Software Metrics
&
Software Reliability

UNIT III

 U3. 1

Learning Objectives

- Software Metrics
  - Software measurements
  - What & Why
  - Token Count
  - Halstead Software Science Measures
  - Design Metrics
  - Data Structure Metrics
  - Information Flow Metrics

 U3. 2

Learning Objectives

- Software Reliability
  - Importance
  - Hardware Reliability & Software Reliability
  - Failure and Faults
  - Reliability Models
  - Basic Model
  - Logarithmic Poisson Model
  - Software Quality Models
  - CMM & ISO 9001

 U3. 3

Software Metrics and Measurements

Objectives

- What is Measurement
- Metrics
- Why Measure
- Question of interest
- Total Quality Management (TQM)
- Types of Software Metrics
- Metrics Program Approach
- Size Metrics
  - LOC
  - FPA
  - Halstead's Software Science
  - Data Structure Metrics
- Information flow metrics

What is Measurement

- Process by which numbers are assigned to attributes of entities in the real world to describe them.
- An entity is an object (e.g. person, room) or an event (e.g. testing phase) in the real world.
- An attribute is a feature or property of an entity.
- Each of the software entity may have multiple attributes. (e.g. code inspected, number of defects found, duration of the project.)

Metrics

- Metrics are measurements
- collection of data about
  - project activities
  - Resources
  - deliverables.
- A unit of measurement of a software product or software related process
- All engineering discipline have metrics.
- Metrics help
  - estimate projects
  - measure project progress
  - measure quality

Why Measure

- When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers,your knowledge is of a meager and unsatisfactory kind.

  - *Lord Kelvin*
- You can't control, what you can't measure

  - *De Marco*
- If you wish to improve, you have got to measure it

  -Both ISO 9000 and CMM(I) require Metrics

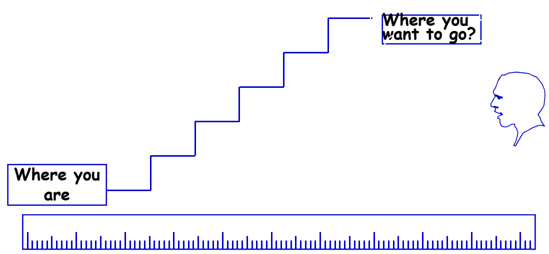
Question of Interest

- How to measure the size of a software?
- How much will it cost to develop a software?
- How many bugs can we expect?
- When can we stop testing?
- When can we release the software?
- What is the complexity of a module?
- What is the module strength & coupling?
- What is the reliability at the time of release?
- Which test technique is most effective?
- Are we testing hard or are we testing smart?
- Do we have a strong program or a weak test suite?

Total Quality Management (TQM)

- A style of management aimed at achieving long-term success by linking quality with customer satisfaction.
- Basic key elements
  - customer focus
  - continuous improvement
  - *measurement and analysis*

Continuous Improvement



MEASURE - ANALYSE - IMPROVE

Types of Software Metrics (1)

Direct Metrics	Indirect Metrics
Size (LOC or FP)	Efficiency
Cost	Productivity
Effort	Reliability
Errors	Functionality
Defects	Complexity
	Maintainability

Types of Software Metrics

- Process
- Product
- Project

Process Metrics

- Effort required in the process
- Time to produce product
- Effectiveness of defect removal during development
- Number of defects found during testing
- Maturity of process

Project Metrics

Describe the characteristics of the project & its execution

- Schedule
- Size
- Effort
- No. of Software developers
- Staffing pattern over the life cycle of the product
- Productivity

Product Metrics

Describe the characteristics of the product

- Complexity
- performance
- Functionality, Usability, Efficiency, Reliability, Portability, Maintainability (ISO 9126)
- Defect Density
- Size etc.

Measuring Quality

- Correctness — the degree to which a program operates according to specification
- Maintainability—the degree to which a program is amenable to change
- Integrity—the degree to which a program is impervious to outside attack
- Usability—the degree to which a program is easy to use

Metrics Program Approach

- Establish the goal of data collection
- Develop a list of questions of interest
- Establish Data categories
- Design and test data collection form
- Collect and validate data
- Analyse data

Size Metrics

- Allows for estimation of effort, time scale & total number of faults
- Lines of code (LOC)
- Useful if, same language & coding style etc.
- Token Count
- Function Point

Size Metrics

```
SUBROUTINE SORT(X,N)
INTEGER X(100),   N, I, J, SAVE, IM1
C THIS ROUTINE SORTS ARRAY X INTO ASCENDING
C ORDER
IF (N.LT.2) GOTO 200
 DO 210 I = 2, N
    IM1 = I-1
    DO 220 J = 1, IM1
       IF (X(1) .GE. X(J)) GOTO 220
          SAVE = X(1)
           X(I) = X(J)
           X(J) = SAVE
220   CONTINUE
210    CONTINUE
200     RETURN
END
```


Size Metrics

- Halstead Software Science Metrics/Token Count
- To evaluate mental effort & time required to create a program
- How compactly a program is expressed

Halstead's Software Science

- A comprehensive collection of metrics
- Predicated on the number of operators and operands within a component or program
- Count
- Occurrence
- total of these tokens

 U3. 22

Software Science Measures cont..

Program Length, $N = N_1 + N_2$

Vocabulary, $n = n_1 + n_2$

Predicted Length, $N\text{^} = (n_1 * \log_2 n_1) + (n_2 * \log_2 n_2)$

Program Volume, $V = N * \log_2 n$

Potential volume, $V^* = (2 + n_2^*) \log_2 (2 + n_2^*)$

- program with minimum size

n_1 / n_2 - Number of unique operators / operands
N_1 / N_2 -Total occurrences of operators /operands

 U3. 23

Software Science Measures cont..

Program Level, L = V*/V

- Ranges 0-1, highest possible level is 1

Estimated Level, L^ =2 n2 / (n1 N2)

Difficulty, D= 1/L

Estimated Difficulty, D^= 1/L^

n1 / n2 - Number of unique operators / operands

N1 / N2 -Total occurrences of operators /operands

 U3. 24

Software Science Measures cont..

Effort, $E = V/\hat{L} = V * \hat{D}$ elementary mental discrimination

Time, $T = E/\hat{a}$; $\hat{a} = 18$ (John Stroud 5-20 per second)

Predicted number of bugs $B = V/3000$

Language level, $\lambda = L^* V^* = L^2 V$

n_1 / n_2 - Number of unique operators / operands

N_1 / N_2 -Total occurrences of operators /operands

Size Metrics

```
SUBROUTINE SORT(X,N)
INTEGER X(100),   N, I, J, SAVE, IM1
C THIS ROUTINE SORTS ARRAY X INTO ASCENDING
C ORDER
IF (N.LT.2) GOTO 200
 DO 210 I = 2, N
    IM1 = I-1
    DO 220 J = 1, IM1
       IF (X(1) .GE. X(J)) GOTO 220
          SAVE = X(1)
          X(I) = X(J)
          X(J) = SAVE
220   CONTINUE
210   CONTINUE
200    RETURN
END
```


Size Metrics

Operator	Occurrences	Operands	Occurrences
SUBROUTINE	1	SORT	1
()	10	X	8
,	8	N	4
INTEGER	1	100	1
IF	2	I	6
.LT.	1	J	5
GOTO	2	SAVE	3
DO	2	IM1	3
=	6	2	2
-	1	200	2
.GE.	1	210	2
CONTINUE	2	1	2
RETURN	1	220	3
End of line	13	-	-
$n_1 = 14$	$N_1 = 51$	$n_2 = 13$	$N_2 = 42$

Software Science Metrics

$N_1 = 51$; $N_2 = 42$; Calculate

- Program Length
- Vocabulary
- Program Volume
- Estimated Statement Count
- Predicted Length
- Potential Volume
- Program Level
- Estimated Level
- Difficulty
- Estimated Difficulty
- Effort
- Time

 U3. 28

Software Science Metrics cont..

$N_1 = 51$; $N_2 = 42$

Program Length = $N_1 + N_2 = 93$

Vocabulary, $n = n_1 + n_2 = 14+13 = 27$

Program Volume, $V = N * \log_2 n = 93 * \log_2 27 = 442$ bits

Estimated Statement Count $S_s = N/C = 93/7=13$

- C is constant 7 for FORTRAN

 U3. 29

Software Science Metrics cont..

$N_1 = 51$; $N_2 = 42$

Predicted Length, $N\hat{}= (n_1 \log_2 n_1) + (n_2 \log_2 n_2)$

$= 14*\log_2 14+13* \log_2 13$

$= 14*3.81+13*3.70 = 101.45$

n_2^*,Unique input output parameter= 3

- X: array holding the integer to be sorted, used as an input & output
- N: the size of the array to be sorted

 U3. 30

Software Science Metrics cont..

Potential volume, program with min. size ,

- $V^* = (2+n_2^*) \log_2 (2+n_2^*)$
  $=5 \log_2 5 = 11.6$

Program Level, L $= V^*/V$
$= 11.6/442 = 0.026$

Estimated Level, $L^\wedge = 2 n_2 / (n_1 N_2) = (2*13)/(14*42) = 0.044$

Difficulty, $D = 1/L = 1/0.026 = 38.5$

Estimated Difficulty, $D^\wedge = 1/L^\wedge = 1/0.044 = 22.72$

Software Science Metrics cont..

