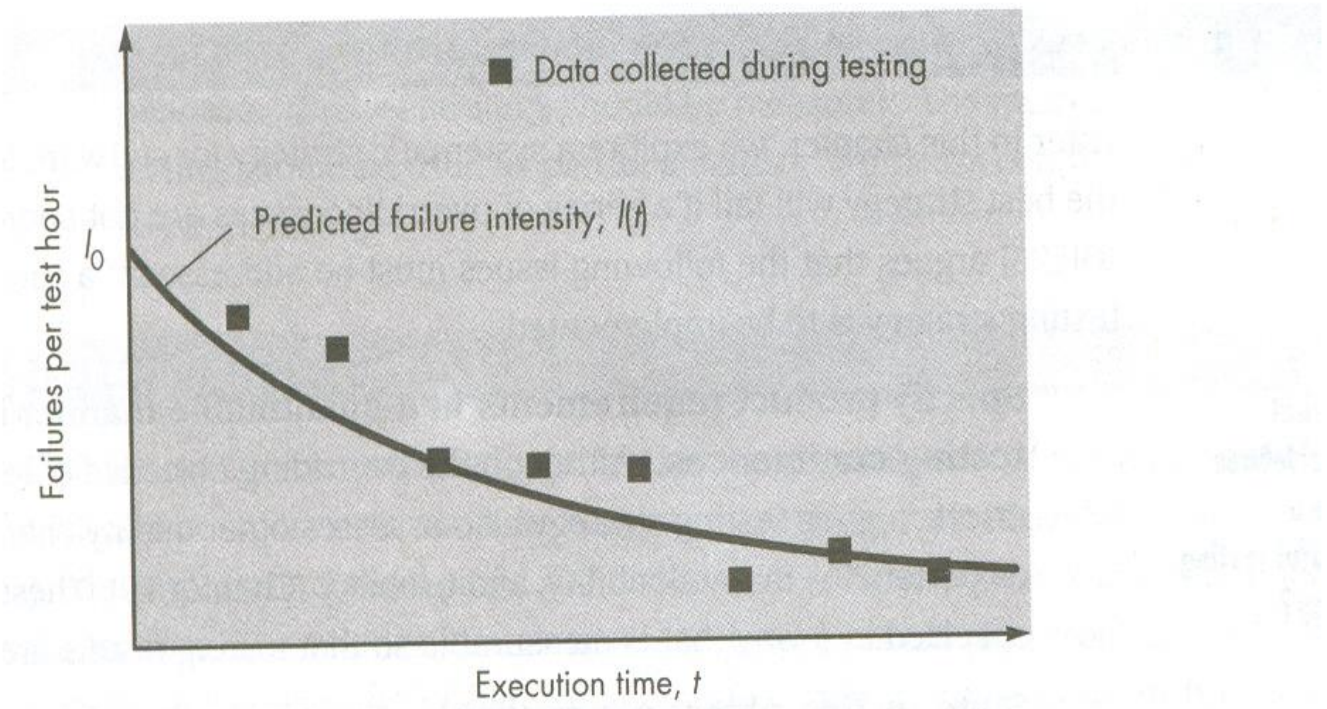
<u>13.1.5 Criteria for Completion of Testing</u> :   when testing is complete? No definite answer, every time a user executes a program, it is tested, statistical modeling & s/w reliability theory used to model s/w failure as function of execution time, s/w failure as a function of time is calculated for this

f(t) = (1/p) ln[Io pt + 1]

f(t) = no of failures expected after s/w tested for execution time t
Io = initial s/w failure intensity at beginning of testing
P = exponential reduction in failure intensity

Instantaneous failure intensity calculated as
L(t) = Io / (Io pt + 1)

Can predict drop-off of errors as testing progress, error intensity plotted against predicted curve, if reasonably close then testing time predicted to acceptably low failure intensity



13.2 STRATEGIC ISSUES :

For successful s/w testing

- Specify product requirements in a quantifiable manner long before testing commences. Good testing also assess product for quality attributes such as portability, maintainability etc, specified in measurable way

- State testing objectives explicitly. Objectives stated in measurable terms ex test effectiveness, test coverage, mean time to failure, cost to find & fix defects etc in test plan
- Understand the users of s/w & develop user profile for each user category. Use-case for interaction scenario, reduce testing efforts, focus on actual use of product
- Develop a testing plan that emphasizes rapid cycle testing. Rapid cycle tests, increments of functionality, feed back used for quality control & test strategy
- Build robust s/w that is designed to test itself. Use of antibugging techniques in s/w, design accommodate automated & regression testing
- Use effective FTR as a filter prior to testing. Uncovers errors, reduce testing effort & time
- Conduct FTR to assess the test strategy & test cases themselves. Uncovers inconsistencies, omissions & errors, saves time & improves quality
- Develop a continuous improvement approach for the testing process. Collection of Metrics

13.3 TESTING STRATEGIES FOR CONVENTIONAL SOFTWARE

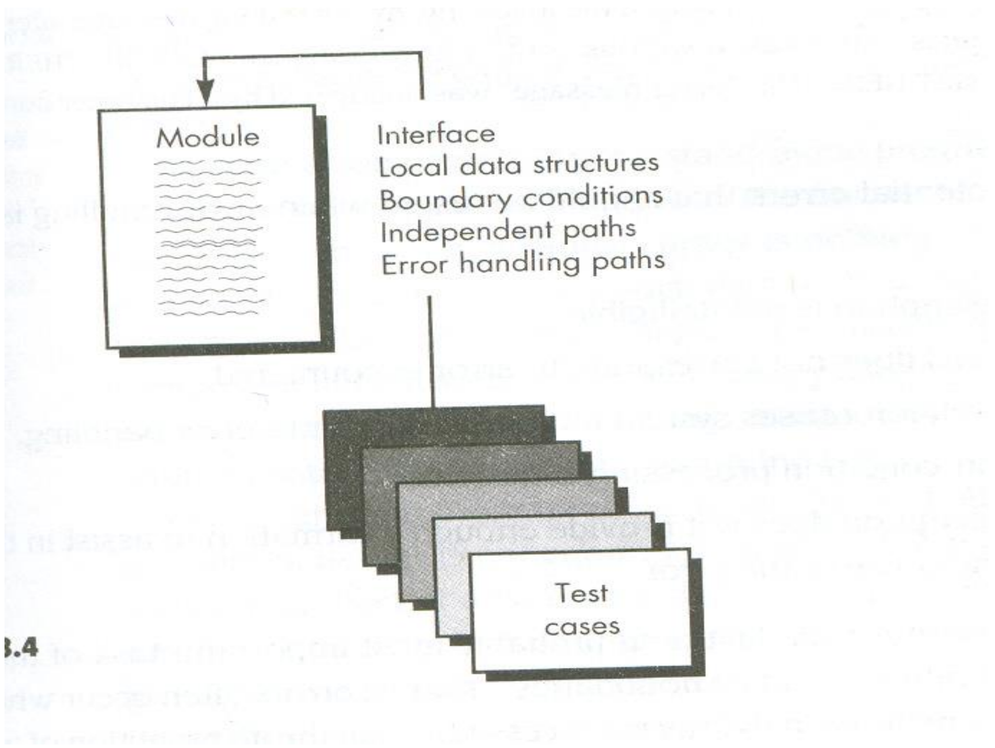
Many strategies, one is to wait till system is fully constructed & then find errors, do not work, one approach is to perform testing on daily basis, effective, another approach falls b/w two, incremental testing, begins with testing of individual program unit

13.3.1 UNIT TESTING :

Verification of smallest unit, imp control paths tested to uncover errors within boundary of module, test & errors uncovered are limited, focus on internal process logic & data structure

Unit Test Consideration :

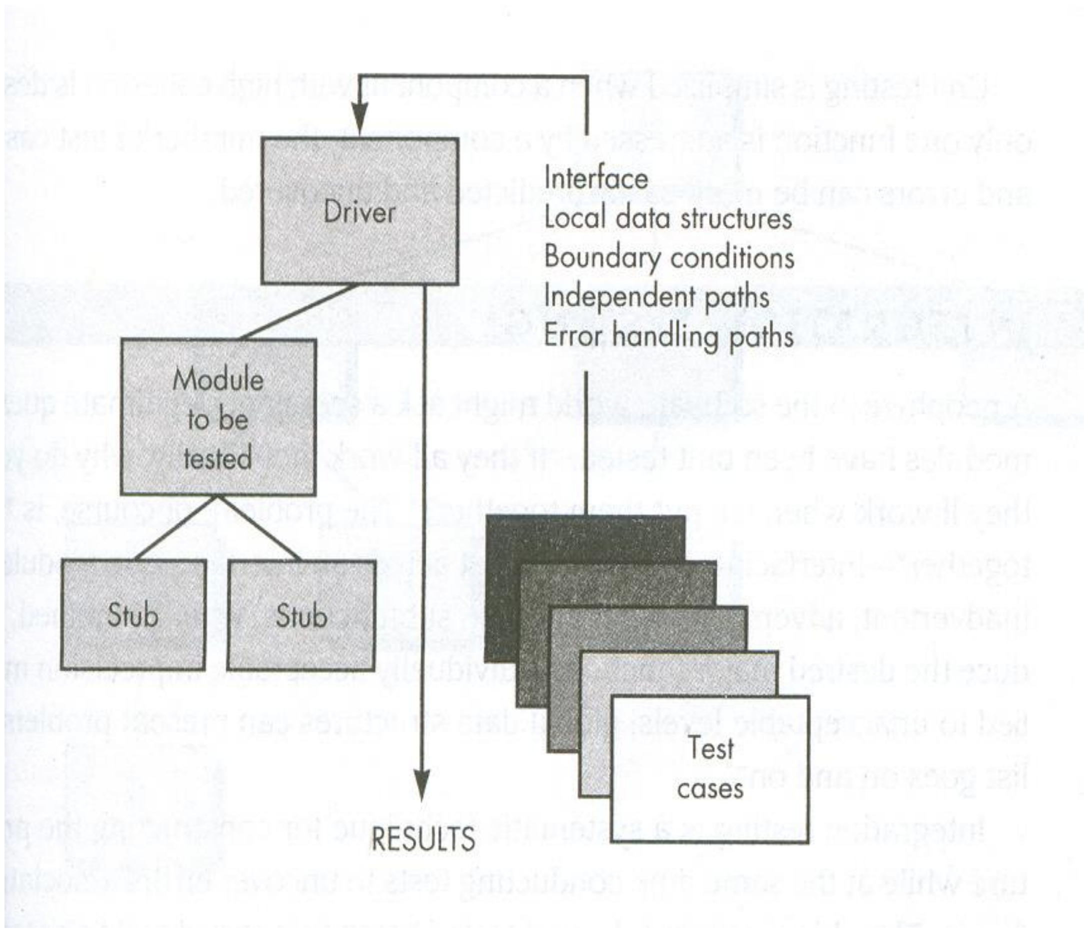
Module i/f tested for proper info flow into & out of module, If data do not enter or exit properly, local DS to check data maintains integrity in all steps, impact of local DS to global DS, boundary conditions tested to check operations at boundary, all independent paths exercised to ensure all statements executed at least once, error-handling path tested,



Test must uncover errors due to erroneous computation, incorrect comparisons or improper control flow, computational errors are (1) misunderstood or incorrect arithmetic precedence (2) mixed mode operation (3) incorrect initialization (4) precision inaccuracy (5) incorrect symbolic representation, test for incorrect comparison & improper control flow errors are (1) comparison of different data types (2) incorrect logical operator or precedence (3) expected equality when not possible (4) incorrect comparison of variables (5) improper or nonexistence of loop termination (6) exit not provided in diverse condition (7) improperly modified loop variables

Boundary testing most imp, s/w often fails at boundary, test case for DS, control flow & data values just below, at & just above max & min
Antibugging technique is during programming, errors anticipated & error-handling paths setup or terminate the processing cleanly when error occurs, these error-handling also must be tested, errors encountered are (1) unintelligible error description (2) no correspondence to error noted & error occurred (3) error cause system intervention before error handle executes (4) exception condition processing improper

Unit Test Procedures : Unit test after source code developed, reviewed & verified, each test case coupled with expected results, a module is not a stand alone, a driver or stub s/w developed for each unit test

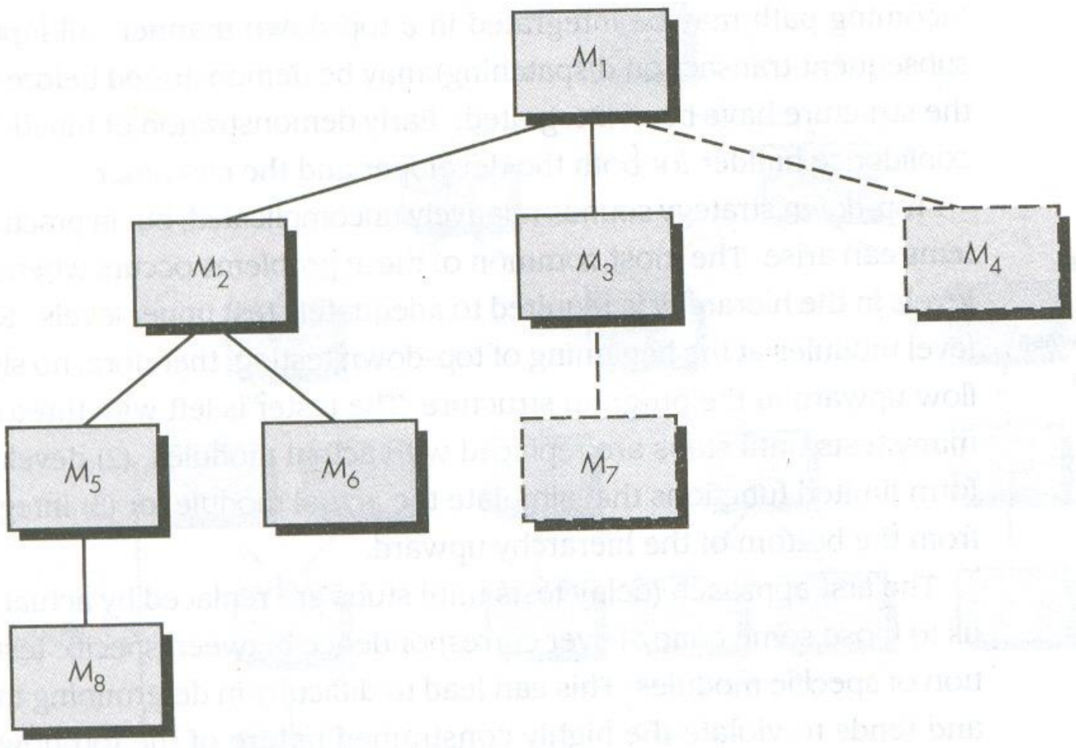


Driver, a main program accepts test data, passes to test module & prints results, Stub replace subordinate modules of test module, a dummy subprogram, do minimal data manipulation, print verification & returns control to test module, drivers & stubs are overhead to program, keep them simple, many compo not tested adequately with driver & stub then postpone until integration, Unit testing simple if module with high cohesion, less no of test cases & errors easily uncovered

13.3.2 INTEGRATION TESTING

Data can be lost across i/f, one module may have adverse effect on others, combined subprograms may not produce desired result, acceptable imprecision might get magnified, problem in global DS, for these Integration testing, systematically construct program structure & conducts tests for i/f errors, 2 methods, combine all subprograms & construct whole structure & then test, chaos results, difficult to isolate causes of errors, incremental integration better

Top-down Integration : incremental program structure by moving downward, begin with main program, subordinate modules in depth-first or breadth-first manner, d-f integrate all modules on a major control path, b-f incorporate modules level by level



Integration process follows series of steps

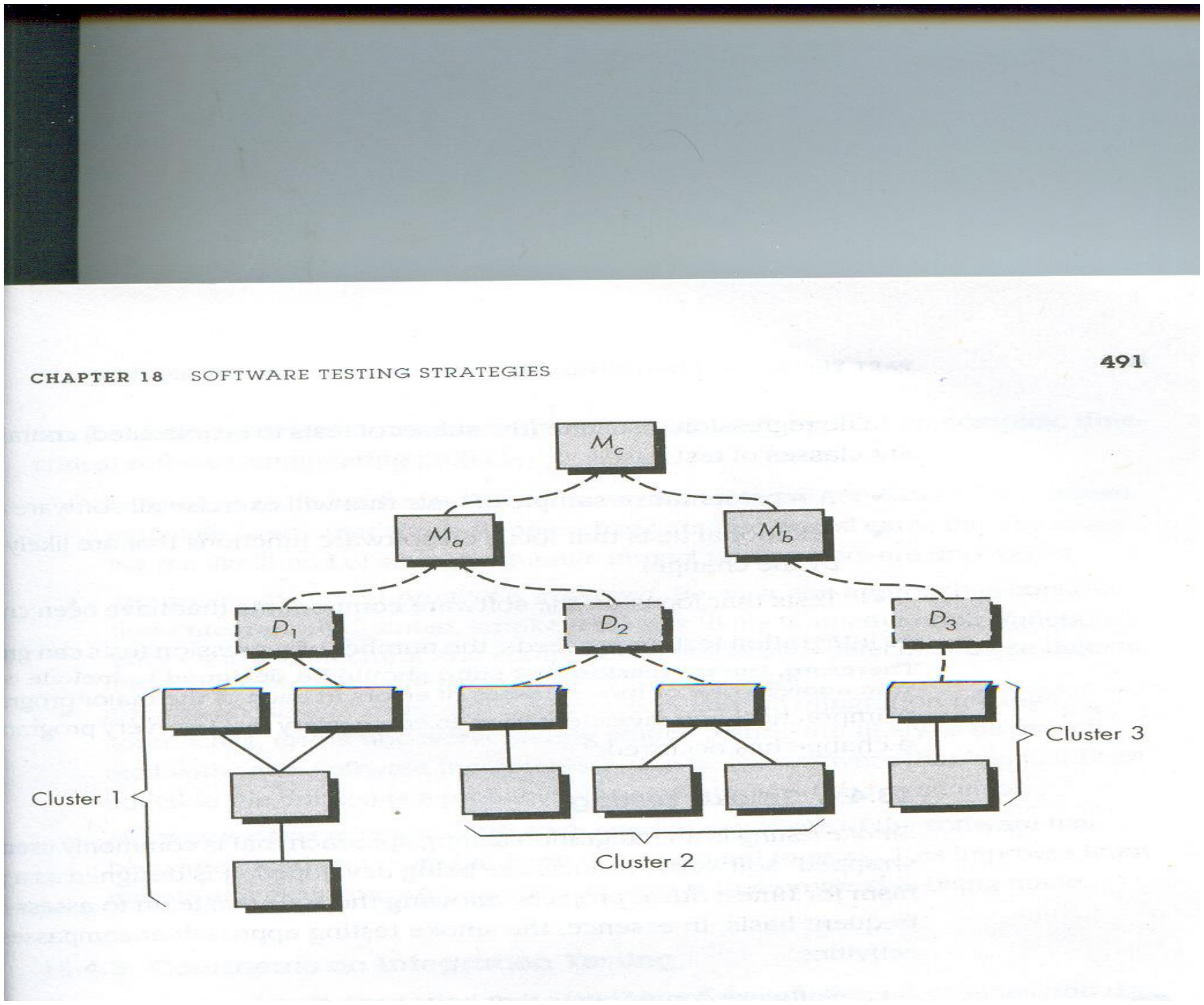
1. Main control module as a test driver & stubs substituted for direct subordinate of main module
2. Depending on approach, subordinate stubs replaced with actual module, one at a time
3. Test as each component integrated
4. Another stub is replaced
5. Regression testing to ensure new errors not introduced

Top-down approach verifies major control points early in test, early recognition of problem in major control, d-f checks complete function of s/w

Problem in top-down approach when processing at low level require adequate testing of upper levels, stubs in place of subordinate, actual data do not flow upward, either delay some test until after integration of actual subordinate or develop stubs with limited functionality of actual module or use bottoms up integration

<u>Bottom-up Integration</u> : Starts construction & testing from lowest level, need for stub eliminated, steps are

- Low level components combined into clusters that perform a specific sub function
- Driver for i/p/& o/p
- Cluster tested
- Drivers removed & clusters combined moving upward



Components combined to form 1,2,& 3 clusters, tested using driver, super ordinate of 1 & 2 i.e. Ma in place of D1 & D2, Mb replaces D3, then after Ma & Mb integrated to Mc, moving upward drivers in less no

Regression Testing : each new module added for integration testing changes s/w, new data paths, new I/O, new control logic, may cause problem with functions working flawlessly earlier, regression testing is re-execution of subset of tests conducted, to ensure that changes do not propagate side effects, whenever s/w debugged, configuration changes, regression testing to ensure new errors not introduced, done manually by executing subset of all test cases or automated capture/playback tools, enables to capture tests & results, regression test suite contains 3 classes of test cases (1) tests that will exercise all s/w functions (2) additional tests that focus on functions likely to be affected by change (3) tests of components that are changed
With progression of integration testing, regression test suite can grow large, include only those tests that address errors in major functions

Smoke Testing : used when shrink-wrapped s/w is developed, for time critical projects, allows to assess project frequently, activities are

1. S/w components coded are integrated into build, includes all data files, libraries, reusable modules & constructed modules to implement program function
2. Tests designed to expose errors in that function i.e. in build, errors that can affect project schedule are identified
3. Build integrated with other builds, smoke tested daily, diff integration approach

Beneficial on complex, time critical s/w

- integration risk minimized. Incompatibilities & stopping errors uncovered early, reduce serious schedule impact
- Quality of end product is improved. Constructive approach, uncovers functional, architectural & component design defects
- Error diagnosis & correction are simplified. Errors uncovered are associated with newly attached s/w
- Progress is easier to assess. Each day more s/w integrated & tested, improves team morale

Strategic options : advantage of one strategy may be disadvantage of other, disadvantage of top-down is requirements of stubs, adv is major control functions tested early, disadvantage of bottom-up is program as an entity do not exist until last module, adv is easy test case design & removal of stubs

Strategy depends on s/w characteristics, project schedule, combining top-down tests for upper level of programs & bottom-up for lower level is best. Critical module identified during testing, characteristics are (1) addresses several s/w requirements (2) high level of control (3) complex or error prone (4) has definite performance requirements, testes as early as possible

Integration test Documentation : plan for integration of s/w & description of tests in test specification, contains test plan & test procedure, part of s/w configuration, testing divided into phases & builds, address specific functions of s/w, criteria & tests applied are
Interface integrity : Internal & external i/f tested with module integration
Functional validity : test for functional errors
Information content : tests for errors in local & global DS
Performance : to verify performance bounds
Also contain schedule of integration, overhead s/w etc, start & end date of each phase, test env & resources, detail testing procedure to accomplish test plan, order of integration & tests, list of test cases & expected results etc
Actual test results, problem or peculiarity recorded in test specification

13.4 TEST STRATEGIES FOR OO S/W

Objective of testing to find more no of errors with manageable amount of efforts in realistic time span, testing strategy & tactics diff for OO s/w

