



JAMAL MOHAMED COLLEGE (AUTONOMOUS)

DEPARTMENT OF COMPUTER APPLICATIONS

TRICHY-20.

SEMESTER - VI: CORE XV: SOFTWARE ENGINEERING

Course Code: 20UCA6CC15

UNIT-5

Quality Concepts – Software quality assurance – Software reviews – Formal technical reviews – Software measurement – Metrics for software quality

TOPIC-1 QUALITY CONCEPTS:-

- Variation control is the heart of quality control.
- A manufacturer wants to minimize the variation among the products that are produced, even when doing something relatively simple like duplicating diskettes.

In this concept, we should understand some clear ideas about the following:-

1. **Quality**
2. **Quality Control**
3. **Quality Assurance**
4. **Cost of Quality**

Let us discuss one by one.

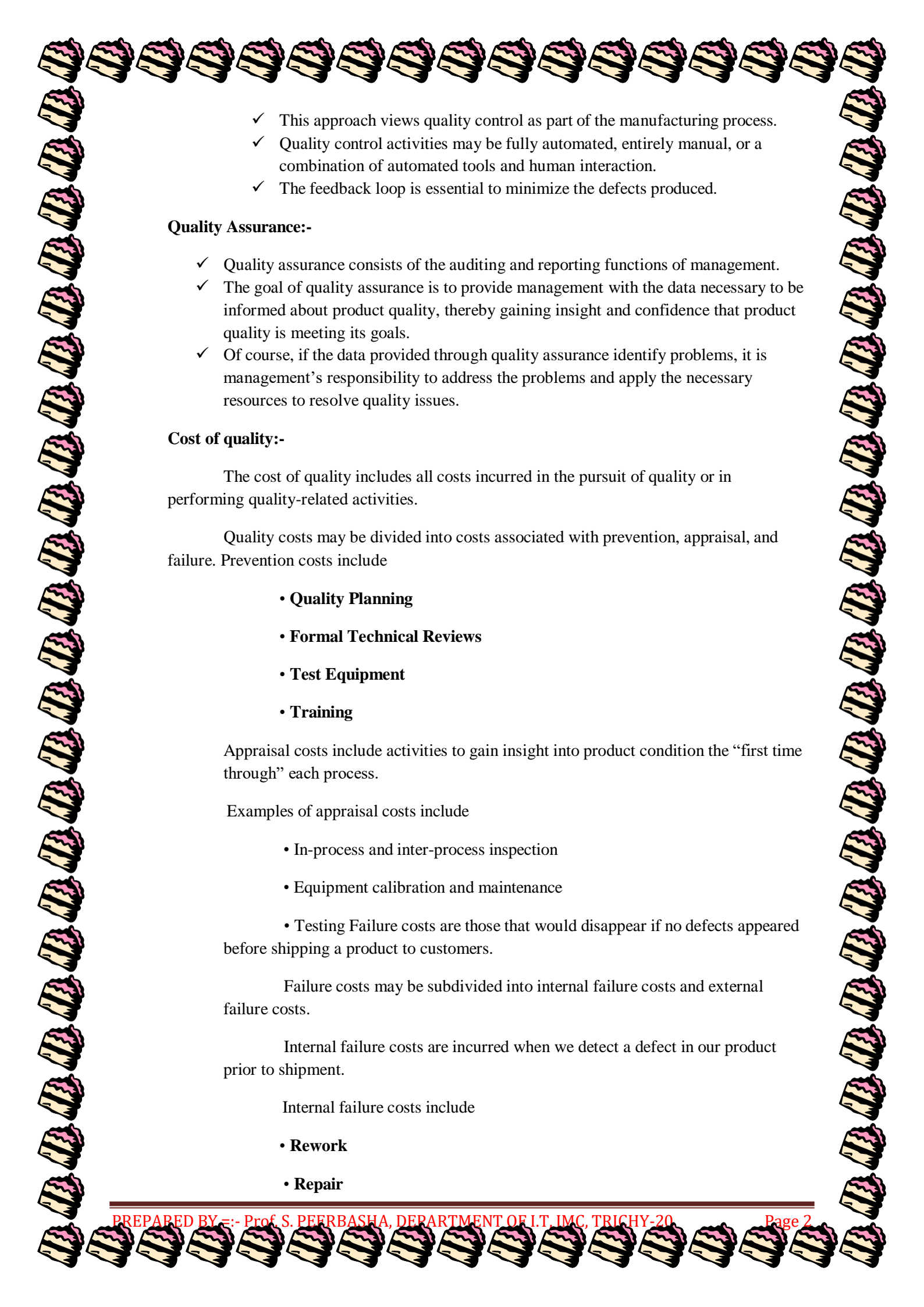
Quality:-

- ✓ The American Heritage Dictionary defines quality as “a characteristic or attribute of something.”
- ✓ When we examine an item based on its measurable characteristics, two kinds of quality may be encountered: quality of design and quality of conformance.
- ✓ Quality of design refers to the characteristics that designers specify for an item. The grade of materials, tolerances, and performance specifications all contribute to the quality of design.
- ✓ Quality of conformance is the degree to which the design specifications are followed during manufacturing.

User satisfaction = compliant product + good quality + delivery within budget and schedule

Quality Control:-

- ✓ Quality control involves the series of inspections, reviews, and tests used throughout the software process to ensure each work product meets the requirements placed upon it.
- ✓ Quality control includes a feedback loop to the process that created the work product.
- ✓ The combination of measurement and feedback allows us to tune the process when the work products created fail to meet their specifications.