Effort, $E = V/L^\wedge = V * D^\wedge$
$= 442/0.044$
$= 10,045$

Time, T $= E/\hat{a}$
$= 10045/18$
$= 558$ sec.
$= 10$ minutes

Data Structure Metrics

- A count of the amount of data input to, processed in and output from software
- Proposed metrics
  - Amount of data
  - The usage of data within modules
  - Degree to which data is shared among modules

Examples of Input, Internal and Output Data

Program	Data input	Internal data	Data Output
Payroll	Name/SSN/Pay Rate/ No. of hrs Worked	Withholding Rates Overtime Factors Insurance premium rates	Gross pay withholding, net pay, pay ledger

Amount of Data

```
SUBROUTINE SORT(X,N)
INTEGER X(100),   N, I, J, SAVE, IM1
C THIS ROUTINE SORTS ARRAY X INTO ASCENDING
C ORDER
IF (N.LT.2) GOTO 200
  DO 210 I = 2, N
      IM1 = I-1
      DO 220 J = 1, IM1
          IF (X(1) .GE. X(J)) GOTO 220
                SAVE = X(1)
                 X(I) = X(J)
                 X(J) = SAVE
220  CONTINUE
210   CONTINUE
200     RETURN
END
```

Amount of Data

- Count the number of entries for variables in the cross reference list.
  - Excludes the variables that are defined but never used
- Count of variables is referred as VARS
- For the sample program SORT
  - VARS = 6
  - X, N, I, J, SAVE, IM1
- Definition of VARS
  - A variable is a string of alphanumeric characters that is defined by a developer and that is used to represent some value during compilation or execution

Amount of Data

1	Program payday (input, output)
2	type check = record
3	gross: real;
4	tax : real
5	net : real
6	end;
7	var pay : check;
8	hours, rate : real;
9	Begin
10	while not eof(input) do begin
11	readln (hours, rate);
12	pay.gross := hours * rate;
13	pay.tax := 0.25 * pay.gross;
14	pay.net := pay.gross – pay.tax;
15	writeln (pay.gross, pay.tax, pay.net)
16	End
17	end

Amount of Data cont..

Check	2	7			
Gross	3	12	13	14	15
Hours	8	11	12		
Net	5	14	15		
Pay	7	12	13	13	14
	14	14	15	15	15
Rate	8	11	12		
tax	4	13	14	15	

Amount of Data cont..

Halstead introduce a metric that he referred to as n2 that includes all variables, constants and labels

n2 = VARS+ unique constants+labels

For payday program

- 7 variables (check, gross, hours, net, pay, rate, tax)
- 1 constant (0.25)
- 1 label (payday)
- n2 = 9

N2 total occurrences of operands

- For payday, N2 = 32

Amount of Data

```
SUBROUTINE SORT(X,N)
INTEGER X(100),   N, I, J, SAVE, IM1
C THIS ROUTINE SORTS ARRAY X INTO ASCENDING
C ORDER
IF (N.LT.2) GOTO 200
  DO 210 I = 2, N
      IM1 = I-1
      DO 220 J = 1, IM1
          IF (X(1) .GE. X(J)) GOTO 220
                SAVE = X(1)
                X(I) = X(J)
                X(J) = SAVE
220   CONTINUE
210    CONTINUE
200     RETURN
END
```

Amount of Data cont..

Halstead introduce a metric that he referred to as n_2 that includes all variables, constants and labels

n_2 = VARS+ unique constants+labels

For sort program

- 6 variables (X,N,I,J, SAVE,IM1)
- 3 constants (1,2,100)
- 4 labels (SORT, 200, 210, 220)
- $n_2 = 13$

The Usage of Data within a Module

- Live Variables
  - More data items that a programmer must keep track of when constructing a statement, the more difficult it is to construct
- Interest is to find live variables
- **A variable is live**
  - From the beginning of a procedure to end of the procedure
  - At a particular statement only if it is referenced a certain no. of statements before and after that statement
  - From its first to its last reference within a procedure

Usage of Data

Line	
01	SUBROUTINE SORT(X,N)
02	INTEGER X(100), N, I, J, SAVE, IM1
03	C THIS ROUTINE SORTS ARRAY X INTO ASCENDING
04	C ORDER
05	IF (N.LT.2) GOTO 200
06	DO 210 I = 2, N
07	IM1 = I-1
08	DO 220 J = 1, IM1
09	IF (X(1) .GE. X(J)) GOTO 220
10	SAVE = X(1)
11	X(I) = X(J)
12	X(J) = SAVE
13	220 CONTINUE
14	210 CONTINUE
15	200 RETURN
16	END

Live Variables for SORT Program

Line	Live Variable	Count
5	N	1
6	N, I	2
7	I, IM1	2
8	I, IM1, J	3
9	I, J, X	3
10	I, J, X, SAVE	4
11	I, J, X, SAVE	4
12	J, X, SAVE	3
		22

Live Variables for SORT Program cont..

Avg. No. of live variables

= Total Live Variables/ count of executable statements

Variable Spans

Two variables can be alive for the same number of statements, but their use in a program can be markedly different.

```
   -------
21 read (a,b);
   --------
32 x:= a;
   --------
45 y:=a-b;
   --------
53 z:= a;
   ---------
60 writeln(a,b)
   ---------
```

How often a variable is used is called variable span (SP).

Variable Spans cont..

Number of statements between two successive references to the same variable

a = 4 spans

- 10, 12, 7, 6
- Avg. span size of a = 8.75

b = 2 spans

- 23, 14
- Avg. span size of b = 18.5

Making Program-wide Metrics

To characterize the average number of live variables for a program having modules

$LV_{program} = \Sigma^{m}_{i=1} LV_i/m$

- LV_i is the average live variable metric computed from the i^{th} module

The average span size SP for a program of n spans

$SP_{program} = \Sigma^{n}_{i=1} SP_i/n$

Program Weakness

Weakness of module can be computed using formula

$WM = LV * \gamma$

LV : average number of live variables

LV = Sum of count of live variables/ Count of exe. statements

γ : average life of variables

γ = Sum of count of live variables/ Total No. of variables

Program Weakness cont..

Can be used to estimate the testability and maintainability

Program weakness

- $WP = (\Sigma_{i=1}^{m} WM_i)/m$
- WM_i : weakness of ith module
- WP : weakness of the program
- M : number of modules in the program

Simply average the weakness of various modules doesn't consider the effect of module coupling

Program Weakness (WP)

Line No.	Program Instructions
1	Program FIRST (input, output);
2	procedure module_compute;
3	var a, b, c : integer;
4	begin
5	readln(a,b);
6	a := a + 5;
7	b := b - 5;
8	c := a * b;
9	writeln(a, b,c);
10	end;
11	begin
12	module_compute;
13	end.

WP of First Program (compute_module)

Line No.	Live Variables in the line	Count
4	---	0
5	a,b	2
6	a,b	2
7	a,b	2
8	a,b,c	3
9	a,b,c	3
10	---	0
	Total Count =	12

WP of First Program (compute_module) cont..

- Total number of executable statement in module_compute: 7
- Total number of variables in module_compute : 3
- LV1 (for module_compute) = 12/7
- γ1 (for module_compute) = 12/3 = 4
- WM1(for module_compute) = LV1* γ1 = 12/7 * 4 = 48/7

Program Weakness of First Program (main)

Line No.	Live Variables in the Line	Count
11	---	0
12	---	0
13	---	0
	Total count =	0

WP of First Program (main) cont..

Total number of executable statement in main: 3

Total number of variables in main : 0

LV2 (for main) = 0

γ2 (for main) = 0

WM2(for main) = LV2* γ2 = 0

WPFIRST = (WM1+WM2)/2 = (48/2+0)/2 = 3.43

Example

Line No.	Program Instructions
1	Program SECOND (input, output);
2	var a, b, c : integer;
3	procedure module_compute(var a, b, c : integer);
4	begin
5	a := a + 5;
6	b := b - 5;
7	c := a * b;
8	end;
9	begin
10	readln(a,b);
11	module_compute(a, b, c);
12	writeln(a, b,c);
13	end.

WP of Second Program (compute_module)

Line No.	Live Variables in the line	Count
4	---	0
5	a	1
6	a,b	2
7	a,b,c	3
8	---	0
	Total Count =	6

WP of 2nd Program (compute_module) cont..

- Total number of executable statement in module_compute: 5
- Total number of variables in module_compute : 3
- LV1 (for module_compute) = 6/5
- γ1 (for module_compute) = 6/3 = 2
- WM1(for module_compute) = LV1* γ1
  = 6/5 * 2 = 12/5

Information Flow Metrics

- Component : Any element identified by decomposing a system into its constituent parts
- What the components do and how they are fitted together , influences the complexity of the system
- Cohesion : the degree to which a component performs a single function
- Coupling : the term used to describe the degree of linkage between one component to others in the same system

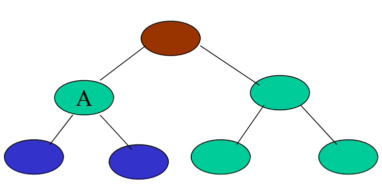
Information Flow Metrics cont..

Information flow metrics model the degree of cohesion and coupling for a particular system component;

- original work done by Henry and Kafura

Basic information flow model

- IF metrics are applied to the component of system design



Information Flow Metrics cont..

- The simplest model of IF for component A
- "FAN IN" is simply a count of the number of other components that can call, or pass control, to component A
- "FAN OUT" is the number of components that are called by component A
- Using the following formula we can derive a measure called as INFORMATION FLOW index of component A, abbreviated as IF(A)

$$IF(A) = [FAN\ IN(A) * FAN\ OUT(A)]^2$$

- Include a power component to model the non- linear nature of complexity

More Sophisticated Information Flow Model

- Ince and shepperd and Kitchenham have done a lot of work to help in the practical application of Henry and Kafura's proposal
- All these work is summarized by Goodman in to a more sophisticated IF model
- The only difference between the simple and the sophisticated IF models lies in the definition of FAN IN and FAN OUT

More Sophisticated Information Flow Model..