13.4.1 Unit testing in OO context : concept of unit with OO changed, each class & its instance combine attributes & operations, unit testing focuses on encapsulated class, operation within class are smallest testable unit, a class contain no of diff operations & an operation exist as part of diff classes, tactic for unit test changes
a single operation can not be tested isolation, ex An operation X defined in superclass, inherited by a no of subclasses, each subclass uses operation X but within context of private attributes & operations of that subclass, thus operation X varies in each subclass, operation X to be tested in context of each subclass, standalone testing will be ineffective, unit testing focus on algorithmic details & data of module, OO testing focus on operations encapsulated in the class & state behavior of class

13.4.2 Integration Testing in OO Context : OO s/w do not have obvious hierarchical control structure, top-down & bottom-up integration meaningless, integrating classes one at a time impossible, because direct & indirect interactions b/w components of class, 2 diff strategies (1) thread-based strategy- integrated set of

classes that respond to one i/p or event for system, each thread integrated & tested individually (2) user-based testing- begins construction of system by testing those classes that use very few server classes, these classes are independent classes, after testing independent classes next layer of classes called dependent classes which use independent classes are tested, these sequence of testing layers of dependent classes continue until entire system is constructed
Use of drivers & stubs also changes while testing OO system, drivers used to test operations at lowest level & testing whole group of classes, driver replaces user i/f to check system functionality before implementation, stubs used when collaboration of classes is required but collaborating class is not implemented,
Cluster testing if integration testing of OO s/w, cluster of collaborating classes tested to uncover errors in collaboration

13.5 VALIDATION TESTING :

After integration testing s/w is assembled, validation testing checks if s/w functions according to customer requirements, validation criteria defined in s/w requirements specification provides base for testing

13.5.1 Validation Test Criteria : black-box testing, conformance to requirements, test plan & test procedures defines classes of tests & test cases, checks functional requirements satisfied, behavioral characteristics achieved, human engineer & other requirements such as error recovery, compatibility, maintainability meet, After each validation test, 2 possible conditions (1) function & performance confirms (2) deviation from specification, deviation or error at this stage rarely corrected before schedule

13.5.2 Configuration Review : ensure all s/w configuration elements developed & cataloged

13.5.3 Alpha & Beta testing : impossible to assess hoe customer will use program, instruction misinterpreted, strange combination of data, o/p not understandable to user, acceptance test conducted, customer validates requirements, conducted by end users, conducted over period of time to uncover errors that might degrade the product
If many users, alpha & beta testing to uncover errors that only end-users can find, alpha test conducted at developer’s site by user, controlled env, natural setting of developer, looking over the shoulder, records errors & usage problems

Beta test at customer site by end users, developer not present, live application of s/w, env not controlled, customer records problems & reports to developer

13.6 SYSTEM TESTING :

S/w incorporated with other system elements, h/w, people, env, system integration & validation tests, not solely by s/w engineers, system testing problems, finger-pointing, when error uncovered each developer blame others, engineers must (1) design error-handling paths (2) conduct series of tests (3) record results of test (4) participate in planning & design of system elements
A series of different tests, fully exercise computer based system, verify system elements integrated properly & perform allocated functions

13.6.1 Recovery Testing : computer-based system must recover from faults, resume processing within pre specified time, fault tolerance – processing faults do not cause overall system function to cease, recovery testing forces s/w to fail in diff ways & verify recovery is performed, if recovery automatic, re-initialization, checkpoint mechanism, data recovery, restart evaluated for correctness, if require human intervention MTTR evaluated.

13.6.2 Security Testing : computer-based system that have sensitive info is target for illegal penetration, spans a broad range of activities – hackers who attempt to penetrate system fro fun, disgruntle employee attempt for revenge, dishonest for illicit personal gain
Security testing verify protection mechanism, test play role of individual who penetrates the system, tester may acquire password through external clerical means, may attack system with custom s/w designed to break down any defense, confuse the system & stopping service to others, purposely cause some system errors & penetrate during recovery, browse insecure data, given time & resources good security testing penetrates a system, cost of penetration more than value of info

13.6.3 Stress Testing : White box 7 black-box evaluate normal program functions & performance, stress test confront program with abnormal situation, execute system demanding abnormal quantity, frequency or volume of resources, ex tests generated (1) demand ten interrupts in second (2) i/p data rate increased (3) require max memory or other resources (4) excessive disk data fetching etc

Sensitivity testing – some time small range of data within bounds may cause extreme or erroneous processing & performance degradation, sensitivity testing uncovers such data combinations

<u>13.6.4 Performance Testing</u> : for real & embedded systems, s/w provides required functions but not conform performance requirements, performance testing test run-time performance of an integrated system, occur throughout all tests, performance of individual module to full system, require special h/w & s/w, monitor execution intervals, log events, machine states

13.7 THE ART OF DEBUGGING :

As a consequence of testing, begins with test, removal of errors, external errors & internal cause may not have obvious relationship

<u>13.7.1 Debugging Process</u> : begins with testing, result assessed & checked with expected results, if not same there is problem, underlying cause must be found & error correction, either cause found & corrected or not found, design special test case & validate, debugging is difficult because

(1) symptom & cause geographically remote
(2) symptom disappear when another error corrected
(3) symptom may caused by non-errors
(4) may caused by human error, not easily traceable
(5) may result of timing problem rather than processing
(6) difficult to reproduce i/p conditions
(7) intermittent symptom common in embedded system
(8) causes distributed across a no of tasks running on diff processors

range from mild to catastrophic, pressure increase with errors, introduction of more errors while fixing one error

<u>13.7.2 Psychological Consideration</u> :

<u>13.7.3 Debugging Approaches</u> : objective is to find & correct cause of error, require systematic evaluation, intuition & luck, 3 approaches (1) brute force (2) backtracking & (3) cause elimination

Brute force more common, less efficient, when all else fail, memory dumps taken, run-time traces invoked, program loaded with write statements, to find clue to cause of errors, waste of time & effort

Backtracking for small programs, begins at symptom, source code traced backward until cause found, if large program no of paths to trace & unmanageable

Cause elimination work on binary partitioning, data related to error organized, cause hypothesis devised, & data used to prove or disprove hypothesis, alternately list of all possible causes developed & tests conducted to eliminate each

Debugging tools like debugging compilers, aids, automatic test case generators etc available, a fresh viewpoint, unclouded by frustration do wonders
Bugs found are corrected but may introduce new errors (1) cause of bug reproduces in other parts of program (2) new bug introduction while fixing (3) prevention of bugs

14. TESTING TECTICS

Critical element of s/w quality assurance, reviews specification, design & code, organizations expend 30-40% of total project effort on testing

14.1 S/W TESTING FUNDAMENTALS :

Testing demolishes the s/w constructed by s/w engineer; Good test has high probability of finding errors with min efforts

<u>Testability</u> : s/w engineer develop s/w with testability, testability – how easily a program can be tested, testing is difficult, metrics used to measure testability of s/w, the characteristics of a testable s/w are

- Operability : the better it works, more efficiently tested
    - the system has few bugs
    - bugs don't block execution of tests
    - product evolves in functional stages – simultaneous coding & testing
- Observability : what you see is what you get
    - distinct o/p for each i/p
    - system states & variables visible & queriable during execution
    - factors affecting o/p visible
    - incorrect o/p identified
    - internal errors detected through self –testing
    - internal errors reported
    - source code accessible
- Controllability : better control of s/w lead to automated & optimized testing
    - All possible o/p generated through some combination of i/p
    - All code is executable through some combination of i/p
    - S/w & h/w states & variables controlled by test engineer
    - I/p & o/p formats consistent & structured
    - Tests specified, automated & reproduced
- Decomposability : scope of testing limited so isolation of problems faster
    - S/w system from independent modules
    - Modules tested independently
- Simplicity : less to test is quick to test
    - Functional simplicity – min necessary to meet requirements

    - Structural simplicity – archi modularized to limit propagation of errors
    - Code simplicity – coding std
- Stability : as few as possible changes, less disturbance to testing
    - Change infrequent
    - Controlled change
    - Change do not invalidate existing tests
    - Recovery well
- Understandability : efficient test with more info
    - Design well understood
    - Dependencies b/w components well understood
    - Changes communicated

Technical documents accessible, well organized, specific & detailed, accurate

Test Characteristics
1. A good test has a high probability of finding errors. Must understand s/w, develop mental picture of how s/w might fail, diff classes of failure probed, ex, in GUI failure – failure to recognize proper mouse position
2. A good test is not redundant. Limited time & resources, every test have different purpose, ex. Valid & invalid passwords
3. A good test should be best combination. Test having similar intent, time & resource limitations, execution of some of them uncover the whole class of errors
4. A good test should not be too simple, nor complex. Each test executed separately

14.2 BLACK-BOX & WHITE-BOX TESTING:

Sometimes testing as an afterthought, incomplete test cases, must find errors with minimum effort & time, no of methods, ensure completeness of tests, any engineered product tested in 2 ways :
(1) Black box testing - specific function for which product designed, test each function fully operational, search errors in functions
Tests conducted at i/f, checks functions for given i/p correct o/p produced, maintenance of database, not bother with internal logic
(2) White box testing - internal working of the product, internal operations for performance according to spec, internal component tested for performance
Examines procedures closely, testing of logical paths, executes specific set of conditions &/or loops, status of program checked

Exhaustive testing, no of possible paths very large, limited no of important logical paths exercised

14.3 WHITE-BOX TESTING :

Glass box testing uses structure of procedural design to derive test cases that (1) guarantee all independent paths exercised at least once (2) executrices all logical decision for true & false (3) execute all loops at boundary & within bound (4) exercise internal DS for validity

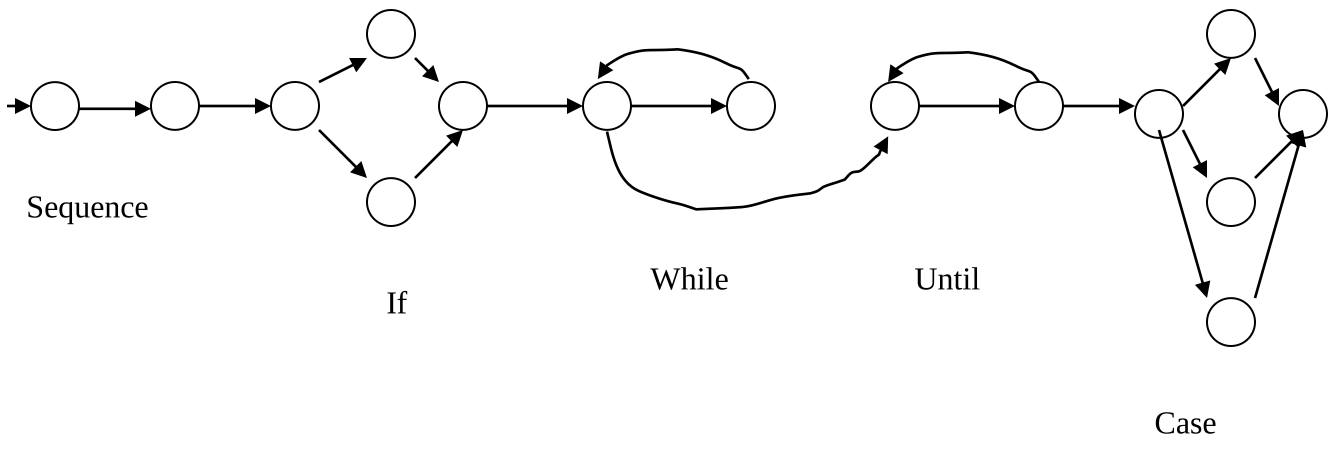
Nature of defects :

- logical errors & incorrect assumptions are inversely proportional to probability that a program path will be executed - errors introduced during design & coding of function, conditions & controls, special case processing fall into cracks
- We often believe that a logical path is not likely to be executed when, in fact, it may be executed on a regular basis – assumption lead to design errors uncovered during testing
- Typographical errors are random – many typing errors uncovered in syntax & spellchecking, uncovered typo lay exist on different paths

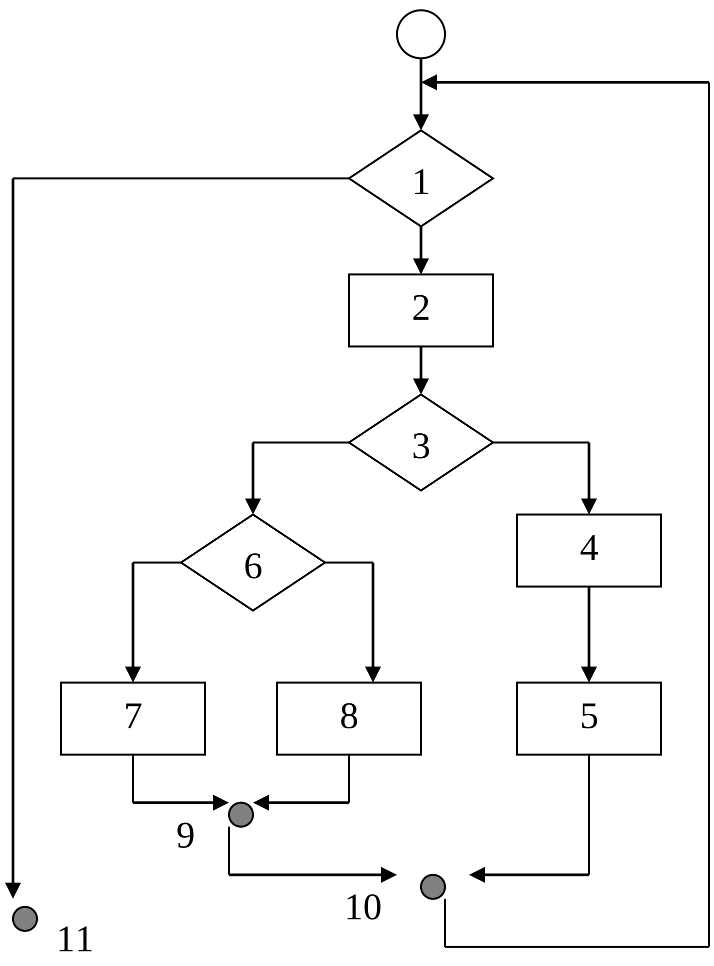
For such errors, white-box testing

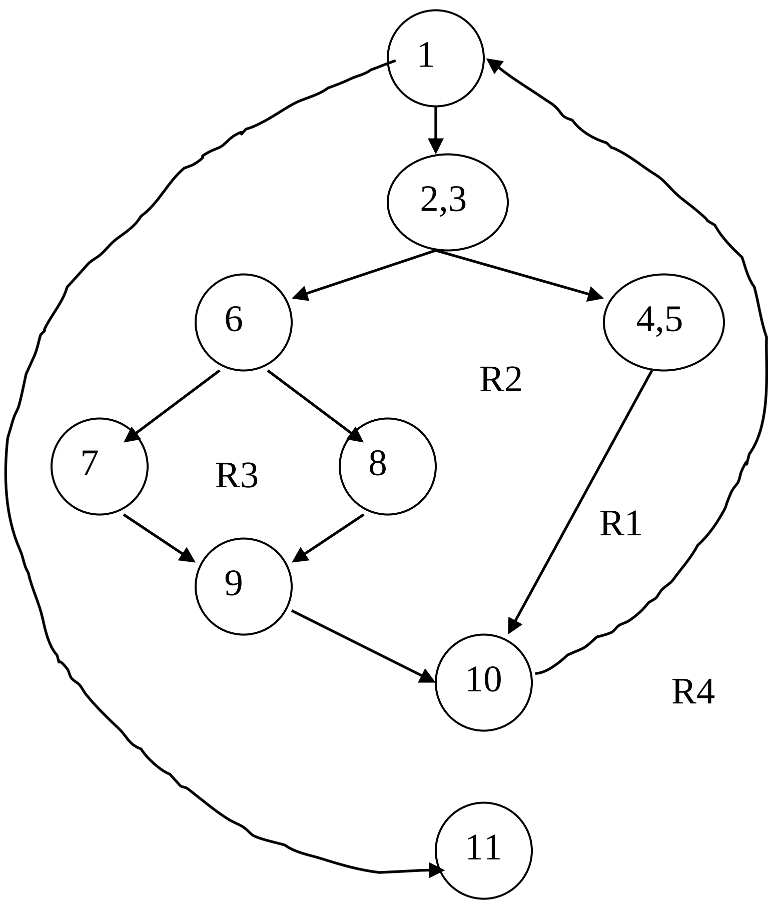
14.4 BASIS PATH TESTING : white-box testing technique, derives logical complexity measure of a procedural design & define basis set of execution path, basis set exercise every statement of program

<u>14.4.1 Flow Graph Notation</u> : simple notation for logical control flow, each structured construct has flow graph symbol



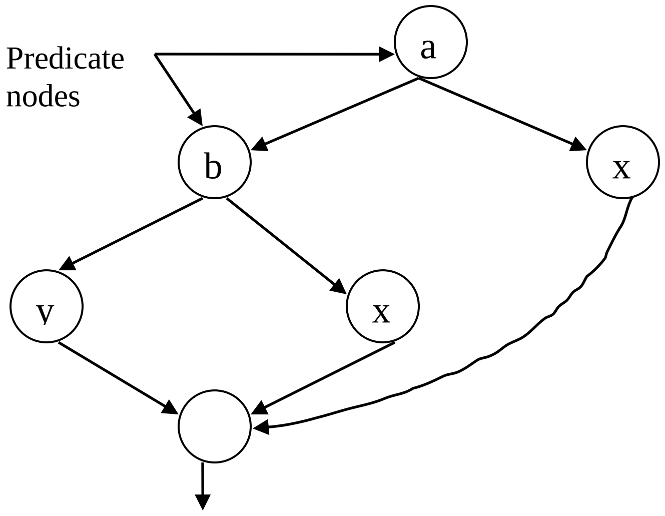
Ex. A flowchart depicts a procedural design, circle as flow graph nodes, represents statements, arrow as edges/links,





Each edge complete at nodes, areas bounded by nodes are region, also include area o/s graph, with compound condition (when not, and, or) flow graph complicated, separate node for each condition, node containing conditions are predicate nodes, two or more edges immerging

IF a OR b
  then procedure x
  else procedure y
ENDIF

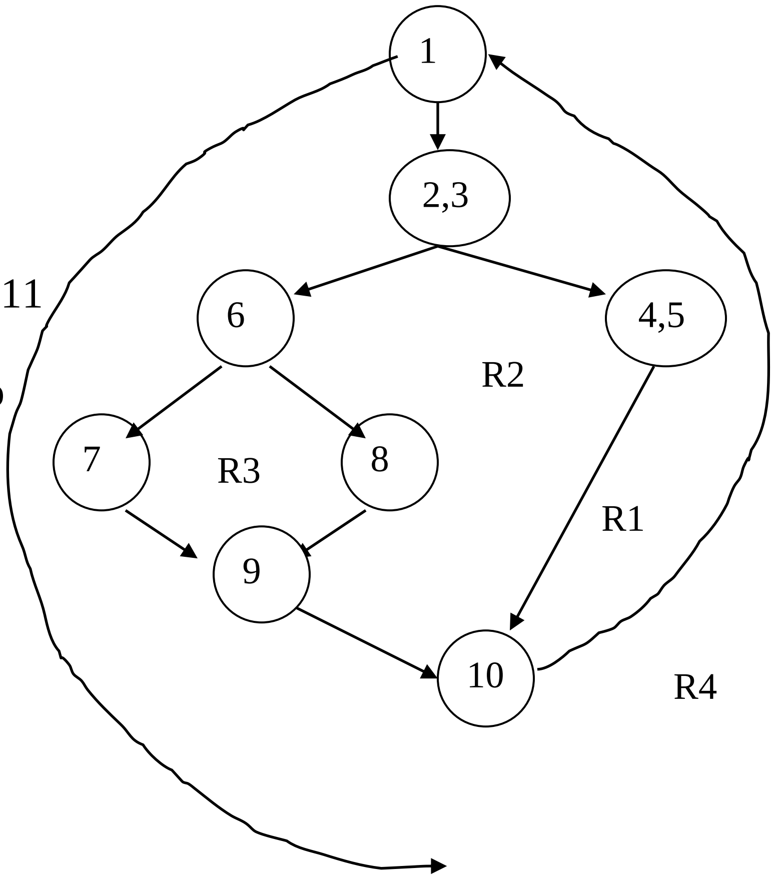


<u>Cyclomatic Complexity</u> : a s/w metric, quantitative measure of logical complexity, used with basis path testing – Cyclomatic complexity defines no of independent paths in basis set of program & gives upper bound of no of tests to be conducted, ensures all statements executed

<u>14.4.2 Independent program paths</u> – path that introduces at least one new set of processing statements or new condition, in flow graph independent path have at least one edge not traversed before, ex the above fig have following independent paths

Path 1 : 1-11
Path 2 : 1-2-3-4-5-10-1-11
Path 3 : 1-2-3-6-8-9-10-1-11
Path 4 : 1-2-3-6-7-9-10-1-11

Path 1-2-3-4-5-10-1-2-3-6-8-9-10-1-11
Not considered as independent path
Combination of specified paths & no
New edge



Path 1,2,3, 7 4 form basis set, test designed for these path, every statement executed, every condition at there true & false side, basis set not unique, computation of Cyclomatic complexity decides no of paths to look for, computed in three ways

1. The no of regions of flow graph correspond to Cyclomatic complexity
2. Cyclomatic complexity V(G) defined as

   V(G) = E – N + 2 E is no of edges, N is no of nodes

3. Cyclomatic complexity V(G)

   V(G) = P + 1 P is no of predicate nodes

Using each method

1. Flow graph has 4 regions
2. V(G) = 11 edges – 9 nodes + 2 = 4
3. V(G) = 3 predicate nodes + 1 = 4

V(G) provides upper bound of independent paths, tests designed to guarantee execution of all statements

