UNIT-02
Measures, Metrics and Indicators
UNIT-02/LECTURE-01

Measures, Metrics and Indicators: : [RGPV/June 2014,2012(7),June 2011(5)]

Measure: Quantitative indication of the extent, amount, dimension, or size of some attribute of a product or process. A single data point.

Metrics: The degree to which a system, component, or process possesses a given attribute. Relates several measures (e.g. average number of errors found per person hour.)

Indicators: A combination of metrics that provides insight into the software process, project or product.

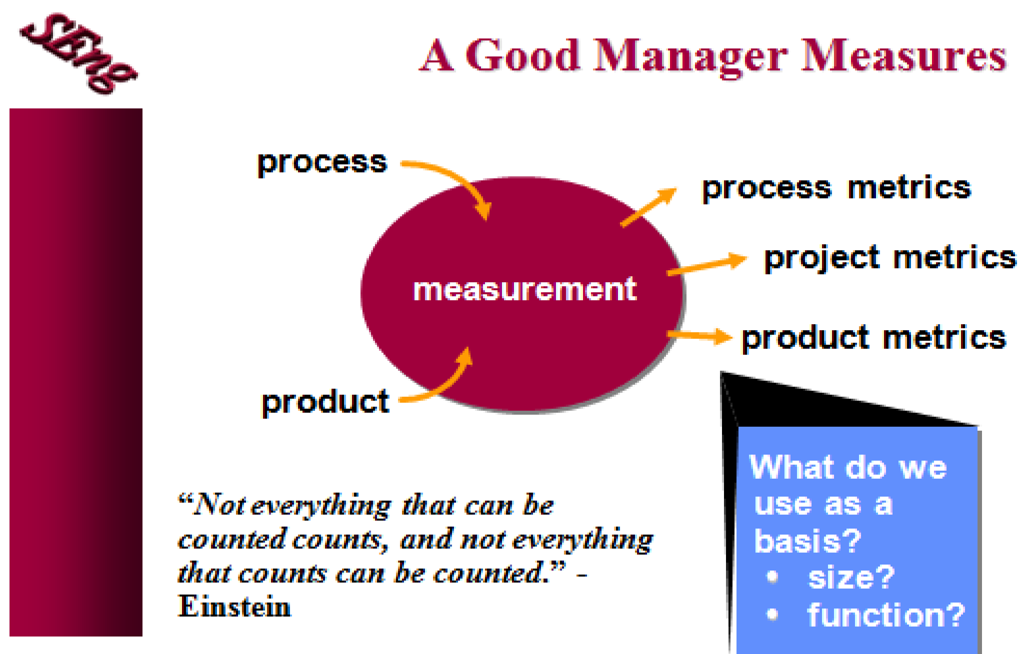
Direct Metrics: Immediately measurable attributes (e.g. line of code, execution speed, defects reported).

Indirect Metrics: Aspects that are not immediately quantifiable (e.g. functionality, quantity, reliability).

Faults:

Errors: Faults found by the practitioners during software development.

Defects: Faults found by the customers after release.



Process Metrics:-

Focus on quality achieved as a consequence of a repeatable or managed process. Strategic and Long Term.

Statistical Software Process Improvement (SSPI). Error Categorization and Analysis:

- All errors and defects are categorized by origin
- The cost to correct each error and defect is recorded
- The number of errors and defects in each category is computed
- Data is analyzed to find categories that result in the highest cost to the organization
- Plans are developed to modify the process

Defect Removal Efficiency (DRE). Relationship between errors (E) and defects (D). The ideal is a DRE of 1:

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

Project metrics:-

Used by a project manager and software team to adapt project work flow and technical activities. Tactical and Short Term.

Purpose:

- Minimize the development schedule by making the necessary adjustments to avoid delays and mitigate problems.
- Assess product quality on an ongoing basis.

Metrics:

- Effort or time per SE task
- Errors uncovered per review hour
- Scheduled vs. actual milestone dates
- Number of changes and their characteristics
- Distribution of effort on SE tasks

Project metrics:-

- Focus on the quality of deliverables.
- Product metrics are combined across several projects to produce process metrics

Metrics for the product:

- Measures of the Analysis Model
- Complexity of the Design Model

1. Internal algorithmic complexity
2. Architectural complexity

we dont take any liability for the notes correctness. http://www.rgpvonline.com

3. Data flow complexity

- **Code metrics**

Metrics Guideline:-

- **Use** common sense and organizational sensitivity when interpreting metrics data
- Provide regular feedback to the individuals and teams who have worked to collect measures and metrics.
- Don't use metrics to appraise individuals
- Work with practitioners and teams to set clear goals and metrics that will be used to achieve them
- Never use metrics to threaten individuals or teams
- Metrics data that indicate a problem area should not be considered "negative." These data are merely an indicator for process improvement
- Don't obsess on a single metric to the exclusion of other important metrics

Normalization for Metrics

- How does an organization combine metrics that come from different individuals or projects?
- Depend on the size and complexity of the project
- Normalization: compensate for complexity aspects particular to a product
- Normalization approaches:
  - Size oriented (lines of code approach)
  - Function oriented (function point approach)

Typical Normalized Metrics

Project	LOC	FP	Effort (P/M)	R(000)	Pp. doc	Errors	Defects	People
alpha	12100	189	24	168	365	134	29	3
beta	27200	388	62	440	1224	321	86	5
gamma	20200	631	43	314	1050	256	64	6

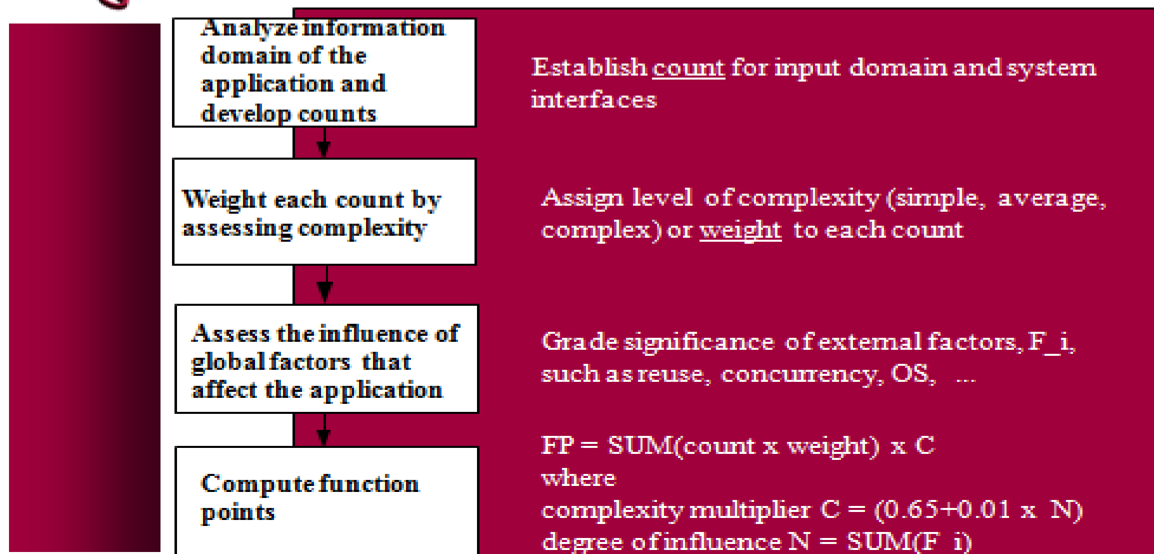
- **Size-Oriented:**

– errors per KLOC (thousand lines of code), defects per KLOC, R per LOC, page of documentation per KLOC, errors / person-month, LOC per person-month, R / page of documentation

- **Function-Oriented:**
  – errors per FP, defects per FP, R per FP, pages of documentation per FP, FP per person-month

Why Opt for FP Measures? : [RGPV/Jun 2014(7)]

- Independent of programming language. Some programming languages are more compact, e.g. C++ vs. Assembler
- Use readily countable characteristics of the "information domain" of the problem
- Does not "penalize" inventive implementations that require fewer LOC than others
- Makes it easier to accommodate reuse and object-oriented approaches
- Original FP approach good for typical Information Systems applications (interaction complexity)
- Variants (Extended FP and 3D FP) more suitable for real-time and scientific software (algorithm and state transition complexity)



Analyzing the Information Domain

measurement parameter	count		weighting factor simple	avg.	complex		
number of user inputs	☐	X	3	4	6	=	☐
number of user outputs	☐	X	4	5	7	=	☐
number of user inquiries	☐	X	3	4	6	=	☐
number of files	☐	X	7	10	15	=	☐
number of ext.interfaces	☐	X	5	7	10	=	☐
count-total →							☐
complexity multiplier							☐
function points →							☐

Taking Complexity into Account

- **Complexity Adjustment Values (F_i) are rated on a scale of 0 (not important) to 5 (very important):**
  1. Does the system require reliable backup and recovery?
  2. Are data communications required?
  3. Are there distributed processing functions?
  4. Is performance critical?
  5. System to be run in an existing, heavily utilized environment?
  6. Does the system require on-line data entry?
  7. On-line entry requires input over multiple screens or operations?
  8. Are the master files updated on-line?
  9. Are the inputs, outputs, files, or inquiries complex?
  10. Is the internal processing complex?
  11. Is the code designed to be reusable?
  12. Are conversion and instillation included in the design?
  13. Multiple installations in different organizations?
  14. Is the application designed to facilitate change and ease-of-use?