- 
- ✓ This approach views quality control as part of the manufacturing process.
 - ✓ Quality control activities may be fully automated, entirely manual, or a combination of automated tools and human interaction.
 - ✓ The feedback loop is essential to minimize the defects produced.

Quality Assurance:-

- ✓ Quality assurance consists of the auditing and reporting functions of management.
- ✓ The goal of quality assurance is to provide management with the data necessary to be informed about product quality, thereby gaining insight and confidence that product quality is meeting its goals.
- ✓ Of course, if the data provided through quality assurance identify problems, it is management's responsibility to address the problems and apply the necessary resources to resolve quality issues.

Cost of quality:-

The cost of quality includes all costs incurred in the pursuit of quality or in performing quality-related activities.

Quality costs may be divided into costs associated with prevention, appraisal, and failure. Prevention costs include

- **Quality Planning**
- **Formal Technical Reviews**
- **Test Equipment**
- **Training**

Appraisal costs include activities to gain insight into product condition the "first time through" each process.

Examples of appraisal costs include

- In-process and inter-process inspection
- Equipment calibration and maintenance
- Testing Failure costs are those that would disappear if no defects appeared before shipping a product to customers.

Failure costs may be subdivided into internal failure costs and external failure costs.

Internal failure costs are incurred when we detect a defect in our product prior to shipment.

Internal failure costs include

- **Rework**
- **Repair**



- **Failure mode analysis**

External failure costs are associated with defects found after the product has been shipped to the customer.

Examples of external failure costs are:-

- **Complaint resolution**
- **Product return and replacement**
- **Help line support**
- **Warranty work**

TOPIC-2 SOFTWARE QUALITY ASSURANCE

The software quality emphasis the following three points.

1. Software requirements are the foundation from which quality is measured. Lack of conformance to requirements is lack of quality.
2. Specified standards define a set of development criteria that guide the manner in which software is engineered. If the criteria are not followed, lack of quality will almost surely result.
3. A set of implicit requirements often goes unmentioned (e.g., the desire for ease of use and good maintainability). If software conforms to its explicit requirements but fails to meet implicit requirements, software quality is suspect.

The main concept to discuss here is about:-

- **BACKGROUND ISSUES**
- **SQA ACTIVITIES**

BACKGROUND ISSUES:-

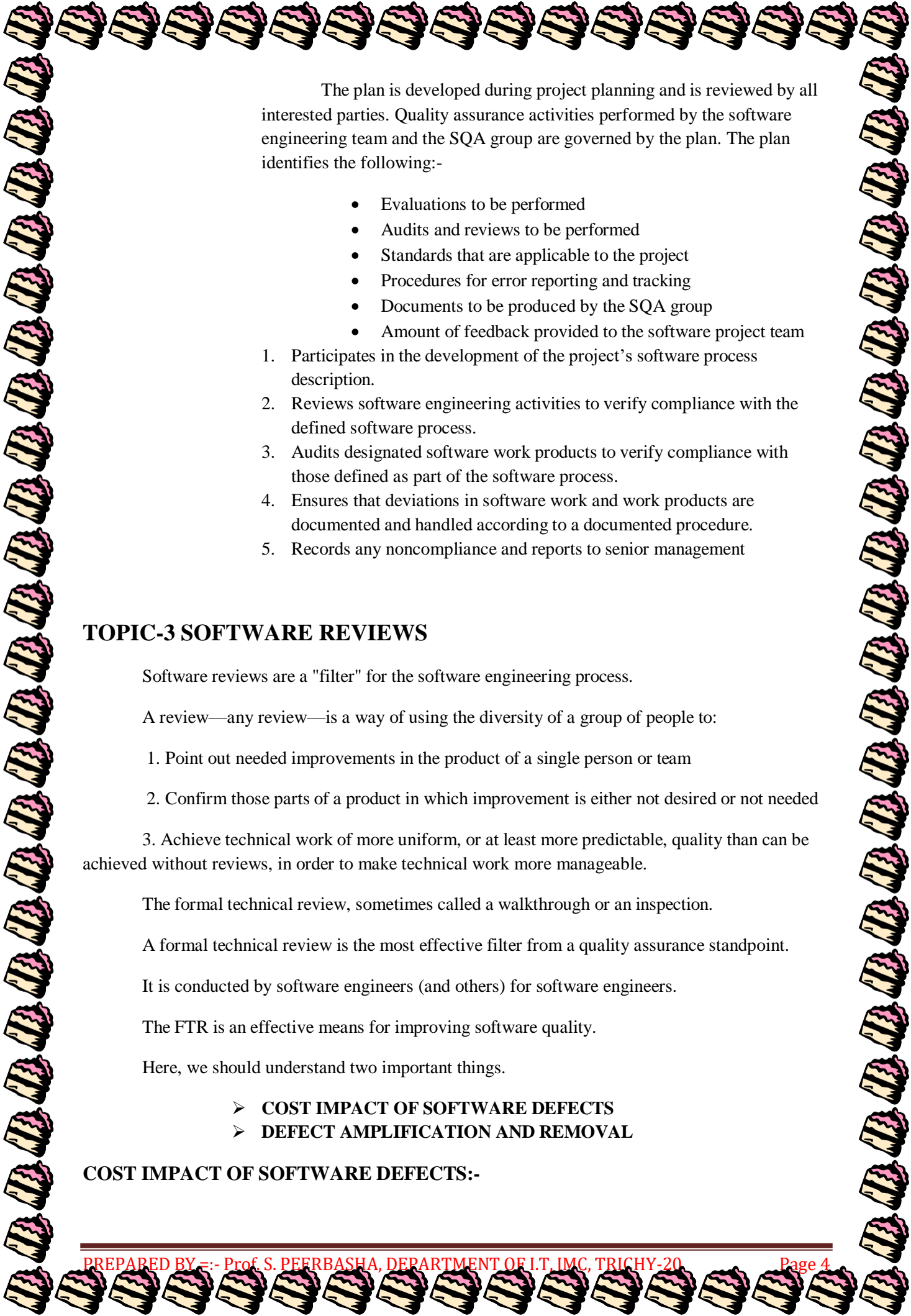
Quality assurance is an essential activity for any business that produces products to be used by others.