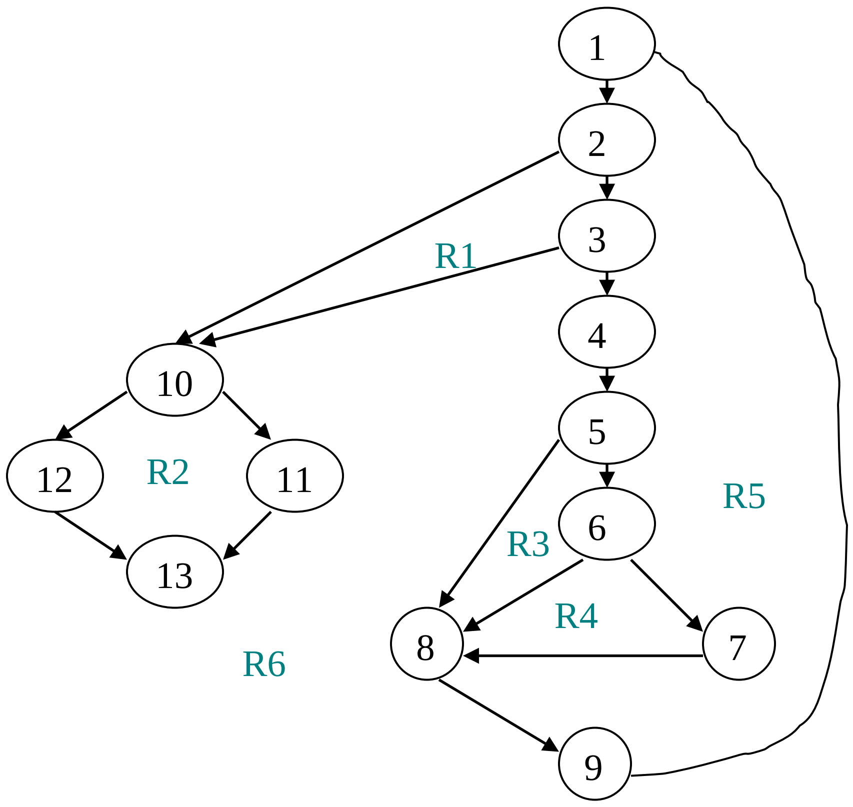
14.4.3 Deriving Test Cases : for procedure Average as an example, contains compound condition & loop

```
PROCEDURE average;

   INTERPHASE RETURENS average, total.input, total.valid;
   INTERPHASE ACCEPTS value, minimum, maximum;

   TYPE value[1:100] IS SCALAR ARRAY;
   TYPE average, total.input, total.valid, minimum, maximum, sum IS SCALAR
   TYPE I IS INTEGER;

   I = 1;
   total.input = total .valid = 0
   sum = 0
   DO WHILE value[i] <> -999 AND total.input < 100
        Increment total.input by 1;
        IF value[i] > minimum AND value [i] <= maximum
            THEN increment total.valid by 1;
            Sum = sum + value[i]
        ELSE skip

        ENDIF
        Increment i by 1;
   ENDDO
   IF total.valid > 0
       THEN average = sum / total.valid;
    Else average = -999

   ENDIF
END average
```

```
PROCEDURE average;

    INTERPHASE RETURENS average, total.input, total.valid;
    INTERPHASE ACCEPTS value, minimum, maximum;

    TYPE value[1:100] IS SCALAR ARRAY;
    TYPE average, total.input, total.valid, minimum, maximum, sum IS SCALAR
    TYPE I IS INTEGER;

    I = 1;
1   total.input = total .valid = 0                  2              3
    sum = 0

    DO WHILE value[i] <> -999 AND total.input < 100          5              6
  4       Increment total.input by 1;

          IF value[i] > minimum AND value [i] <= maximum

              THEN increment total.valid by 1;
   7                  Sum = sum + value[i]
              ELSE skip

  8       ENDIF
          Increment i by 1;
 9
    ENDDO
    IF total.valid > 0   10

 11     THEN average = sum / total.valid;

 12     Else average = -999

13  ENDIF
END average
```

Steps are

1. using design code as a foundation, draw flow graph –



2. Determine Cyclomatic complexity of flow graph
   V(G) = 6 regions
   V(G) = 17 edges – 13 nodes +2 = 6
   V(G) = 5 predicate nodes + 1 = 6

3. Determine basis set of independent paths: from Cyclomatic complexity 6 paths

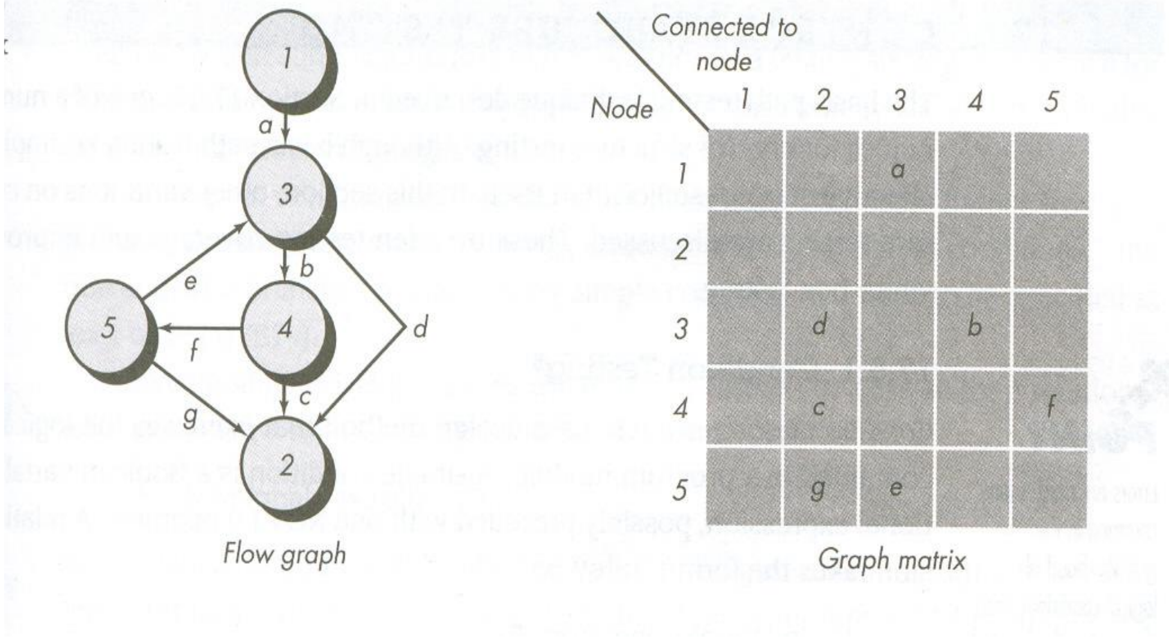
   Path 1 : 1-2-10-11-13
   Path 2 : 1-2-10-12-13
   Path 3 : 1-2-3-10-11-13
   Path 4 : 1-2-3-4-5-8-9-2-…
   Path 5 : 1-2-3-4-5-6-8-9-2-…
   Path 6 : 1-2-3-4-5-6-7-8-9-2-…

4. prepare test cases that will force execution of each path in basis set : data chosen in manner that each path tested
   path 1 test case :
   value (k) = valid i/p where k<i for 2 <=i=100

value(i) = -999 where 2 <=i=100
expected results : correct average based on k values & proper totals
path 2 test case :
value(1) : -999
expected result : average = -999
path 3 test case :
attempt to process 101 or more values, first 100 values valid
expected result : same as test case 1
path 4 test case :
value(i) = valid i/p where i<100
value(k) < minimum where k<i
expected results : correct average of k values & proper totals
path 5 test case :
value (i) = valid i/p where i<100
value k > maximum where k <= i
expected result : correct average based on n values & proper totals
path 6 test case :
value (i) = valid i/p, i<=100
expected results : correct average based on n i/p & proper totals

test case executed & compared with expected results, some independent path not tested in standalone manner, combination of data, as another path

<u>14.4.4 Graph Matrices</u> : to assist basis path, graph matrix a s/w tool, a square matrix, size acc to no of nodes, row & col corresponds to nodes, matrix entries corresponds to connection b/w nodes, ex

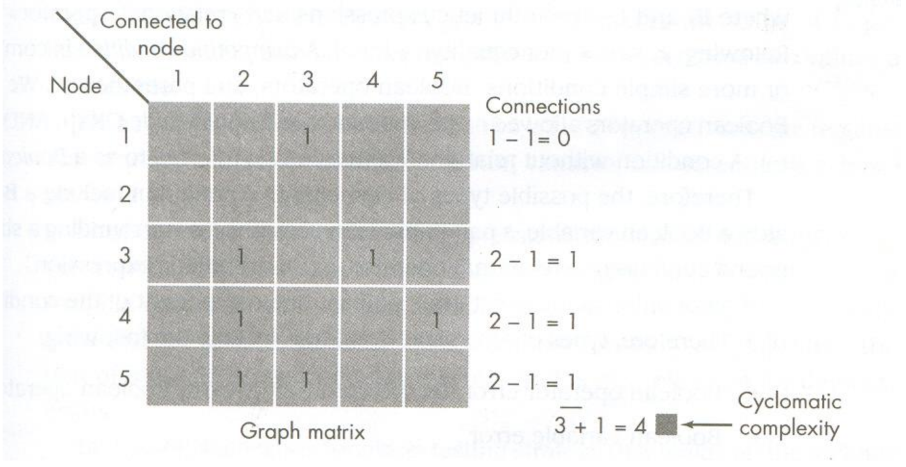


$$\begin{array}{c|ccccc}
\text{Node} & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & & & a & & \\
2 & & & & & \\
3 & & d & & b & \\
4 & & c & & & f \\
5 & & g & e & & \\
\end{array}$$

Adding link weight, lead to evaluate program control structure, link weight either 0 or 1, other properties assigned to link weight are

- prob that link will be executed
- processing time spend traversing a link
- memory required
- resources required

for our ex, matrix given weight for connection, a connection matrix, row containing 2 or more entries predicate nodes, arithmetic performed determines Cyclomatic complexity, other algo exist for graph matrix, design of test cases



$$\begin{array}{c|ccccc|l}
\text{Node} & 1 & 2 & 3 & 4 & 5 & \text{Connections} \\
\hline
1 & & & 1 & & & 1-1=0 \\
2 & & & & & & \\
3 & & 1 & & 1 & & 2-1=1 \\
4 & & 1 & & & 1 & 2-1=1 \\
5 & & 1 & 1 & & & 2-1=1 \\
\end{array}$$

$$3+1=4 \blacksquare \longleftarrow \text{Cyclomatic complexity}$$

14.5 CONTROL STRUCTURE TESTING :

Variation of control structure testing

14.5.1 Condition Testing : exercise logical conditions, generally condition a Boolean var or relational expression, may precede with NOT, relational expression form

E1 <relational-operator> E2

Relational operator are <, >, =, <=, >=, <>

Compound condition contain 2 or more conditions, Boolean operators (AND, OR, NOT) & parenthesis, condition without relational expression is Boolean expression

Elements of condition are Boolean operator, Boolean var, Boolean parenthesis, relational operator, arithmetic expression, type of errors of conditions are

- Boolean operator error (incorrect/missing/extra)
- Boolean var error
- Boolean parenthesis error
- Relational operator error
- Arithmetic expression error

Condition testing test conditions, detect errors in conditions & program as well

Branch testing simple one, for compound condition c, true & false branches of c & every simple condition executed

Domain testing 3 or 4 tests for relational expression of the form

E1 <relational-operator> E2

E1 >, = & < E2tested, if relational-operator incorrect, these three test detect it, to detect errors in E1 & E2, E1 > or < E2 with small difference, for Boolean expression with n var 2^n tests, domain test useful with small n

Error-sensitive tests for Boolean expression, for single expression test case less than 2^n that detect errors, one technique is *Branch & relational operator* , uses condition constraints for condition C, condition constraints is n simple conditions defined as (D1, D2, …, Dn), Di = constraint , an outcome of ith simple condition by executing C, outcome of each condition of C satisfy corresponding constraint D, for a Boolean var B specify a constraint that it must be true or false, for relational expression , symbols <, =, > etc specify constraint on outcome of expression,

Ex C1: B1 & B2 B1, B2 = Boolean var

Form of condition C1 (D1, D2), D1 & D2 = T or F, value (T,F) is condition constraint for C1, test by making B1= T & B2 = F, BRO testing tests (T,T), (F,T) & (T,F), if C1 incorrect due to incorrect Boolean operator, atleast one condition force C1 to fail

Ex C2: B1 & (E3 = E4)

Condition constraint for C2 (D1, D2), D1 = t or f & D2 = <, >, =
Thus constraint set will be (f, =(t, =),), (t, <), (t, >)

Ex C3: (E1 > E2) & (E3 = E4)
Condition constraint (D1, D2), D1, D2 = >, =, <
Constraint set will be (>, ==), (=, =), (<, =), (>, >), (>, <)

<u>14.5.2 Data Flow Testing</u> : selects test paths according to location of definition & use of var, no of strategies, assume each statement assigned unique statement no & each function does not modify parameters & global var for a statement s
DEF(S) = {X / Statement S contains a definition of X}
USE(S) = {X / statement S contains a use of X}

If statement S a if or loop, its DEF set empty & USE set based on condition of S, definition of var X at S is *live* at statement S' if there is a path from S to S' & no other definition of X in between
Definition-use (DU) chain of var X is [X, S, S']

DU testing strategy - Simple data flow testing covers every DU chain, do not guarantee coverage of all branches, for ex, if-then-else, then part has no definition of any var & else do not exist, then else branch not covered , useful for test paths containing if & loop, ex

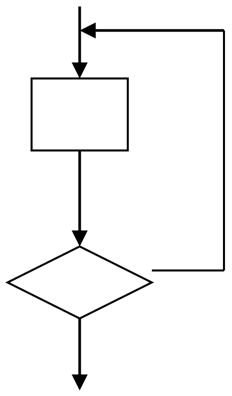
```
Proc x
   B1 :
          Do while C1
                 If C2
                 Then if C4
                               Then B4;
                               Else B5;
                        Endif;
                 Else
                        If C3
                               Then B2;
                               Else B3;
                        Endif;
                 Endif;
          Enddo;
          B6;
End proc;
```

First find out definition & use of var in each condition/block, assume var X defined in the last statement of block B1, B2, B3, B4 & B5 & used in first statement of blocks B2, B3, B4, B5 & B6, DU strategy executes shortest path from each Bi, there are 25 DU chains but only 5 paths will be covered, method effective for error detection

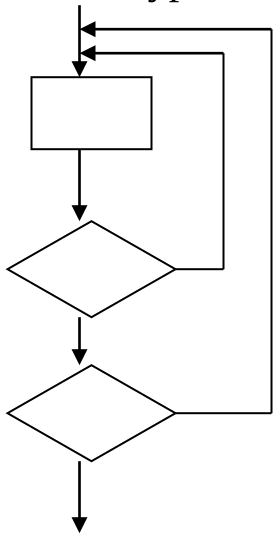
<u>14.5.3 Loop Testing</u> : a white box testing strategy, exclusively tests loops, 4 classes of loops

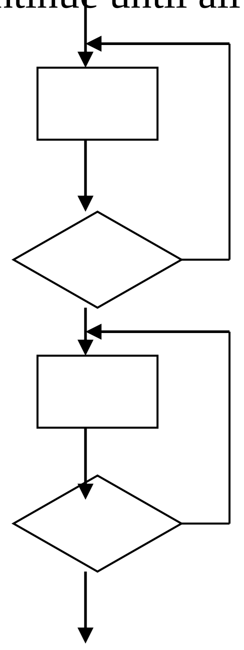
(1) Simple loops – tests applied to simple loops of n iteration are

- skip the loop entirely
- only one pass through loop
- two passes through loop
- m passes through loop
- n-1, n, n+1 passes through loop



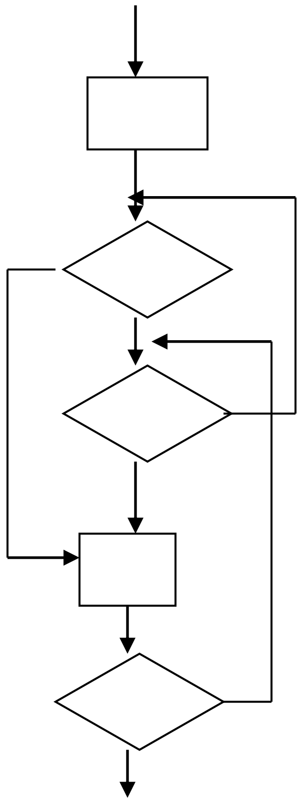
(2) Nested loop – no of test will grow if applied simple loops testing, to reduce no of tests (1) start at the innermost loop, set all other loops to min values (2) conduct simple loop test for innermost loop, outer at their min (3) work outward, conducting test for next loop, outer to min value & others at typical value (4) continue until all loops tested





(3) Concatenated loops : tested using simple loop testing, each loop independent of other

(4) Unstructured loops : use structured programming to structure the loops

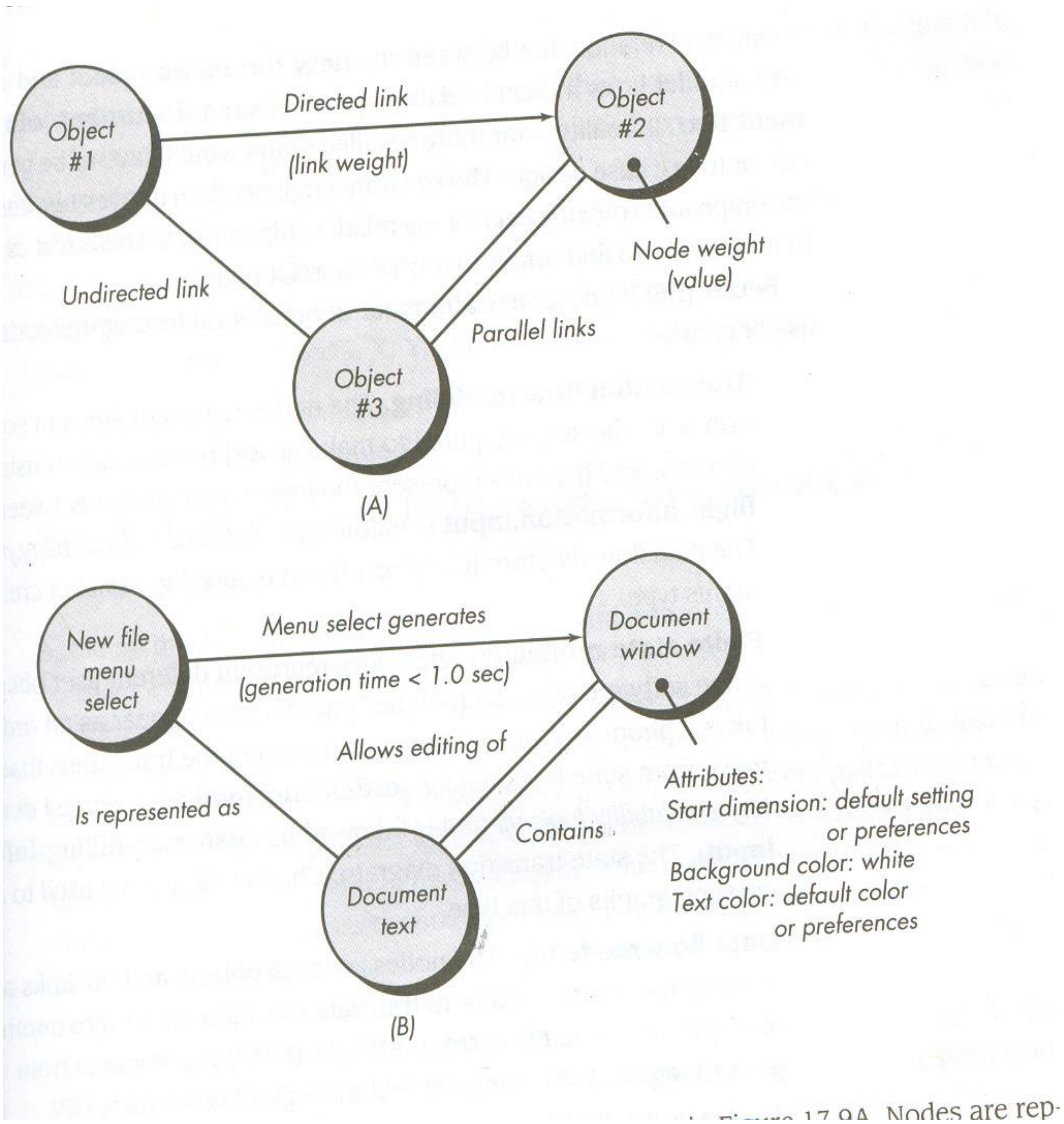


14.6 BLACK BOX TESTING :

Behavioral testing, focus on functional requirement, not an alternative of white-box testing, uncover different class of errors (1) incorrect / missing functions (2) i/f errors (3) errors in DS & database access (4) behavior / performance error (5) initialization & termination errors, applied in later stage, test case satisfy following criteria : (1) reduce no of test cases designed to achieve reasonable testing (2) tells us about presence or absence of classes of errors

14.6.1 Graph-Based Testing Methods :

First understand objects & relationship b/w them, next define series of tests to verify "all objects have expected relationship to one another", a graph of imp objects (as nodes) & their relationship (as links) created, node weight – some properties of node ex data value, link weight – characteristic of link



Directional link – relationship moves in one direction, bidirectional link – relationship in both direction, parallel links when a no of different relationship ex, a word processing application where object #1 = new file menu, object # 2 = document window, object #3 = document text

Menu selection of new file generates document window, node weight are list of properties expected, link weight indicate window generated in less than 1.0

second, undirected link b/w new file menu & document text & parallel links b/w document window & document text, derive test cases by traversing graph & covering each relationship, to find errors in relationships, behavioral testing methods applied to graph
Transaction flow modeling – nodes represent steps in transaction, links logical connection
Finite state modeling – nodes are different states of s/w, links are transition
Data flow modeling – nodes are data object, links transformation of one data object into other
Timing modeling – nodes program object, links sequential connection, specify required execution time

Testing begin with defining nodes & node weight, link & link weights, loops may present, loop testing applied, each relationship studied, transitivity determine, x to compute y, y to compute z => x to compute z
If a link is bi-directional, test, symmetry also checked, UNDO of any application have limited symmetry, after action completes, exceptions where undo not allowed, reflexive relationship – each node has relation with it self, all nodes & links are covered

<u>14.6.2 Equivalence Partitioning</u> : Divide i/p domain in classes of data, uncovers classes of errors & reduce test cases, if a set of objects are related in reflexive, symmetric & transitive manner than equivalence class is present, a set of valid & invalid states of i/p conditions developed, i.p condition may be a numeric value, range of values or Boolean condition, guidelines for defining equi class

- if i/p condition specifies a range, one valid & 2 invalid equivalence classes
- if single value, one valid & 2 invalid
- if a member of a set, one valid & 1 invalid
- if Boolean, 1 valid & 1 invalid

Ex area code – blank or three digit number
I/p condition – Boolean – may or may not present
I/p condition – range – values b/w defined range