we dont take any liability for the notes correctness. http://www.rgpvonline.com

Exercise: Function Points[RGPV/Jun 2013(7)]

- Compute the function point value for a project with the following information domain characteristics:

Number of user inputs: 32

Number of user outputs: 60

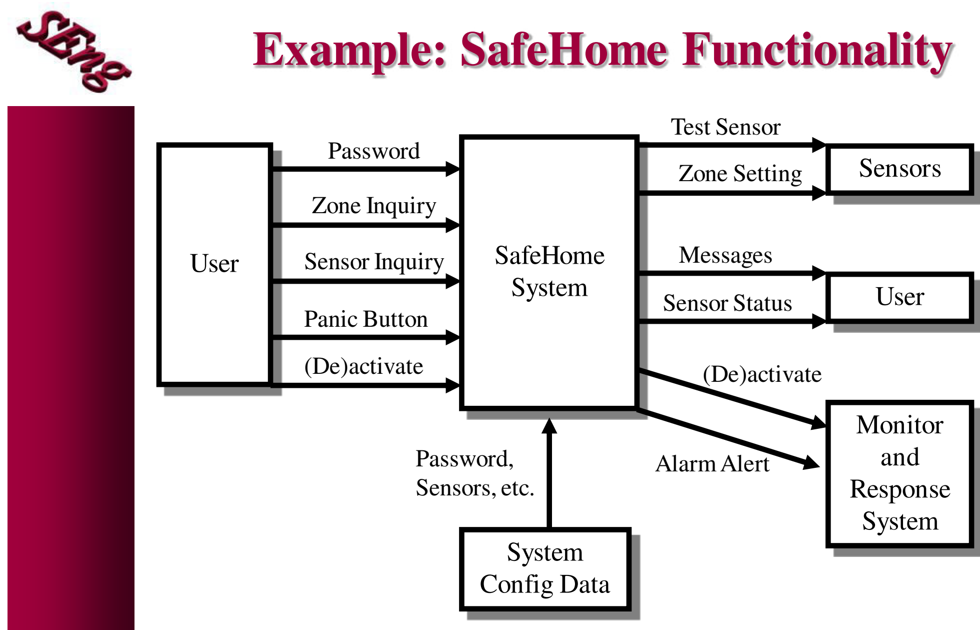
Number of user enquiries: 24

Number of files: 8

Number of external interfaces: 2

Assume that weights are average and external complexity adjustment values are not important.

Answer: 32*4+60*5+24*4+8*10+2*7



SEng

Example: SafeHome FP Calc

measurement parameter	count		simple	avg.	complex		
			weighting factor				
number of user inputs	3	X	3	4	6	=	9
number of user outputs	2	X	4	5	7	=	8
number of user inquiries	2	X	3	4	6	=	6
number of files	1	X	7	10	15	=	7
number of ext.interfaces	4	X	5	7	10	=	22
count-total							52
complexity multiplier $[0.65 + 0.01 \times \sum F_i] = [0.65 + 0.46]$							1.11
function points							58

Exercise: Function Points

- Compute the function point total for your project. Hint: The complexity adjustment values should be low ( )
- Some appropriate complexity factors are (each scores 0-5):
  1. Is performance critical?
  2. Does the system require on-line data entry?
  3. On-line entry requires input over multiple screens or operations?
  4. Are the inputs, outputs, files, or inquiries complex?
  5. Is the internal processing complex?
  6. Is the code designed to be reusable?
  7. Is the application designed to facilitate change and ease-of-use?

we dont take any liability for the notes correctness. http://www.rgpvonline.com

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	What are the software metrics? show why & how software metrics can improve software process.	June 2012	10
Q.2	Write short notes on Software metrics.	June 2011	5
Q.3	Explain the need for software measures & describe various metrics	June 2014	7
Q.4	Compute the function point value for a project with the following information domain characteristics: Number of user inputs: 32 Number of user outputs: 60 Number of user enquiries: 24 Number of files: 8 Number of external interfaces: 2 Assume that weights are average and external complexity adjustment values are not important.	June 2013	7

UNIT-02/LECTURE-02

OO Metrics: Distinguishing Characteristics

- **The following characteristics require that special OO metrics be developed:**
  - **Encapsulation —** Concentrate on classes rather than functions
  - **Information hiding —** An information hiding metric will provide an indication of quality
  - **Inheritance —** A pivotal indication of complexity
  - **Abstraction —** Metrics need to measure a class at different levels of abstraction and from different viewpoints
  - **Conclusion: the class is the fundamental unit of measurement**

OO Project Metrics

- **Number of Scenario Scripts (Use Cases):**
  - Number of use-cases is directly proportional the number of classes needed to meet requirements
  - A strong indicator of program size
- **Number of Key Classes (Class Diagram):**
  - A key class focuses directly on the problem domain
  - NOT likely to be implemented via reuse
  - Typically 20-40% of all classes are key, the rest support infrastructure (e.g. GUI, communications, databases)
- **Number of Subsystems (Package Diagram):**
  - Provides insight into resource allocation, scheduling for parallel development and overall integration effort

OO Analysis and Design Metrics

- **Related to Analysis and Design Principles**
- **Complexity:**
  - **Weighted Methods per Class (WMC):** Assume that *n* methods with cyclomatic complexity c1,c2……….cn are defined for a class C:

$$WMC = \sum c_i$$

  - **Depth of the Inheritance Tree (DIT):** The maximum length from a leaf to the root of the tree. Large DIT leads to greater design complexity but promotes reuse

- **Number of Children (NOC):** Total number of children for each class. Large NOC may dilute abstraction and increase testing

Further OOA&D Metrics[RGPV/Jun 2014(7)]

- **Coupling:**
  - **Coupling between Object Classes (COB):** Total number of collaborations listed for each class in CRC cards. Keep COB low because high values complicate modification and testing
  - **Response for a Class (RFC):** Set of methods potentially executed in response to a message received by a class. High RFC implies test and design complexity
- **Cohesion:**
  - **Lack of Cohesion in Methods (LCOM):** Number of methods in a class that access one or more of the same attributes. High LCOM means tightly coupled methods

OO Testability Metrics

- **Encapsulation:**
  - **Percent Public and Protected (PAP):** Percentage of attributes that are public. Public attributes can be inherited and accessed externally. High PAP means more side effects
  - **Public Access to Data members (PAD):** Number of classes that access another classes attributes. Violates encapsulation
  - **Inheritance:**
  - **Number of Root Classes (NRC):** Count of distinct class hierarchies. Must all be tested separately
  - **Fan In (FIN):** The number of superclasses associated with a class. FIN > 1 indicates multiple inheritance. Must be avoided
  - **Number of Children (NOC) and Depth of Inheritance Tree (DIT):** Superclasses need to be retested for each subclass

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

Quality Metrics

- Measures conformance to explicit requirements, following specified standards, satisfying of implicit requirements
- Software quality can be difficult to measure and is often highly subjective

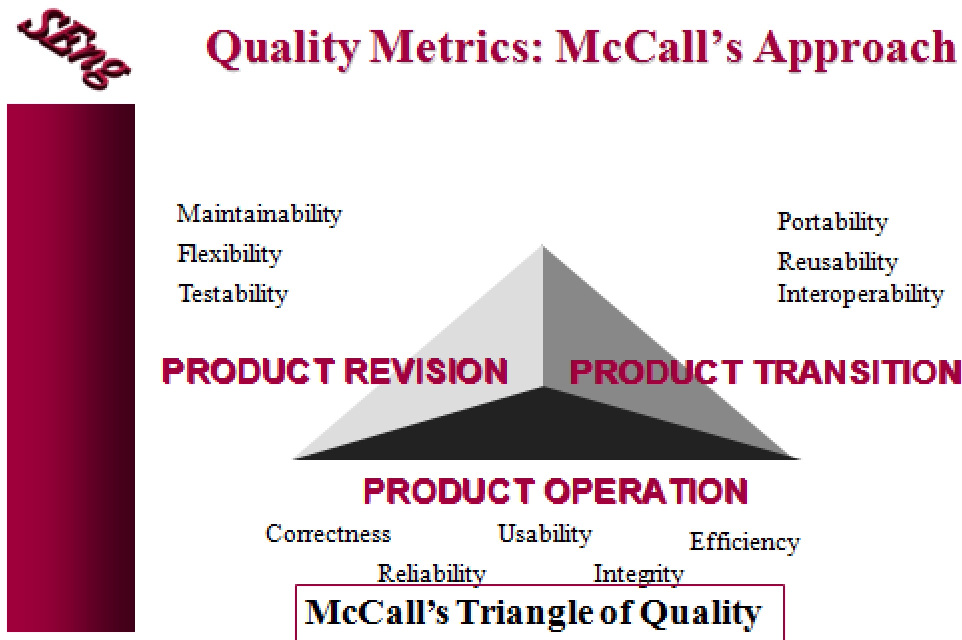
- Correctness:
  - The degree to which a program operates according to specification
  - Metric = Defects per FP

2. Maintainability:
   - The degree to which a program is amenable to change
   - Metric = Mean Time to Change. Average time taken to analyze, design, implement and distribute a change