The history of quality assurance in software development parallels the history of quality in hardware manufacturing. During the early days of computing (1950s and 1960s), quality was the sole responsibility of the programmer. Standards for quality assurance for software were introduced in military contract software development during the 1970s

SQA ACTIVITIES:-

Software quality assurance is composed of a variety of tasks associated with two different constituencies—the software engineers who do technical work and an SQA group that has responsibility for quality assurance planning, oversight, record keeping, analysis, and reporting.

Prepares a SQA plan for a project:-



The plan is developed during project planning and is reviewed by all interested parties. Quality assurance activities performed by the software engineering team and the SQA group are governed by the plan. The plan identifies the following:-

- Evaluations to be performed
 - Audits and reviews to be performed
 - Standards that are applicable to the project
 - Procedures for error reporting and tracking
 - Documents to be produced by the SQA group
 - Amount of feedback provided to the software project team
1. Participates in the development of the project's software process description.
 2. Reviews software engineering activities to verify compliance with the defined software process.
 3. Audits designated software work products to verify compliance with those defined as part of the software process.
 4. Ensures that deviations in software work and work products are documented and handled according to a documented procedure.
 5. Records any noncompliance and reports to senior management

TOPIC-3 SOFTWARE REVIEWS

Software reviews are a "filter" for the software engineering process.

A review—any review—is a way of using the diversity of a group of people to:

1. Point out needed improvements in the product of a single person or team
2. Confirm those parts of a product in which improvement is either not desired or not needed
3. Achieve technical work of more uniform, or at least more predictable, quality than can be achieved without reviews, in order to make technical work more manageable.

The formal technical review, sometimes called a walkthrough or an inspection.

A formal technical review is the most effective filter from a quality assurance standpoint.

It is conducted by software engineers (and others) for software engineers.

The FTR is an effective means for improving software quality.

Here, we should understand two important things.

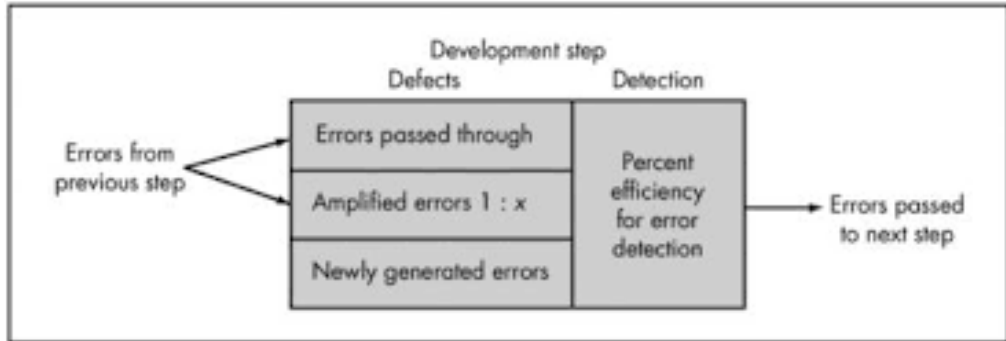
- **COST IMPACT OF SOFTWARE DEFECTS**
- **DEFECT AMPLIFICATION AND REMOVAL**

COST IMPACT OF SOFTWARE DEFECTS:-

(a) A defect in a hardware device or component; for example, a short circuit or broken wire.

(b) An incorrect step, process, or data definition in a computer program.

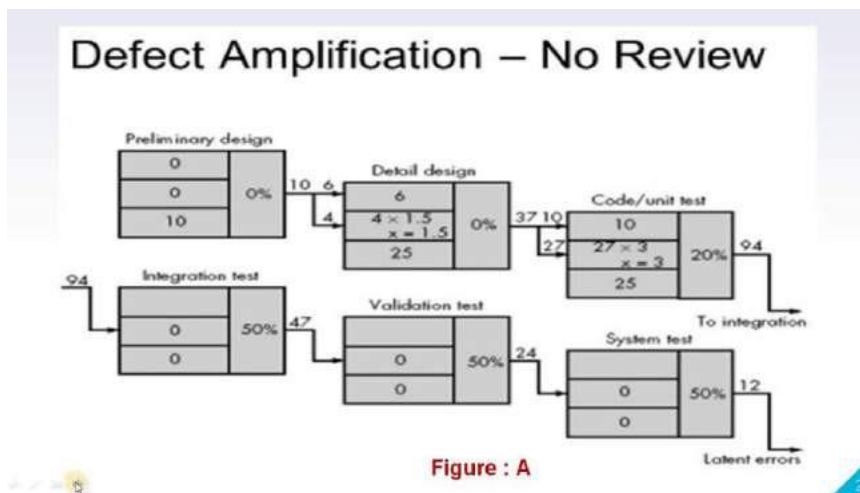
The primary objective of formal technical reviews is to find errors during the process so that they do not become defects after release of the software.