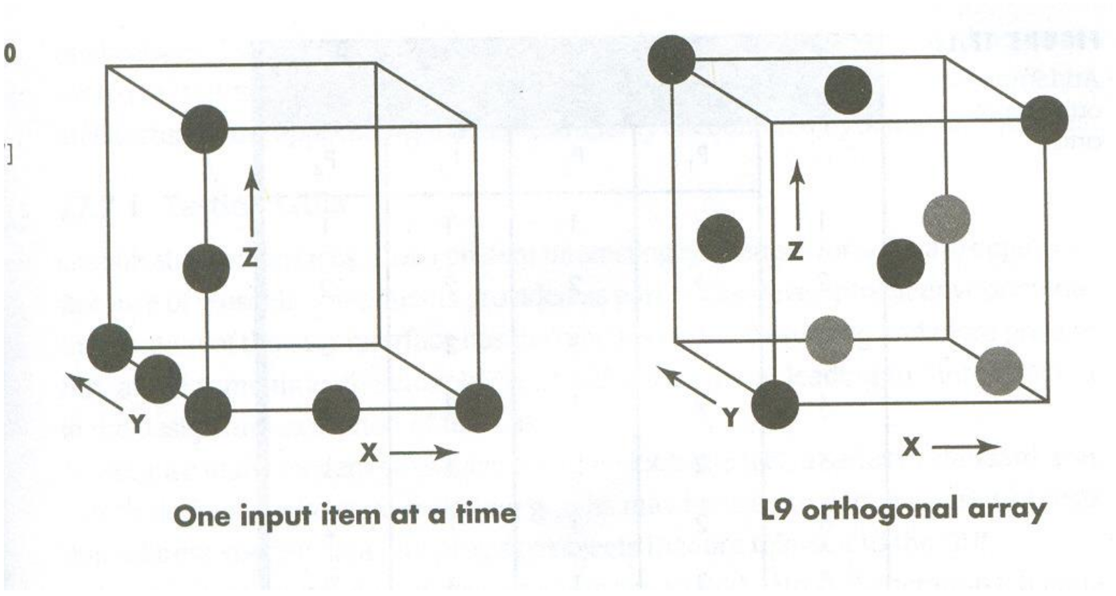
Based on these info test case developed as above

14.6.3 Boundary Value Analysis : greater no of errors at boundaries of i/p domain, exercise boundary values, complement equivalence class, select test cases at the edges of class, also for o/p domain

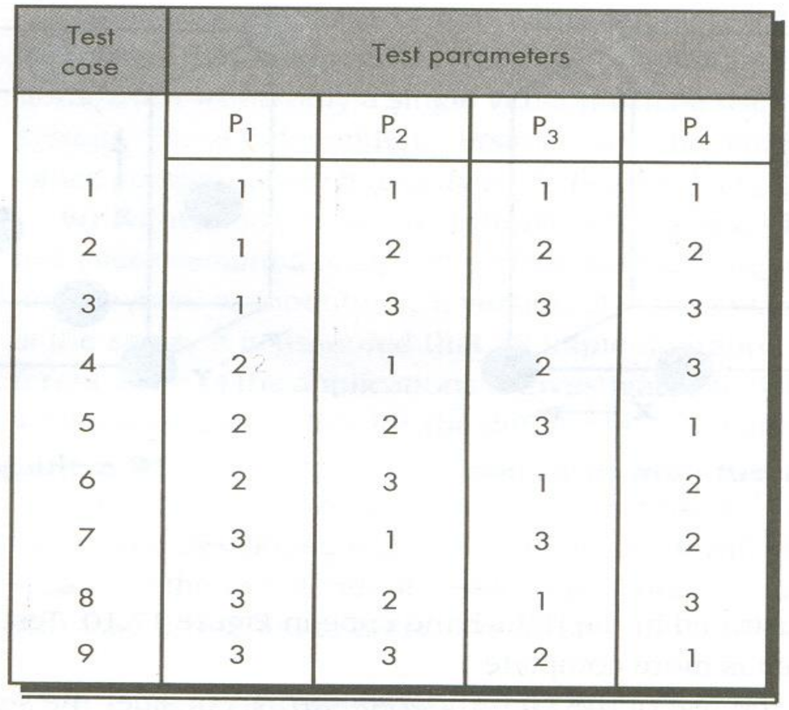
- if i/p specify a range bounded by a & b, test case with values a & b & below & above a & b
- if i/p no of values, test case that exercise min & max no, above & below
- for o/p domain also apply above two
- if DS has some boundary, exercise it to it

14.6.4 Orthogonal Array Testing : if no of i/p limited & bounded, possible to consider all i/p permutation & test i/p domain, if large i/p then testing exhaustive, impractical or impossible

Orthogonal testing finds errors with faulty logic within s/w, if there are 3 parameters having 3 possible values than $3^3 = 27$ different combinations, with orthogonal array test L9 orthogonal array test cases created, dispersed uniformly throughout test domain



For ex four i/p parameters, p1, p2, p3 & p4, p1 takes three values 1,2 & 3
P2, p3 & p4 also takes three values 1, 2 or 3 if permutation then $3^4 = 81$,
Whereas with orthogonal test case reduces to 9 tests



Test case	Test parameters			
	P_1	P_2	P_3	P_4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

-detect & isolate single mode faults, ex p1 = 1 cause errors, by analyzing identify the cause
-detect all double mode faults – problem with two parameter values
- multimode faults

14.7 OBJECT-ORIENTED TESTING METHODS

Tests for errors occur as classes collaborate with one another, tests exercised class operations & examines if errors exist in collaborations, as classes integrated – use-based & fault –based testing, finally use-cases to uncover errors at validation level, conventional test cases as i/p-process-o/p but OO test focus on sequence of operation to exercise state of classes

14.7.1. The Teat Case Design implication of OO Concepts

Attributes & operations are encapsulated, testing inside class, a little difficult to test encapsulation, built-in operations required to repost the state of an object, inheritance add challenge for testing, each new context of reuse requires testing, multiple inheritance complicates testing by increasing no of context, if a subclass used in different context than superclass, a new set of test to be designed

14.7.2 Applicability of Conventional Test Case Design Methods

White-box testing methods – basis path, loop testing, etc ensures that every statement in a class tested, black-box testing methods appropriate for OO system testing

14.7.3Fault-Based Testing

For uncovering faults, test-case exercised code to find possible faults, faults means anything that is likely to go wrong, perceived about fault & tested, integration testing for faults in operation calls 7 message connections, 3 types of faults – unexpected result, wrong operation/ message used, incorrect invocation, integration testing applied to attributes & operations, behavior of object tested through its attributes, integration testing finds errors in client objects not server,

14.7.4 Test Cases & Class Hierarchy

Inheritance complicates testing, ex. A class BASE contains operations *inherited()* & *redefined()* a class DERIVED redefines *redefined()* in local context, DERIVED::redefined() should be tested as it is a new design & new code, here code do not change but the behavior of *redefined()* changes, only a subset of test for DERIVED::redefined(), BASE::redefined() & DERIVED::redefined() are 2 diff operations, each require testing, possible faults are – integration, condition, boundary fault etc, tests for BASE::redefined() applied to DERIVED

14.7.5Scenario-Based Testing

Fault-based testing misses (1) incorrect specifications & (2) interaction among subsystems, incorrect specification is nonconformance to requirements, error in subsystem interaction occurs when behavior of one subsystem leads to failure of other, scenario-based testing focus on what user does not what product does, user tasks captured & applied as test
To uncover interaction errors, test cases must be more complex & realistic, exercise multiple subsystem in a single test, ex scenario-based test for text editor

Use-case : fix the final draft
Background : while reading final draft new errors discovered, events are

1. print entire document
2. move around document, changing certain pages
3. as each page is changed, it's printed
4. sometimes a series of pages printed

user needs (1) method for printing single page (2) method for printing range of pages

14.7.6Testing Surface Structure & Deep Structure

Surface structure are externally observable structures of OO program, obvious to end-user, instead of functions users given objects to manipulate, user scenario collected & unconventional testing
Deep structure refer to internal technical details, testing to exercise dependencies, behavior & communication mechanism, testing with the help of analysis & design model

14.8 TESTING METHODS APPLICABLE AT THE CLASS LEVEL
Testing focuses on a single class & methods

14.8.1 Random Testing for OO Classes
Ex banking application, Account class operations : *open(), setup(), deposit(), withdraw(), balance(), close()* etc, certain constraints like before deposit open an account etc, every permutation of operations performed, min life of an account is – open-setup-deposit-withdraw-close, a variety of operation sequences generated randomly
Teat case 1: open-setup-deposit-deposit-balance-summarize-withdraw-close
Test case 2: open-setup-deposit-withdraw-deposit-balance-creditlimit-withdraw-close

14.8.2 Partition Testing at the Class Level

reduces no of test cases, i/p & o/p are categorized, test case for each category, ex state based partitioning categorizes class operations based on their ability to change state of class, account class state operations are *: deposit()* & *withdraw(),* nonstate operations are *balance(), summarize()* & *creditlimit()* test to change state & do not change state

test case 1: open-setup-deposit-deposit-withdraw-withdraw-close
test case 2: open-setup-deposit-summarize-creditlimit-withdraw-close

Attribute based partitioning categorizes class operations on attributes, account class have balance & creditlimit, 3 partitions (1) operations that use creditlimit (2) operations that modify creditlimit (3) operation that do not use or modify creditlimit

Category based partitioning, generic functions, intial functions – open(), setup(), computational are deposit(), withdraw()

14.9 INTERCLASS TEST CASE DESIGN

Integration of OO system complexes test case, banking example as in fig, class collaboration testing by random & partitioning methods, scenario-based & behavioral testing

fig

14.91. Multiple Class Testing

Steps for generating multiple class random test cases

1. For each client class, use list of class operations to generate random test sequences, operation will send messages to other server classes
2. For each message generated, determine collaborator class & corresponding operation in server
3. For each server object determine messages transmitted
4. For each message, determine next level of op invoked & add into test sequence

14.9.2 Tests Derived from Behavior Models

use state diagram for behavior of a class, fig describe state diagram of class Account, initially empty acct & setup acct state, majority of behavior during working acct state, withdrawal & close cause account class to nonworking acct & dead acct states
fig

14.10 TESTING FOR SPECIALIZED ENV, ARCHI & APPLICATIONS

14.10.1 Testing GUIs – challenging, reusable components, building is less time consuming but complexity of GUI grows, testing difficult to develop & carry, series of std tests, finite state modeling to derive tests for specific data & programs, no of i/ps, use of automated tools

14.10.2 Testing Client Server Applications – distributed nature, performance in transaction processing, no of diff h/w platforms, complex n/w communications, service to multiple clients, centralized database, coordination of communication, all this make testing difficult, more time & cost, testing at 3 diff levels (1) individual client tested in disconnected mode (2) Client & server s/w tested but n/w operations not exercised (3) complete client/server archi including n/w operation & performance, different approaches are

- Application function tests – functionality of client application tested in standalone fashion
- Server tests – coordination & data mgmt functions of server, server performance
- Database tests – accuracy & integrity of data stored by server, transactions posted by clients examined to ensure data properly stored, updated & retrieved
- Transaction tests – tests for each class of transactions processed according to requirements, correctness of processing & performance issues
- N/w communications tests – verify communication among nodes on n/w correctly, message passing, transactions & related n/w traffic without error, n/w security tests

14.10.3 Testing documentation & help facilities – two phases review & inspection of document for clarity & live test in conjunction with actual program, tests for accurate description, accurate examples, consistent terminology, consistent menu descriptions, easy location of guidance, etc

14.10.4 Testing for real-time systems – along with white & black box testing, event handling & timing of data, parallel tasks, data in one state give accurate result, in different state give fault,
Task testing – test each task independently, conventional testing strategy, uncovers errors in logic & function

Behavioral testing – using system model, behavior of real-time system simulated & examined, serve as bases for test case design for behavior of real-time systems
Intertask testing – testing time-related errors, asynchronous tasks communicate with each other are tested with diff data rates & processing load to teat intertask synchronization errors, tasks that communicate via message queue or data store tested for errors in size of data storage areas
System testing – s/w & h/w integrated & system tested for errors at s/w/h/w interface, real-time system deals interrupts, using state diagram & control specifications a list if all possible interrupts & their processing, tests to assess following system characteristics

- interrupt priority properly assigned & handled
- processing of each interrupt
- performance of each interrupt-handling procedure acc to requirements
- high volume of interrupts arriving at critical time creates problem in function & performance

Global data used for transfer info for interrupt processing tested for side effects

15. PRODUCT METRICS

Metrics refer to a broad range of measurement for s/w, to improve s/w process continuously, to assist in estimation, quality control, productivity assessment, project control, to assess quality of technical work product & assist in tactical decision

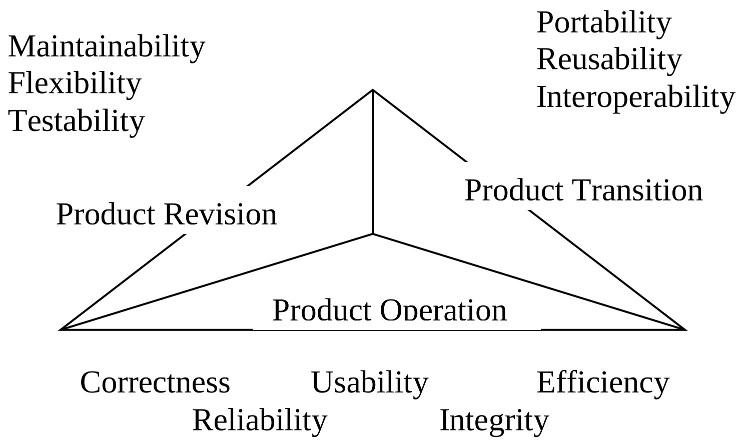
15.1 S/W QUALITY

S/w quality is conformance to explicitly stated functional & performance requirements, explicitly documented development stds, & implicit characteristics of professionally developed s/w, 3 imp points

1. S/w requirements are foundation from which quality is measured, lack of conformance to requirements is lack of quality
2. Specified std define set of development criteria that guides s/w engineering, if criteria not followed, lack of quality
3. Set of implicit requirements, if fail to meet s/w quality is suspect

15.1.1 McCall's Quality Factors

Factors affecting quality categorized in (1) factors directly measured (ex defects uncovered) & (2) factors measured indirectly (ex usability & maintainability) , s/w quality factors focus on 3 aspects of s/w (10 its operational characteristics, (2) ability to undergo change (3) adaptability to new environment, factors of quality are



Correctness : an extent to which s/w satisfies its specification & fulfills customer's mission

Reliability : The extent to which program expected to perform intended function with required precision
Efficiency : The amount of computing resources & code required by a program to perform its function
Integrity : extent of unauthorized access control
Usability : effort required to learn, operate, prepare i/p & interpret o/p
Maintainability : Ease with which a program corrected, adapted & enhanced
Flexibility : effort to modify operational program
Testability : Effort to test program to ensure error free operation
Portability : transfer a program from one h/w or s/w to another
Reusability : Extent to which a program of part of it reused in other applications
Interoperability : Effort to couple one system to another

Difficult to develop direct measures of these quality factors

15.1.2 ISO 9126 Quality Factors

To identify quality attributes for computer s/w, 6 key attributes
Functionality. The degree to which s/w satisfies stated needs, - Suitability, accuracy, interoperability, compliance & security
Reliability. The amt of time s/w is available for use, - maturity, fault tolerance, recoverability
Usability. Degree to which s/w is easy to use, - understandability, learn ability, operability
Efficiency. Degree to which s/w makes optimal use of system resources – time behavior, resource behavior
Maintainability. Ease with which repair made to s/w – analyzability, changeability, stability
Portability. Ease with which s/w transposed from one env to another – adaptability, install ability, conformance, replace ability

15.1.3 The Transition to a Quantitative View

Precise measure of qualitative factors, set of s/w metrics for quantitative measure of quality factors

15.2 A FRAMEWORK FOR PRODUCT METRICS

Primary concern is productivity – a measure of s/w development o/p in effort & time & quality metrics – a measure of fitness for use of the product

15.2.1 MEASURE, METRICS & INDICATOR :

Measure : provide a quantitative indication of the extent, amount, capacity or size of some attributes of a product/process
Measurement : act of determining a measure
Metric : a quantitative measure of the degree to which a product possesses a given attribute

Single data point like no of errors in a single module is a measure, measurement as a result of collection of one/more data points ex. no of modules reviewed for errors, metric relates individual measures ex avg. no of errors per module

S/w engineer collects no of measure develops metrics & indicators obtained, is a single/combination of metrics, provide insight to s/w process, project or product, project manager adjusts project ex. 4 s/w team on a project, metric of errors found per person hour, formal review, 40% more errors, provides an indicator that formal review better

15.2.2 The Challenge of Product Metrics

Researchers attempted to develop a single metric, dozens of complexity measures, each take a different view of complexity, ex attractiveness of a car

15.2.3 Measurement Principles

Metrics provide (1) Assistance in evaluation of analysis & design model (2) Provide indication of complexity of procedural design & source code (3) facilitate design of more effective testing, measurement have 5 activities

- Formulation – The derivation of s/w measures & metrics appropriate for s/w being considered
- Collection – mechanism that accumulate data required to derive formulated metrics
- Analysis – computation of metrics & application of mathematical tools
- Interpretation – evaluation of metrics to gain insight about quality

- Feedback – recommendations from interpretation of metrics transmitted to s/w team

Before using s/w metrics characterization & validation, principles to be followed are

- A metric should have desirable mathematical property – metric's value within meaningful range
- When a metric represents a s/w characteristics that increases when positive trait occur or decreases when undesirable traits occur are encountered, the value of metrics should increase or decrease
- Each metrics should be validated empirically in a wide variety of contexts before using to make decisions- measure factor of interest independently of other factors

Formulation, characterization & validation of metrics should follow (1) whenever possible, data collection & analysis should be automated (2) valid statistical techniques applied to establish relationship b/w internal product attributes & external quality characteristics (3) interpretative guidelines & recommendation established for each metric

15.2.4 Goal-Oriented S/w Measurement

Goal/Question/metric (GQM) paradigm developed to identify meaningful metrics for s/w process, emphasize on (1) establish an explicit measurement goal specific to process activity or product being assessed (2) define a set of questions that are answered to achieve the goal (3) identify well-formulated metrics that help answer the questions

A goal definition template to define each measurement goal

Analyze [the name of the activity or attribute to be measured] **for the purpose of** [the overall objective of the analysis**] with respect to** [the aspect of the activity or attribute that is considered] **from the viewpoint of** [the people who have an interest in the measurement] **in the context of** [the env in which measurement takes place]

For ex template for safehome security s/w

Analyze the Safehome Software Architecture **for the purpose of** evaluating architectural components **with respect to** the ability to make Safe-Home more extensible **from the viewpoint of** the s/w engineers performing work **in the context of** product enhancement over the next three years

After explicitly defining goal, a set of questions developed answers help s/w team to determine whether measurement goal achieved or not ex for safe-home ex questions might be
Q1: Are archi compo characterized in manner that compartmentalizes functions & related data?
Q2: Is complexity of each compo within bound that will facilitate modifications & extension?
Each question answered quantitatively, for first answer metric indicating cohesion of archi is useful, for second answer Cyclomatic complexity

15.2.5 The Attributes of Effective S/w Metrics

Many metrics, not all practical, some demand complex measurements, others are not graspable, others violate basic intuition of quality, attributes for effective s/w metrics are

- Simple & computable – easy to learn& computation not require inordinate effort or time
- Empirically & intuitively persuasive – satisfy engineer's notion about product attribute under consideration
- Consistent & objective – yield unambiguous results
- Consistent in use of units & dimensions- no bizarre combination of units
- Programming language independent – based on analysis or design model or structure of program
- Effective mechanism for high-quality feedback – lead to higher quality product

15.2.6 The Product Metrics landscape

Metrics for Analysis Model – address various aspects of analysis model
Functionality Delivered – provides indirect measure of functionality within s/w
System Size – measure overall size of system from info available in analysis model
Specification quality - indication of specificity & completeness of requirements specifications
system from info available in analysis model
Specification quality - indication of specificity & completeness of requirements specifications

Metrics of one model may be used later in s/w engineering activities

15.3 METRICS FOR THE ANALYSIS MODEL

These metrics examine analysis model for predicting size of resultant system, size indicates design complexity, coding, integration & testing efforts

15.3.1 Function-based Metrics

Measures functionality delivered by the application as normalization value, FP used to (1) estimate cost & effort (2) predict no of errors during testing (3) forecast no of compo & no of source lines, functionality is not a direct measure, derived indirectly from other direct measures, function point derived from direct measurement of s/w information domain & complexity assessment, computation done by filling table

			Weighting factor				
Measurement parameter	count		simple	average	complex		
No. of user i/p	☐	X	3	4	6	=	☐
No. of user o/p	☐	X	4	5	7	=	☐
No. of inquiries	☐	X	3	4	6	=	☐
No. of files	☐	X	7	10	15	=	☐
No. of external i/f	☐	X	5	7	10	=	☐
Count = total --							☐

Five info domain characteristics are determined & placed in table
No of external i/p : user or other application i/p that provide application oriented data to s/w, different that inquiries
No of external o/p : user o/p that provide application oriented info, reports, screen, error messages
No of external inquiries : online i/p that result in immediate s/w response as on-line o/p
No. of internal logical files : logical grouping of data, reside within boundary, maintained via external i/p

No. of external interface files : a logical grouping of data residing externally of application ex tapes, disks, devices that used to transmit info to system counted

Complexity value is associated with each count, simple, avg. or complex entries (criteria developed by organization) FP is calculated :

FP = count-total * [0.65 + 0.01 * Σfi]

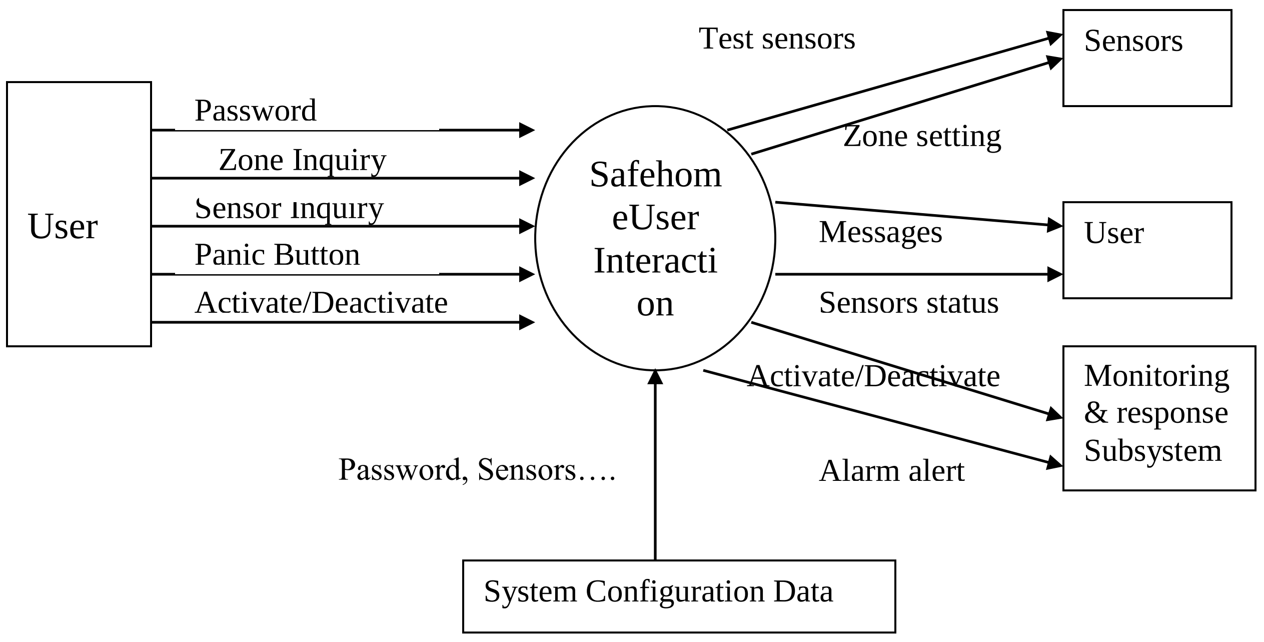
Fi (I=1 to 14) = complexity adjustment values derived from questions noted below.