Quality Metrics: Further Measures

3. Integrity:
   - The degree to which a program is impervious to outside attack
   - Summed over all types of security attacks, *i*, where *t* = threat (probability that an attack of type *i* will occur within a given time) and *s* = security (probability that an attack of type *i* will be repelled)
4. Usability:
   - The degree to which a program is easy to use.
   - Metric = (1) the skill required to learn the system, (2) the time required to become moderately proficient, (3) the net increase in productivity, (4) assessment of the users attitude to the system
   - Covered in HCI course

we dont take any liability for the notes correctness. http://www.rgpvonline.com



Quality Metrics: Deriving McCall's Quality Metrics

- Assess a set of quality factors on a scale of 0 (low) to 10 (high)
- Each of McCall's Quality Metrics is a weighted sum of different quality factors
- Weighting is determined by product requirements
- Example:
  - **Correctness = Completeness + Consistency + Traceability**
  - **Completeness** is the degree to which full implementation of required function has been achieved
  - **Consistency** is the use of uniform design and documentation techniques
  - **Traceability** is the ability to trace program components back to analysis
  - This technique depends on good objective evaluators because quality factor scores can be subjective

Managing Variation

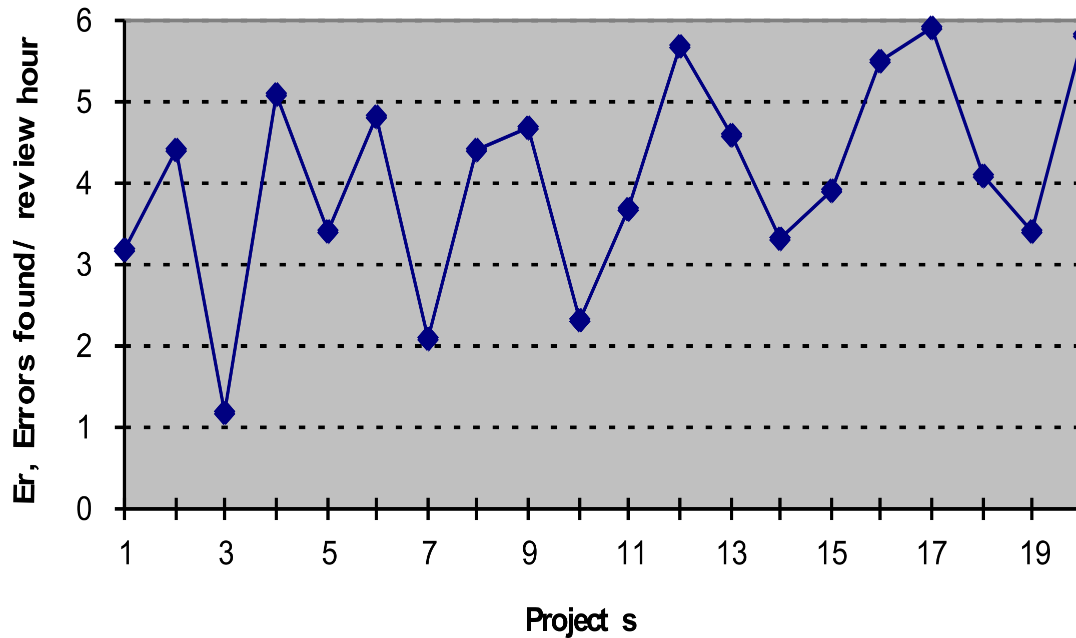
- How can we determine if metrics collected over a series of projects improve (or degrade) as a consequence of improvements in the process rather than noise?
- Statistical Process Control:
  - Analyzes the dispersion (variability) and location (moving average)
  - Determine if metrics are:

(a) Stable (the process exhibits only natural or controlled changes) or

we dont take any liability for the notes correctness. http://www.rgpvonline.com

(b) Unstable (process exhibits out of control changes and metrics cannot be used to predict changes)

Control Chart



Compare sequences of metrics values against mean and standard deviation. E.g. metric is unstable if eight consecutive values lie on one side of the mean.

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Compute the function point value for a project with the following: No of external inputs:32 No of external outputs:60 No of external inquiries:24 No of external interface files:2 No of internal logical files:8 Assume that all complexity adjustment values are average.	JUNE 2013	7
Q.2	Discuss the impact of cohesion, coupling in design phases.	JUNE 2014	7

UNIT-02/LECTURE-03

Software Reliability:

- Probability of failure-free operation for a specified time in a specified environment for a given purpose
- This means quite different things depending on the system and the users of that system
- Informally, reliability is a measure of how well system users think it provides the services they require.

Software reliability

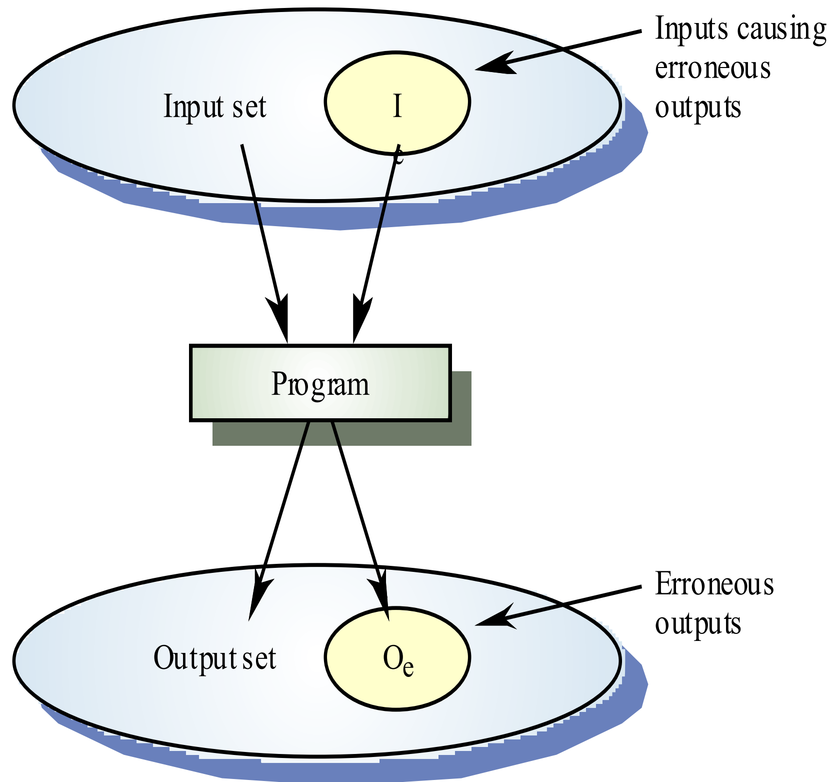
- – Cannot be defined objectively
  - Reliability measurements which are quoted out of context are not meaningful
- – Requires operational profile for its definition
  - The operational profile defines the expected pattern of software usage
- – Must consider fault consequences
  - Not all faults are equally serious. System is perceived as more unreliable if there are more serious faults

Failures and faults

- – A failure corresponds to unexpected run-time behaviour observed by a user of the software.
- – A fault is a static software characteristic which causes a failure to occur.
- – Faults need not necessarily cause failures. They only do so if the faulty part of the software is used.
- – If a user does not notice a failure, is it a failure? Remember most users don't know the software specification.

we dont take any liability for the notes correctness. http://www.rgpvonline.com

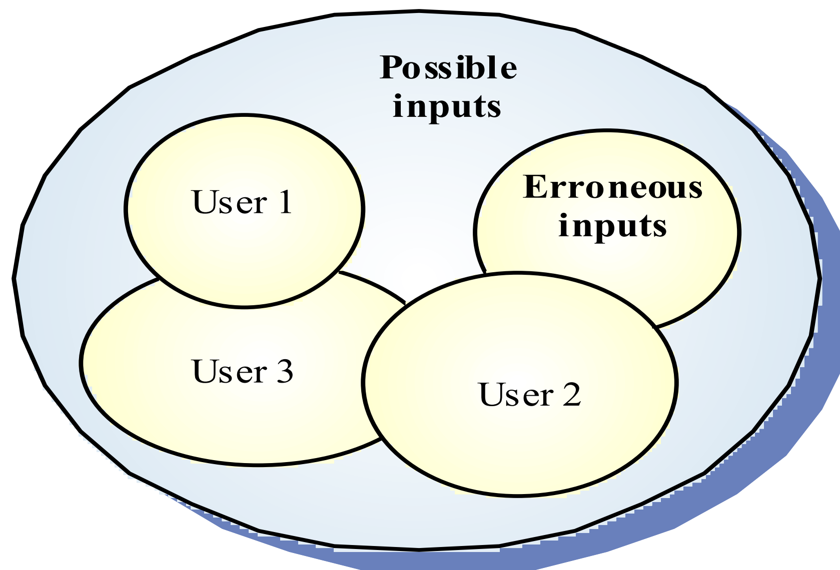
Input/output mapping



Reliability improvement

- Reliability is improved when software faults which occur in the most frequently used parts of the software are removed
- Removing x% of software faults will not necessarily lead to an x% reliability improvement
- In a study, removing 60% of software defects actually led to a 3% reliability improvement
- Removing faults with serious consequences is the most important objective

Reliability perception



Reliability and formal methods

- The use of formal methods of development may lead to more reliable systems as it can be proved that the system conforms to its specification
- The development of a formal specification forces a detailed analysis of the system which discovers anomalies and omissions in the specification
- However, formal methods may not actually improve reliability
- The specification may not reflect the real requirements of system users
- A formal specification may hide problems because users don't understand it
- Program proofs usually contain errors
- The proof may make assumptions about the system's environment and use which are incorrect