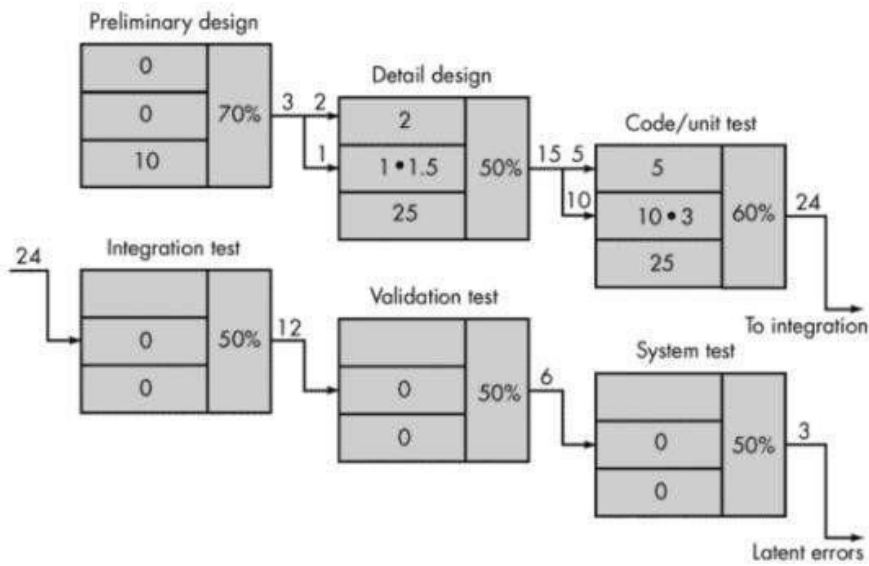
DEFECT AMPLIFICATION AND REMOVAL

A defect amplification model can be used to illustrate the generation and detection of errors during the preliminary design, detail design, and coding steps of the software engineering process.

DEFECT AMPLIFICATION MODEL WITHOUT REVIEWS (NO REVIEWS):-



DEFECT AMPLIFICATION MODEL WITH REVIEWS:-



TOPIC-4 FORMAL TECHNICAL REVIEWS

A formal technical review is a software quality assurance activity performed by software engineers.

The objectives of the FTR are:-

- ✓ To uncover errors in function, logic, or implementation for any representation of the software.
- ✓ To verify that that the software under review meets its requirements.
- ✓ To ensure that the software has been represented according to predefined standards.
- ✓ To achieve software that is developed in a uniform manner.
- ✓ To make projects more manageable.

The FTR is actually a class of reviews that includes walkthroughs, inspections, round-robin reviews and other small group technical assessments of software.

Each FTR is conducted as a meeting and will be successful only if it is properly planned, controlled, and attended.

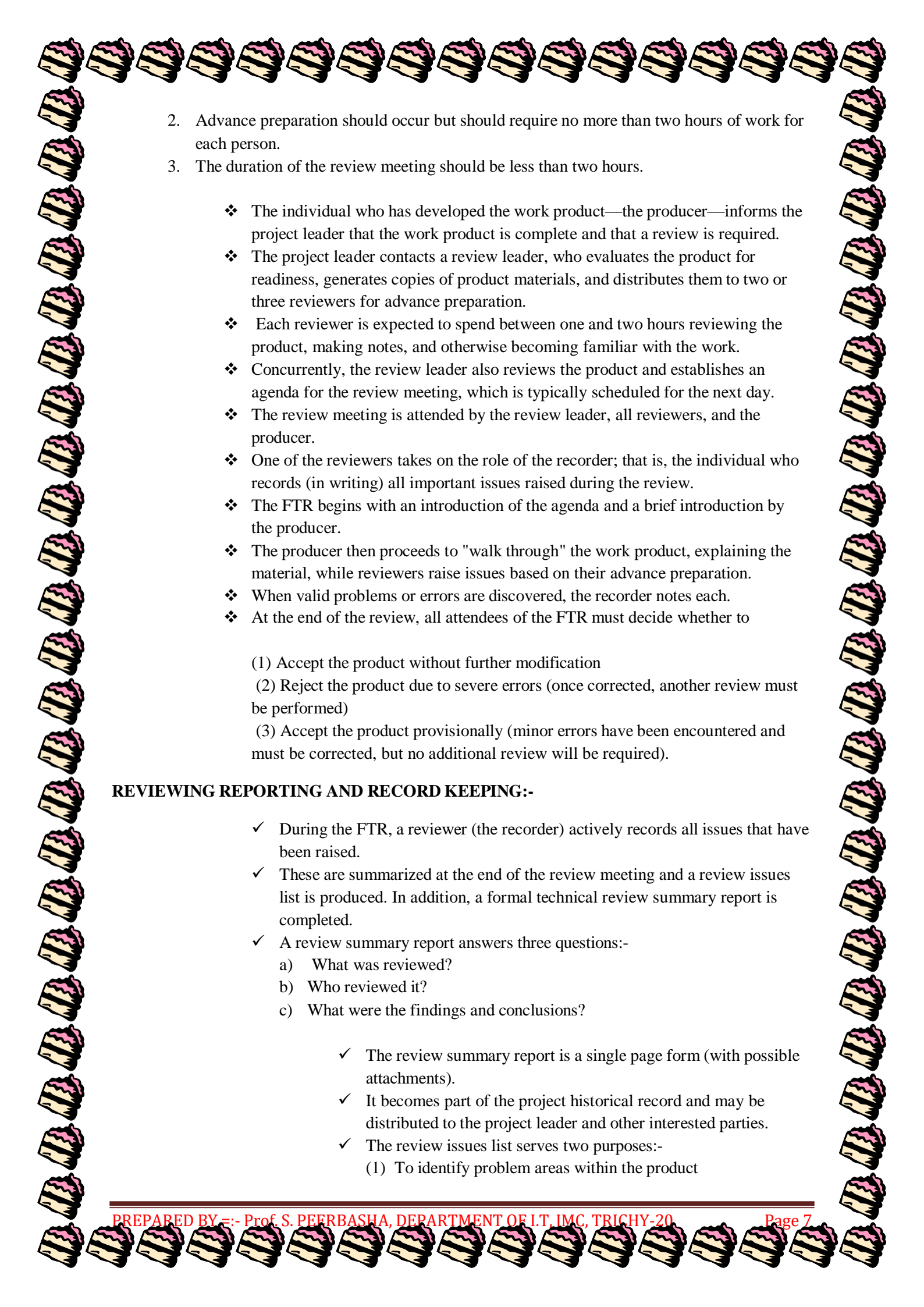
Here, we discuss about the three important things. They are:-