0	1	2	3	4	5
No influence	Incidental	Moderate	Average	Significant	Essential

Fi :

1. Does system require reliable backup & recovery?
2. Is data communication required?
3. Are there distributed processing functions?
4. Is performance critical?
5. Heavily utilized operational environment
6. On-line data entry
7. Multiple screen
8. On-line master file updating
9. Complex i/p, o/p, files & inquiries
10. Complex internal processing
11. Reusable code
12. Conversion & installation included in design
13. Multiple installation in diff org
14. Change facility for user

For ex consider analysis model of safehome s/w, function manages user interaction, accept user password to activate or deactivate system, allows inquiry for status of sensors, function displays messages & sends signals to various compo of security system, DFD



3 external i/ps – password, panic button & activate/deactivate
2 external inquiries – zone inquiry & sensor inquiry
1 internal logical file – system configuration file
2 external o/p – message & sensor status
4 external i/f files – test sensor, zone setting, activate/deactivate & alarm alert

Measurement parameter			count	simple	average	complex	
No. of user i/p	3	X	3	4	6	=	9
No. of user o/p	2	X	4	5	7	=	8
No. of inquiries	2	X	3	4	6	=	6
No. of files	1	X	7	10	15	=	7
No. of external i/f	4	X	5	7	10	=	20
Count_ total ---							50

FP = count_ total * [0.65 + 0.01 * ∑(Fi)]

Fi(i= 1 to 14) are value adjustment factors, for this example ∑(Fi)=46, a moderately complex system

FP = 50 * [0.65 + 0.01 * 46] = 56

Based on this FP, project team estimates project size based on past data, suppose 1 FP translates into 60 lines of code & 12 FP produced by each person-month efforts, such data helps project team members

FP obtained is used to normalize s/w productivity, quality etc

- errors/FP
- defect/FP
- cost/FP
- pages of docu/FP
- FP/person-month

15.3.2 Metrics for Specification Quality

A list of characteristics to assess quality of analysis model & requirements specification – specificity, completeness, correctness, correctness, traceability, modifiability, consistency, achievevability, understandability, verifiability, precision, reusability etc, all qualitative in nature, each represented using metrics, ex assume n_r requirements in specification

$$n_r = n_f + n_{nf}$$

n_f – no of functional requirements
n_{nf} – no of nonfunctional requirements

To determine specificity of requirements, a metric based on consistency

$$Q_1 = n_{ui} / n_r$$

n_{ui} – no of requirements

MANAGING SOFTWARE PROJECTS

21. PROJECT MANAGEMENT CONCEPTS

21.1 THE MANAGEMENT SPECTRUM

S/w project management focuses on people, product, process & project

21.1.1 People : Highly skilled s/w people, an important requirement, PM-CMM to "enhance the readiness of s/w org to undertake complex application by attracting, grow, motivate, deploy & retain talent, PM-CMM defines key practicing area – recruiting, selection, recruiting, performance management, training, compensation, career development, organization

21.1.2 Product : before project planning, product objective & scope be established, consider alternate solutions, identify technical & management constraints, without this impossible to estimate cost, assess risk, breakdown project tasks & schedule

Objective states overall goal for product from customer viewpoint & scope identifies primary data, function & behavior of the product and binding them. After that alternate solution to be considered for best approach

21.1.3 Process : provide a framework, plan of s/w development, no. of tasks & umbrella activities

21.1.4 Project : planned & controlled projects to manage complexity, in 1998 26% failed & 46% experienced cost & schedule overrun, to avoid failure s/w project manager & engineer must avoid common warning signs, understand critical success factor, develop an approach for planning, monitoring & controlling project

21.2 PEOPLE

S/w development team, important contributor, a primary asset, participants & their way for effective SE

21.2.1 *The Stakeholders* : five categories

1. Senior manager : defines business issues, significant influence on project
2. Project manager : plans, motivates, organize & control developers
3. Practitioners : technical skills
4. Customers : specify requirements
5. End users : interact with s/w

Project team must be organized to maximize skills & ability of each person, a job of team leader

21.2.2 *Team leaders* : Competent developer a poor team leader, characteristics are

1. Motivation : encourage technical people, produce at best ability
2. Organization : ability to mold existing process, initial concept to final product
3. Ideas/ innovation : ability to encourage people to create & feel creative

Successful project leader apply problem solving mgmt, should concentrate & understand the problem, manage the flow of ideas & control quality

For effective project manager following characteristics are :

1. Problem solving : can diagnose technical & organizational issues, structure a solution, motivate others to develop solution, past experience, flexible to change directions if initial attempts fails
2. Managerial identity : take charge of project, confident to assume control, & assurance to technical people
3. Achievement : to optimize productivity of a team, reward initiative accomplishments
4. Influence & team building : read people, understand signals & react to needs of people, remain under control in high stress conditions

21.2.3 *The s/w team* : best team structure depends on mgmt style, no of people in the team & their skill level & problem difficulty :

Factors to be considered while organizing SE team are :

- Difficulty of the problem
- Size of the resultant program
- Time the team will stay together
- Degree of modularization
- Required quality & reliability
- Rigidity of delivery date

- Degree of communicate required

Classification of team structure :

- Closed paradigm : traditional hierarchy of authority (CC), well for projects similar to past projects, less innovative
- Random paradigm : loose structure, depends on individual initiative, best in innovative & technical breakthrough, not good for orderly performance
- Open paradigm : control of closed paradigm & innovation of random paradigm, work performed collaboratively with heavy communication, best for complex problems
- Synchronous paradigm : rely on modularity of problem, team members work on different modules, little communication

Very first s/w team CC, senior engineer in nucleus, plan, coordinate & review technical activities, technical staff, analysis & development, backup engineer to support senior engineer, a s/w librarian, maintains & control elements of s/w configuration, documentation, source listing, catalogue & index reusable s/w modules
Democratic s/w team has appeal, creative energy must be converted to high performance, to achieve high performance team

1. team leaders must have trust in one another
2. distribution of skills must be appropriate to proble
3. high cohesiveness

Objective of project manager, create a team that exhibit cohesiveness, regardless of team organization, jelled teams share common goal & common culture, But many team suffer team toxicity, the factors playing are

- Frenzied work atmosphere, team members waste enrgy, lose focus on objectives
- High frustration caused by personal, business or technological factors, causing friction among members
- Poorly coordinated procedures, improper process model, becomes road block
- Unclear definition of role resulting in lack of accountability, leads to finger pointing
- Continuous & repeated failure, leads to loss of confidence & low morale

Antitoxins :

To avoid frenzied env, team must have access to all info, major goals & objectives should not be modified unless absolutely necessary, bad news delivered to team earliest
When unable to control the situation, frustration in s/w people, can avoid frustration by giving responsibility for decision making, more control over process & technical decision
Chose appropriate s/w process by (1) being certain that characteristics of s/w be addressed by process chosen (2) allowing team to select process
Project manager should clearly define roles & responsibilities before project, team must establish mechanism for accountability & define steps for corrective approach when failure occurs
Failure occurs, Key to avoid atmosphere of failure is feedback & problem solving, failure of a member-failure of a team, corrective action

21.2.3 Agile Teams
Agile philosophy encourage customer satisfaction & early incremental delivery of s/w, small highly motivated project tams, informal methods, minimal SE work products & development simplicity
Team adopts characteristics of successful SE team & avoids toxins that create problems, stress on individual competence & group collaboration, agile teams are self-organizing, significant autonomy for project mgmt & technical decisions, minimal planning, as project proceeds focus on individual competency to benefit the project, brief daily meetings to coordinate & synchronize work, based on info in meeting team adapts approach for increment of work

21.2.5 Coordination & Communication Issue :

Large scale development leads to complexity, confusion & difficulties in coordinating team members, continuous changes create uncertainty, interoperability b/w many systems leads communications b/w s/w. scale, uncertainty & interoperability the characteristic of modern s/w, to deal with them – methods to coordinate people effectively, formal & informal communication b/w team members & b/w multiple teams, formal communication through writing, structured meetings, impersonal communication channel, informal communication more personal, SE team share ideas, ask for help & interact

21.3 THE PRODUCT :

Project manager would like to estimate the project quantitatively & plan but information unavailable, requirement analysis provide info but time consuming & changing requirements, product & problem must be examination at the very beginning & scope to be established & bounded

21.3.1 S/W SCOPE : first project management activity, defined as

Context : how s/w fit in larger system, constraints

Information objective : customer visible data objects are to be produced as o/p & what data objects required for i/p

Function & performance : functions required to transform i/p to o/p, special performance characteristics

Scope must be clear & understandable at the management & technical levels, bounded scope - quantitative data such as no of users, size of mailing list, response time etc explicitly stated, constraints & limitation must be noted

21.3.2 PROBLEM DECOMPOSITION : core activity of s/w requirement analysis, two major areas (1) the functionality to be delivered (2) process used to deliver it

Complex problem partitioned into smaller problem which is more manageable, applies as project planning begins, s/w functions to be evaluated & refined before estimation, affects cost & schedule

21.4 THE PROCESS :

Characteristics of s/w process – definition, development & support applied to all s/w., problem is to select appropriate model, project manager decide on appropriate process model depending on (1) customer requirement & end user (2) characteristic of product & (3) project env, after selecting process model team defines preliminary project plan, process decomposition begins & complete plan for every task

21.4.1 Melding The Product With The Process : melding of product with process as planning begins, each function to be developed must pass thro set of framework activities defined for organization, for ex organization has selected Customer communication, planning, risk analysis, engineering, construction & release, customer evaluation as their framework activities

Each framework activity applied to each function, a matrix for major functions as column & activities in row, SE work tasks are entered in the cells of matrix, project manager then estimate resources for each cell, start & end dates for tasks & work product

Common process framework activities	Customer communication			Planning				Modelling					Construction							Deployment
S/w engineering tasks																				
Product Functions																				
Text i/p																				
Editing & formatting																				
Auomatic copy edi																				
Page layout capabi																				
Automatic indexin																				
File management																				
Document production																				

21.4.2 PROCESS DECOMPOSITION : s/w team flexible to choose process model best for project, small project similar to past one best using linear model, strict time constraints & modular then RAD model, tight deadline & do not require full functionality then incremental model

After selection process model, adaptation of common process framework, Process decomposition required, for a simple project customer communication activity tasks are

1. Develop list of clarification issue

2. Meet customer to clarify
3. Develop statement of scope
4. Review scope
5. Modify scope

Complex project require long list of tasks

1. Review customer request
2. Plan & schedule formal meeting
3. Conduct research to specify proposed solution& existing approaches
4. Prepare working document & agenda for formal meeting
5. Conduct meeting
6. Jointly develop mini specs for data function & behavior
7. Review mini spec for correctness & consistency
8. Assemble mini spec in a scooping document
9. Review document
10. Modify if required

21.5 THE PROJECT :

To manage a project one must know what can go wrong & how to right it, signs that indicates that project is in jeopardy

1. S/w people unable to understand customer needs
2. Poorly defined product scope
3. Poorly managed changes
4. Changes in technology chosen
5. Change in business needs
6. Unrealistic deadlines
7. Users resistant to new product
8. Sponsorship is lost
9. Lack of appropriate skills
10. Avoidance of best practice & past experience

A 90 –90 rule to s/w development project, first 90% development absorbs 90% of efforts & time, other 10 % takes another 90%, approaches to avoid problems are

1. Start on right foot : work hard to understand the problem, set realistic objects & expectations, build right team, give autonomy, authority & technology

2. Maintain momentum : a good start slowly disintegrate, to maintain momentum, incentives to be paid, emphasize on quality, mgmt stay out of team’s way
3. Track progress : track progress as work products are produced & approved, use of s/w process & project measures to assess progress
4. Make smart decisions : decisions must be simple, use of s/w components, identify & avoid obvious risks, allocate more time to complex tasks
5. conduct postmortem analysis : a consistent mechanism for extracting lessons learned from each project, evaluate planned & actual schedule, collect & analyze s/w project metrics, feedback from team members & customer

21.6 THE W^5HH PRINCIPLE

Boehm suggested approach to address project objectives, milestones & schedules, responsibility, mgmt & technical approach & sources, WWWWWHH is a series of questions that lead to definition of project characteristics & project plan

1. Why system is being developed? : assess validity of reasons for s/w
2. What will be done : establishes task set required for project
3. By when? : establish project schedule
4. Who is responsible for a function? : role & responsibility of each member in s/w team
5. Where are they located in the organization? : responsibility of other persons
6. How will the jobs done technically & managerially? : mgmt & technical strategy
7. How much of each resource needed? : estimation of resources

S/w project mgmt is an umbrella activity, begins before technical activity, continues throughout definition, development & support

22. METRICS FOR PROCESS AND PROJECTS

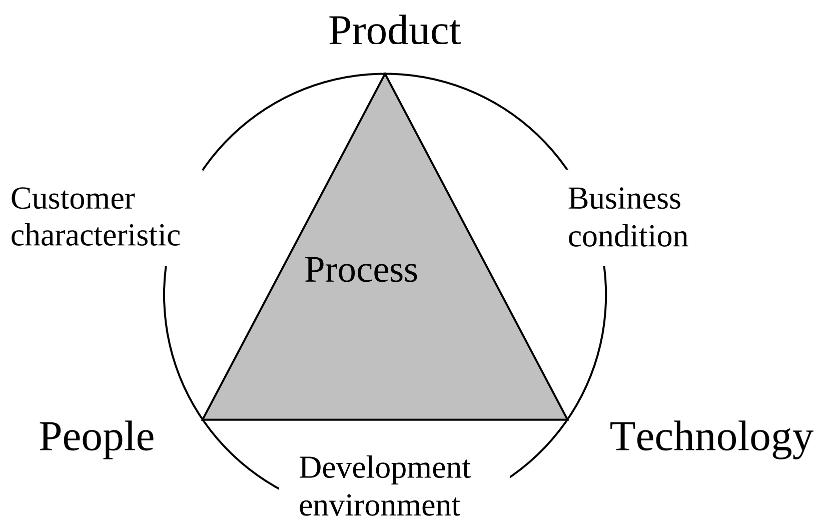
In engineering measures like weight, power consumption, temperature etc, in s/w what to measure & how to evaluate, Metrics must be collected & process & project indicator gained, process indicator enable org to gain insight of efficacy of existing process, collected across all projects, project indicators enable project manager to assess status, track potential risk, uncover problem areas, adjust work flow, evaluate team for quality

22.1 METRICS IN THE PROCESS & PROJECT DOMAIN

Process metrics collected for all projects & for long period of time, set process indicator that lead to process improvement, project metrics enable project manager to (1) assess status of ongoing project (2) track potential risks (3) uncover problem areas(4) adjust work flow (5) evaluate quality control of work products

22.1.1 PROCESS METRICS & S/W PROCESS IMPROVEMENT :

Process is one of the controllable factor to improve s/w quality & org performance, process resides in the center of a triangle connecting three activities having influence on s/w quality & org performance, skill & motivation of people, complexity of product & technology used



Process triangle within circle of env conditions, includes develop env (case tools), business conditions (dead lines, rules), customer characteristics (easy of communication)

Indirect measure efficiency of process, matrices based on outcome of process such as errors, defects, work products delivered, human effort, time, schedule conformance, Also measure characteristics of SE tasks ex effort & time spent performing umbrella activities

Private & public use of process data, s/w eng sensitive to private data, must used as an indicator for individual, private metrics are defect rate, errors found, such data helps to improve individual, some process metrics private to project team but public to team members, defects per major s/w function, technical review, provide indicator to improve team performance, private metrics combines to form team data & becomes public data ex project level defect rate, effort etc, helps improve process maturity level, not to be misused

- Common sense in interpretation
- Regular feedback to teams
- Do not use to upraise individual
- Set clear goals & metrics
- Not to threaten
- Not to be considered –ve
- Do not rely on single metric

More adept org uses SSPI, s/w failure analysis by collecting info about all errors & defects

22.1.2 Project Metrics

S/w process metrics used for strategic purpose, project metrics for tactical purpose, project metrics & indicators used by project manager & team to adapt project workflow & technical activities, First application of project metrics for estimation, metrics from past projects to find effort & time estimates, as project proceeds effort & time expended compared to estimates, data used to monitor & control project, metrics such as production rate, review hours, FP, delivered source lines, errors uncovered, technical metrics to assess design quality

Project metrics used to minimize development schedule by making adjustments, mitigate potential problems & risks and also used to assess product quality, & to improve quality, a quality improves defects minimized,
Leads to educed amount of rework, leads to reduction in project cost

A fishbone diagram to diagnose, spine represents quality factor under consideration, each rib indicate potential cause for quality problem, diagram is extended upon cause noted

22.2 S/W MEASUREMENT :

2 types of measurements - Direct (length) & indirect (quality) measures, direct measure of a se process is cost & efforts applied, direct measure for product LOC, execution speed, memory size, defects reported, indirect product measures are functionality, quality, complexity, efficiency, reliability, maintainability etc, direct measures are easy to collect, indirect measures are difficult to assess & measured indirectly

Project metrics consolidated to develop process metrics, problem is to combine metrics from different individuals & projects ex. Two teams on different projects, record & categorize all errors individually, combined to develop team metrics, 1st team found 342 errors, 2nd 184 errors, which team is more effective, depend on size & complexity of problem

22.2.1 SIZE ORIENTED METRICS :

Co

Size of the s/w produced, a table of size oriented measure

Project	LOC	Effort	Cost (000)	P. of doc	Errors	Defects	People
Alpha	12,100	24	168	365	134	29	3
Beta	27,200	62	440	1224	321	86	5
Gamma	20,200	43	314	1050	256	64	6

Lists s/w development project completed & measures, for project alpha, to develop metric, lines of code used as normalization value, different size-oriented metrics developed are :

- errors/ KLOC
- defects/KLOC
- cost/KLOC
- Pages of documentation / KLOC

Others are errors/person-month, LOC/person-month, cost/page of documentation

Not accepted as best measure of s/w development process, LOC are programming language dependent

22.2.2 FUNCTIONS ORIENTED METRICS :

Measures functionality delivered, widely used is FP, computation on characteristics of s/w info domain & complexity, controversial proponents claim FP language independent, opponents claim method require subjective data & no direct physical meaning

22.2.3 RECONCILING DIFFERENT METRICS APPROCHES :

Relationship b/w LOC & FP depend on programming language & quality of design, rough estimation b/w avg. no of code required to build one FP

Programming Language	LOC/FP
Assembly	320
C	128
Cobol	105
Fortran	105
Pascal	90
Ada	70
Oo lang	30
4 GL	20
Code generators	15
Spreadsheets	6

1 LOC of 4GL, 3-5 times more functionality, conventional prog lang, metrics LOC/p-m, FP/p-m, not to compare b/w groups, many factors influence productivity, must be considered
people factor : size & expertise
problem factor : complexity & constraints
Process factor : techniques used, CASE tools
product factor : reliability & performance
resource factor : available tools, h/w & s/w

increase in one productivity factor, increases s/w development productivity

22.2.4 OBJECT-ORIENTED METRICS

conventional s/w metrics for OO projects, but not enough for incremental or evolutionary process, sets of metrics for OO projects :
No of Scenario Scripts – scenario script is a detailed sequence of steps, describe interaction b/w user & application, no of scenario script directly correlated to size of projects & no of test cases
No of Key Classes – Highly independent compo defined early in OO analysis, are central to problem, no of key classes indicate amount of effort required to develop s/w, also amount of reuse
No of Support Classes – required to implement system but not immediately related to problem, ex UI classes, database access, support classes for each key class, no of support class indicate effort & reuse
Avg No of Support Classes per Key Class – estimation is made easy, 2-3 support classes for each key class in GUI application, 1-2 support classes for non-GUI
No of Subsystem – an aggregation of classes that support a function visible to end-user, identification of subsystem makes scheduling easy

22.2.5 Use-Case Oriented Metrics
Use-case applied as normalization measure, use-cases designed early in process, makes estimation easy, use-cases define user visible functions & features that are basic to system, independent of programming lang, no of use-cases directly proportional to no of LOC & test cases

22.2.6 Web Engineering Project Metrics

22.3 METRICS FOR S/W QUALITY

Goal of s/w eng to produce high quality product, to achieve goal s/w eng must apply effective method along with modern tools, quality depends on requirements specification, design, coding & testing, s/w eng should assess quality of analysis & design model, source code & test cases, project manager evaluate quality as project progresses, collection of private metrics converted in project metrics, measurement of errors & defects at project level, compute DRE for each activity

22.3.1 MEASURING QUALITY :

To measure s/w quality 4 measure as useful indicators
Correctness : program must operate correctly otherwise useless to users, correctness is a degree to which s/w performs required function, measure is defects/KLOC, a defect is lack of conformance
Maintainability : require more effort than any other se activity, ease with which a program corrected, adapted & enhanced, not a direct measure, a time-oriented metrics mean-time-to-change - time to analyze change, design change & implement change, low MMTC are more maintainable, cost oriented metric – Spoilage - calculates cost to correct defects, ratio of spoilage to project cost is plotted to decide improvement
Integrity : prevent hackers & viruses, an ability to withstand attacks on security, attacks can be on programs/data/documents, to measure integrity 2 attributes, threat - probability of an attack of specific type within given time, security – prob attack of specific type repelled, estimated from past data, integrity defined as

$$\text{Integrity} = \Sigma [(1-\text{threat}) * (1 - \text{security})]$$

Usability : user-friendly, to quantify user friendliness usability is measured, 4 characteristics (1) physical & intellectual skill required to learn system(2) time required to become moderately efficient(3) increase in productivity is the system is used (4) assessment of user attitude towards system

22.3.2 DEFECT REMOVAL EFFICIENCY :

A quality metric provide filtering ability for QA activity

$$DRE = E / (E + D)$$

E = errors found
D = defects found

Ideally DRE = 1, as E increases D decreases, encourages s/w team to find as many errors, within a project allows to assess teams ability to find errors before they move to next activity, analysis model errors pass to design, in this context