Reliability and efficiency

- As reliability increases system efficiency tends to decrease
- To make a system more reliable, redundant code must be includes carrying out run-time checks, etc. This tends to slow it down
- Reliability is usually more important than efficiency
- No need to utilise hardware to fullest extent (erdvė) as computers are cheap and fast
- Unreliable software isn't used

- Hard to improve unreliable systems
- Software failure costs often far exceed system costs
- Costs of data loss are very high

Reliability metrics

- Hardware metrics not really suitable for software as they are based on component failures and the need to repair or replace a component once it has failed. The design is assumed to be correct.
- Software failures are always design failures. Often the system continues to be available in spite of the fact that a failure has occurred.
- **Probability of failure on demand**
    - This is a measure of the likelihood that the system will fail when a service request is made
    - POFOD = 0.001 means 1 out of 1000 service requests result in failure
    - Relevant for safety-critical or non-stop systems

– **Rate of fault occurrence (ROCOF)**

- Frequency of occurrence of unexpected behaviour
- ROCOF of 0.02 means 2 failures are likely in each 100 operational time units
- Relevant for operating systems, transaction processing systems

– **Mean time to failure**

- Measure of the time between observed failures
- MTTF of 500 means that the time between failures is 500 time units
- Relevant for systems with long transactions e.g. CAD systems

– **Availability**

- Measure of how likely the system is available for use. Takes repair/restart time into account
- Availability of 0.998 means software is available for 998 out of 1000 time units
- Relevant for continuously running systems e.g. telephone switching systems

Reliability measurement

– Measure the number of system failures for a given number of system inputs

- Used to compute POFOD

– Measure the time (or number of transactions) between system failures

- Used to compute ROCOF and MTTF

– Measure the time to restart after failure

- Used to compute AVAIL

UNIT-02/LECTURE-04

Reliability specification Reliability requirements are only rarely expressed in a quantitative, verifiable way.

- To verify reliability metrics, an operational profile must be specified as part of the test plan.
- Reliability is dynamic – reliability specifications related to the source code are meaningless.
  - No more than N faults/1000 lines.
  - This is only useful for a post-delivery process analysis.

COCOMO MODEL: [RGPV/JUNE 2014,2013,2011(7)]

COCOMO, Constructive Cost Model is static single-variable model. Barry Boehm introduced COCOMO models. There is a hierarchy of these models.

- The **Constructive Cost Model** (COCOMO) is the most widely used software estimation model in the world. It
- The COCOMO model predicts the **effort** and **duration** of a project based on inputs relating to the size of the resulting systems and a number of "**cost drives**" that affect productivity.

Effort

- **Effort Equation**
  - $PM = C * (KDSI)^n$ (person-months)
    - where PM = number of person-month (=152 working hours),
    - C = a constant,
    - KDSI = thousands of "delivered source instructions" (DSI) and

n = a constant.

Productivity

- **Productivity equation**
  - (DSI) / (PM)
    - where PM = number of person-month (=152 working hours),
    - DSI = "delivered source instructions"

Schedule

- Schedule equation
  - $TDEV = C * (PM)^n$ (months)
    - where TDEV = number of months estimated for software development.

Average Staffing

- Average Staffing Equation
  - (PM) / (TDEV) (FSP)
    - where FSP means Full-time-equivalent Software Personnel.

- COCOMO is defined in terms of three different models:
  - the Basic model,
  - the Intermediate model, and
  - the Detailed model.
- The more complex models account for more factors that influence software projects, and make more accurate estimates.

The Development mode

The most important factors contributing to a project's duration and cost is the Development Mode

- Organic Mode: The project is developed in a familiar, stable environment, and the product is similar to previously developed products. The product is relatively small, and requires little innovation.
- Semidetached Mode: The project's characteristics are intermediate between Organic and Embedded
- Embedded Mode**:** The project is characterized by tight, inflexible constraints and interface requirements. An embedded mode project will require a great deal of innovation.

Modes

Feature	Organic	Semidetached	Embedded
Organizational understanding of product and objectives	Thorough	Considerable	General
Experience in working with related software systems	Extensive	Considerable	Moderate
Need for software conformance with pre-established requirements	Basic	Considerable	Full
Need for software conformance with external interface specifications	Basic	Considerable	Full

SEG3300 A&B W2004 R.L. Probert 16

we dont take any liability for the notes correctness. http://www.rgpvonline.com

Modes (.)

Feature	Organic	Semidetached	Embedded
Concurrent development of associated new hardware and operational procedures	Some	Moderate	Extensive
Need for innovative data processing architectures, algorithms	Minimal	Some	Considerable
Premium on early completion	Low	Medium	High
Product size range	<50 KDSI	<300KDSI	All

SEG3300 A&B W2004 R.L. Probert 17

Cost Estimation Process

Cost=SizeOfTheProject x Productivity

Model 1:

Basic COCOMO model is static single-valued model that computes software development effort (and cost) as a function of program size expressed in estimated lines of code.

Model 2:

Intermediate COCOMO model computes software development effort as a function of program size and a set of "cost drivers" that include subjective assessments of product, hardware, personnel, and project attributes.

Model 3:

Advanced COCOMO model incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step, like analysis, design, etc.

COCOMO can be applied to the following software project's categories.

Organic mode:

These projects are very simple and have small team size. The team has a good application experience work to a set of less than rigid requirements. A thermal analysis program developed for a heat transfer group is an example of this.

we dont take any liability for the notes correctness. http://www.rgpvonline.com

Semi-detached mode:

These are intermediate in size and complexity. Here the team has mixed experience to meet a mix of rigid and less than rigid requirements. A transaction processing system with fixed requirements for terminal hardware and database software is an example of this.

Embedded mode:

Software projects that must be developed within a set of tight hardware, software, and operational constraints. For example, flight control software for aircraft.

The basic COCOMO model takes the form

$$E = a_b \, (KLOC)\exp(b_b)$$
$$D = C_b(E)\exp(d_b)$$

where,

E is the effort applied in person-month,
D is the development time in chronological month,
KLOC is the estimated number of delivered lines (expressed in thousands) of code for project,
The ***ab*** and ***cb*** and the exponents ***bb*** and ***db*** are given in the table below.

Software Project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	.38
Semi-detached	3.0	1.12	2.5	.35
Embedded	3.6	1.20	2.5	.32

Basic COCOMO

The basic model is extended to consider a set of “cost drivers attributes”. These attributes can be grouped together into four categories.

1. Product attributes

    a) Required software reliability.
    b) Complexity of the project.
    c) Size of application database.
2. Hardware attributes

    a) Run-time performance constraints.
    b) Volatility of the virtual machine environment.
    c) Required turnaround time.

d) Memory constraints.

3. Personnel attribute

   a) Analyst capability.
   b) Software engineer capability.
   c) Virtual machine experience.
   d) Application experience.
   e) Programming language experience.

4. Project attributes

   a) Application of software engineering methods.
   b) Use of software tools.
   c) Required development schedule.

Each of the 15 attributes is rated on a 6-point scale that ranges from “very low” to “very high” in importance or value. Based on the rating, an effort multiplier is determined from the tables given by Boehm. The product of all multipliers results in an effort adjustment factor (EAF). Typical values of EAF range from 0.9 to 1.4.

Example : Problem Statement same as LOC problem refer section 3.2.1

KLOC = 10.9
E = ab (KLOC)exp(bb)
= 2.4(10.9)exp(1.05)
= 29.5 person-month

D = Cb(E)exp(db)
= 2.5(29.5)exp(.38)
= 9.04 months

The intermediate COCOMO model takes the following form.

E = ai(LOC)exp(bi) X EAF

Where,
E is the effort applied in person-months,
LOC is the estimated number of delivered lines of code for the project.
The coefficient ***ai*** and the exponent ***bi*** are given in the table below.

Software project	a_i	b_i
Organic	3.2	1.05
Semidetached	3.0	1.12
Embedded	2.8	1.20

INTERMEDIATE COCOMO MODEL

Software project	a_i	b_i
organic	3.2	1.05
Semi-detached	3.0	1.12
embedded	2.8	1.20

COCOMO represents a comprehensive empirical model for software estimation. However, Boehm's own comments [BOE81] about COCOMO (and by extension all models) should be heeded:

Today, a software cost estimation model is doing well if it can estimate software development costs within 20% of actual costs, 70% of the time, and on its own turf (that is, within the class of projects to which it has been calibrated ... This is not as precise as we might like, but it is accurate enough to provide a good deal of help in software engineering economic analysis and decision making.

To illustrate the use of the COCOMO model, we apply the basic model to the CAD software example [described in SEPA, 5/e]. Using the LOC estimate and the coefficients noted in Table 5.1, we use the basic model to get:

$E = 2.4\ (KLOC)^{1.05}$

$= 2.4\ (33.2)^{1.05}$

= 95 person-months

This value is considerably higher than the estimates derived using LOC. Because the COCOMO model assumes considerably lower LOC/pm levels than those discussed in SEPA, 5/e, the results are not surprising. To be useful in the context of the example problem, the COCOMO model would have to be recalibrated to the local environment.

To compute project duration , we use the effort estimate described above:

$D = 2.5\ E^{0.35}$

$= 2.5\ (95)^{0.35}$

= 12.3 months

we dont take any liability for the notes correctness. http://www.rgpvonline.com

The value for project duration enables the planner to determine a recommended number of people, *N,* for the project:

N = E/D

= 95/12.3

~ 8 people

In reality, the planner may decide to use only four people and extend the project duration accordingly.

