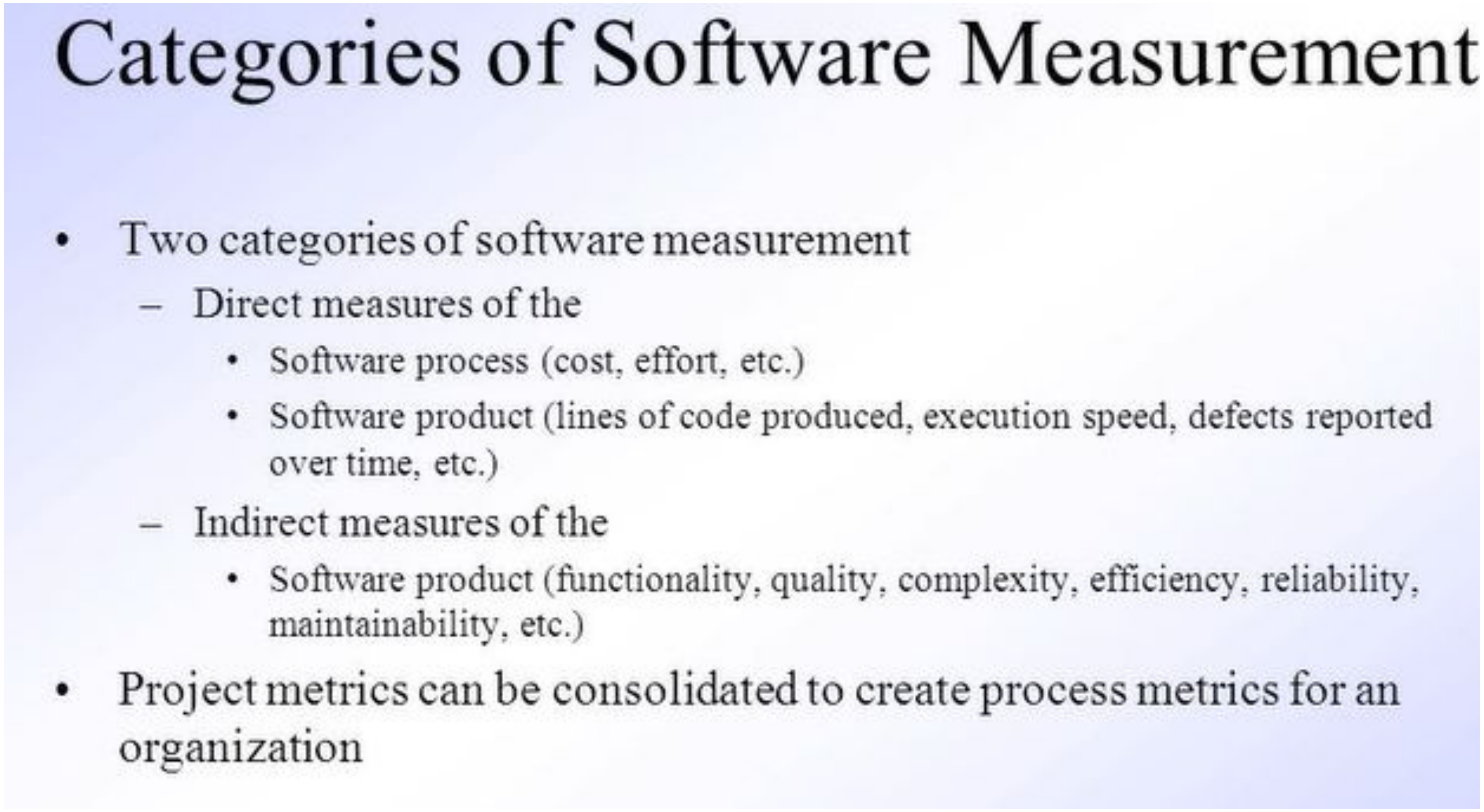