For a Component A let :

a = the number of components that call A

b = the number of parameters passed to A from components higher in the hierarchy

c = the number of parameters passed to A from components lower in the hierarchy

d = the number of data elements read by component A.

Then :

FAN IN(A) = a + b + c +d

More Sophisticated Information Flow Model..

Also let :

e = the number of components called by A

f = the number of parameters passed from A to components higher in the hierarchy

g = the number of parameters passed from A to components lower in the hierarchy;

h = the number of data elements written to by A.

Then:

FAN OUT (A) = e + f + g + h

 U3. 64

What we Learnt

- ✓ What is Measurement
- ✓ Metrics
- ✓ Why Measure
- ✓ Question of interest
- ✓ Total Quality Management (TQM)
- ✓ Types of Software Metrics
- ✓ Metrics Program Approach
- ✓ Size Metrics
- ✓ Information flow metrics

 U3. 65

Software Reliability

Probability of Failure-Free Operations of a Computer Program for a Specified time in a Specified Environment

 U3. 66

Learning Objectives

- Basic concepts
- Software Quality
  - McCall Software Quality model
  - Boehm Software quality Model
  - ISO 9126
- Software Reliability Models
  - Basic Execution time Model
  - Logarithmic Poisson Execution Time Model
  - ✓ Calendar Time Component
  - ✓ Resource Usage

 U3. 67

Software Reliability – Alternate Definitions

- Informally denotes a product's trustworthiness or dependability.
- Probability of the product working "correctly" over a given period of time

 U3. 68

Software Reliability

- a software product having a large number of defects is unreliable.
- reliability of a system improves if the number of defects is reduced.

 U3. 69

Reliability

- User's concern
  - An important attribute determining the quality of the product.
- Demand for quantitative estimation of reliability before making buying decision.

Hardware vs. Software Reliability

- Hardware failures inherently different from software failures.
- Most hardware failures are due to component wear and tear:
  - some component no longer functions as specified.

Hardware vs. Software Reliability..

- A logic gate can be stuck at 1 or 0,
  - or a resistor might short circuit.
- To fix hardware faults:
  - replace or repair the failed part.
- Software faults are latent:
  - system will continue to fail unless changes are made to the software design and code.

Hardware vs. Software Reliability..

- Because of the difference in effect of faults, metrics appropriate for hardware reliability measurements are not good software reliability metrics

Hardware vs. Software Reliability

When a hardware is repaired:

- its reliability is maintained

When software is repaired:

- its reliability may increase or decrease.

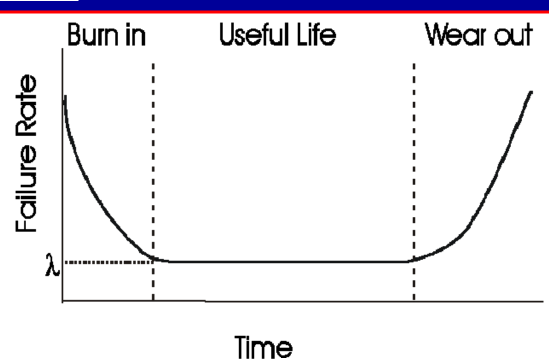
Goal of hardware reliability study :

- stability (i.e. interfailure times remains constant)

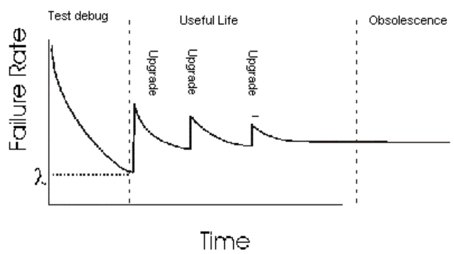
Goal of software reliability study

- reliability growth (i.e. interfailure times increases)

Bathtub Curve for Hardware Reliability



Revised Bathtub Curve for Software Reliability



Why Software Obsolete

- Change in environment
- Change in infrastructure/technology
- Major change in requirements
- Increase in complexity
- Extremely difficult to maintain
- Deterioration in structure of the code
- slow execution speed
- Poor graphical user interfaces

Importance

Most important characteristics of software

- Quality
- Cost
- Schedule

How to measure quality?

- Reliability
  - ✓connected with defects
  - ✓clearly observer-dependent
  - ✓cannot be determined absolutely

Failures and Faults

- Failure
- Program in its functioning has not met user requirements in some way
- Fault
  - The defect in a program that, when executed under particular conditions, causes a failure
  - Can be source of more than on e failure
  - Property of program rather than property of its execution or behaviour

Mean Value Function

- Reliability quantities usually defined with respect to time
- Three types of time
  - Execution time
  - Calendar time
  - Clock time (wait time + execution time)

Mean Value Function cont..

- Four general ways of characterizing failure occurrences in time
  - Time of failure
  - Time interval between failure
  - Cumulative failure experienced up to a given time
  - Failure experienced in a time interval

Time Based Failure Specification

Failure Number	Failure Time (Sec)	Failure Interval (Sec)
1	8	8
2	18	10
3	25	7
4	36	11
5	45	9
6	57	12
7	71	14
8	86	15
9	104	18
10	124	20
11	143	19
12	169	26
13	197	28
14	222	25
15	250	28

 U3. 82

Failure Based Failure Specifications

Time (Sec)	Cumulative Failures	Failures In Interval (30 Sec)
30	3	3
60	6	3
90	8	2
120	9	1
150	11	2
180	12	1
210	13	1
240	14	1

 U3. 83

Probability Distribution At Times Ta And Tb

Value Of Random Variable (Failures In Time Period)	Probability	
	Elapsed time ta = 1 hr	Elapsed time tb = 5 hrs
0	0.10	0.01
1	0.18	0.02
2	0.22	0.03
3	0.16	0.04
4	0.11	0.05
5	0.08	0.07
6	0.05	0.09
7	0.04	0.12
8	0.03	0.16
9	0.02	0.13
10	0.01	0.10
11	0	0.07
12	0	0.05
13	0	0.03
14	0	0.02
15	0	0.01
Mean Failures	**3.04**	**7.77**

 U3. 84

Mean Value Function cont..

Two different viewpoints for the time variation

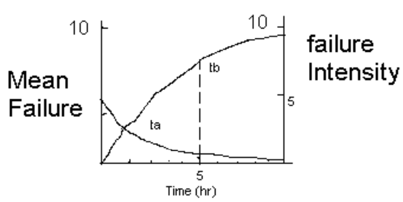
Mean value function

- Average cumulative failures associated with each time point

Failure intensity function

- Number of failures per unit time
- Rate of change of the mean value function
- 0.01 failure/hr or 1 failure/100 hr

Mean Value & Failure Intensity Functions cont..

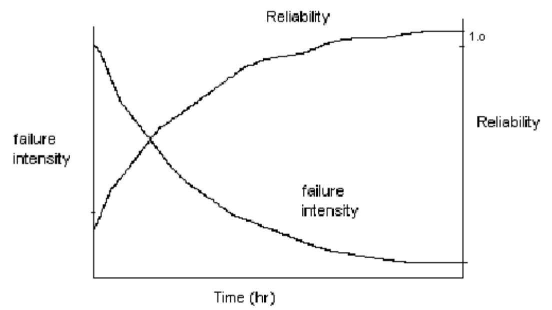


- Failure behavior affected by two principal factors
- The number of faults in the software being executed
- The execution environment or the operational profile of execution

Reliability and Failure Intensity

- Probability of failure-free operations of a computer program for a specified time in a specified environment. Example
- Time sharing system may have a reliability of 0.95 for 10 hrs when employed by the average user
- An equivalent statement is that the failure intensity of 0.05 failure/hr

Reliability and Failure Intensity



 U3. 88

Uses of Reliability Studies

- To evaluate software engineering technology quantitatively
- Evaluating development status during the test phases of a project
- To monitor the operational performance of software
- Quantitative understanding of software quality

 U3. 89

Quality

comparative- dictionary

a degree or level of innate excellence

quantitative - manufacturing

extent of departure from the standard, degree of conformity

fitness-for-purpose- Juran

a measure of usefulness and practicality

subjective - artistic...

a measure of the skill of the creator and value to the person for whom it was created

 U3. 90

Software Quality

- Conformance of requirements
- Fitness for the purpose
- Level of satisfaction
- Consider a software product:
  - functionally correct
  - but unusable user interface.
    - ✓Functional correctness alone does not determine quality

Modern View of Quality

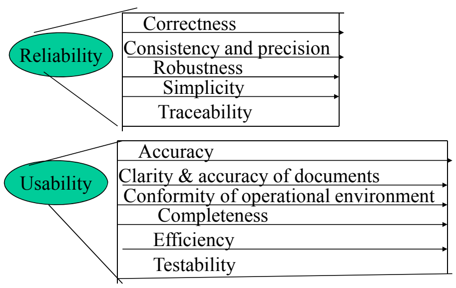
Associates several quality factors with a software product :

- Correctness
  - ✓**if different requirements as specified in the SRS document have been correctly implemented.**
  - ✓**Accuracy of results.**
- Reliability
- Efficiency
  - ✓**Amount of computing resources and code required by software to perform a function**
- Portability
  - ✓**Work on different operating systems**
  - ✓**Or on different machines**
  - ✓**Or with other software products**

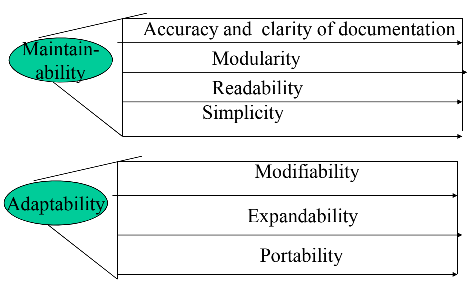
Modern View of Quality cont..