UNIT-02/LECTURE-05

Relation between LOC and FP:

- Relationship:
  - *LOC = Language Factor * FP*
  - where
    - LOC (Lines of Code)
    - FP (Function Points)

Relation between LOC and FPs

Language	LOC/FP
assembly	320
C	128
Cobol	105
Fortan	105
Pascal	90
Ada	70
OO languages	30
4GL languages	20

Assuming LOC's per FP for:

Java = 53,

C++ = 64

*aKLOC = FP * LOC_per_FP / 1000*

It means for the SpellChekcer Example: (Java)

LOC=52.25*53=2769.25 LOC or 2.76 KLOC

Effort Computation

- The Basic COCOMO model computes effort as a function of program size. The Basic COCOMO equation is:
  - *E = aKLOC^b*
-
- Effort for three modes of Basic COCOMO.

Mode	a	b
Organic	**2.4**	**1.05**
Semi-detached	**3.0**	**1.12**
Embedded	**3.6**	**1.20**

Example

Mode	Effort Formula
Organic	$E = 2.4 * (S^{1.05})$
Semidetached	$E = 3.0 * (S^{1.12})$
Embedded	$E = 3.6 * (S^{1.20})$

Size = 200 KLOC

Effort = a * Size^{b}

Organic — $E = 2.4 * (200^{1.05}) = 626$ staff-months

Semidetached — $E = 3.0 * (200^{1.12}) = 1133$ staff-months

Embedded — $E = 3.6 * (200^{1.20}) = 2077$ staff-months

- The intermediate COCOMO model computes effort as a function of program size and a set of cost drivers. The Intermediate COCOMO equation is:
  - *E = aKLOC^b*EAF*
- Effort for three modes of intermediate COCOMO.

Mode	a	b
Organic	**3.2**	**1.05**
Semi-detached	**3.0**	**1.12**
Embedded	**2.8**	**1.20**

we dont take any liability for the notes correctness.

Effort Adjustment Factor

Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
Required Reliability	.75	.88	1.00	1.15	1.40	1.40
Database Size	.94	.94	1.00	1.08	1.16	1.16
Product Complexity	.70	.85	1.00	1.15	1.30	1.65
Execution Time Constraint	1.00	1.00	1.00	1.11	1.30	1.66
Main Storage Constraint	1.00	1.00	1.00	1.06	1.21	1.56
Virtual Machine Volatility	.87	.87	1.00	1.15	1.30	1.30
Comp Turn Around Time	.87	.87	1.00	1.07	1.15	1.15
Analyst Capability	1.46	1.19	1.00	.86	.71	.71
Application Experience	1.29	1.13	1.00	.91	.82	.82
Programmers Capability	1.42	1.17	1.00	.86	.70	.70
Virtual machine Experience	1.21	1.10	1.00	.90	.90	.90
Language Experience	1.14	1.07	1.00	.95	.95	.95
Modern Prog Practices	1.24	1.10	1.00	.91	.82	.82
SW Tools	1.24	1.10	1.00	.91	.83	.83
Required Dev Schedule	1.23	1.08	1.00	1.04	1.10	1,10

Total EAF = Product of the selected factors

Adjusted value of Effort: Adjusted Person Months:

APM = (Total EAF) * PM

	Organic	Semidetached	Embedded
a	3.2	3.0	2.8
b	1.05	1.12	1.20

Mode	Effort Formula
Organic	$E = 3.2 * (S^{1.05}) * C$
Semidetached	$E = 3.0 * (S^{1.12}) * C$
Embedded	$E = 2.8 * (S^{1.20}) * C$

e.g. Size = 200 KLOC

Effort = a * $Size^b$ * C

Cost drivers:

Low reliability .88

High product complexity 1.15

Low application experience 1.13

High programming language experience .95

C = .88 * 1.15 * 1.13 * .95 = 1.086

Organic — E = 3.2 * ($200^{1.05}$) * 1.086 = 906 staff-months

Semidetached — E = 3.0 * ($200^{1.12}$) * 1.086 = 1231 staff-months

Embedded — E = 2.8 * ($200^{1.20}$) * 1.086 = 1755 staff-months

Software Development Time

Development Time Equation Parameter Table:

Parameter	Organic	Semi-detached	Embedded
C	**2.5**	**2.5**	**2.5**
D	**0.38**	**0.35**	**0.32**

Development Time, TDEV = C * (APM **D)

Number of Personnel, NP = APM / TDEV

Distribution of Effort

- A development process typically consists of the following stages:
- \- Requirements Analysis
- \- Design (High Level + Detailed)
- \- Implementation & Coding
- \- Testing (Unit + Integration)

The following table gives the recommended percentage distribution of Effort (APM) and TDEV for these stages:

we dont take any liability for the notes correctness. http://www.rgpvonline.com

Percentage Distribution of Effort and Time Table:

	Req Analysis	Design, HLD + DD	Implementation	Testing	
Effort	23%	29%	22%	21%	100 %
TDEV	39%	25%	15%	21%	100 %

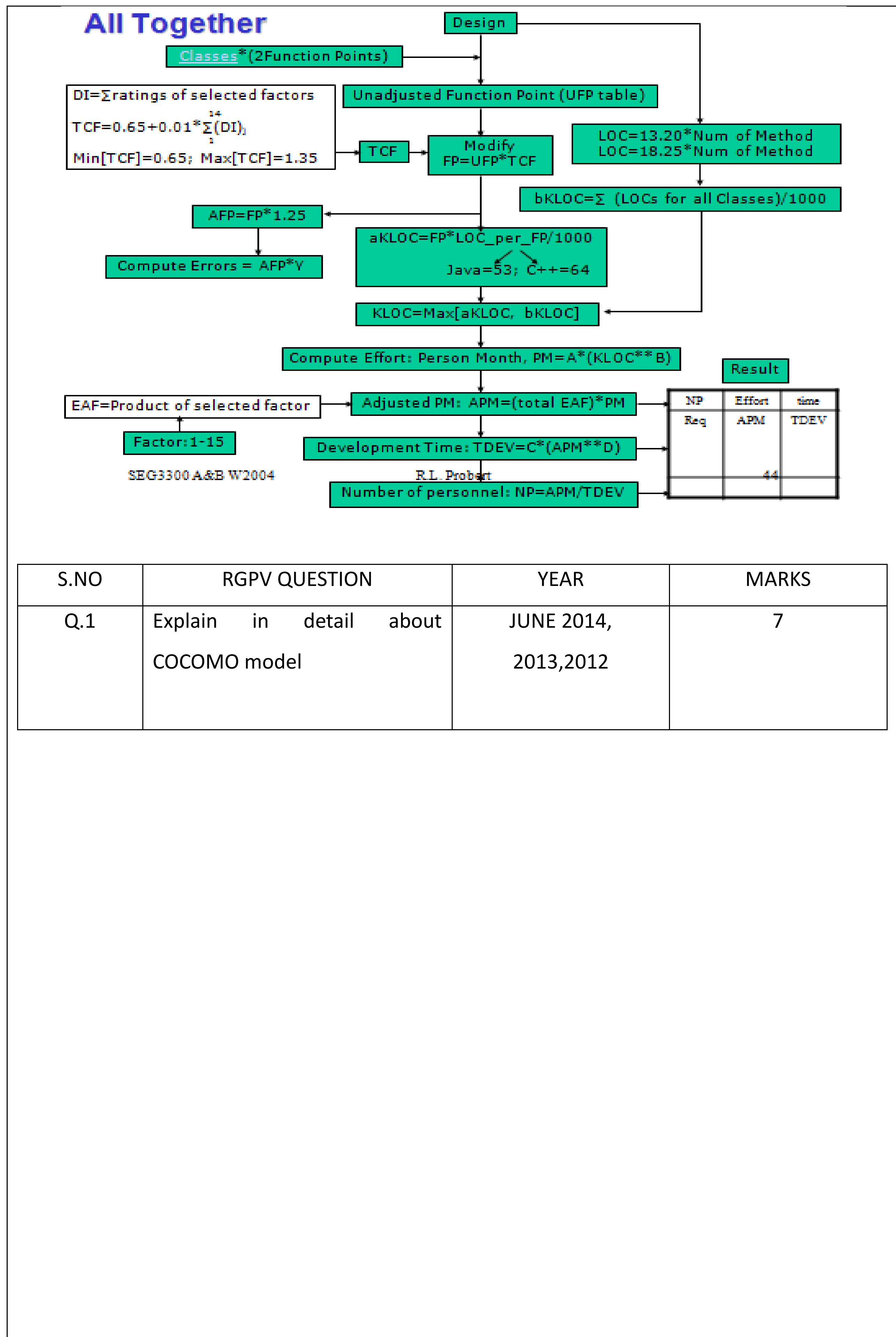
- Calculate the estimated number of errors in your design, i.e.total errors found in requirements, specifications, code, user manuals, and bad fixes:
  - Adjust the Function Point calculated in step1

*AFP = FP ** 1.25*

  - Use the following table for calculating error estimates
  -

Error Type	Error / AFP
Requirements	**1**
Design	**1.25**
Implementation	**1.75**
Documentation	**0.6**
Due to Bug Fixes	**0.4**

we dont take any liability for the notes correctness. http://www.rgpvonline.com



S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Explain in detail about COCOMO model	JUNE 2014, 2013,2012	7

UNIT-02/LECTURE-06

COCOMOII:

Constructive Cost Model II (COCOMO® II) is a model that allows one to estimate the cost, effort, and schedule when planning a new software development activity. COCOMO® II is the latest major extension to the original COCOMO® (COCOMO® 81) model published in 1981. It consists of three submodels, each one offering increased fidelity the further along one is in the project planning and design process. Listed in increasing fidelity, these submodels are called the Applications Composition, Early Design, and Post-architecture models.