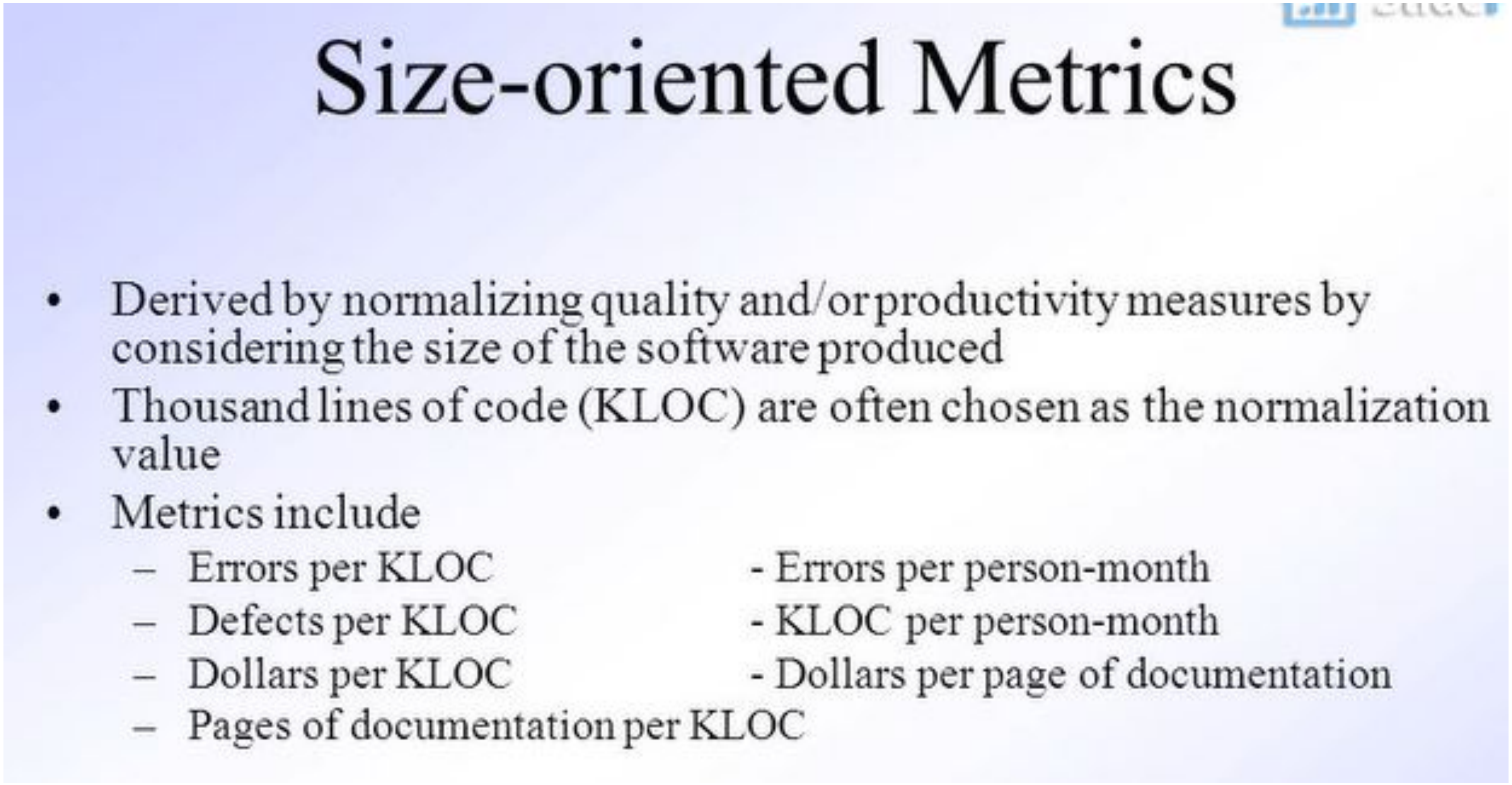