1. **THE REVIEW MEETING**
2. **REVIEW REPORTING AND RECORD KEEPING**
3. **REVIEW GUIDELINES**

THE REVIEW MEETING:-

The Review Meeting will be conducted with the following constraints.

1. Between three and five people (typically) should be involved in the review.

- 
2. Advance preparation should occur but should require no more than two hours of work for each person.
 3. The duration of the review meeting should be less than two hours.

- ❖ The individual who has developed the work product—the producer—informs the project leader that the work product is complete and that a review is required.
- ❖ The project leader contacts a review leader, who evaluates the product for readiness, generates copies of product materials, and distributes them to two or three reviewers for advance preparation.
- ❖ Each reviewer is expected to spend between one and two hours reviewing the product, making notes, and otherwise becoming familiar with the work.
- ❖ Concurrently, the review leader also reviews the product and establishes an agenda for the review meeting, which is typically scheduled for the next day.
- ❖ The review meeting is attended by the review leader, all reviewers, and the producer.
- ❖ One of the reviewers takes on the role of the recorder; that is, the individual who records (in writing) all important issues raised during the review.
- ❖ The FTR begins with an introduction of the agenda and a brief introduction by the producer.
- ❖ The producer then proceeds to "walk through" the work product, explaining the material, while reviewers raise issues based on their advance preparation.
- ❖ When valid problems or errors are discovered, the recorder notes each.
- ❖ At the end of the review, all attendees of the FTR must decide whether to

(1) Accept the product without further modification

(2) Reject the product due to severe errors (once corrected, another review must be performed)

(3) Accept the product provisionally (minor errors have been encountered and must be corrected, but no additional review will be required).

REVIEWING REPORTING AND RECORD KEEPING:-

- ✓ During the FTR, a reviewer (the recorder) actively records all issues that have been raised.
- ✓ These are summarized at the end of the review meeting and a review issues list is produced. In addition, a formal technical review summary report is completed.
- ✓ A review summary report answers three questions:-
 - a) What was reviewed?
 - b) Who reviewed it?
 - c) What were the findings and conclusions?

- ✓ The review summary report is a single page form (with possible attachments).

- ✓ It becomes part of the project historical record and may be distributed to the project leader and other interested parties.

- ✓ The review issues list serves two purposes:-

(1) To identify problem areas within the product

(2) To serve as an action item checklist that guides the producer as corrections are made. An issues list is normally attached to the summary report.

REVIEW GUIDELINES:-

1. Review the product, not the producer.
2. Set an agenda and maintain it.
3. Limit debate and rebuttal.
4. Enunciate problem areas, but don't attempt to solve every problem noted.
5. Take written notes.
6. Limit the number of participants and insist upon advance preparation.
7. Develop a checklist for each product that is likely to be reviewed.
8. Allocate resources and schedule time for FTRs.
9. Conduct meaningful training for all reviewers.
10. Review your early reviews.

TOPIC-5 SOFTWARE MEASUREMENT

- ✓ Measurements in the physical world can be categorized in two ways: direct measures (e.g., the length of a bolt) and indirect measures (e.g., the "quality" of bolts produced, measured by counting rejects).
- ✓ Software metrics can be categorized similarly.
- ✓ Direct measures of the software engineering process include cost and effort applied.
- ✓ Direct measures of the product include lines of code (LOC) produced, execution speed, memory size, and defects reported over some set period of time.
- ✓ Indirect measures of the product include functionality, quality, complexity, efficiency, reliability, maintainability.