COCOMO® II can be used for the following major decision situations

- Making investment or other financial decisions involving a software development effort
- Setting project budgets and schedules as a basis for planning and control
- Deciding on or negotiating tradeoffs among software cost, schedule, functionality, performance or quality factors
- Making software cost and schedule risk management decisions
- Deciding which parts of a software system to develop, reuse, lease, or purchase
- Making legacy software inventory decisions: what parts to modify, phase out, outsource, etc
- Setting mixed investment strategies to improve organization's software capability, via reuse, tools, process maturity, outsourcing, etc
- Deciding how to implement a process improvement strategy, such as that provided in the SEI CMM

The original COCOMO® model was first published by Dr. Barry Boehm in 1981, and reflected the software development practices of the day. In the ensuing decade and a half, software development techniques changed dramatically. These changes included a move away from mainframe overnight batch processing to desktop-based real-time turnaround; a greatly increased emphasis on reusing existing software and building new systems using off-the-shelf software components; and spending as much effort to design and manage the software development process as was once spent creating the software product.

we dont take any liability for the notes correctness. http://www.rgpvonline.com

These changes and others began to make applying the original COCOMO® model problematic. The solution to the problem was to reinvent the model for the 1990s. After several years and the combined efforts of USC-CSSE, ISR at UC Irvine, and the COCOMO® II Project Affiliate Organizations, the result is COCOMO® II, a revised cost estimation model reflecting the changes in professional software development practice that have come about since the 1970s. This new, improved COCOMO® is now ready to assist professional software cost estimators for many years to come.

About the Nomenclature

The original model published in 1981 went by the simple name of COCOMO®. This is an acronym derived from the first two letters of each word in the longer phrase Constructive Cost Model. The word constructive refers to the fact that the model helps an estimator better understand the complexities of the software job to be done, and by its openness permits the estimator to know exactly why the model gives the estimate it does. Not surprisingly, the new model (composed of all three submodels) was initially given the name COCOMO® 2.0. However, after some confusion in how to designate subsequent releases of the software implementation of the new model, the name was permanently changed to COCOMO® II. To further avoid confusion, the original COCOMO® model was also then re-designated COCOMO® 81. All references to COCOMO® found in books and literature published before 1995 refer to what is now called COCOMO® 81. Most references to COCOMO® published from 1995 onward refer to what is now called COCOMO® II.

If in examining a reference you are still unsure as to which model is being discussed, there are a few obvious clues. If in the context of discussing COCOMO® these terms are used: Basic, Intermediate, or Detailed for model names; Organic, Semidetached, or Embedded for development mode, then the model being discussed is COCOMO® 81. However, if the model names mentioned are Application Composition, Early Design, or Post-architecture; or if there is mention of scale factors Precedentedness (PREC), Development Flexibility (FLEX), Architecture/Risk Resolution (RESL), Team Cohesion (TEAM), or Process Maturity (PMAT), then the model being discussed is COCOMO® II.

Reverse engineering :

Reverse engineering, the process of taking a software program's binary code and recreating it

so as to trace it back to the original source code, is being widely used in computer hardware and software to enhance product features or fix certain bugs. For example, the programmer writes the code in a high-level language such as C, C++ etc. (you can learn basic C programming with this beginners course); as computers do not speak these languages, the code written in these programming languages needs to be assembled in a format that is machine specific. In short, the code written in high level language needs to be interpreted into low level or machine language.

The process of converting the code written in high level language into a low level language without changing the original program is known as reverse engineering. It's similar to disassembling the parts of a vehicle to understand the basic functioning of the machine and internal parts etc. And thereafter making appropriate adjustments to give rise to a better performing or superior vehicle.

If we have a look at the subject of reverse engineering in the context of software engineering, we will find that it is the practice of analyzing the software system to extract the actual design and implementation information. A typical reverse engineering scenario would comprise of a software module that has been worked on for years and carries the line of business in its code; but the original source code might be lost, leaving the developers only with the binary code. In such a case, reverse engineering skills would be used by software engineers to detect probable virus and malware to eventually protect the intellectual property of the company. Learn more protecting Intellectual Property in this course.

At the turn of the century, when the software world was hit by the technology crisis Y2K, programmers weren't equipped with reverse engineering skills. Since then, research has been carried out to analyse what kind of development activities can be brought under the category of reverse engineering so that they can be taught to the programmers. Researchers have revealed that reverse engineering basically comes under two categories-software development and software testing. A number of reverse engineering exercises have been developed since then in this regard to provide baseline education in reversing the machine code.

Reverse Engineering

Reverse engineering can be applied to several aspects of the software and hardware

development activities to convey different meanings. In general, it is defined as the process of creating representations of systems at a higher level of abstraction and understanding the basic working principle and structure of the systems under study. With the help of reverse engineering, the software system that is under consideration can be examined thoroughly. There are two types of reverse engineering; in the first type, the source code is available, but high-level aspects of the program are no longer available. The efforts that are made to discover the source code for the software that is being developed is known as reverse engineering. In the second case, the source code for the software is no longer available; here, the process of discovering the possible source code is known as reverse engineering. To avoid copyright infringement, reverse engineering makes use of a technique called clean room design.

In the world of reverse engineering, we often hear about black box testing. Even though the tester has an API, their ultimate goal is to find the bugs by hitting the product hard from outside. Learn more about different software testing techniques in this course.

Apart from this, the main purpose of reverse engineering is to audit the security, remove the copy protection, customize the embedded systems, and include additional features without spending much and other similar activities.

Where is Reverse Engineering Used?

Reverse engineering is used in a variety of fields such as software design, software testing, programming etc.

In software design, reverse engineering enables the developer or programmer to add new features to the existing software with or without knowing the source code. Different techniques are used to incorporate new features into the existing software.

- Reverse engineering is also very beneficial in software testing, as most of the virus programmers don’t leave behind instructions on how they wrote the code, what they have set out to accomplish etc. Reverse engineering helps the testers to study the virus and other malware code. The field of software testing, while very extensive, is also interesting and requires vast experience to study and analyze virus code. Learn more about software test design in this course.
- The third category where reverse engineering is widely used is in software security.

Reverse engineering techniques are used to make sure that the system does not have any major vulnerabilities and security flaws. The main purpose of reverse engineering is to make the system robust so as to protect it from spywares and hackers. Infact, this can be taken a step forward to Ethical hacking, whereby you try to hack your own system to identify vulnerabilities. You can learn more about Ethical hacking with this course.

- While one needs a vast amount of knowledge to become a successful reverse engineer, he or she can definitely have a lucrative career in this field by starting off with the basics. It is highly suggested that you first become familiar with assembly level language and gain significant amount of practical knowledge in the field of software designing and testing to become a successful software engineer. Learn how to kick-start your career in this interesting field by visiting our online course agile testing for reverse engineering applications.

Reverse Engineering Tools

As mentioned above, reverse engineering is the process of analyzing the software to determine its components and their relationships. The process of reverse engineering is accomplished by making use of some tools that are categorized into debuggers or disassemblers, hex editors, monitoring and decompile tools:

Disassemblers – A 36artridges36 is used to convert binary code into assembly code and also used to extract strings, imported and exported functions, libraries etc. The disassemblers convert the machine language into a user-friendly format. There are different dissemblers that specialize in certain things.

Debuggers – This tool expands the functionality of a 36artridges36 by supporting the CPU registers, the hex duping of the program, view of stack etc. Using debuggers, the programmers can set breakpoints and edit the assembly code at run time. Debuggers analyse the binary in a similar way as the disassemblers and allow the reverser to step through the code by running one line at a time to investigate the results.

Hex Editors – These editors allow the binary to be viewed in the editor and change it as per the requirements of the software. There are different types of hex editors available that are used for different functions.

PE and Resource Viewer – The binary code is designed to run on a windows based machine and has a very specific data which tells how to set up and initialize a program. All the programs that run on windows should have a portable executable that supports the DLLs the program needs to borrow from.

Reverse engineering has developed significantly and taken a positive approach to creating descriptive data set of the original object. Today, there are numerous legitimate applications of reverse engineering. Due to the development of numerous digitizing devices, reverse engineering software enables programmers to manipulate the data into a useful form. The kind of applications in which reverse engineering is used ranges from mechanical to digital, each with its own advantages and applications. Reverse engineering is also beneficial for business owners as they can incorporate advanced features into their software to meet the demands of the growing markets.

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Write short note on Reverse engineering?	June 2013	5

we dont take any liability for the notes correctness.

UNIT-02/LECTURE-07

What is reverse engineering (RE)? : [RGPV/JUNE 2(5)]

Reverse Engineering (RE): “disassemble or analyze in detail in order to discover concepts involved in manufacture.” – “reverse engineer.” The Merriam-Webster Dictionary, New ed. 2004.

Reverse engineering is “the process of discovering the technological principles of a mechanical application through analysis of its structure, function and operation. That involves sometimes taking something apart and analyzing its workings in detail, usually with the intention to construct a new device or program that does the same thing without actually copying anything from the original.”