$DRE_i = E_i / (E_i + E_{i+1})$

E_i = No of errors in SE activity i

E_{I+1} = No of errors in SE activity I+1, not discovered in activity i

DREi = 1, filter out errors before they pass to next activity

22.4 INTEGRATING METRICS WITHIN S/W PROCESS :

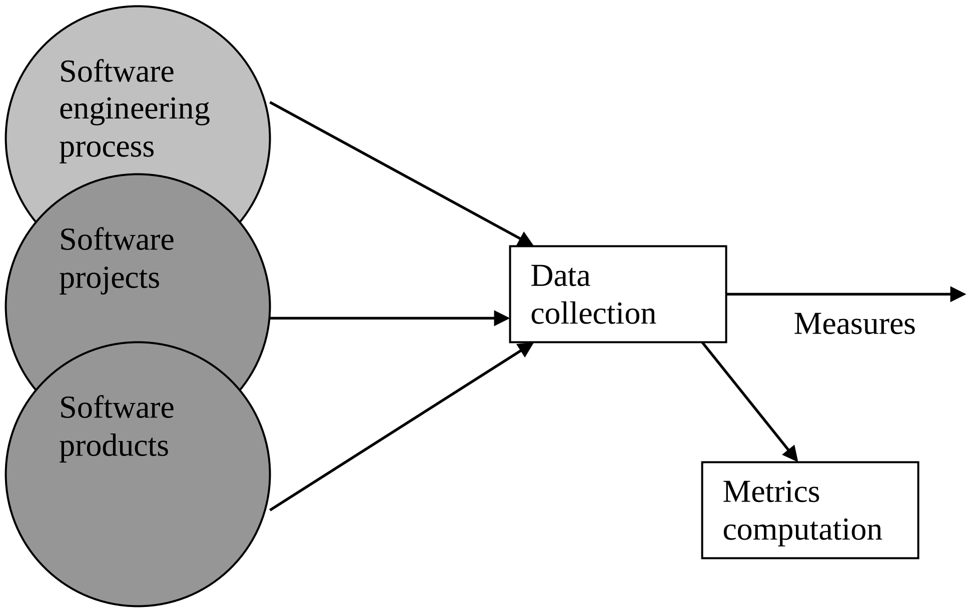
Majority s/w developers do not fill requirement of s/w measure

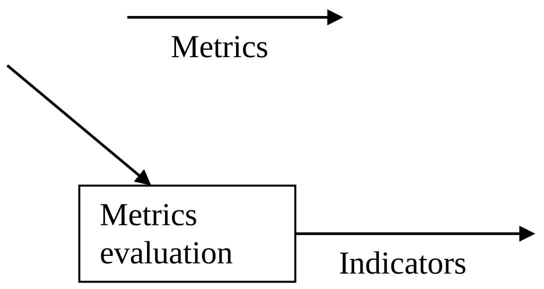
22.4.1 Arguments for Software Metrics

Measurement to determine improvement, management determine improvement by evaluating productivity & quality, measurement used to establish a process baseline to measure improvement, project managers interested in project estimation, high quality systems & maintain schedules, by setting project base line easier to measure all, collection of quality metrics enable organization to eliminate causes of defects having greatest impact on development, After completion of design, benefit from metrics

22.4.2 Establishing a Baseline

Baseline consist of data collected from past projects, baseline data must have following attributes (1) reasonably accurate data (2) data collected for as many projects as possible (3) consistent measures (4) apply to similar works





22.4.3 Metrics Collection, Computation & Evaluation

To establish a baseline, data are collected in ongoing manner, past project measures collected, metrics computed, evaluated & applied during estimation, technical work, project control etc, metrics produces indicators to guide project

22.5 METRICS FOR SMALL ORGANIZATIONS

many s/ org have less than 20 developers, unrealistic to develop comprehensive s/w metrics programs, all org should measure & use metrics to improve their s/w process & quality & timeliness of product, s/w process must be – simple, customized to local needs & should add value, how to chose a simple metric that add value & meet the needs of org, begin by focusing on result then measurement,

23. ESTIMATION

23.1 OBSERVATIONS ON ESTIMATION

A set of activities, first activity is to estimate – work to be done, resources & time, estimation is associated with some degree of uncertainty, for estimation - experience, access to historical info, courage to commit quantitative measure

Project complexity has strong effect on uncertainty, a relative measure, affected by past experience ex a first time developer find some application highly complex whereas developer who has developed similar applications in past find it run of the mill, measure of complexity such as FP applied in early stages, project size affect accuracy of planning, as size increases interdependency b/w diff elements increases, difficult decomposition, degree of structural uncertainty is risk to estimation, structure refers to degree to which requirements solidified, easy compartmentalized of functions etc,
Available historical info strong influence on estimation risks, one can improve areas of problem, s/w metrics of past projects applied to estimate with assurance,
Risk is degree of uncertainty in quantitative estimates of resources, cost & schedule, if scope poorly understood or changing requirements, uncertainty & risk becomes high, complete definition of project (function, performance & i/f), not to be rigid about estimation, modern iterative SE approaches revise estimation according to change

Objective is to provide framework to make reasonable estimates of resources, cost & schedule, done at the beginning of the project, updated regularly as project progresses, should include best & worst situations

23.2 THE PROJECT PLANNING PROCESS

Objective is to provide a framework that enables manager to reasonably estimate resources, cost & schedule, define best-case & worst-case scenario

23.3 S/W SCOPE & FEASIBILITY:

Describes functions & features of s/w, i/p & o/p data, the content delivered to users, & performance, constraints, interfaces & reliability. Scope defined using either

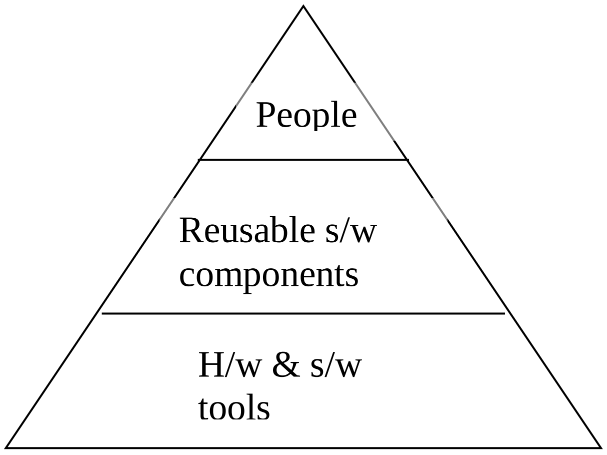
1. A narrative description of s/w scope developed after communication
2. A set of use-cases developed by end-users

Functions in scope evaluated & refined to provide more detail before estimation as cost & schedule estimation are functionality oriented, decomposition of functionality, performance reflect processing & response time requirements, constraints identify limits on s/w

After establishing scope, is the project feasible?, not everything imaginable is feasible, technology : is project technically feasible, can defects be reduced using technologies, finance : is it financially feasible, complete development at a cost given by org, can customer afford it, time : time-to-market to beat competition, resources : org having needed resources for successful implementation task of project team to find if project can work in dimensions given above

23.4 RESOURCES :

Second activity, resources for s/w development,



h/w & s/w provide infrastructure & base, reusable components reduce time & cost, people, the primary resource, characteristics to be defined – description of resource, statement of availability, time when required, duration of application

23.4.1 Human resources : skills required for development, organizational position & specialty to be specified, for small project single person may perform all the activities, no. of persons required can be estimated after estimating development effort

23.4.2 Reusable s/w components: component based SE, components must be cataloged, standardized & validated, four categories considered & plan accordingly

1 Off-the-shelf compo : existing s/w, acquired from third party or developed internally, ready to use & fully validated

2 Full-experience compo : existing specification, design, code or test data from past projects having similar to requirements, team have full experience with application area, modification low risk

3 Partial-experience compo : existing specification, design, code or test data from past project having relative to requirements, quite modification, limited experience with application area, modification high risk

4 New components : built for project

Guidelines to be followed

1. off-the-shelf compo meets requirement, use them, cost for acquiring & integrating will be less that building, low risk
2. full experience compo, risk for modifying & integrating low, use them
3. partial experience compo, analyzed before use, extensive modification, avoid, more costly

consider at the planning stage, Better to specify s/w resources early, alternatives

23.4.3 Environmental resources :

Env incorporates h/w & s/w, h/w provides a platform that support tools to develop s/w, time of requirement to be specified, supporting h/w also must be specified

23.5 S/W PROJECT ESTIMATION :

In early days, s/w was less costly, errors had little impact, today s/w most expensive element, cost & effort estimation not easy, variability lead to profit & loss, variables like (h), technical, env, political, affect cost & effort, a series of systematic steps provide estimations, for estimating a no of options :

1. delay estimation until late

   Not a practical option
2. estimate on similar project that are complete

   if similarity is there in project & env, not a good indicator for future results
3. simple decomposition technique to generate cost & effort estimate

   Divide & conquer, decomposition into major function & SE activities
4. use empirical model

   Based on experience, along with decomposition tech provide valuable estimation
5. use automated estimation tools

   Employ decomposition/empirical model, characteristics of org & s/w

Estimation must be based on historical data

23.6 DECOMPOSITION TECHNIQUES :

Before decomposition, estimate size

23.6.1 S/W SIZING : accuracy of estimate depend upon (1) degree of accurate size estimate (2) ability to translate size into human effort, time & cost (3) degree to which project planning reflects ability of s/w team (4) stability of requirement & env

Size estimation a major challenge, quantifiable outcome of s/w, LOC or FP, different approaches :

1. Fuzzy logic sizing : approximate reasoning technique, identify type of application, establish magnitude on qualitative scale, assess to historical data
2. FP sizing : estimation on information domain

3. Standard component sizing : s/w composed of standard components, screens, modules, reports, files, no. of occurrence of each compo, with help of historical data size of each component id decided, combined value
4. Change of sizing : existing s/w modified as part of project, no & type of changes

All these approaches must be used & combined to get expected value

23.6.2 PROBLEM BASED ESTIMATION :

LOC & FP, (1) used as an estimation variable to size each element of s/w (2) as baseline metrics from past project to estimate cost & efforts

LOC & FP both have some common characteristics, s/w eng begins planning with project scope – attempts to decompose s/w into functions - LOC & FP for each function estimated - Baseline productivity metrics applied to find cost & effort for each function – combine to produce project estimates

LOC/p-m or FP/p-m average computed by project domain, projects should be grouped by team size, complexity & others, new project allocated to a domain & then domain avg is applied to generate estimates

LOC/p-m or FP/p-m differ in level of detail, for using LOC decomposition is absolutely necessary, more partitioning leads to more accurate estimate, in FP estimates decomposition works differently, focus on info domain – i/p. o/p, files, inquiries etc, regardless of variable & using historical data developer estimate optimistic, most likely & pessimistic size value for each function or information domain, 3-point or expected value as weighted avg of optimistic, most likely & pessimistic estimates

$$EV = (S_{opt} + 4S_m + S_{pess}) / 6$$

Example of LOC based estimation :

A CAD application for mechanical component, to be run on engineering workstation, interface with computer graphics peripherals - mouse, digitizer, color display, laser printer, scope is developed as :

CAD s/w accepts 2 & 3 – dim geometric data from engineer thro user interface, geometric data & supporting info in CAD database, design analysis module will produce o/p, display on graphic device, s/w control & interact peripheral devices

This scope is not bound, concrete detail & quantitative bounding,

Major functions identified are :
1. user i/f & control facilities
2. 2 –dim geometric analysis
3. 3 – dim geometric analysis
4. database mgmt
5. computer graphics display facilities
6. peripheral control functions
7. design analysis module

LOC for each function estimated using 3-function tech as :

Function	Estimated LOC
UICF	2,300
2DGA	5,300
3DGA	6,800
DBM	3,350
CGDF	4,950
PC	2,100
DAM	8,400
TOTAL LOC	33,200

33,200 LOC for the s/w, from historical data, avg productivity 620/LOC & labor cost 8000 $ /month, project estimation cost 4,31,000 $ for 54 p-m

FP based estimation :

Focus on information domain, estimates i/p, o/p, etc for CAD s/w, weighing factor applied & estimates are :

Information domain	Opt.	Likely	Pees.	Est. count	Weight	FP-count
No of i/p	20	24	30	24	4	97
No of /p	12	15	22	16	5	78
No of inq	16	22	28	22	4	88
No of files	4	4	5	4	10	42
Ext i/f	2	2	3	2	7	15

Count total						320

Complexity adjustment factors computed as shown below

Factor	Value
Backup & recovery	4
Data communication	2
Distributed processing	0
Performance critical	4
Existing operating env	3
On-line data entry	4
I/p transaction over multiple screens	5
On-line file updation	3
Complex info domain	5
Complex internal processing	5
Reuse of code design	4
Conversion/installation in design	3
Multiple installation	5
Application designed for change	5
Complexity adjustment factor	1.17

Finally FP is derived as:

$$FP = \text{count-total} * [0.65 * 0.01 * \sum fi]$$
$$= 375$$

Historical data show avg productivity 6.5 FP/p-m & cost per FP 1230$, estimated project cost 4,61,000$ & 58 p-m

23.6.5 PROCESS BASED ESTIMATION :

Estimation on the process that will be used, process decomposed in small activities/ tasks, functions & required s/w activities listed, estimates efforts for each activity, labor rate applied for computing cost, for whole project , compared with other estimates, similarity, reliability

For the CAD s/w complete process table of estimates of effort for each activities is

Activity →	CC	Planning	Risk Analysis	Engineering		Construct & release		CE	Totals
Tasks				Analysis	Design	Code	Test		
Functions↓									
UICF				0.5	2.5	0.4	5.0	n/a	8.4
2DGA				0.75	4.0	0.6	2.0	n/a	7.35
3DGA				0.5	4.0	1.0	3.0	n/a	8.5
CGDF				0.5	3.0	1.0	1.5	n/a	6.0
DBM				0.5	3.0	0.75	1.5	n/a	5.75
PCF				0.25	2.0	0.5	1.5	n/a	4.25
DAM				0.5	2.0	0.5	2.0	n/a	5.0
Total	0.25	0.25	0.25	3.5	20.5	4.5	16/5		46.0
% Effort	1%	1%	1%	8%	45%	10%	36%		

53% of efforts on engineering, Totals provide for estimated efforts for each activity & tasks, labor rate associated, estimated project cost 3,68,000$ & effort 46 p-m

3 diff estimates range from 46 p-m to 58 p-m, avg 53 p-m, 13 % variation, if wide variation, reevaluation of info used to make estimates, causes can be (1) scope is not understood (2) inappropriate or misapplied estimate techniques

23.7 EMPERICAL ESTIMATION MODELS :

Empirically derived formulas for s/w estimation such as function of LOC & FP, LOC & FP estimated using earlier approaches & plugged in estimation model, empirical data supporting such models derived from limited sample of projects, no single model appropriate for all s/w

1. structure of estimation model :

a typical model, derived on regression analysis on data from past project, overall structure is

$$E = A + B * (ev)^{C}$$

Where A,B & C = empirically derived constants
E = effort in p-m
ev = estimation variable like LOC or FP

Majority models use some project adjustment component, to enable E to be adjusted to other project characteristics like complexity, skill etc
LOC oriented models :

$E = 5.2 * (KLOC)^{0.91}$ Walston-felix model
$E = 5.5 + 0.73 (KLOC)^{1.16}$ Baily-Basili model
$E = 3.2 * (KLOC)^{1.05}$ Boehm simple model
$E = 5.288 * (KLOC)^{1.047}$ Doty model for KLOC>9

FP oriented models :

$E = -13.39 + 0.0545$ FP Albretch & Gaffiney model
$E = 60.62 * 7.728 * 10^{-8}$ FP Kemerer model
$E = 585.7 + 15.12$ FP

Each model give different result for same value of LOC or FP, locally calibrated

2. COCOMO MODEL :

Constructive COst MOdel , s/w estimation model derived by Barry Boehm, original model evolved to COCOMO II, a hierarchy of models address following areas

Application composition model : Used in early stages, when prototyping of user i/f, s/w & system interaction, performance assessment, to evaluate technology maturity
Early design stage model : after stabilized requirements & basic architecture
Post-architecture-stage-model : during construction

COCOMO requires size info, 3 sizing options as part of model hierarchy – object points, FP & LOC, Application composition model uses object points, an indirect s/w measure computed using counts of screens, reports & components required to build application, complexity associated with each, complexity – a function of no & source of client & server data tables required to generate screen or report,

Object Type	Complexity Weight		
	Simple	Medium	Difficult

Screen	1	2	3
Report	2	5	8
3GL Compo			10

Total count, if component based development/s/w reuse than % of reuse estimated & object point count adjusted

NOP = (Object points) * [(100 - %reuse) / 100]

NOP = New object point

To estimate effort on NOP, productivity rate is derived as

Prod = NOP /p-m

Developer's Experience/Capability	Very low	Low	Nominal	High	Very high
Environment maturity/capability	Very low	Low	Nominal	High	Very high
PROD	4	7	13	25	50

Productivity rate uses developer experience & env maturity, effort can be derived as

Estimated effort = NOP/PROD

Advanced COCOMO model considers cost drivers

3. The S/w equation :

Multivariable model, derived from productivity data of 4000 projects

$$E = [\ LOC / B^{0.333} / P\ ]^{3} * (1/t^{4})$$

E = effort in p-m
T = project duaration in months
B = special skill factor, for KLOC = 5 – 15, B = 0.16
KLOC > 70, B = 0.39

P = productivity parameter that reflects
Process maturity
S/w engineering practice
Programming lang used
S/w env
Skill & experience of team
Complexity of project

P = 2000 for real-time embedded s/w
P = 10,000 for telecommunication & system s/w
P = 28,000 for business application

Derived from historic data another modification, minimum development time is defined as

t_{min} = 8,14 (LOC/P)$^{0.43}$ in months
E = 180 Bt^3 in p-m

Using P = 12,000 for CAD s/w

t_{min} = 8.14(33,200 / 12.000) $^{0.43}$ = 12.6 months
E = 180 * 0.28 * (0.15) ^{3} = 58 p-m

THE MAKE-BUY DECISION :

Cost effective to acquire than develop, options available are

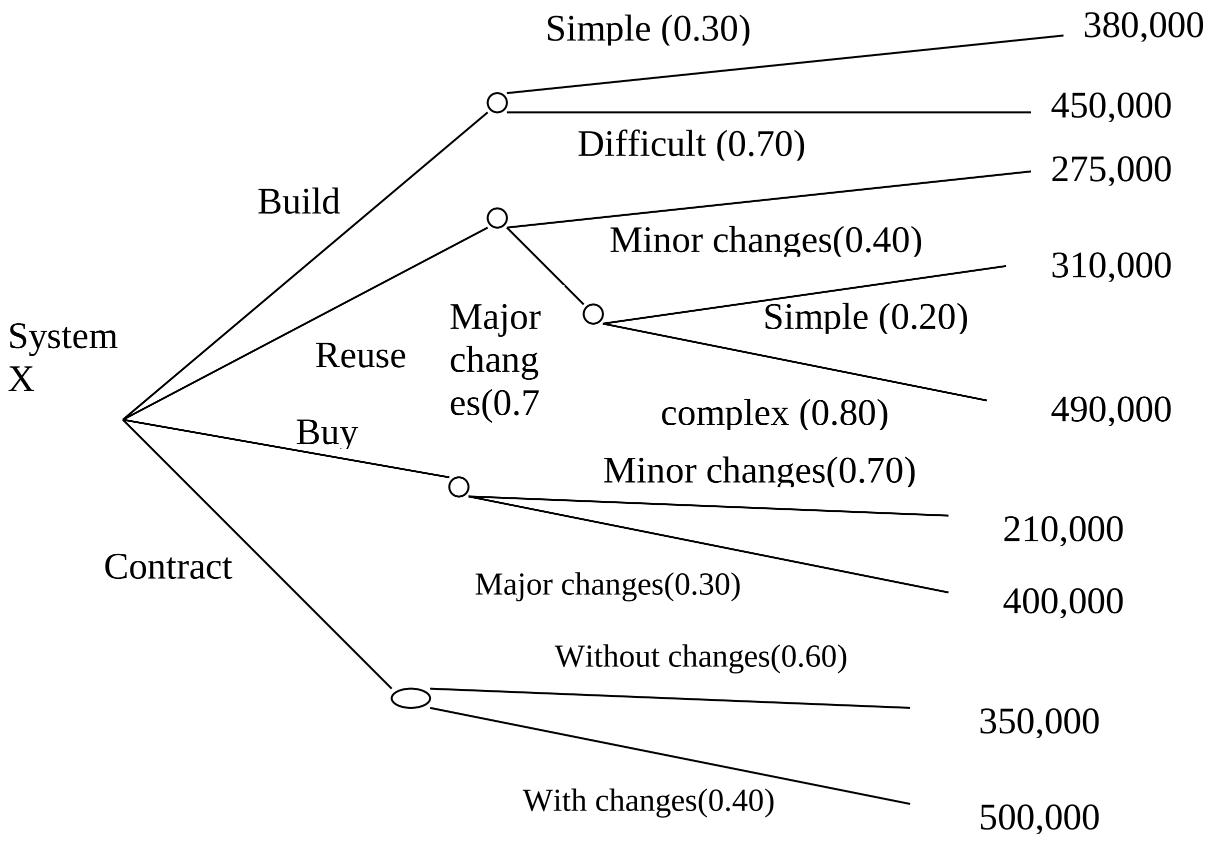
1. s/w to be purchased off-the-shelf
2. full/ partial experience s/w component to be acquired & modified & integrated
3. outside contract

Guidelines to follow :

1. define function & performance of desired s/w
2. estimate cost to develop & date of delivery
3. select 3-4 candidate applications that best meet specifications
4. select reusable component that assist in constructing application
5. compare candidate s/w
6. evaluate each option
7. contact other user & ask for opinion

Final make-buy decision is made on (1) delivery date rigidity (2) cost of acquisition & customization (3) cost of support

A decision tree analysis can be performed to weight alternatives as shown in the figure



System X to be developed, options available are : (1) build from scratch (2) reuse partial experience component (3) buy available s/w product & modify to meet the needs (4) third party contract, to build from scratch, 70% prob of difficult job, estimate cost for difficult development 4,50,000, for simple development cost estimated 3,80,000, expected cost computed along any branch is :

Expected cost = ∑(path prob * estimated path cost)

For build path,
Exp cost = 0.30 * 3,80,000 + 0.70 * 4,80,000
= 4,29,000
similarly cost for reuse path = 3,82,000

buy path = 2,67,000
third party = 4,10,000

lowest is buy option, not only cost but other criteria considered, availability, experience, requirement conformance, changes to be done etc

OUTSOURCING :

Contracted to third party, lower cost & higher quality, positive side cost saving, negative side co loses control over s/w & competitiveness handed to third party

AUTOMATED ESTIMATION TOOLS :

Uses decomposition techniques & empirical estimation models, Allow to estimate cost & effort, perform analysis on different situations - delivery date, staffing, provide 6 generic functions :

1. Sizing of project deliverables : size of product, includes external representation such as screens & reports, s/w itself KLOC, functionality FP, description i.e. documents
2. Selecting project activities : process frame work is se task set is specified
3. Predicting Staffing level : no of people available
4. predicting s/w effort : effort required for product
5. Predicting s/w cost :
6. predicting s/w schedule : when effort, staffing & project activities are known, schedule can be derived

Accurate project estimation generally make use of 2 or 3 techniques. To derive accurate estimate, planner should compare & combine estimates derived in different techniques along with historical data.