- Usability
  - ✓A software product has good usability, if different categories of users (i.e. both expert and novice users) can easily invoke the functions of the product.
- Reusability
  - ✓Different modules of the product can easily be reused to develop new products.
- Maintainability
  - ✓If faults can be corrected
  - ✓Functionalities can be added or modified (customized)

Software Quality Domain and Attributes



Software Quality Attributes



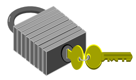
McCall Software Quality Model

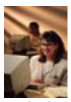
- Introduced in 1977
- Two levels of quality attributes
  - Quality Factors
  - Quality criteria
- Quality factors are organized in three product quality factors
  - Operation
  - Revision
  - Transition

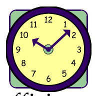
Operation Factors


correctness

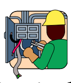

reliability

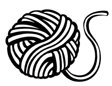

integrity


usability


efficiency

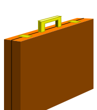
Revision Factors

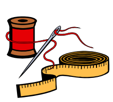

maintainability

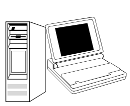

flexibility


testability

Transition Factors


portability


reusability


interoperability

McCall Quality Criteria

Usability
- Operability
- Training
- Communicativeness
- I/O volume
- I/O rate

Integrity
- Access Control
- Access Audit

Efficiency
- Storage efficiency
- Execution efficiency

Correctness
- Traceability
- Completeness
- Consistency

- **Reliability**
  - Accuracy
  - Error tolerance
  - Consistency
  - Simplicity
- **Maintainability**
  - Consistency
  - Simplicity
  - Conciseness
  - Self descriptiveness
  - Modularity
- **Testability**
  - Simplicity
  - Instrumentation
- **Flexibility**
  - Expandability
  - Generality
  - Self descriptiveness
  - Modularity
- **Reusability**
  - Generality
  - Self Descriptiveness
  - Modularity
  - Machine independence
  - Software system independence
- **Portability**
  - Self Descriptiveness
  - Modularity
  - Machine independence
  - S/W system independence
- **Interoperability**
  - Modularity
  - Comm. Commonality
  - Data Commonality

 U3. 100

Boehm Software Quality Model

- Introduced in 1978
- Defined three levels for quality attributes
  - Primary uses
  - Intermediate constructs
  - Primitive constructs
- Includes the needs and expectations of user as does McCall's also includes characteristics of hardware that are missing in McCall's Model

 U3. 101

Boehm Software Quality Model cont..

Primary uses
A. General utility
B. As is utility
C. Maintainability

Intermediate Constructs
1. Portability(A)
2. Reliability(B)
3. Efficiency (B)
4. Human Engineering (B)
5. Testability (A,C)
6. Understandability (A,C)
7. Modifiability (A,C)

- Primitive Constructs

Device independence(1)
Completeness(1,2)
Accuracy(2)
Consistency(2,6)
Device efficiency(3)
Accessibility(3,4,5)
Communicativeness(4,5)
Structured ness(5,6,7)
Self descriptiveness(5,6)
Conciseness(6)
Legibility(6)
Augment ability(7)

 U3. 102

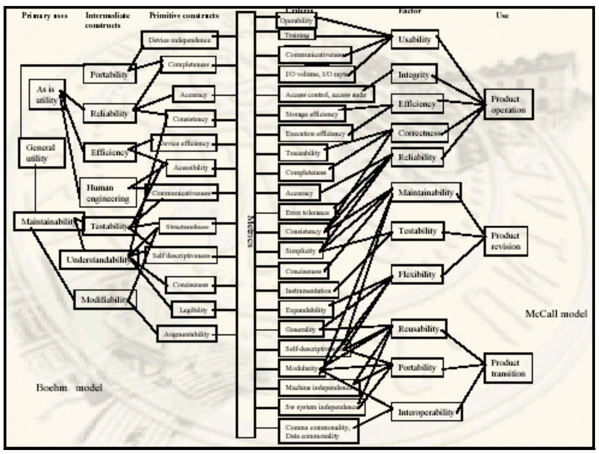
Boehm vs McCall Quality Model

Boehm

- **Basic user requirements(3)**: "as is" utility,general usability , maintainability
- **Quality factors (7):**portability, reliability, efficiency, usability, testability, understandability, flexibility
- **Quality characteristics(12**)

McCall

- **Basic user requirements(3)**: product operation,product revision, product transition
- **Quality characteristics(11):** usability, integrity, efficiency, correctness, reliability, maintainability, testability, flexibility, reusability, portability, interoperability
- **Quality characteristics(22)**



ISO 9126

- Software specialists have looked for a standard that would unambiguously define the quality attributes of the software
- 1991: ISO 9126: "Software Product Evaluation: Quality Characteristics and Guidelines for their Use"
- Consolidates many views of software QUALITY
- ISO hierarchy is strict and non-overlapping unlike McCall's and Boehm's

ISO 9126 cont..

Requirements for the quality characteristics in the new of standard:

- To cover together all aspects of software quality resulting from ISO quality definition
- To describe the product quality with a minimum of overlap
- To be as close as possible to the established technology
- To form a set of not more than six to eight characteristics for reason of clarity and handling
- To identify areas of attributes of software products for further refinement

Software Quality Characteristics in ISO 9126

- **Functionality**
  - to meet stated and implied need
- **Usability**
  - to be understood, learned and used
- **Reliability**
  - To maintain a specified level of performance
- **Portability**
  - To be adapted for different specified environment
- **Maintainability**
  - To be modified for the purposes of making corrections, improvements, or adaptations
- **Efficiency**
  - To provide appropriate performance relative to the amount of resources used

Quality Characteristics and Attributes ISO 9126

Functionality
- Suitability
- Accuracy
- Interoperability
- Security

Reliability
- Maturity
- Fault tolerance
- Recoverability

Usability
- Understandability
- Learnability
- Operability

•**Efficiency**
- –Time behaviour
- –Resource behaviour

•**Maintainability**
- –Analyzability
- –Changeability
- –Stability
- –Testability

•**Portability**
- –Adaptability
- –Installability
- –Conformance
- –Replaceability

Reliability Models

Principal factors that affect the software reliability

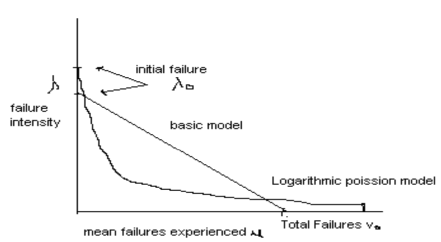
- Fault introduction
  - ✓Depends on characteristics of developed code and development process
- Fault removal
  - ✓Depends on time, operational profile and the quality of repair activity
- Environment
  - ✓Directly depends on operational profile

Software reliability model specifies the general form of the dependence of the failure process on above factors.

Reliability Models cont..

- Based on failure rather fault
- Execution time denoted by Γ
- Calendar Time denoted by T
- The variation in time between successive failures described in terms of function $\mu(\Gamma)$ denotes the average no. of failure upto time Γ
- $\lambda(\Gamma)$ = failure intensity function (Average no. of failure per unit time at time Γ
- **The reliability of program increases through fault correction and hence the failure intensity decreases**

Failure Intensity Functions for Basic Model(RBM) & Logarithmic Poisson Model (RLPM)



Failure Intensity Functions for RBM & RLPM..

RBM

- Decrement in failure intensity functions remains constant whether it is first failure that is being fixed or the last

RLPM

- Decrement in failure intensity functions becomes smaller with failure experienced; it decreases exponentially

Failure Intensity Functions for RBM

- Have failure intensity as a function of failures experienced is

$\lambda(\mu) = \lambda_0 [1- \mu/v_0]$

- λ_0 : initial failure intensity at the start of execution
- μ: the average or expected number of failures experienced at a given point in time
- v_0 :total number of failures that would occur in infinite time

Failure Intensity Functions for RBM Example

- Assume that a program will experienced 100 failures in infinite time. It has now experienced 50. The initial failure intensity was 10 failures/CPU hr. Determine the value of the current failure intensity

  $\lambda(\mu) = \lambda_0 [1- \mu/v_0]$

  $\lambda_0 = 10$; $\mu = 50$; $v_0 = 100$

  $\lambda(\mu) = 10[1\text{-}50/100] = 5$ failures/cpu hr.

Decrement of failure intensity per failure

$d\lambda/d\mu = -\lambda_0/v_0 = -10/100 = -0.1$/cpu hr.

Failure Intensity Functions for RLPM

- Have failure intensity as a function of failures experienced is

$\lambda(\mu) = \lambda_0 \exp(-\theta\mu)$

  - λ_0 : initial failure intensity at the start of execution
  - μ : the average or expected number of failures experienced at a given point in time
  - θ : failure intensity decay parameter

Failure Intensity Functions for RLPM cont..

- Assume initial failure intensity is 10 failures/CPU hr. the failure decay parameter is 0.02/failure .We assume that 50 failures have been experienced. Determine the value of the current failure intensity

  $\lambda(\mu) = \lambda_0 \exp(-\theta\mu)$

  $\lambda_0 = 10$; $\mu = 50$; $\theta = 0.02$

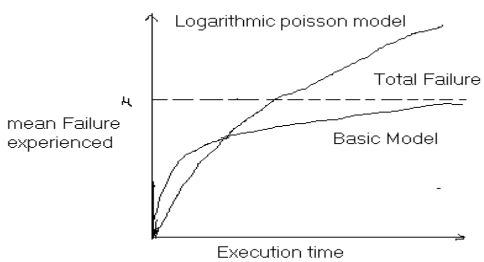
  $\lambda(\mu) = 10 \exp[(-0.02)*(50)] = 3.68$ failures/cpu hr.