The software can be measured in terms of the following three metrics:-

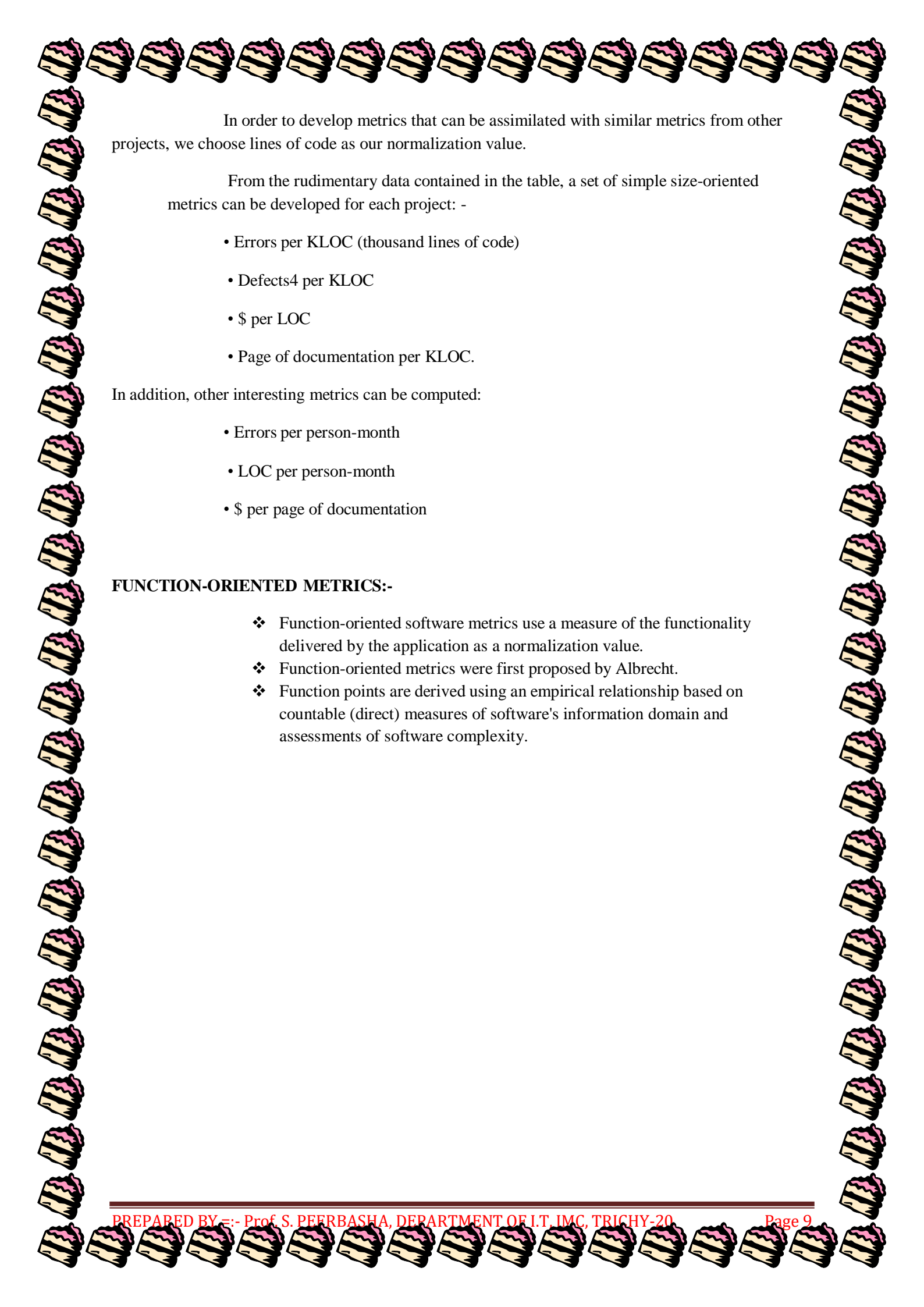
1. **SIZE ORIENTED METRICS**
2. **FUNCTION ORIENTED METRICS**
3. **EXTENDED FUNCTION POINT METRICS**

SIZE ORIENTED METRICS:-

- ❖ Size-oriented software metrics are derived by normalizing quality and/or productivity measures by considering the size of the software that has been produced.

Project	LOC	Effort	\$(000)	Pp doc	Errors	Defects	People
alpha	12,100	24	168	365	134	29	3
beta	27,200	62	440	1224	321	86	5
gamma	20,200	43	314	1050	256	64	6

Fig. Size-Oriented Metrics



In order to develop metrics that can be assimilated with similar metrics from other projects, we choose lines of code as our normalization value.

From the rudimentary data contained in the table, a set of simple size-oriented metrics can be developed for each project: -

- Errors per KLOC (thousand lines of code)
- Defects4 per KLOC
- \$ per LOC
- Page of documentation per KLOC.

In addition, other interesting metrics can be computed:

- Errors per person-month
- LOC per person-month
- \$ per page of documentation

FUNCTION-ORIENTED METRICS:-

- ❖ Function-oriented software metrics use a measure of the functionality delivered by the application as a normalization value.
- ❖ Function-oriented metrics were first proposed by Albrecht.
- ❖ Function points are derived using an empirical relationship based on countable (direct) measures of software's information domain and assessments of software complexity.

Measurement parameter	Count	Weighting factor				
		Simple	Average	Complex		
Number of user inputs	<input type="text"/>	x 3	4	6	=	<input type="text"/>
Number of user outputs	<input type="text"/>	x 4	5	7	=	<input type="text"/>
Number of user inquiries	<input type="text"/>	x 3	4	6	=	<input type="text"/>
Number of files	<input type="text"/>	x 7	10	15	=	<input type="text"/>
Number of external interfaces	<input type="text"/>	x 5	7	10	=	<input type="text"/>
Count-total	→					<input type="text"/>

Number of user inputs:-

→ Each user input that provides distinct application oriented data to the software is counted. Inputs should be distinguished from inquiries, which are counted separately.

Number of user outputs:-

→ Each user output that provides application oriented information to the user is counted. In this context output refers to reports, screens, error messages, etc. Individual data items within a report are not counted separately.