5. **What are some common uses for reverse engineering?**

- As a learning tool.
- As a way to make new compatible products that are cheaper than what’s currently on the market.
- for making software interoperate more effectively or to bridge different operating systems or databases.
- To uncover the uncoordinated features of commercial products

.This kind of inquiry engages individuals in a constructive learning process about the operation of systems and products. The process of taking something apart and revealing the way in which it works is an effective way to learn how to build a technology or make improvements to it.

According to A Methodology for Reverse Engineering, reverse engineering consists of the following steps:

- Observe and assess the mechanisms that make the device work.
- Dissect and study the inner workings of a mechanical device.
- Compare the actual device to your observations and suggest improvement.

Through reverse engineering, a researcher can gather the technical data necessary for the documentation of the operation of a technology or component of a system. When reverse engineering software, researchers are able to examine the strength of systems and identify their weaknesses in terms of performance, security, and interoperability. The reverse engineering

process allows researchers to understand both how a program works and also what aspects of the program contribute to its not working. Independent manufacturers can participate in a competitive market that rewards the improvements made on dominant products. For example, security audits, which allow users of software to better protect their systems and networks by revealing security flaws, require reverse engineering. The creation of better designs and the interoperability of existing products often begin with reverse engineering.

6. How is reverse engineering implemented legally?

There are two basic legalities associated with reverse engineering:

a. Copyright Protection – protects only the look and shape of a product.
b. Patent Protection – protects the the idea behind the functioning of a new product.

According to npd-solutions a patent is no more than a warning sign to a competitor to discourage competition. Also npd-solutions says that if there is merit in an idea, a competitor will do one of the following:

- Negotiate a license to use the idea.
- Claim that the idea is not novel and is an obvious step for anyone experienced in the particular field.
- Make a subtle change and claim that the changed product is not protected by patent.

Commonly, RE is performed using the clean-room or Chinese wall. Clean-room, reverse engineering is conducted in a sequential manner:

- A team of engineers are sent to disassemble the product to investigate and describe what it does in as much detail as possible at a somewhat high level of abstraction.
- Description is given to another group who has no previous or current knowledge of the product.
- Second party then builds product from the given description. This product might achieve the same end effect but will probably have a different solution approach.

7. What are some legal cases and ethical issues involving reverse engineering?

New court cases reveal that reverse engineering practices which are used to achieve interoperability with an 39artridges39ly created computer program, are legal and ethical. In

December, 2002, Lexmark filed suit against SCC, accusing it of violating copyright law as well as the DMCA. SCC reverse engineered the code contained in Lexmark printer 40artridge so that it could manufacture compatible cartiges. According to Computerworld , Lexmark "alleged that SCC's Smartek chips include Lexmark software that is protected by copyright. The software handles communication between Lexmark printers and toner 40artridges; without it, refurbished toner 40artridges won't work with Lexmark's printers." The court ruled that "copyright law shouldn't be used to inhibit interoperability between one vendor's products and those of its rivals. In a ruling from the U.S. Copyright Office in October 2003, the Copyright Office said "the DMCA doesn't block software developers from using reverse engineering to access digitally protected copyright material if they do so to achieve interoperability with an independently created computer program."

8. Is reverse engineering unethical?

This issue is largely debated and does not seem to have a clear cut answer. The number one argument against reverse engineering is that of intellectual property. If an individual or an organization produces a product or idea, is it ok for others to "disassemble" the product in order to discover the inner workings? Lexmark does not think so. Since Lexmark and companies like them spend time and money to develop products, they find it unethical that others can reverse engineer their products. There are also products like Bit Keeper that have been hurt by reverse engineering practices. Why should companies and individuals spend major resources to gather intellectual property that may be reversed engineered by competitors at a fraction of the cost?

There are also benefits to reverse engineering. Reverse engineering might be used as a way to allow products to interoperate. Also reverse engineering can be used as a check so that computer software isn't performing harmful, unethical, or illegal activities.

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Write short note on Reverse engineering.	June.2013	5

UNIT-02/LECTURE-08

Project Scheduling

Overview

The chapter describes the process of building and monitoring schedules for software development projects. To build complex software systems, many engineering tasks need to occur in parallel with one another to complete the project on time. The output from one task often determines when another may begin. Software engineers need to build activity networks that take these task interdependencies into account. Managers find that it is difficult to ensure that a team is working on the most appropriate tasks without building a detailed schedule and sticking to it. This requires that tasks are assigned to people, milestones are created, resources are allocated for each task, and progress is tracked.

Root Causes for Late Software

- Unrealistic deadline established outside the team
- Changing customer requirements not reflected in schedule changes
- Underestimating the resources required to complete the project
- Risks that were not considered when project began
- Technical difficulties that complete not have been predicted in advance
- Human difficulties that complete not have been predicted in advance
- Miscommunication among project staff resulting in delays
- Failure by project management to recognize project failing behind schedule and failure to take corrective action to correct problems

How to Deal With Unrealistic Schedule Demands

1. Perform a detailed project estimate for project effort and duration using historical data.
2. Use an incremental process model that will deliver critical functionality imposed by deadline, but delay other requested functionality.
3. Meet with the customer and explain why the deadline is unrealistic using your estimates based on prior team performance.
4. Offer an incremental development and delivery strategy as an alternative to increasing resources or allowing the schedule to slip beyond the deadline.

Project Scheduling Perspectives

- Project scheduling is an activity that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks.
- One view is that the end-date for the software release is set externally and that the software organization is constrained to distribute effort in the prescribed time frame.
- Another view is that the rough chronological bounds have been discussed by the developers and customers, but the end-date is best set by the developer after carefully considering how to best use the resources needed to meet the customer’s needs.
- Schedules evolve over time as the first macroscopic schedules is refined into the detailed schedule for each planned software increment.

we dont take any liability for the notes correctness. http://www.rgpvonline.com

Software Project Scheduling Principles

- Compartmentalization – the product and process must be decomposed into a manageable number of activities and tasks
- Interdependency – tasks that can be completed in parallel must be separated from those that must completed serially
- Time allocation – every task has start and completion dates that take the task interdependencies into account
- Effort validation – project manager must ensure that on any given day there are enough staff members assigned to completed the tasks within the time estimated in the project plan
- Defined Responsibilities – every scheduled task needs to be assigned to a specific team member
- Defined outcomes – every task in the schedule needs to have a defined outcome (usually a work product or deliverable)
- Defined milestones – a milestone is accomplished when one or more work products from an engineering task have passed quality review

Relationship Between People and Effort : [RGPV/JUNE 2013 (7)]

- Adding people to a project after it is behind schedule often causes the schedule to slip further
- The relationship between the number of people on a project and overall productivity is not linear (e.g. 3 people do not produce 3 times the work of 1 person, if the people have to work in cooperation with one another)
- The main reasons for using more than 1 person on a project are to get the job done more rapidly and to improve software quality.

Project Effort Distribution

- The 40-20-40 rule:
  - 40% front-end analysis and design
  - 20% coding
  - 40% back-end testing
- Generally accepted guidelines are:
  - 02-03 % planning
  - 10-25 % requirements analysis
  - 20-25 % design
  - 15-20 % coding
  - 30-40 % testing and debugging

Software Project Types

1. Concept development – initiated to explore new business concept or new application of technology
2. New application development – new product requested by customer
3. Application enhancement – major modifications to function, performance, or interfaces (observable to user)
4. Application maintenance – correcting, adapting, or extending existing software (not

we dont take any liability for the notes correctness. http://www.rgpvonline.com

immediately obvious to user)

5. Reengineering – rebuilding all (or part) of a legacy system

Factors Affecting Task Set

- Size of project
- Number of potential users
- Mission criticality
- Application longevity
- Requirement stability
- Ease of customer/developer communication
- Maturity of applicable technology
- Performance constraints
- Embedded/non-embedded characteristics
- Project staffing
- Reengineering factors

Concept Development Tasks

- **Concept scoping** – determine overall project scope
- **Preliminary concept planning** – establishes development team's ability to undertake the proposed work
- **Technology risk assessment** – evaluates the risk associated with the technology implied by the software scope
- **Proof of concept** – demonstrates the feasibility of the technology in the software context
- **Concept implementation** – concept represented in a form that can be used to sell it to the customer
- **Customer reaction to concept** – solicits feedback on new technology from customer

Scheduling

- Task networks (activity networks) are graphic representations can be of the task interdependencies and can help define a rough schedule for particular project
- Scheduling tools should be used to schedule any non-trivial project.
- *Program evaluation and review technique* (PERT) and *critical path method* (CPM)) are quantitative techniques that allow software planners to identify the chain of dependent tasks in the project *work breakdown structure* (WBS) that determine the project duration time.
- *Timeline* (Gantt) *charts* enable software planners to determine what tasks will be need to be conducted at a given point in time (based on estimates for effort, start time, and duration for each task).
- The best indicator of progress is the completion and successful review of a defined software work product.
- Time-boxing is the practice of deciding a priori the fixed amount of time that can be spent on each task. When the task's time limit is exceeded, development moves on to the next task (with the hope that a majority of the critical work was completed before time ran out).