Decrement of failure intensity per failure

$d\lambda/d\mu = -\lambda_0\,\theta \exp(-\theta\mu) = -10(0.02)\exp(-0.02*50)$

$= -0.2$/cpu hr.

Mean Failure Experienced vs Execution Time for RBM & RLPM



Mean Failure Experienced vs Execution Time (RBM)

- The expected number of failures experienced as a function of execution time be denoted by Γ

  $\mu(\Gamma) = v_0\,[1- \exp(-\lambda_0\,\Gamma\,/v_0)]$

- initial failure intensity 10 failures/CPU hr and total 100 failures calculate the failure experienced after 10 cpu hr of execution

  $\mu(\Gamma) = 100[1-\exp(-10*10/100)] = 63$ failures

  after 100 cpu

  $\mu(\Gamma) = 100[1-\exp(-10*100/100)] = 100$ failures

Mean Failure Experienced vs Execution Time (RLPM)

- The expected number of failures experienced as a function of execution time be denoted by Γ

  $\mu(\Gamma) = [\ln (\lambda_0\,\theta\,\Gamma +1)]/\,\theta$

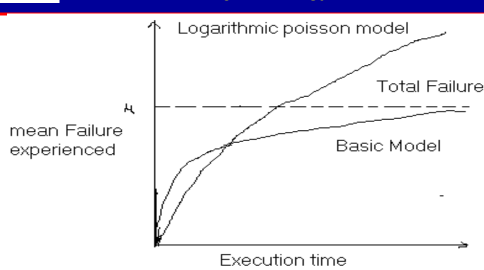
- initial failure intensity 10 failures/CPU hr and decay parameter 0.02 /failure calculate the failure experienced after 10 cpu hr of execution

  $\mu(\Gamma) = \ln[(10)(0.02)(10) +1]/0.02 = 55$ failures

  after 100 cpu hr

  $\mu(\Gamma) = \ln[(10)(0.02)(100) +1]/0.02 = 152$ failures

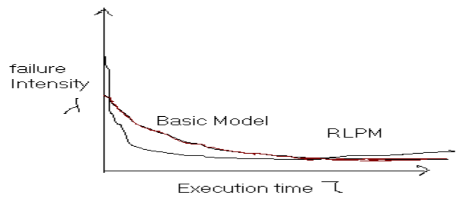
Mean Failure Experienced vs Execution Time for RBM & RLPM



RBM
After 10 cpu hrs : 63 failure
After 100 cpu hrs : 100 failures (almost)

RLPM
After 10 cpu hrs : 55 failure
After 100 cpu hrs : 152 failures

Failure Intensity at a Given Execution Time (RBM & RPLM)



 U3. 121

Failure Intensity at a Given Exe. Time (RBM)

- Present failure intensity at any given value of execution time

  $\lambda(\Gamma) = \lambda_0 \exp(-\lambda_0 \Gamma / v_0)]$

- initial failure intensity 10 failures/CPU hr and total 100 failures calculate the failure intensity after 10 cpu hr of execution

  $\lambda(\Gamma)$= 10 exp(-10*10/100)= 3.68 failures/cpu hr.

  after 100 cpu hr

$\lambda(\Gamma)$= 10 exp(-10*100/100)= 0.000454 failures/cpu hr.

 U3. 122

Failure Intensity at a Given Execution Time (RPLM)

- Present failure intensity at any given value of execution tie

  $\lambda(\Gamma) = \lambda_0 \exp(\lambda_0 \theta \Gamma + 1)$

- initial failure intensity 10 failures/CPU hr and decay parameter 0.02/failure calculate the failure intensity after 10 cpu hr of execution

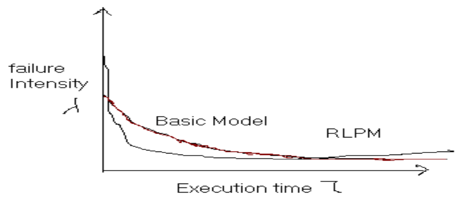
$\lambda(\Gamma)$= 10 /[10(0.02)(10)+1) = 3.33 failures/cpu hr.

after 100 cpu hr

$\lambda(\Gamma)$= 10 /[10(0.02)(100)+1) = 0.476 failures/cpu hr.

 U3. 123

Failure Intensity at a Given Execution Time (RBM & RPLM)



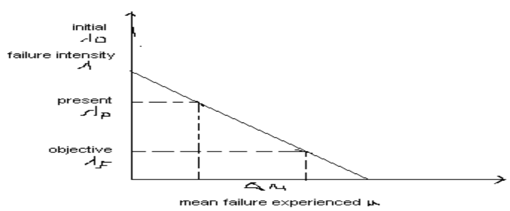
RBM
After 10 cpu hrs : 3.68 fail./cpuhr
After 100 cpu hrs : 0.0000454 fail./cpuhr

RLPM
After 10 cpu hrs : 3.33 fail./cpuhr
After 100 cpu hrs : 0.476 fail./cpuhr

Additional Number of Failures (RBM)

- Equation to use the objective and the present value of failure intensity to determine the additional expected number of failures that must be experienced to reach that objective

дμ = $v_0/\lambda_0 (\lambda_p - \lambda_F)$



Relations for the Additional Number of Failures (RBM) cont..

- Determine the expected number of failures that will be experienced between a present failure intensity of 3.68 failures/cpu hr and an objective of0.000454 failure/cpu hr using the same parameters initial failure intensity 10 failures/CPU hr and total 100 failures

дμ = $v_0/\lambda_0 (\lambda_p - \lambda_F)$

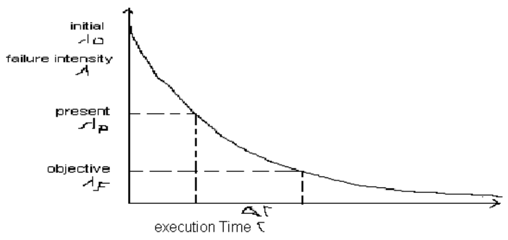
= (100/10)(3.68-0.000454)

= 10(3.68) = 37 failures

Relations for the Additional Execution Time (RBM)

- To determine the additional execution time дΓ required to reach the failure intensity objective

дΓ = $v_0/\lambda_0 \ln(\lambda_p / \lambda_F)$



Additional Number of Failures & Additional Execution Time For RLPM

дμ = $(1/\theta)\ln(\lambda_p / \lambda_F)$

дΓ = $(1/\theta)\,[(1/\lambda_F)-(1/\lambda_p)]$

Additional Number of Failures for RLPM

- Determine the expected number of failures that will be experienced between a present failure intensity of 3.33 failures/cpu hr and an objective of 0.476failure/cpu hr using the same parameters initial failure intensity 10 failures/CPU hr and decay parameter 0.02/failure

дμ $= (1/\theta)\ln(\lambda_p/\lambda_F)$

$= (1/0.02)\ln(3.33/0.476)$

$= 10(3.68) = 97$ failures

Calendar Time Component

- Relates execution time and calendar time by determining the calendar time to execution time ratio at any given point of time
- Ratio is based on constraints that are involved in applying resources to a project
- Have greater significance at testing & repair phase
- Rate of testing is constrained by
  - Failure identification/ test team personnel
  - Failure correction or debugging personnel
  - Computer time available

Calendar Time Component cont..

Based on a debugging model, which takes into account

- Resources used in operating the program for a given execution time and processing an associated quantity of failures
- Resource quantity available
- The degree to which a resource can be utilized

Resource Usage

	Usage Parameters requirements per		Planned parameter	
Resources	**CPU hr**	**Failure**	**Quantities available**	**Utilization**
Failure Identification	θ_i	μ_I	P_i	
1 **Personnel**				
Failure Correction Personnel	0	μ_f	P_f	p_f
Computer Time	θ_c	μ_C	P_c	p_c

Resource Usage cont..

- Resource usage is linearly proportional to execution time and mean failure experienced

Let x_r be the usage of resource r. then

$$x_r = \theta_r\Gamma + \mu_r\mu$$

θ_r :Resource usage per CPU hr

μ_r : Resource usage per failure

Resource usage per unit execution time

$$dx_r/d\Gamma = \theta_r + \mu_r\lambda$$

$$X_c = \theta_c\text{Д}\Gamma + \mu_c\text{Д}\mu$$

$$X_f = \mu_f\text{Д}\mu$$

$$X_I = \theta_I\text{Д}\Gamma + \mu_I\text{Д}\mu$$

Resource Usage cont..

- Computer time required per unit execution time will normally be greater than 1
- Failure intensity decreases with testing

Calendar Time Component cont..

- Suppose the test team runs test cases for 8 CPU hr and identifies 20 failures. The effort required per hr of execution time is 6 person hr. Each failure requires 2 hr on the average to verify and determine its nature. Calculate the total failure identification effort required

$x_r = \theta_r\Gamma + \mu_r\mu = 6(8)+2(20) = 48+40$

$= 88$ person hr.