25. RISK MANAGEMENT

The future of project, changes in requirement, technology & other connected entities affecting timeliness & success, choice of methods & tools, no. of people, emphasis on quality

25.1 REACTIVE Vs. PROACTIVE RISK STRATEGY

REACTIVE RISK STRATEGY : monitors for likely risks, resources set aside, nothing until problem arises

PROACTIVE RISK STRATEGY : before technical work, identification of potential risks, probability & impact, priority decided, plan to manage risks, primary objective to avoid, a contingency plan

25.2 S/W RISKS : two characteristics,
Uncertainty : risk may or may not happen, no 100% probability
Loss : if risk becomes real, unwanted consequences / losses
Necessary to quantify uncertainty & degree of losses, categories are :

1. Project risk : threatens project plan, project schedule to slip, cost increase, risks are - potential budgetary, schedule, personnel, resources & requirement problems & their impact
2. Technical risks : threatens quality & timeliness, implementation difficult, risks are - potential design, implementation, interfacing & maintenance problem, specification ambiguity, technical uncertainty & obsolescence, cause is underestimation
3. Business risks : viability of s/w, (1) market risk (2) strategic risks (3) sales risk (4) management risk (5) budget risk

Other categorization is :

1. Known risks : uncovered by careful evaluation of project plan, business & technical env, other info - unrealistic delivery data, poor development env

2. Predictable risks : from past project experience, poor communication, dilution of staff effort, staff turnover etc
3. Unpredictable risks : do occur, difficult to identify in advance

25.3 RISK IDENTIFICATION

Systematic attempt to specify threats to project plan - schedule, estimates etc., identify known & predictable risks - steps to avoid or control, two distinct type associated with each category, <u>generic:</u> that threat every s/w project, <u>product specific</u> : identified by understanding technology, people & env of the project, examine project plan & scope, finding out special characteristics of the product that threatens project plan

To identify risks, risks item checklist, focus on known & predictable risks, subcategories are :

1. Product size : overall size of s/w
2. Business impact – constraints by mgmt & market
3. Customer characteristics : customer sophistication & communication
4. Process definition – degree of s/w process defined & followed
5. Development env : availability & quality of tools
6. Technology to be built – complexity & new tech
7. Staff size & experience – technical & project experience

A list of questions, allows estimation of risks, characteristics of risks, risk component & drivers along with occurrence probability,

25.3.1 ASSESSING OVERALL PROJECT RISK

Following questions from survey of s/w project managers, ordered by importance to success of project

1. S/w & customer manager formally committed to support the project?
2. End-user enthusiastic for project & product
3. Requirement fully understood by team & customer
4. Customer involved in definition of requirements
5. Realistic expectation of end-users
6. Stable project scope

7. Right mix of skills in team
8. Stable project requirements
9. Experience with technology to be implemented
10. Adequate no of developer in team
11. Agreement of customer & users in importance of project

If any question has negative answer, risk mitigation, monitoring & mgmt steps,

25.3.2 RISK COMPONENTS & DRIVERS :

Project manager must identify risk drivers that affect risk components – performance, cost, support & schedule, defined as :

1. Performance risk : degree of uncertainty that product will meet its requirements & fit for its intended use
2. Cost risks : project budget be maintained
3. Support risks : s/w easy to correct, adapt & enhance
4. Schedule risks : project schedule be maintained & on time delivery

Impact of risk drivers on risk components in 4 categories – negligible, marginal, critical & catastrophic, figure indicates potential consequences of errors & failure to achieve desired result

COMPONENTS → CATEGORY ↓		PERFORMANCE	SUPPORT	COST	SCHEDULE
CATASTROPHIC	1	Failure to meet requirement, result in mission failure		Failure result in increased cost & schedule delay	
	2	Degradation to achieve technical performance	Nonresponsiveness / unsupported s/w	Financial shortage, budget overrun	Unachievable delivery data
CRITICAL	1 2				
MARGINAL	1 2				
NEGLIGIBLE	1 2				

25.4 RISK PROJECTION :

Risk projection/ estimation, rate each risk for its probability/likelihood that risk on (1) risk is real & (2) consequences of problems if risks occur, risk projection activities are : (1) establish a scale that indicate probability of risk (2) delineate consequences of risk (3) impact of risk on project & product (4) analyze risk projection

25.4.1 Developing a Risk Table : Risk table a technique for risk projection, ex

Risks	**Category**	**Probability**	**Impact**	**RMMM**
Size estimate significantly low	PS	60%	2	
Larger no of users than planned	Ps	30%	3	
Less reuse than plan	PS	70%	2	
End users resist system	BU	40%	3	
Delivery deadline tight	BU	50%	2	
Funding will be lost	CU	40%	1	
Change in customer requirements	PS	80%	2	
Technology do not meet expectation	TE	30%	1	
Lack of training on tools	DE	80%	3	
Staff inexperienced	ST	30%	2	
Staff turnover may be high	ST	60%	2	

1. catastrophic
2. critical
3. marginal
4. negligible

Risks are listed from risk item checklist, categorized, probability computed, impact assessed by assessing risk components, impact on each category assessed & then averaged, sorting on probability & impact, risk prioritization, project manager define cut-off line, risk above line to be considered, below line to create second order priority

Risks above line must be managed, Risk Mitigation, Monitoring & Management plan, assess risk drivers on qualitative scale – impossible, improbable, probable & frequent, mathematical probability with each qualitative value

25.4.2 ASSESSING RISK IMPACT : determined by three factors – nature, scope & timing, nature indicates problem ex poorly defined i/f to h/w lead integration problem, scope refers severity & distribution, timing refers when & how long, consequences of risk can be determined :

1. Determine avg prob of occurrence for each risk component
2. Using table, determine impact for each compo
3. Complete the risk table & analyze results

Risk exposure is determined by

RE = P * C, P = probability of occurrence of risk
C = cost if risk becomes real

After estimation cost of risk, each risk exposure calculated, total provides a way to adjust cost of project, predicts increase in staff resource,
Iterative application of risk projection & analysis, reevaluation in new circumstances

25.5 RISK REFINEMENT :
During early stages, risks stated generally, with time more knowledge of project & risk, refine risk in detailed risks, easier to mitigate, monitor & manage, refinement helps to isolate underlying risks, easier analysis & response

25.6 RISK MITIGATION, MONITORING & MANAGEMENT :

Risk analysis to assist project team developing strategy dealing risk, 3 points for effective strategy are :

1. Risk avoidance
2. Risk monitoring
3. Risk management & contingency planning

Risk avoidance prime objective, developing risk mitigation plan, for ex high staff turnover risk r1, based on past data & intuition, the prob l1 is 70%, impact x1 critical impact on cost & schedule, to mitigate risk develop a strategy for reducing turnover, possible steps :

1. Determine causes for high turnover ex poor working conditions, low pay
2. Mitigate causes under management control before project
3. Develop technique to ensure continuity if people leave during project
4. Disperse information to all team
5. Documentation std
6. Review with all team
7. Define backup staff

Risk monitoring as risk progresses, monitor factors that provide indication of risk becoming more/less, for high turnover monitor :
1. Team members attitude
2. Degree to which team has jelled
3. Interpersonal relationship b/w members
4. Potential problems with compensation & benefits
5. Availability of jobs

Also monitor effectiveness of risk mitigation steps, ex. document std step, documents developed in time

Risk management & contingency planning after mitigation efforts fail & risk real, ex in mid project, no of people plan to leave, if mitigation plan followed then backup available, information documented, dispersed information to all members, temporary focusing resources to speed up, people to leave transfer their knowledge in meetings, documents, audio, video etc

RMMM incur extra cost, 30-40 risks for large project, risk management for all is a project itself, 20% risks leads 80% of other, 20% must be identified

25.7 RMMM PLAN :

Risk management strategy as a s/w project plan or separate RMMM plan, documents all work performed for risk analysis, part of project plan, with beginning of project, risk mitigation & monitoring, risk mitigation a problem avoidance activity, risk monitoring has 3 prime objectives (1) assess whether predicted risk occurs (2) ensure risk aversion steps applied (3) collect info for future risk analysis. Problem tracked to more than one risk, risk monitoring determines origin of risk outline :

I. Introduction
   1. scope & purpose of document
   2. overview of major risk
   3. responsibilities
      a. management
      b. technical staff

II. project risk table
   1. description of all risks above cut-off line
   2. factors influencing prob & impact

III. risk mitigation, monitoring & management
   n. risk # n
      a. mitigation
         1. general strategy
         2. specific steps to mitigate risk
      b. Monitoring
         1. factors to be monitored
         2. monitoring approach
      c. management
         1. contingency plan
         2. special consideration

IV. RMMM plan iteration schedule

V. Summary

Risk analysis – identification, projection, assessment, management & monitoring – absorb significant amount of project planning effort, worth it

26. QUALITY MANAGEMENT

High quality product, SQA an umbrella activity, throughout s/w process, encompasses (1) a quality management approach (2) effective Se technology (3) FTR (4) multitired testing strategies (5) control of s/w docu (6) procedure to assure compliance with stds (7) measurement & reporting mechanism

26.1 QUALITY CONCEPT

26.1 QUALITY CONCEPT : variations b/w two products, variation control, in s/w minimize difference b/w predicted & actual resources - staff, equipment's etc, minimize the variance in no of release, minimize differences in speed & accuracy of customer support

26.1.1 QUALITY :a characteristic/ attribute of something
Quality of design : characteristics specified such as grade of material, tolerance, performance etc, higher grade material, better performance, more tolerance increases product quality, if according to specification, for s/w quality of design encompasses requirements, specification & design of system,
Quality of conformance : degree to which design specification followed during manufacturing, greater degree of conformance, higher quality, s/w quality of conformance focuses n implementation, if implementation follows design & meets requirements, conformance quality high
Other factors also to be considered
User satisfaction = compliant product + good quality + within budget & schedule

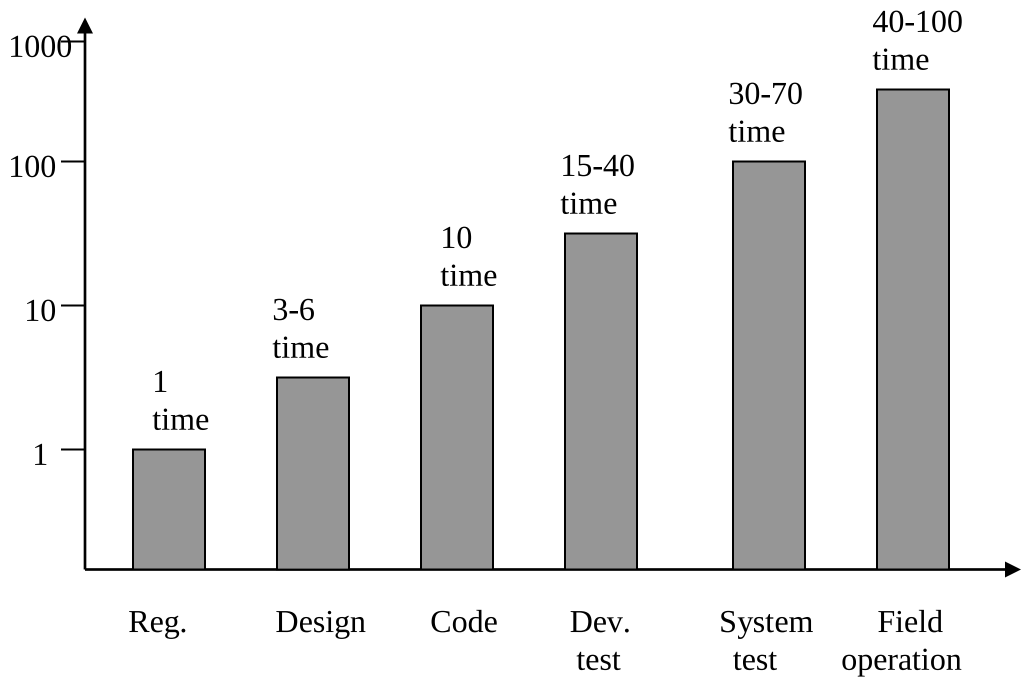
26.1.2 QUALITY CONTROL : A series of inspection, reviews & tests throughout development cycle, ensure product meets requirements, a feedback to process creating product, allows to tune process, QC automated or manual or combination of both, key concept is all the products have defined & measurable specifications, to compare o/p, feedback essential to minimize defects

26.1.3 QUALITY ASSURANCE : auditing & reporting function, management provided data about product quality & gain insight & confidence about product quality meets goal, management responsibility to address problem & apply necessary steps to resolve quality issue

26.1.4 COST OF QUALITY : all cost incurred to gain quality & activities, study provide baseline for current cost of quality, means to reduce cost & normalization value for comparison, divided in prevention, appraisal & failure cost, prevention cost are :

1. Quality planning
2. FTR
3. Test equipment
4. Training
5. Appraisal – activities to gain insight of product conditions, costs are :
6. Inprocess & interprocess inspection
7. Equipment calibration & maintenance
8. Testing
9. Failure – internal failure cost for detecting errors & external failure cost associated with defects, internal failure costs are :
10. Rework
11. Repair
12. Failure mode analysis
13. External failures are :
14. Complaint resolution
15. Product return & replacement
16. Help line support
17. Warranty work

Relative cost to find & repair defect increases from prevention to detection & from internal to external failure, study of IBM, prevention cost was 91$ / defect & fixing defects after delivery was 18 times more



26.2 S/W QUALITY ASURANCE :

S/w quality : conformance to explicitly stated functional & performance requirements, explicitly documented development `stds & implicit characteristics expected of the s/w i.e.

1. S/w requirements - foundation to measure quality, lack of conformance = lack of quality
2. Specified stds define set of development criteria for s/w, if criteria not followed = lack of quality
3. Implicit requirements, ex maintainability

First formal quality assurance & control functions by Bell Labs in 1916, STD for s/w quality assurance in 1970, responsibility of s/w engineer, project manager, customer, sales persons, SQA group look at product from customer point of view

26.2.2 SQA ACTIVITIES : variety of tasks associated with s/w engineer who do technical work & SQA group responsible for quality assurance planning, record keeping, analysis & reporting

S/w engineer performs QA, applying technical methods & measures, conducting FTR & testing, SQA group assist engineers to achieve high quality product, set of activities :

1. Prepare a SQA plan for project : developed during project planning, governs QA activities, identifies :
2. Evaluations to be performed
3. Audits & reviews to be performed
4. Stds applicable to project
5. Procedures for error tracking & reporting
6. Documents to be produced
7. Amt of feedback to be provided to team

- Participates in development of process description : SE tam select process, SQA examines for compliance with policy, internal & external s/w std
- Review SE activities to verify compliance with s/w process : SQA group identifies, documents & tracks deviation & corrections made
- Audit designated work products to verify compliance with those defined as part of s/w process : reviews work products, identifies & track deviations, correction & report to manager
- Ensures that deviation in s/w work & product are documented & handled according to documented procedure : deviation in project plan, process description, applicable standards or technical work product
- Record any noncompliance & report to management : tracked until resolved

Coordinates the control & management of changes, collect & analyze s/w metrics

26.3 S/W REVIEWS :

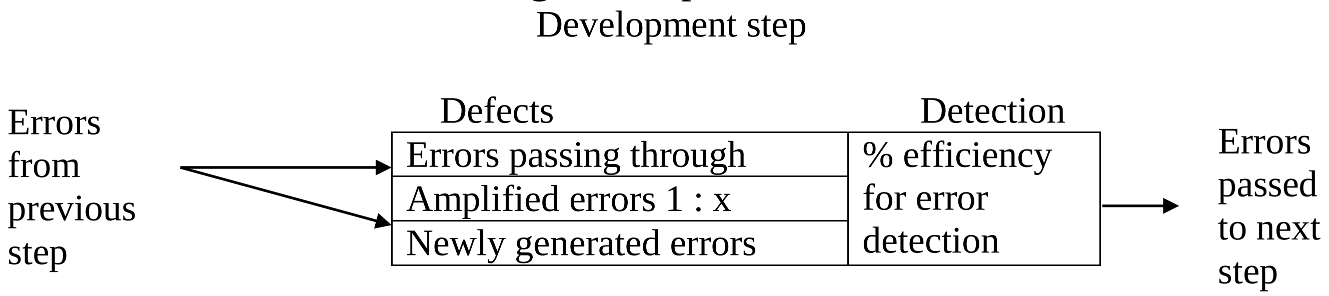
Filter for SE process, applied at various points, uncover errors, purifies product, review necessary to

1. Point out needed improvement in product
2. Confirm part of product not requiring improvement
3. Achieve uniform & predictable technical work

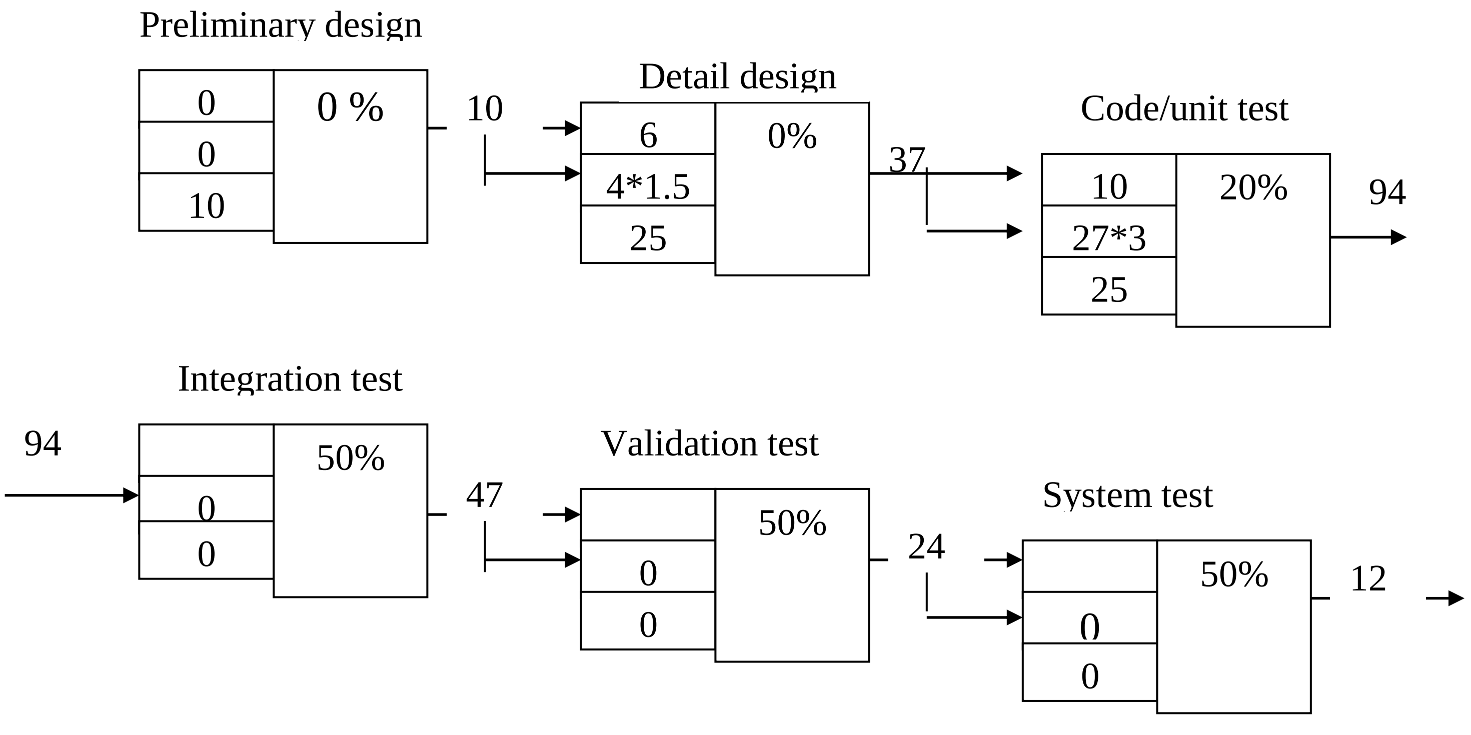
Many different types of reviews - informal meetings, a formal presentation, most imp FTR or walkthrough, provide effective means to improve s/w quality

26.3.1 COST IMPACT OF S/W DEFECTS : fault/defects a quality problem, primary objective of FTR to find errors before release, to discover errors before propagate to next step, design activity introduces 50-65% of all errors, FTR help uncover 75% of design errors, reduce cost of maintenance, error uncovered during design cost 1 unit to correct, before testing 6.5 units, during testing 15 units & after release 60 – 100 units

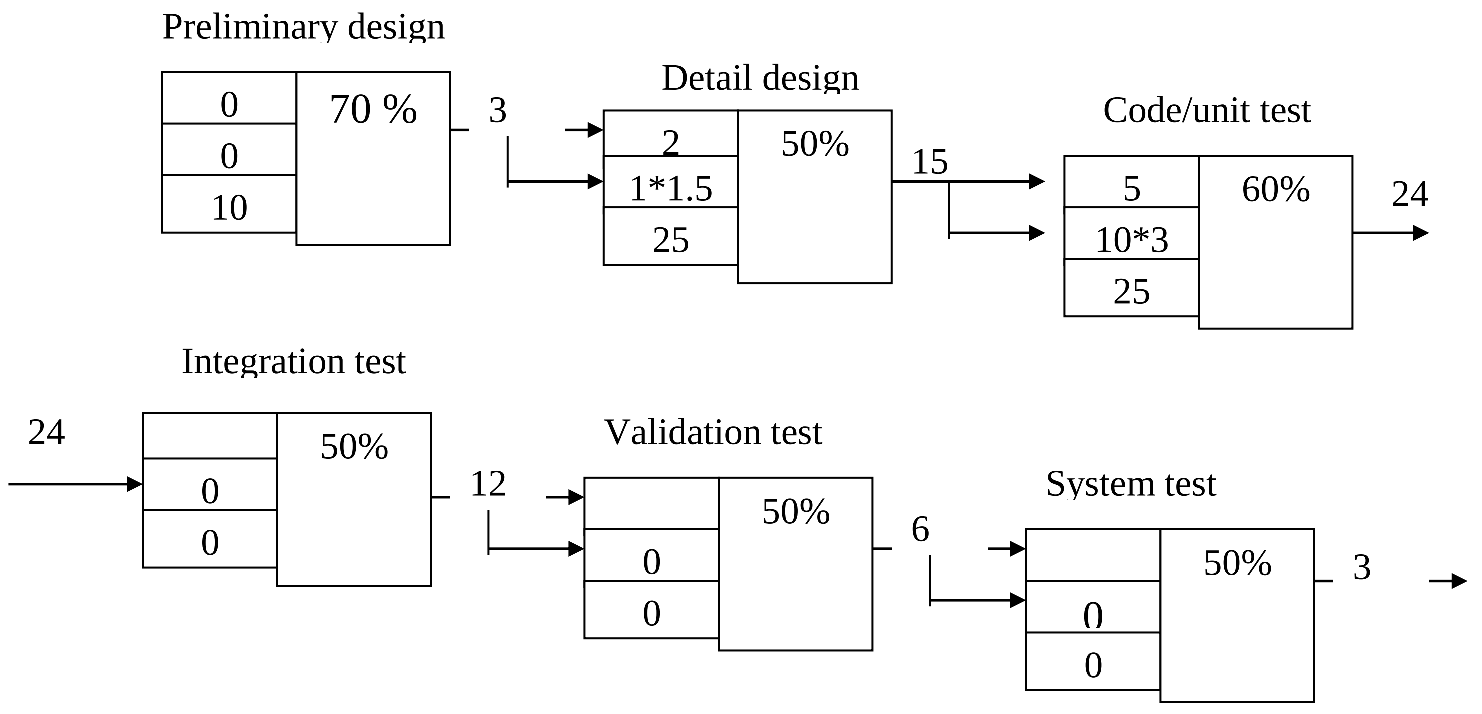
26.3.2 DEFECT AMPLIFICATION & REMOVAL : defect amplification model to detect errors during diff steps