Number of user inquiries:-

→ An inquiry is defined as an on-line input that results in the generation of some immediate software response in the form of an on-line output. Each distinct inquiry is counted.

Number of files:-

→ Each logical master file (i.e., a logical grouping of data that may be one part of a large database or a separate file) is counted.


Number of external interfaces:-

→ All machine readable interfaces (e.g., data files on storage media) that are used to transmit information to another system are counted.

To compute function points (FP), the following relationship is used:-

$$FP = \text{count total} [0.65 + 0.01 \sum(F_i)]$$

Where, count total is the sum of all FP entries obtained.



The F_i ($i = 1$ to 14) are "complexity adjustment values" based on responses to the following questions.

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. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require on-line data entry?
7. Does the on-line data entry require the input transaction to be built 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 installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?

Once function points have been calculated, they are used in a manner analogous to LOC as a way to normalize measures for software productivity, quality, and other attributes: -

- Errors per FP
- Defects per FP.
- \$ per FP
- Pages of documentation per FP
- FP per person-month.

EXTENDED FUNCTION POINT METRICS:-

A function point extension called feature points is a superset of the function point measure that can be applied to systems and engineering software applications.

To compute the feature point, information domain values are again counted and weighted.

In addition, the feature point metric counts new software characteristic— algorithms.

An algorithm is defined as "a bounded computational problem that is included within a specific computer program.

Another function point extension for real-time systems and engineered products has been developed by Boeing.

The Boeing approach integrates the data dimension of software with the functional and control dimensions to provide a function-oriented measure amenable to applications that emphasize function and control capabilities called the 3D function point.

The data dimension will counts the retained data (the internal program data structure; e.g., files) and external data (inputs, outputs, inquiries, and external references) are used along with measures of complexity to derive a data dimension count.

The functional dimension is measured by considering “the number of internal operations required to transform input to output data” .

The control dimension is measured by counting the number of transitions between states.

Note:-

A state represents some externally observable mode of behaviour, and a transition occurs as a result of some event that causes the software or system to change its mode of behaviour (i.e., to change state).

For example, a wireless phone contains software that supports auto dial functions. To enter the auto-dial state from a resting state, the user presses an Auto key on the keypad.

To compute 3D function points, the following relationship is used:-

$$\text{index} = I + O + Q + F + E + T + R$$

Where I, O, Q, F, E, T, and R represent complexity weighted values for the elements discussed already: inputs, outputs, inquiries, internal data structures, external files, transformation, and transitions, respectively.

Semantic statements			
	1-5	6-10	11+
Processing steps			
1-10	low	low	Average
11-20	low	Average	High
21+	Average	High	High


Each complexity weighted value is computed using the following relationship:-

$$\text{Complexity weighted value} = N_{il}W_{il} + N_{ia}W_{ia} + N_{ih}W_{ih}$$

N represent the number of occurrences of element i (e.g., outputs) for each level of complexity (Low, Average, High)

W represents the corresponding weights.

TOPIC-6 METRICS FOR SOFTWARE QUALITY

- 
- ❖ The overriding goal of software engineering is to produce a high-quality system, application, or product.
 - ❖ To achieve this goal, software engineers must apply effective methods coupled with modern tools within the context of a mature software process.
 - ❖ Metrics provide an indication of the effectiveness of individual and group software quality assurance and control activities
 - ❖ Metrics such as work product (e.g., requirements or design) errors per function point, errors uncovered per review hour, and errors uncovered per testing hour provide insight into the efficacy of each of the activities implied by the metric.
 - ❖ Error data can also be used to compute the defect removal efficiency (DRE) for each process framework activity.

The following concepts will be discussed here:-

1. **An overview of factors that affect quality.**
2. **Measuring quality.**
3. **Defect removal efficiency.**

An overview of factors that affect quality:-

The following are the factors assess software that affects quality. They are:-

- (1) Product operation (using it)
- (2) Product revision (changing it)
- (3) Product transition (modifying it to work in a different environment; i.e., "porting" it).

The relationship between these quality factors (what they call a framework) and other aspects of the software engineering process:-

First, the framework provides a mechanism for the project manager to identify what qualities are important.

Secondly, the framework provides a means for quantitatively assessing how well the development is progressing relative to the quality goals established.

Thirdly, the framework provides for more interaction of QA personnel throughout the development effort.