Calendar Time to Execution Time Relationship

- Resource quantities and utilization are assumed to be constant
- Ration of calendar time to execution time can be obtained by dividing the resource usage rate of the limiting resources by the constant quantity of resource available that can be utilized

  $dt/d\Gamma = (1/P_r\rho_r)\ dx_r/d\Gamma = (\theta_r + \mu_r\lambda)/\ P_r\rho_r$

  P_r : resource available

  ρ_r : utilization

$dx_r/d\Gamma = \theta_r + \mu_r\lambda$

Reliability Allocation

- Discuss software reliability apportionment schemes
- Answer the question
  - How reliable should each system module be?
- Provide the reliability guidelines well in advance of any development at the planning and design stage of the system
- Deals with setting of the reliability goals for individual modules such that
  - a specified system reliability goal is met
  - the modules goals are "well balanced" among themselves

Parametric Approach

- Apportion the reliability goal R to n modules such that

$\P^{n}_{i=1} R^{*}_{i} >= R$

  - R : System Reliability goal
  - R^{*}_{i}: Allocated reliability for module i
- Failure rate of every module and hence of the whole system can be assumed constant, as after the development of any software it is not expected to be modified at the user end till there is a crash or till the next version is to be released.

Parametric Approach cont..

If λ_i^* is allocated failure rate for module i and λ is the required failure rate for the system

$$\Sigma^n_{i=1} \lambda^*_i <= \lambda$$

λ^*_i : fraction of the total failure rate let

$$\lambda^*_i = W_i \lambda$$

W_i :Weighing factor factor for module I

$$\Sigma^n_{i=1} W_i = 1 \qquad \text{equ. (1)}$$

$$R^*_i = (R)^{w_i}$$

To sure edq. 1

$$W_i = Z_i / \Sigma^n_{i=1} Z_i$$

Z_i Proportionality factor for module i

 U3. 139

What we Learnt

- ✓ Basic concepts
- ✓ Software Quality
  - ✓ McCall Software Quality model
  - ✓ Boehm Software quality Model
  - ✓ ISO 9126
- ✓ Software Reliability Models
  - ✓ Basic Execution time Model
  - ✓ Logarithmic Poisson Execution Time Model
  - ✓ Calendar Time Component
  - ✓ Resource Usage

 U3. 140

Understanding the Capability Maturity Model for Software

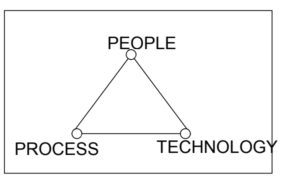
 U3. 141

Learning Objectives

- Quality Leverage Points
- Software process
- Why to Focus on Process
- Immature Process
- Mature Process
- CMM
- Capability vs. Performance
- Maturity Level
- Common Features
- CMM Structure

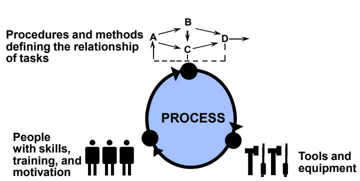
Quality Leverage Points

Major determinants of product cost, schedule, and quality



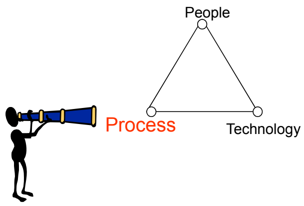
Definition of Software Process

- Process - a sequence of steps performed for a given purpose (IEEE)
- Software process - a set of activities, methods, practices, and transformations that people use to develop and maintain software and the associated products (CMM)



Why Focus on Process ? - 1

•Everyone realizes the importance of
•having a motivated, quality work force but



… even our finest people can't perform at their best when the process is not understood or operating "at its best."

Why Focus on Process ? - 2

The Process management premise :

- The quality of a system is highly influenced by the quality of the process used to acquire, develop, and maintain it.
- This premise implies focus on processes as well as on product.

An Immature Process

•Ad hoc; process improvised by practitioners and their management

•Not rigorously followed or enforced

•Highly dependent on current practitioners

•Low visibility

A Mature Process

- Consistent with the way work actually gets done
- Defined, documented, and continuously improving
- Supported visibly by management and others
- Well controlled—process fidelity is audited and enforced
- Constructive use of product and process measurement
- Disciplined use of technology

 U3. 148

Common Points in the Quality Movement

- Improvement focuses on fixing the process, not on blaming the people.
- Improvement must be measured and periodically reinforced.
- Improvement is a continuous process.
- If level of discomfort is not high enough, things will not change.

 U3. 149

What Is the Capability Maturity Model (CMM)?

- A common-sense application of process management and quality improvement concepts to software development and maintenance
- A model for organizational improvement
- Strategy for improving the software process, irrespective of the actual life cycle model used
- Developed by Software Engineering Institute (SEI) of Carnegie-Mellon University in 1986
- Used to judge the maturity of the software processes of an organization and to identify the key practices that are required to increase the maturity of these processes

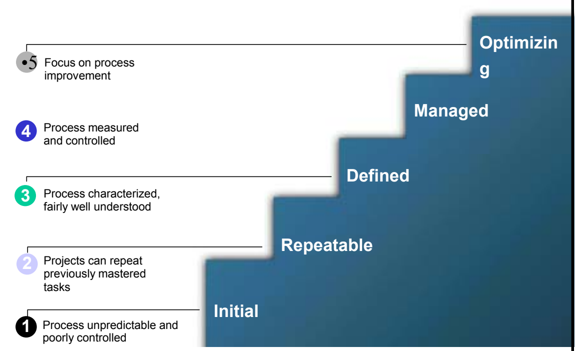
 U3. 150

Capability Versus Performance

- Process capability – the range of expected results that can be achieved by following a process, initially established at the organization level. A predictor of future project outcomes.

- Process performance – a measure of the actual results achieved from following a process. Refers to a particular project in the organization.

The Maturity Levels



Understanding the Initial Maturity Level

- Performance driven by the competence and heroics of the people doing the work

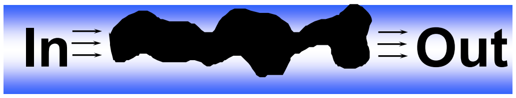
- High quality and exceptional performance possible so long as the best people can be hired

- Unpredictable—for good or ill

- The major problems facing the software organization are managerial, not technical

1

Software Management is a Black Art



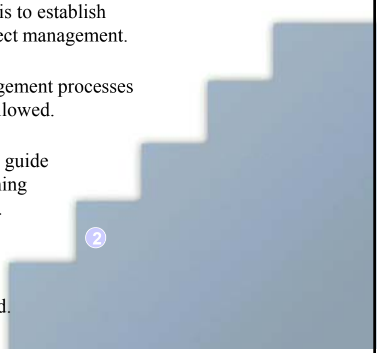
- Requirements flow in.
- A software product is (usually) produced by some amorphous process.
- The product flows out and (hopefully) works.

The Key Process Areas for the Initial Level

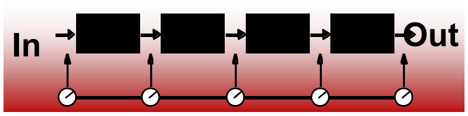
1 No key process areas

Understanding the Repeatable Maturity Level

- The predominant need is to establish effective software project management.
- Software project management processes are documented and followed.
- Organizational policies guide the projects in establishing management processes.
- Successful practices developed on earlier projects can be repeated.



Project Management System is in Place



•Process of building software is a series of black boxes with defined checkpoints (milestones).

The KPAs for the Repeatable Level

Software Configuration Management

2 Software Quality Assurance

Software Subcontract Management

Software Project Tracking and Oversight

Repeatable

Software Project Planning

Requirements Management

Requirements Management (RM)

•Purpose is to establish a common understanding between the customer and the software project of the customer's requirements that will be addressed.

Involves

▪document and control of customer requirements

▪keeping plans, products, and activities consistent with the requirements

Software Project Planning (PP, SPP)

•Purpose is to establish reasonable plans for performing the software engineering and for managing the software project.

Involves

- developing estimates for the work to be performed
- establishing the necessary commitments
- defining the plan to perform the work

 U3. 160

Software Project Tracking and Oversight (PT, PTO)

•Purpose is to provide adequate visibility into actual progress so that management can take effective actions when performance deviates significantly from the plan.

Involves

- tracking and reviewing software accomplishments and results against documented estimates, commitments, and plans
- adjusting plans based on actual accomplishments and results

 U3. 161

Software Subcontract Management (SM)

•Purpose is to select qualified software subcontractors and manage them effectively.

Involves

- selecting a software subcontractor
- establishing commitments with the subcontractor
- tracking and reviewing the subcontractor's performance and results

 U3. 162

Software Quality Assurance (QA, SQA)

•Purpose is to provide management with appropriate visibility into the process being used and the products being built.

Involves

- reviewing and auditing the software products and activities to ensure that they comply with the applicable procedures and standards
- providing the software project and other appropriate managers with the results of those reviews and audits

 U3. 163

Software Configuration Management (CM, SCM)

•Purpose is to establish and maintain the integrity of the products of the software project throughout the software life cycle.

Involves

- identifying configuration items/units
- systematically controlling changes
- maintaining integrity and traceability of the configuration throughout the software life cycle

 U3. 164

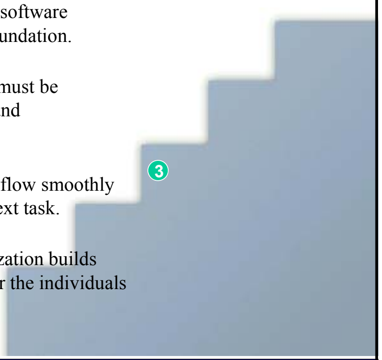
Understanding the Defined Maturity Level

This level builds on the software project management foundation.

To control a process, it must be defined, documented, and understood.

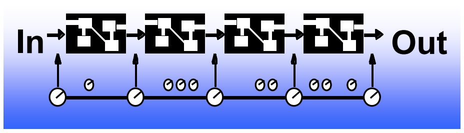
The outputs of one task flow smoothly into the inputs of the next task.