Box represents s/w development step, errors generated, review fail to uncover newly generated errors & from previous step, errors pass through, may get amplified



Defect amplification with no review, each test step uncover & correct 50% of errors, 10 design errors amplified to 94 errors & 12 defects were released



Defect amplification with design & code review at each step, 10 design errors amplified to 24 errors & 3 defects were release, associating hypothetical cost, total cost with review 783 units & without 2177 units, review expends time & effort & money, but provide cost benefit

26.4 FORMAL TCHNICAL REVIEWS :

SQA activity by s/w engineers, objectives are : (1) uncover errors in function, logic or implementation (2) verify requirement conformance (3) ensure s/w is according to predefined std (4) achieve s/w in uniform manner (5) manageable project

Serves as training ground, different approaches for s/w analysis, design & implementation, includes walkthroughs, inspections & reviews, FTR conducted as meeting, properly planned controlled & attended

26.4.1 The Review Meeting : constraints applied are:

1. 3-5 people involved in review
2. Advance preparation not more than 2 hours

3. Duration not more than 2 hours

Focuses on specific part of s/w, higher possibility of uncovering errors, developer finishes work product, informs project leader about review, contacts review leader, evaluates product, distributes copies to 2-3 reviewers, each reviews work product, leader establishes agenda, meeting attended by leader, reviewers & developer, recorder record imp issues raised, meeting begin with intro to agenda & developer, developer gives walkthrough, reviewers raises issues prepared in advance, when valid problem/ errors discovered, recorder notes, at the end all decide on whether (1) accept work product without modification (2) reject product due to severe errors (3) accept provisionally

26.4.2 Review Reporting & Record Keeping : recorder keeps record of all issues raised, summarized at the end, a review issue list produced, a review summary report includes :
What was reviewed
Who reviewed
Findings & conclusion
Part of project historical record, review issue list to (1) identify problem areas within product (2) serve as an action item checklist , guide developer for corrections, a follow-up procedure to check items on issue list corrected

26.4.3 Review Guidelines :

Guidelines for FTR, established in advance, distributed to all reviewers & followed, minimum set are :
1. Review the product not the producer : not to embarrass, proper tone & attitude
2. Set an agenda & maintain it : drift not allowed, on track & schedule
3. Limit debate & rebuttal : may not agree universally, debate issues recorded for further discussion
4. Enunciate problem area but don’t attempt to solve every problem noted : not a problem solving session, solution by developer, after review
5. Take written notes : make notes on board
6. Limit no of participant & insist on advance preparation :
7. Develop a check list for each work product to be reviewed : help to structure FTR meeting, focus on imp issues

8. Allocate resources & time schedule for FTR : as a part of SE process, time for modification
9. Conduct meaningful training of reviewers : for effective review, include process related issues & human psychology
10. Review early reviews : review guidelines must be reviewed

26.4.4 Sample-driven Reviews :

26.5 FORMAL APPROCH TO SQA :

S/w quality is everybody's responsibility, achieved thro' competent analysis, design, coding & testing, application of FTR, multitiered testing, control of work products & changes, applying SE stds, A community argue about more formal approach for s/w quality, computer programs a mathematical object, mathematical proof of correctness applied to check program conformance

26.6 STATISTICAL QUALITY ASSURANCE :

Reflects quality quantitatively, achieved by

1. Collect & categorize info about each s/w defects
2. Track defects to underlying cause
3. Use pareto principle (80 –20), isolate 20%
4. Solve problems causing defects

A s/w development firm collects info of defects for 1 year, errors during development & defects after release, hundreds of errors tracked to one or more of following few causes :

1. Incorrect/erroneous specification (IES)
2. Misinterpreted customer communication (MCC)
3. Intentional deviation from specification (IDS)
4. Violation of programming stds (VPS)
5. Error in data representation (EDR)
6. Inconsistent module interface (IMI)
7. Error in design logic (EDL)
8. Incomplete / erroneous testing (IET)
9. Inaccurate/incomplete docum (IID)
10. Error in programming lang translation of design (PLT)
11. Ambiguous/inconsistent (h)-comp i/f (HCI)

12. Miscellaneous (MIS)

total			Serious	Moderate	Minor
Error	no	%			
IES	205	22%			
MCC	156	17%			
EDR	130	14%			

A table to apply stat QA, indicates IES, MCC & EDR causes 53% of all errors, once vital few causes identified, corrective action, for MCC, technique to improve quality of customer communication & specification, for EDR, CASE tools, new causes moves upward, use of Statistical QA improves quality, up to 50% reduction in defects

Error index ET calculated for major steps, after analysis, design, coding, testing & imle, following data gathered:

Ei = no, of errors found during ith step of SE process
Si = no of serious errors
Mi = moderate error
Ti = minor errors
PS = product size
Ws, Wm, Wt = Weighting factors
Recommended Ws = 10, Wm = 3 & Wt = 1

Phase index is computed at each step as

$$PIi = Ws\,(Si / Ei) + Wm\,(Mi / Ei) + Wt\,(Ti / Ei)$$

Error index is calculated using cumulative phase index :

$$EI = \Sigma\,(I * PIi) / PS$$

Error index & info of table to develop indication of improvement in s/w quality, Stat QA isolates vital few causes for defects, appropriate corrections

26.6.2 Six Sigma of S/w Engineering :

26.7 S/W RELIABILITY :

An imp element of quality, if program repeatedly fails to perform, other quality elements of little consideration, measured using historical & development data; statistically reliability is the prob of failure free operation in specified env for a specified time. 96% prob, failure is nonconformance to requirement

26.7.1 Measures Of Reliability & Availability :

S/w failure due to design & implementation problem, reliability measurement mean time b/w failure is considered

$$MTBF = MTTF + MTTR$$

Better than defects/KLOC for user, each defect do not lead to failure, many errors remain undetected for years, less impact on reliability, s/w availability, prob that a program is operating according to requirements at a given point of time

$$Availability = [MTTF / (MTTF + MTTR)] * 100\%$$

Indirect measure of maintainability

26.7.2 S/w Safety:

S/w as a part of control system, complexity increases, design faults difficult to uncover, s/w safety- an SQA activity that focus on identification & assessment of potential hazards that may affect s/w negatively & cause entire system to fail, early detection lead to elimination /control

A modeling & analysis process, identification of hazard, categorization by criticality & risk, ex hazards for computer assisted automobile control can be :

1. Uncontrolled acceleration, can not be stopped
2. Do not disengage on brake depression
3. Speed lose/gain slow

Once hazard identified, analysis technique to assign severity & prob of occurrence, s/w analyzed in context of entire system, ex errors in user i/p, fault tree analysis, real-time logic or petri net model applied to predict chain of events causing hazard & prob of each event

Next specify safety-related requirements – a list of undesirable events & desires system response to these events, role of s/w in managing undesirable events identified
S/w reliability & s/w safety – related but different from each other, reliability use statistical analysis to find prob of s/w failure, failure may not result in hazard, safety examines failures that lead to hazardous situation

26.8 ISO 9000 QUALITY STANDARDS :

A QA system defined as organizational structure, responsibilities, procedures, processes & resources for implementing Q M, such system help org to ensure their product & services satisfy customer expectation by meeting their specifications, system covers entire life cycle of product – planning, controlling, measuring, testing & reporting, & improving quality
ISO 9000 describes quality assurance elements in general terms, applied to any business, for registration, organization's quality system & operations scrutinized by auditors for compliance to stds, issued certificate

ISO approach to Quality Assurance System :
Treat an enterprise as a n/w of interconnected processes, processes must address stds & documentation practice, docu helps org to understand, control & improve processes, elements included in ISO-9000 QA system are – organizational structure, procedures, processes & resources to implement quality planning, quality control, quality assurance & improvement, it do not describe how to implement these elements

ISO 9001 standard : applies to all engineering discipline including SE, contains 20 requirements that must be present for effective QA system are :
1. Management responsibilities
2. Quality system
3. Contract review
4. Design control
5. Document & data control

6. Purchasing
7. Control of customer supplied product
8. Product identification & traceability
9. Process control
10. Inspection & testing
11. Control of inspection, measuring & test equipments
12. Inspection & test status
13. Control of nonconforming product
14. Corrective & preventive action
15. Handling storage, packaging, preservation & delivery
16. Control of quality records
17. Internal quality audits
18. Training
19. Servicing
20. Statistical techniques

S/w org must establish policies & procedures to fulfill each requirement

26.9 THE SQA PLAN :

A roadmap for s/w QA, Developed by SQA group & project team, IEEE recommended SQA plan :

- I. Purpose of plan
- II. References
- III. Management
  1. Organization
  2. Tasks
  3. Responsibilities
- IV. Documentation
  1. Purpose
  2. Required SE documents
  3. Other documents
- V. Standards, practices & conventions
  1. Purpose
  2. Convention
- VI. Reviews & audits
  1. Purpose

2. Review requirements
    a. S/w requirements review
    b. Design review
    c. S/w verification & validation review
    d. Functional audits
    e. Physical audits
    f. In-process audits
    g. Management reviews

VII. Test
VIII. Problem reporting & corrective action
IX. Tools, techniques & methodology
X. Code control
XI. Media control
XII. Supplier control
XIII. Record collection, maintenance & retention
XIV. Training
XV. Risk management

Initially purpose & scope of documents, list of s/w activities to be covered by SQA, management section describes place of SQA in org, SQ tasks & activities, responsibilities for product quality, documentation describes each work product produced by s/w process - project plan, ERD, specifications & user documents
Std, practice & convention list all std applied for s/w process & metrics collected, review & audit lists reviews & audits to be conducted by SQA group, test refers to s/w test plan & procedures, problem reporting & correction, defines procedure for reporting, tracking & error/defect resolving, remaining parts identifies tools & methods to support SQA activities, s/w configuration mgmt procedures etc.

SQA is umbrella activity applied at each step of s/w process. SQA encompasses procedures for effective application of methods & tools, FTR, testing strategies & techniques, procedures for change control, procedures for assuring compliance with stds & measurement & reporting mechanism

9. S/W CONFIGURATION MANAGEMENT

Changes during s/w development, if not analyzed create confusion, SCM an umbrella activity, change may occur at any time, activities are (1) identify change (2) control change (3) ensure change properly implemented (4) report to others, not maintenance activity, SCM with project beginning, goal is to accommodate changes with ease, reduce efforts

SCM :

Information o/p from s/w process, 3 categories (1) computer programs (2) data (3) documents. SC comprises items of these process o/p, with process progression SC items increases, ex system specification item grows to s/w project plan & requirement specification, each SCI spawn other SCI, change leads to confusion, sources of change are :

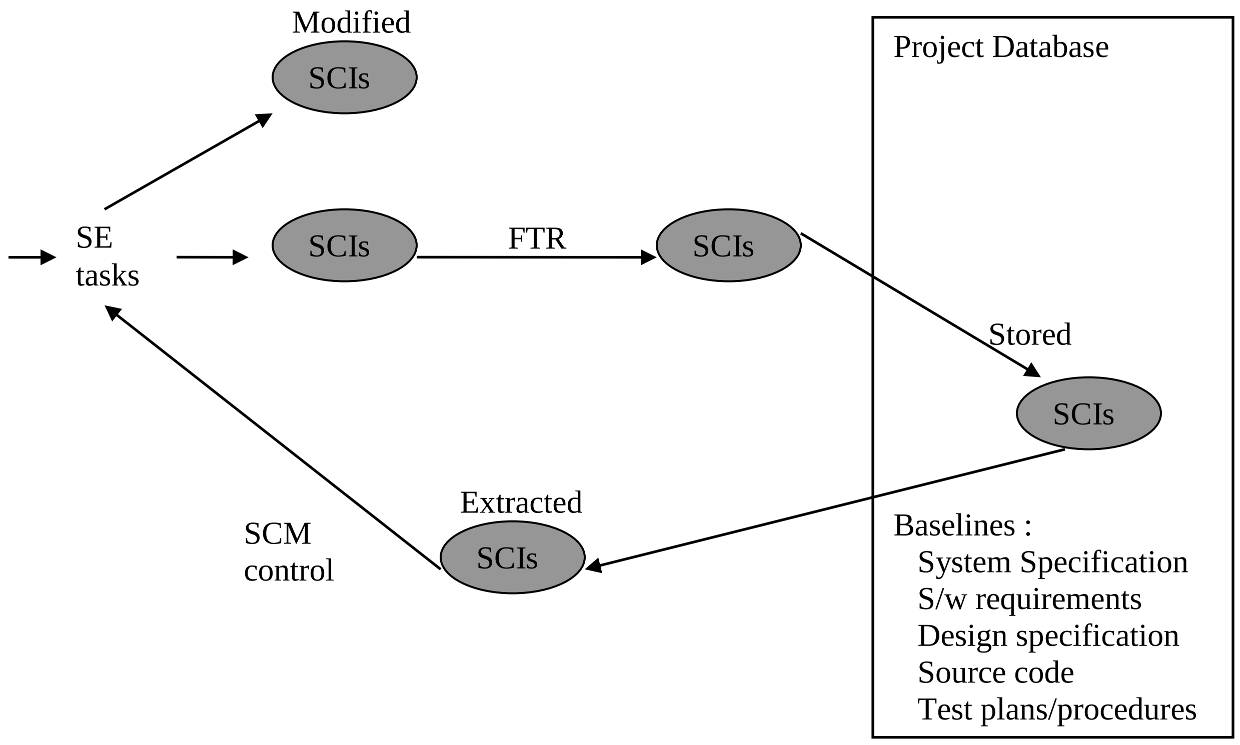
(1) new business or market condition leads to change in product requirement/business rules
(2) new customer needs, modification of data produced, functionality or service delivered by system
(3) reorganization or downsizing of business, project priorities/ SE team organization
(4) budgetary/ scheduling constraints, redefinition of system/product

SCM manage change throughout life of s/w

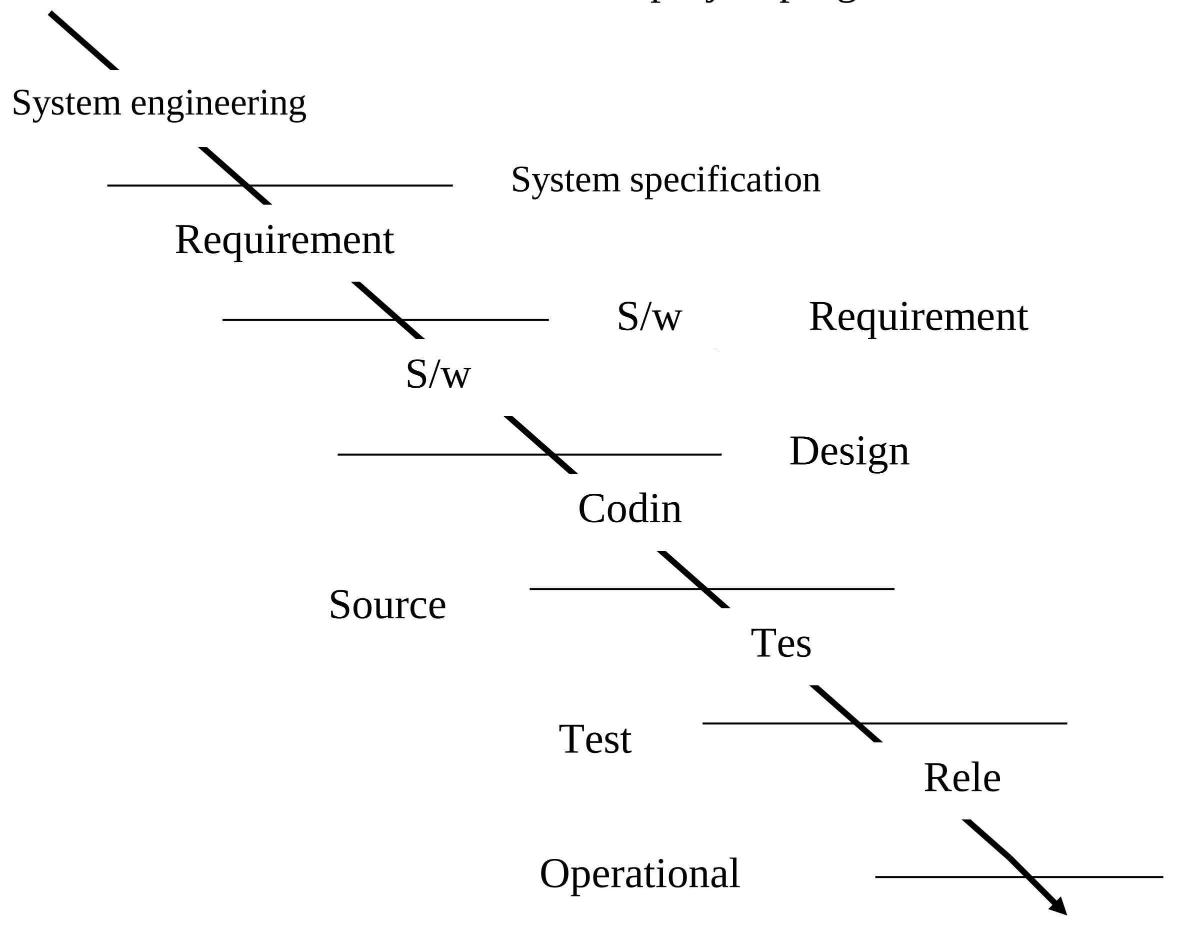
BASELINE :

Change in customer requirement, developer change technical approach, managers modify strategy, knowledge is driving force behind changes, all changes are not justified, baseline – a s/w CM concept help control change without serious impeding justifiable change,
IEEE definition : a specification/product that has been formally reviewed & agreed upon that thereafter serves as the basis for further development & can be changed through formal change control procedure
Before SC item becomes a baseline, changes can be quick & informal, but after establishing baseline change through formal procedure

Baseline a milestone in SE, marked by one or more s/w configuration items, ex elements of design specification, documented & reviewed, errors corrected, approved to become a baseline, further changes in architecture after change evaluation & approval



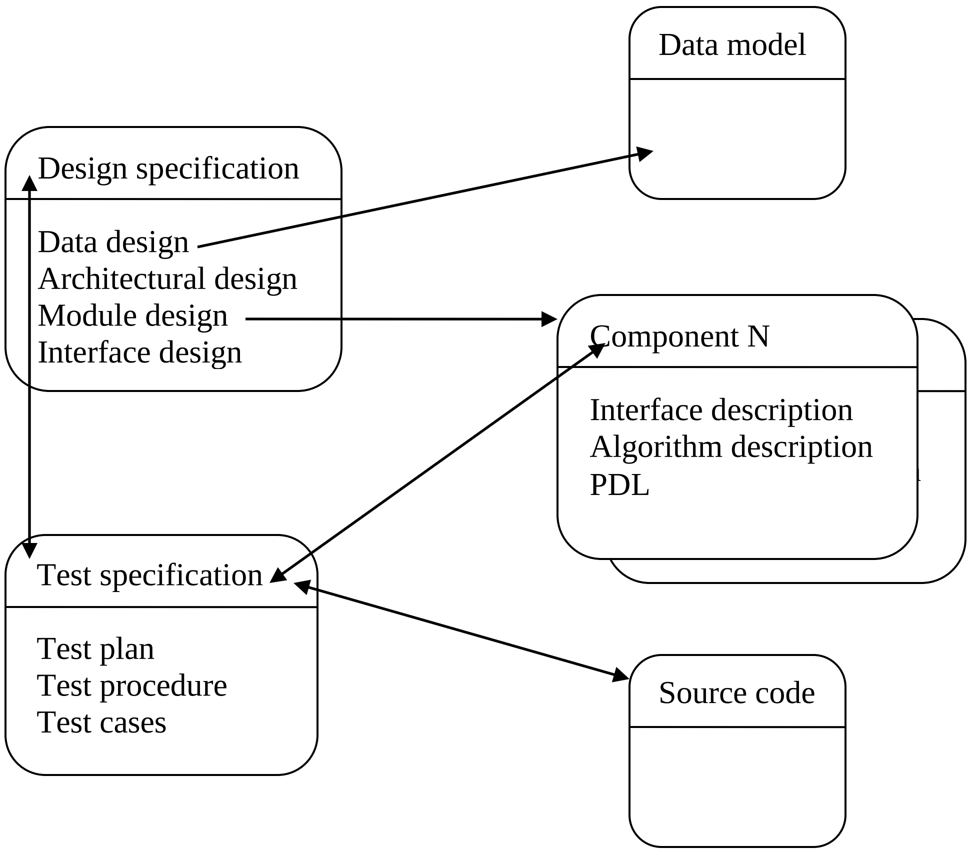
Creation of baseline with different events as project progresses



SE tasks produce many SCI, after review & approval placed in project database/library, when change in SCI, brought from database, modified if SCM controls followed

SC ITEMS :

Sc items- Info created as part of s/w process, can be document, testcase or program component, s/w tools used for development also under SCI – editor, compiler etc, SCI are organized to form configuration object, which has name, attributes & connection to other objects,



Each defined separately but related to others, data model & components are part of object design specification, other modules are interrelated, change in one object, interrelationship enable to determine other affected objects.

SCM technique applied to following SCI :

1. System specification
2. S/w project plan
3. S/w requirements specification
    a. Graphical analysis model
    b. Process specification
    c. Prototypes
    d. Mathematical specifications
4. Preliminary user manual
5. Design specifications
    a. Data design descriptions
    b. Architectural design descriptions
    c. Module design descriptions
    d. I/f design description

    e. Object descriptions
6. Source code listing
7. Test specification
    a. Test plan & procedure
    b. Test cases & recorded results
8. Operation & installation manuals
9. Executable program
    a. Module executable code
    b. Linked modules
10. Database description
    a. Schema & file structure
    b. Initial content
11. As-built user manual
12. Maintenance documents
    a. S/w problem reports
    b. Maintenance requests
    c. Engineering change orders
13. Standards & procedures for s/w engineering

Additionally, s/w tools under configuration control, compilers, editors, CASE tools etc, because they were used to produce source code, documents, they must be available at the time of change