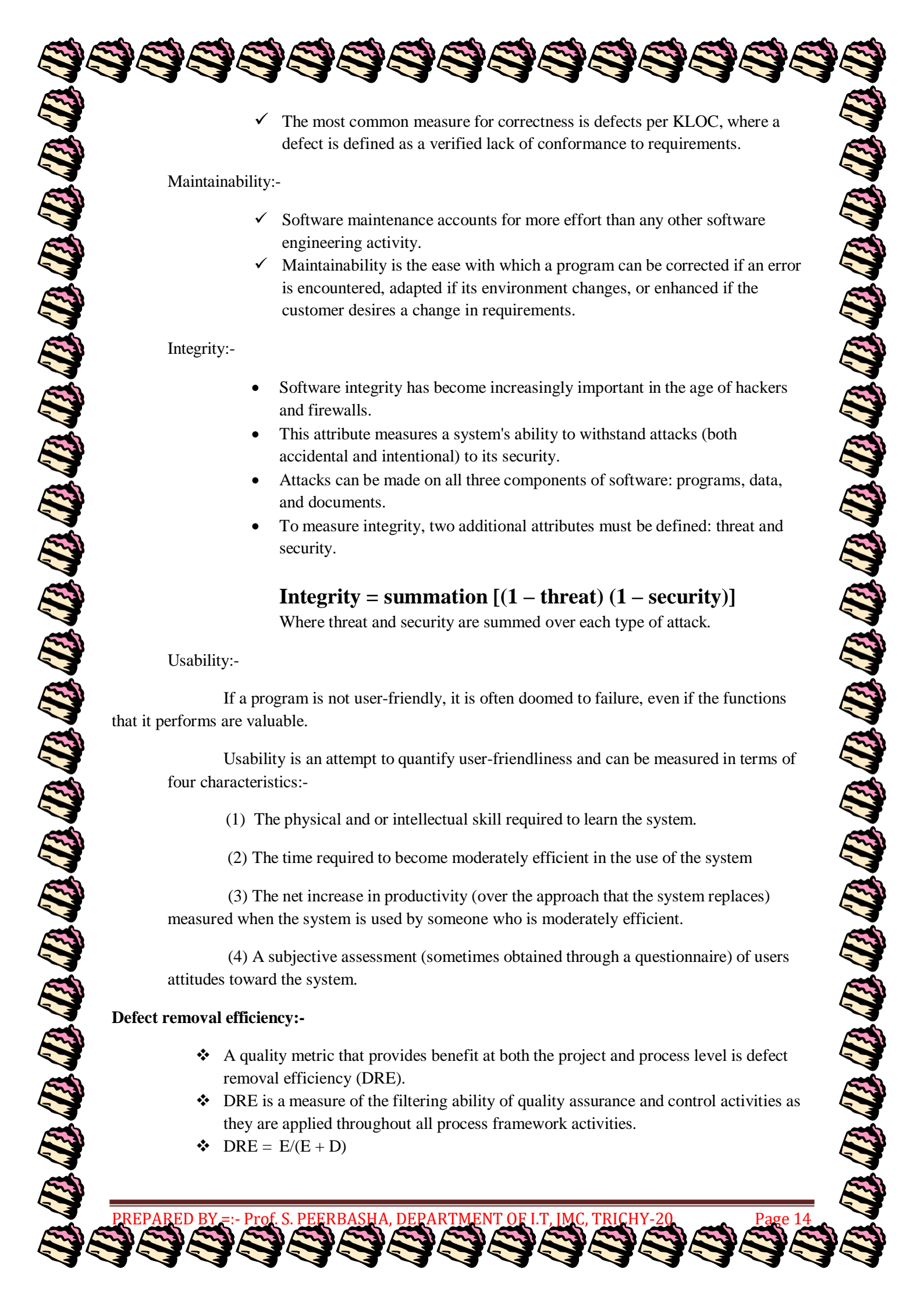
Lastly, quality assurance personal can use indications of poor quality to help identify [better] standards to be enforced in the future.

Measuring quality:-

There are many measures of software quality, correctness, maintainability, integrity, and usability provide useful indicators for the project team.

Correctness:-

- ✓ A program must operate correctly or it provides little value to its users. Correctness is the degree to which the software performs its required function.

- 
- ✓ The most common measure for correctness is defects per KLOC, where a defect is defined as a verified lack of conformance to requirements.

Maintainability:-

- ✓ Software maintenance accounts for more effort than any other software engineering activity.
- ✓ Maintainability is the ease with which a program can be corrected if an error is encountered, adapted if its environment changes, or enhanced if the customer desires a change in requirements.

Integrity:-

- Software integrity has become increasingly important in the age of hackers and firewalls.
- This attribute measures a system's ability to withstand attacks (both accidental and intentional) to its security.
- Attacks can be made on all three components of software: programs, data, and documents.
- To measure integrity, two additional attributes must be defined: threat and security.

Integrity = summation [(1 – threat) (1 – security)]

Where threat and security are summed over each type of attack.

Usability:-

If a program is not user-friendly, it is often doomed to failure, even if the functions that it performs are valuable.

Usability is an attempt to quantify user-friendliness and can be measured in terms of four characteristics:-

- (1) The physical and or intellectual skill required to learn the system.
- (2) The time required to become moderately efficient in the use of the system
- (3) The net increase in productivity (over the approach that the system replaces) measured when the system is used by someone who is moderately efficient.
- (4) A subjective assessment (sometimes obtained through a questionnaire) of users attitudes toward the system.

Defect removal efficiency:-

- ❖ A quality metric that provides benefit at both the project and process level is defect removal efficiency (DRE).
- ❖ DRE is a measure of the filtering ability of quality assurance and control activities as they are applied throughout all process framework activities.
- ❖ $DRE = E/(E + D)$

- ❖ Where E is the number of errors found before delivery of the software to the end-user and D is the number of defects found after delivery.
- ❖ DRE can also be used within the project to assess a team's ability to find errors before they are passed to the next framework activity or software engineering task.
- ❖ $DRE_i = E_i / (E_i + E_{i+1})$
- ❖ Where E_i is the number of errors found during software engineering activity i and E_{i+1} is the number of errors found during software engineering activity i+1 that are traceable to errors that were not discovered in software engineering activity i.

VERY IMPORTANT QUESTIONS FROM THIS UNIT-5

PART-B

1. Write short notes on software quality.
2. Explain the various SQA activities.
3. Write a note on Defect Amplification Model.
4. Explain the various metrics of software quality.

PART-B

1. Discuss the formal technical reviews in detail.
2. Discuss the software measurement in detail.

ALL THE BEST