At this level, the organization builds processes that empower the individuals doing the work.



 U3. 165

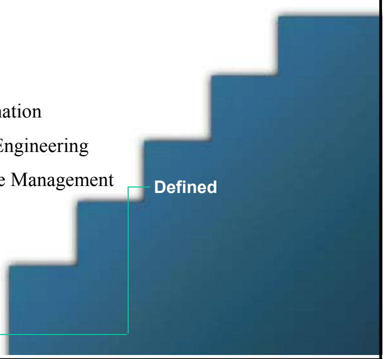
Managed According to a Well-Defined Process



- Roles and responsibilities in the process are understood.
- The production of the software product is visible throughout the software process.

The Key Process Areas for the Defined Level

Peer Reviews
3 Intergroup Coordination
Software Product Engineering
Integrated Software Management
Training Program
Organization Process Definition
Organization Process Focus



Organization Process Focus (PF, OPF)

- Purpose is to establish the organizational responsibility for software process activities that improve the organization's overall software process capability.

Involves

- developing and maintaining an understanding of the organization's and projects' software processes
- coordinating the activities to assess, develop, maintain, and improve these processes

Organization Process Definition (PD,OPD)

•Purpose is to develop and maintain a usable set of software process assets that improve process performance and provide a basis for cumulative, long-term benefits.

Involves

- developing and maintaining the organization's standard software process and related process assets.

Training Program (TP)

•Purpose is to develop the skills and knowledge of individuals so they can perform their roles effectively and efficiently.

Involves

- identifying the training needs of the organization, projects, and individuals
- developing and/or procuring training to address these needs

Integrated Software Management (IM, ISM)

•Purpose is to integrate the project's software engineering and management activities into a coherent, defined software process tailored from the organization's software process assets.

Involves

- developing the project's defined software process by tailoring the organization's standard software process
- managing the software project according to this defined software process

Software Product Engineering (PE, SPE)

- Purpose is to consistently perform a well-defined engineering process that integrates all the software engineering activities to produce correct, consistent software products effectively and efficiently.

Involves

- performing the engineering tasks to build and maintain the software using appropriate tools and methods

 U3. 172

Intergroup Coordination (IC)

- Purpose is to establish a means for the software engineering group to participate actively with the other engineering groups so that the project is better able to satisfy the customer's needs effectively and efficiently.

Involves

- disciplined interaction and coordination of the project engineering groups with each other to address system-level requirements, objectives, and plans

 U3. 173

Peer Reviews (PR)

- Purpose is to remove defects from the software work products early and efficiently. An important corollary effect is to develop a better understanding of the software work products and of defects that might be prevented.

Involves

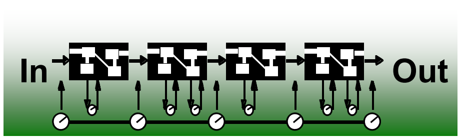
- methodical examination of work products by the producer's peers to identify defects and areas where changes are needed
- identifying products that will undergo a peer review in the project's defined software process

 U3. 174

Understanding the Managed Maturity Level

- Sets quantitative quality goal for both software products and processes
- Can be summarized as "predictable" as the process is measured and operates within measurable limits

Product and Process are Quantitatively Managed



•Management has an objective basis for making decisions.

•Management is able to predict performance within quantified bounds.

The Key Process Areas for the Managed Level

Quantitative Process Management (QP, QPM)

•Purpose is to control the process performance of the software project quantitatively.

Involves

- establishing goals for process performance
- measuring the performance of the project
- analyzing these measurements
- making adjustments to maintain process performance within acceptable limits

Software Quality Management(QM, SQM)

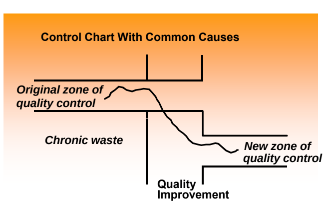
•Purpose is to develop a quantitative understanding of the quality of the project's software products and achieve specific quality goals.

Involves

- defining quality goals for the software products
- establishing plans to achieve these goals
- monitoring and adjusting software plans, software work products, activities, and quality goals to satisfy the needs and desires of customer and end-user

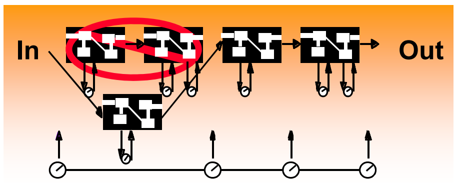
Understanding the Optimizing Maturity Level

•Identify and eliminate chronic causes of poor performance



•Continuously improve the software process

Focus on Continuous Process Improvement



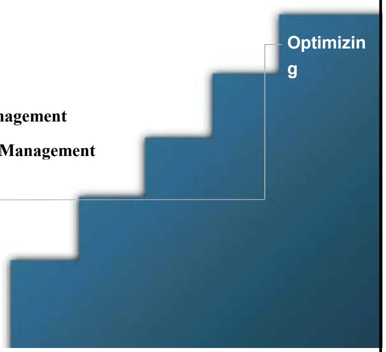
•Disciplined change is a way of life.

The Key Process Areas for the Optimizing Level

5 Process Change Management

Technology Change Management

Defect Prevention



Defect Prevention (DP)

•Purpose is to identify the cause of defects and prevent them from recurring.

Involves

- analyzing defects that were encountered in the past
- taking specific actions to prevent the occurrence of these types of defects in the future

Process Change Management (PC, PCM)

•Purpose is to continuously improve the software processes used in the organization with the intent of improving software quality, increasing productivity, and decreasing the cycle time for product development.

Involves
- defining process improvement goals
- systematically identifying, evaluating, and implementing improvements to the organization's standard software process and the projects' defined software processes

Technology Change Management (TM, TCM)

•Purpose is to identify new technologies (i.e., tools, methods, and processes) and transfer them into the organization in an orderly manner.

Involves
- identifying, selecting, and evaluating new technologies
- incorporating effective technologies into the organization

Maturity Levels Cannot be Skipped

•Each level provides a necessary foundation for improvements undertaken at the next level.

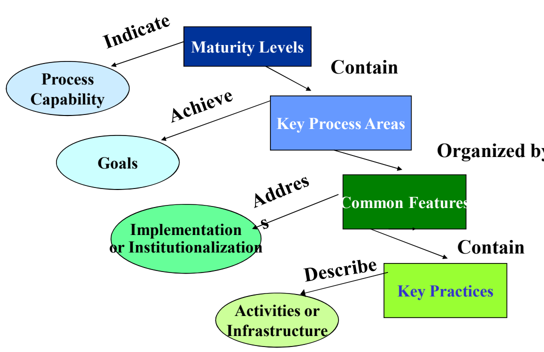
- engineering process is easily sacrificed without management discipline
- detailed measures are inconsistent without a defined process
- effect of process innovation is obscure in a noisy process

Common Features

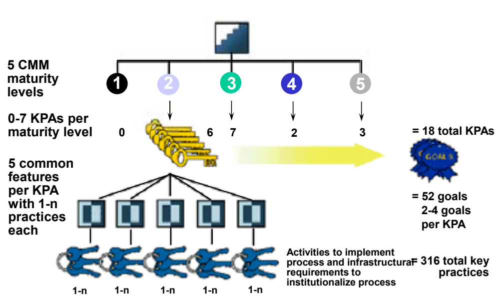
- Indicate whether the implementation and institutionalization of a key process area are effective, repeatable and lasting

1. Commitment to perform (Actions)
2. Ability to perform (Preconditions)
3. Activities performed (roles and procedures)
4. Measurement and analysis (need to measure & analysis of measure)
5. Verifying Implementation(steps to ensure)

The CMM Structure



CMM Structure



CMM Structure

Sl. No.	Structure/ level	Level 1	Level 2	Level 3	Level 4	Level 5	Total	
1.	KPAs (0-7 per ML)	-	6	7	2	3	18	
2.	Goals (1-4 per KPA)	-	20	17	6	9	52	
Common Features	Commitments	-	9	9	3	7	28	316
	Abilities	-	25	25	8	13	71	
	Activities	-	62	50	12	26	150	
	Measurement analysis	-	6	9	2	3	20	
	Verification & Implementation	-	19	15	6	7	47	

 U3. 190

What we Learnt

- The CMM focuses on software management issues.
- Visibility into the process depends on maturity of the process.
- The CMM is a 5-level model and the levels are broken into key process areas.
- Each level builds on the capability of the previous level.

 U3. 191

ISO 9001

QUALITY SYSTEM STANDARD & CERTIFICATION

 U3. 192

Learning Objective

- Quality
- Quality Control
- Quality Assurance
- Quality Management System
- ISO 9000
- ISO 9001
- ISO 9001 Clauses

Quality

Ability of a set of inherent
characteristics
of a
product , system or process
to fulfil requirements of *customers*
and
other interested parties

Quality Control

Part of Quality Management,

focussed on fulfilling

quality requirements

Quality Assurance

Part of quality management,

focussed on providing confidence, that quality requirements will be fulfilled

 U3. 196

Quality Management System

System to establish quality policy

&

quality objectives and to achieve those

objectives

 U3. 197

Quality System

"Say What you do; do what you say"

 U3. 198

ISO 9000

- A different attempt to improve software quality
- Not an industry specific, use by over 130 countries
- Set of documents dealing with quality system that can be used for quality assurance process
- Series of five related standards
  - Industrial activities, design/development, production, installation and servicing

 U3. 199

What is ISO 9001?