Tracking Project Schedules

we dont take any liability for the notes correctness. http://www.rgpvonline.com

- Periodic project status meetings with each team member reporting progress and problems
- Evaluation of results of all work product reviews
- Comparing actual milestone completion dates to scheduled dates
- Comparing actual project task start-dates to scheduled start-dates
- Informal meeting with practitioners to have them asses subjectively progress to date and future problems
- Use earned value analysis to assess progress quantitatively

Tracking Increment Progress for OO Projects

- Technical milestone: OO analysis complete
    - All hierarchy classes defined and reviewed
    - Class attributes and operations are defined and reviewed
    - Class relationships defined and reviewed
    - Behavioral model defined and reviewed
    - Reusable classed identified
- Technical milestone: OO design complete
    - Subsystems defined and reviewed
    - Classes allocated to subsystems and reviewed
    - Task allocation has been established and reviewed
    - Responsibilities and collaborations have been identified
    - Attributes and operations have been designed and reviewed
    - Communication model has been created and reviewed
- Technical milestone: OO programming complete
    - Each new design model class has been implemented
    - Classes extracted from the reuse library have been implemented
    - Prototype or increment has been built
- Technical milestone: OO testing complete
    - The correctness and completeness of the OOA and OOD models has been reviewed
    - Class-responsibility-collaboration network has been developed and reviewed
    - Test cases are designed and class-level tests have been conducted for each class
    - Test cases are designed, cluster testing is completed, and classes have been integrated
    - System level tests are complete

WebApp Project Scheduling

- During the first iteration a macroscopic is developed by allocating effort to specific tasks (it is understood that this is changeable schedule)
- The project is broken up into increments and the increments are refined in to detailed schedules as each is begun (some increments may be developed in parallel)
- Each task on the task list for each increment is adapted in one of four ways as its detailed schedule is created
    - task is applied as is
    - task is eliminated because it is not necessary for the increment
    - new (custom) task is added
    - task is refined (elaborated) into a number of named subtasks that each becomes part of the schedule
- This process continues until each planned increment is completed

we dont take any liability for the notes correctness. http://www.rgpvonline.com

Earned Value Analysis

- Earned value is a quantitative measure given to each task as a percent of project completed so far.
  1. The total hours to complete each project task are estimated (BCWS – budgeted cost of work scheduled)
  2. The effort to complete the project is computed by summing the effort to complete each task (BAC – budget at completion)
  3. Each task is given an earned value based on its estimated percentage contribution to the total (BCWP – budgeted cost of work completed).
- It is compute the actual cost of work performed (ACWP) at any point in the project and to compute progress indicators for both schedule and costs based on these measures

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	What is the role of effort estimation in a project & why is it important to do this estimation early?	JUNE 2013	7
Q.2	Explain problem based estimation technique.	JUNE 2013	10

we dont take any liability for the notes correctness. http://www.rgpvonline.com

UNIT-02/LECTURE-09

Project Scheduling (Basic Principles): [RGPV/JUNE 20113(5)]

Software project scheduling is an activity that distributes estimated effort across the planed project duration by allocating the effort to specific software engineering tasks.

First, a macroscopic schedule is developed.

---> a detailed schedule is redefined for each entry in the macroscopic schedule.

A schedule evolves over time.

Basic principles guide software project scheduling:

- Compartmentalization
- Interdependency
- Time allocation
- Effort allocation
- Effort validation
- Defined responsibilities
- Defined outcomes
- Defined milestones

Defining A Task Set For The Software Project

There is no single set of tasks that is appropriate for all projects.

An effective software process should define a collection of task sets, each

designed to meet the needs of different types of projects.

A task set is a collection of software engineering work

-> tasks, milestones, and deliverables.

Tasks sets are designed to accommodate different types of projects and different degrees of rigor.

Typical project types:

- Concept Development Projects
- New Application Development Projects
- Application Enhancement Projects
- Application Maintenance Projects
- Reengineering Projects

Obtaining Information

Degree of Rigor:

we dont take any liability for the notes correctness. http://www.rgpvonline.com

- Casual
- Structured
- Strict
- Quick Reaction

Defining Adaptation Criteria:

-- This is used to determine the recommended degree of rigor.

Eleven criteria are defined for software projects:

- Size of the project
- Number of potential users
- Mission criticality
- Application longevity
- Ease of customer/developer communication
- Maturity of applicable technology
- Performance constraints
- Embedded/non-embedded characteristics
- Project staffing
- Reengineering factors

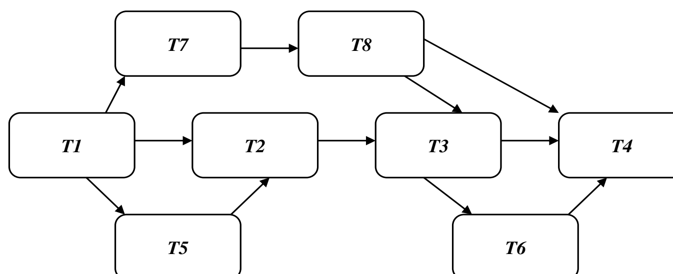
Defining A Task Network

Defining A Task Network

Individual tasks and subtasks have interdependencies based on their sequence.

A task network is a graphic representation of the task flow for a project. Figure 7.3 shows a schematic network for a concept development project.

Critical path:
-- the tasks on a critical path must be completed on schedule to make the whole project on schedule.



Scheduling of a software project does not differ greatly from scheduling of any multitask engineering effort.

Two project scheduling methods:

- Program Evaluation and Review Technique (PERT)
- Critical Path Method (CPM)

Both methods are driven by information developed in earlier project planning activities:

- Estimates of effort
- A decomposition of product function
- The selection of the appropriate process model
- The selection of project type and task set

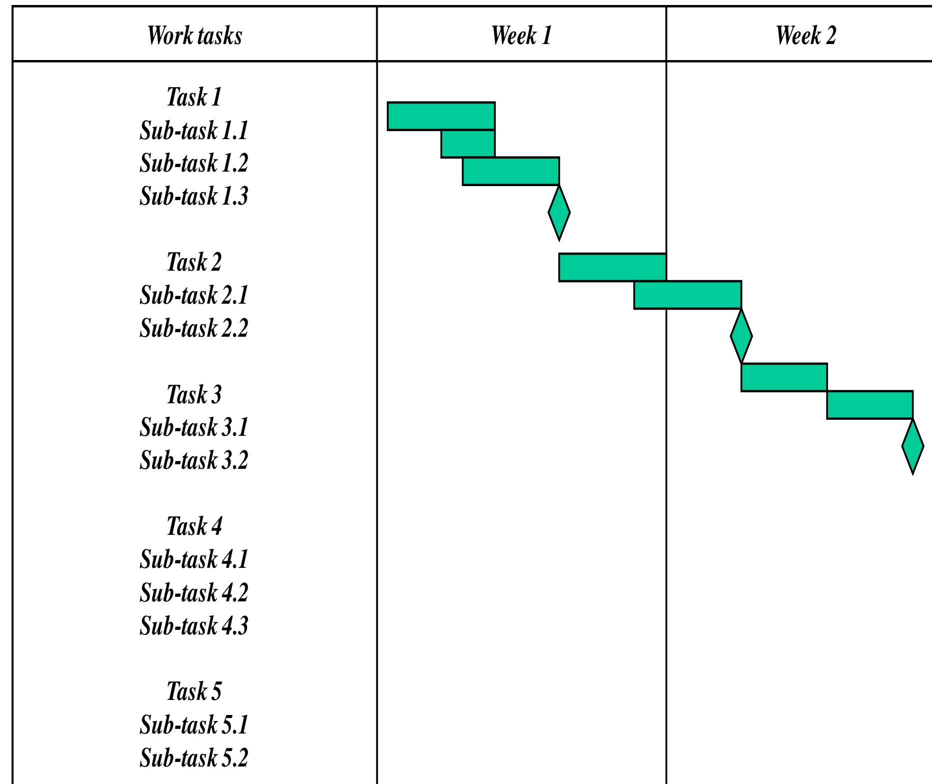
Both methods allow a planer to do:

- determine the critical path
- time estimation
- calculate boundary times for each task

Boundary times:

- the earliest time and latest time to begin a task
- the earliest time and latest time to complete a task
- the total float.

we dont take any liability for the notes correctness. http://www.rgpvonline.com



Tracking the Schedule

The project schedule provides a road map for a software project manager.

It defines the tasks and milestones.

Several ways to track a project schedule:

- conducting periodic project status meeting
- evaluating the review results in the software process
- determine if formal project milestones have been accomplished (Figure 7.4)
- compare actual start date to planned start date for each task
- informal meeting with practitioners

Project manager takes the control of the schedule in the aspects of:

- project staffing

we dont take any liability for the notes correctness. http://www.rgpvonline.com

- project problems
- project resources
- reviews
- project budget

The Project Plan

The software project plan is a relatively brief document that is addressed to a diverse audience.

It must consists the following:

- communication scope, resources, staffing, and customer
- define risks and risk management techniques
- define cost and schedule for management review
- provide an overall approach to software development
- outline how quality will be ensured and change will be managed.

The plan concentrates on a general statement of what and a specific statement of how much and how long.

The purpose of a project plan is:

-> help establish the viability of the software development effort.

The project plan need not be a lengthy and complex document.

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Write short note on task scheduling with an example.	June 2011	5

REFERENCCE

BOOK	**AUTHOR**	**PRIORITY**
Software Engineering	P,S. Pressman	1
Software Engineering	Pankaj jalote	2

we dont take any liability for the notes correctness. http://www.rgpvonline.com