•Specific guidelines to software namely ISO9000-3

•Minimum criteria for an acceptable quality system

•Has much broader scope: Hardware, software, processed, material and services

 U3. 200

International Organization of Standardization

ISO is an organization that develops Standards for use worldwide to help companies plug into the world market.

 U3. 201

Quality Management System Criteria

- An International Group of Business and Quality Professionals determined criteria.
- **basics of good business practice. For example:**
- Set quality goals
- Ensure customer requirements are understood and met
- Train employees
- Control your production processes
- Purchase from suppliers that can provide quality product
- Correct problems and make sure they do not happen again

 U3. 202

Internal Benefits Include

Increased productivity

- Less scrap and rework
- Increased employee satisfaction
- Continual improvement
- Increased profits

 U3. 203

Marketing Benefits Include

- An internationally recognized QMS
- Increased opportunities in specific markets
- Increased customer satisfaction

 U3. 204

What will ISO 9001 do for your employees?

- It will ensure that they have the training and information to do their job correctly.
- Systems will be in place to identify problems
  - find the cause and eliminate it to prevent problems from reoccurring

The ISO 9001 Standard

Section 1: Scope

- Talks about the standard and how it applies to organizations

Section 2: Normative Reference

- References another document that should be used along with the standard, ISO 9000:2000, Quality Management Systems-Fundamentals and Vocabulary

Section 3: Terms and Definitions

- Gives a few new definitions

Section 4: General Requirements

- Gives requirements for the overall Quality Management System

The ISO 9001 Standard cont..

Section 5: Management Responsibility

- Gives requirements for Management and their role in the Quality Management System

Section 6: Resource Management

- Gives requirements for resources including personnel, training, the facility and work environment

Section 7: Product Realization

- Gives requirements for the production of the product or service, including things like planning, customer related processes, design, purchasing and process control

The ISO 9001 Standard cont..

Section 8: Measurement, Analysis and Improvement

- Gives requirements on monitoring processes and improving those processes

ISO 9001 Clauses

Management Responsibility

- Define, document, understand, implement and maintain quality policies

Quality system

- Document quality system, including procedure and instructions

Contract review

- Review contract to determine whether the requirements are adequately defined, agreed with the bid and implemented

Design Control

- Includes planning design activities, identifying inputs and outputs, verifying design and controlling design changes

Document Control

- Distribution and modification of documents be controlled

ISO 9001 Clauses..

Purchasing

- Includes the assessment of potential subcontractors and verification of purchased

Purchaser-Supplied Product

- Any purchaser-supplied material be verified and maintained

Product Identification and Traceability

- Product be identified and traceable during all stages of production, delivery and installation

Process Control

- Production process be defined and planned, that includes carrying out production under controlled condition

ISO 9001 Clauses..

Inspection and Testing

- Incoming material be inspected or certified before use and that in-process inspection and testing be performed. Final inspection and testing be performed prior to release

Inspection , Measuring and test equipment

- Equipment used to demonstrate conformance be controlled, calibrate and maintained

Inspection and Test Status

- The status of inspections and tests be maintained for items as they progress through various processing steps

ISO 9001 Clauses..

Control of nonconforming product

- Nonconforming product be controlled to prevent inadvertent use of installation

Corrective Action

- Potential causes of nonconforming product are eliminated

Handling, storage, packaging and delivery

- Procedures for Handling, storage, packaging and delivery be established and maintained

Quality Records

- Quality records be collected , maintained and dispositioned

ISO 9001 Clauses..

Internal Quality Audits

- Audits be planned and performed

Training

- Training needs to be identified and provided. Records of training are maintained

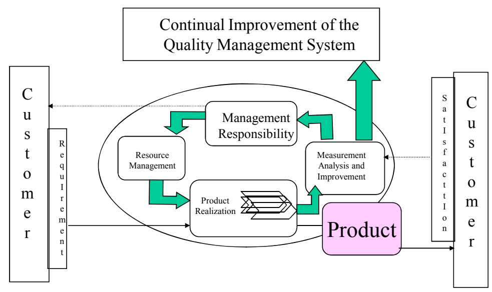
Servicing

- Servicing activities be performed as specified

Statistical technique

- Appropriate , adequate statistical techniques are identified should be used to verify the acceptability of process capability and product characteristics

The Process Model



 U3. 214

What we Learnt

- ✓ Quality
- ✓ Quality Control
- ✓ Quality Assurance
- ✓ Quality Management System
- ✓ ISO 9000
- ✓ ISO 9001
- ✓ ISO 9001 Clauses

 U3. 215

UNIT III Learnings

- ✓ Software Metrics
  - ✓ Software measurements
  - ✓ What & Why
  - ✓ Token Count
  - ✓ Halstead Software Science Measures
  - ✓ Design Metrics
  - ✓ Data Structure Metrics
  - ✓ Information Flow Metrics

 U3. 216

UNIT III Learnings

- ✓ Software Reliability
  - ✓Importance
  - ✓Hardware Reliability & Software Reliability
  - ✓Failure and Faults
  - ✓Reliability Models
  - ✓Software Quality Models
  - ✓CMM & ISO 9001

 U3. 217

Problem 1

- Assume that a program will experience 150 failures in infinite time. It has now experienced 80. The initial failure intensity was 10-failures/CPU hr.
  - (i) Determine current failures intensity
  - (ii) Calculate the failures experienced and failure intensity after 25 and 40 CPU hrs of execution.
  - (iii) Compute additional execution time required to reach the failure intensity objectives of 2-failures/CPU-hr.
  - Assume that initial failure intensity is 10 failure/cpu hrs. The failure

$\lambda(\mu) = \lambda_0 [1- \mu/v_0]$

$\mu(\Gamma) = v_0 [1- \exp(-\lambda_0 \Gamma /v_0)]$

 U3. 218

Problem 2

- Intensity decay parameter is 0.03/failure. We have experienced 75 Failures upto this time. Find the failures experienced and failure intensity after 25 and 50 CPU hours of execution.

 U3. 219

Objective Questions

Q1. Write formula for Language level and Program volume

Q2. Reliability of software is measured at________ phase

Q3. In logarithmic Poisson execution model, 0 is known as_____.

Q4. Define each of following term and derive/show their formula-

(i) Program level (ii) Potential volume (iii) Average life of a variable (iv) FANOUT

Q5. The number of clauses used in ISO9001 is ________.

Q6. Software science measures are developed by _________

Q7. Define Calendar Time Component.

Q8. Why is it important for software developers to make use of measurement to guide their work?

 U3. 220

Short Questions

Q1. What are information flow metrics? Explain the basic information flow model.

Q2. Differentiate between various categories of metrics.

Q3. Write short note on Any Three of following

I. MTBF

II. MTTF

III. Failure intensity

IV. CMM

V. MTBF

VI. Failure intensity

 U3. 221

Short Questions

Q4. Differentiate ISO9126 vs McCall software Quality Model

Q5. Write in short McCall, Boehm's and ISO 9126 software quality models.

Q6. Write Short Note on Resource Usage

Q7. Differentiate Basic Execution Time Model vs. Logarithmic Poisson Execution Time Model

Q8..A program is expected to have 500 faults. It is also assumed that one fault may lead to one failure only. The initial failure intensity was 2 failures/ CPU hr. The program was to be released with a failure intensity objective of 5 failures/100CPU hr. Calculate the number of failures experienced before release.

 U3. 222

Long Questions

Q1. What are software metrices ? Describe information flow based metrics.

Q2. Describe the Mc Call software quality model. How many product quality factors are defined and why?

Q3. Explain Halstead theory of software science. Is it significant in today's scenario of component based software development?

Q4. Write a program for calculation of roots of quadratic equation. Generate Cross reference list for the program and also calculate LV and WM for this program.

Q5. Explain Baehm Software Quality model with help of a block diagram.

Q6. Intensity decay parameter is 0.03/failure. We have experienced 75 Failures upto this time. Find the failures experienced and failure intensity after 25 and 50 CPU hours of execution.

Long Questions..

Q7. Assume that a program will experience 150 failures in infinite time. It has now experienced 80. The initial failure intensity was 10-failures/CPU hr.

(i) Determine current failures intensity

(ii) Calculate the failures experienced and failure intensity after 25 and 40 CPU hrs of execution.

(iii) Compute additional execution time required to reach the failure intensity objectives of 2-failures/CPU-hr.

Assume that initial failure intensity is 10 failure/cpu hrs. The failure

Q8. List the difference of CMM and ISO 9001. Why is it suggested that CMM is the better choice than ISO 9001?

Q9. What are software metrics? Discuss Halistead software sciences metrics along with its limitations.

Long Questions..

Q10. Compare and contrast Basic Execution Time Reliability Model with Logarithmic Poisson Execution Time Model

Q11. Discuss information flow metrics with its limitation. How a more sophisticated Information Flow model can overcome them?

Q12. Discuss CMM structure. What are the common features in each KPA.

Q13. Discuss the various key process areas of CMM at various maturity levels.

Research Problems

Q1. Given a structure with high fan-out, how would you convert it to a structure with a low fan-out

Q2. Discuss some approaches on how you can use metrics to guide you in design to produce a design that is easy to modify.

Q3. Design an experiment to study the Amount of Data, Average Live Variables and Variable Span.

Q4. If you have all the metrics data available for design, how will you use this data? Specify your objectives, the metrics you will use, how you will interpret the value, and what possible actions you will take based on the interpretation.

 U3. 226

References

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

 U3. 227

References..

. 7. Rajib Mall, "Fundamrntal of Software Engineering", PHI, 3rd Ed., 2009.

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

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

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

11. www.ifpug.com

12. www.softwaremetrics.com

13. www.qualityworld.com

14. www.sei.cmu.edu

15. www.spr.com

 U3. 228