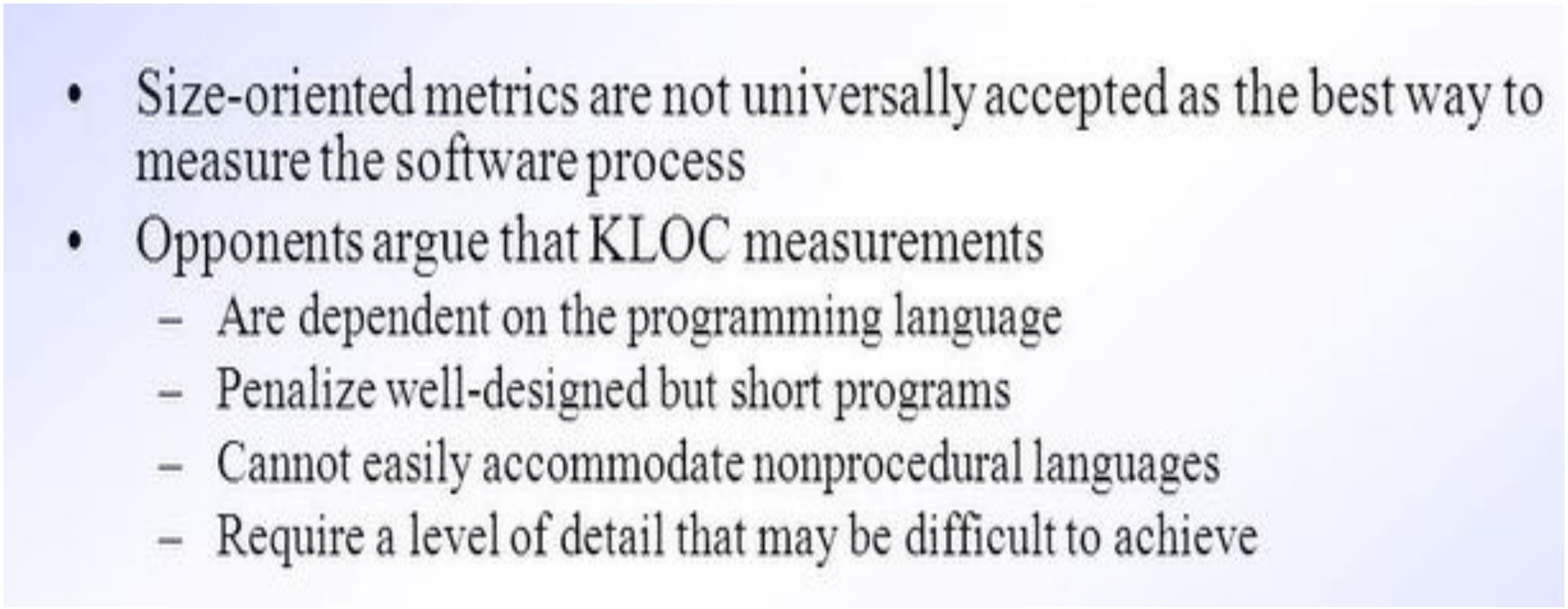
Categories of Software Measurement

- Two categories of software measurement
  - Direct measures of the
    - Software process (cost, effort, etc.)
    - Software product (lines of code produced, execution speed, defects reported over time, etc.)
  - Indirect measures of the
    - Software product (functionality, quality, complexity, efficiency, reliability, maintainability, etc.)
- Project metrics can be consolidated to create process metrics for an organization



Size-oriented Metrics

- Derived by normalizing quality and/or productivity measures by considering the size of the software produced
- Thousand lines of code (KLOC) are often chosen as the normalization value
- Metrics include
  - Errors per KLOC
  - Defects per KLOC
  - Dollars per KLOC
  - Pages of documentation per KLOC
  - Errors per person-month
  - KLOC per person-month
  - Dollars per page of documentation



- Size-oriented metrics are not universally accepted as the best way to measure the software process
- Opponents argue that KLOC measurements
  - Are dependent on the programming language
  - Penalize well-designed but short programs
  - Cannot easily accommodate nonprocedural languages
  - Require a level of detail that may be difficult to achieve

Lines of code (LOC):

A line of code (LOC) metric is based on the measurement of the source lines of the code in a program.
It is simply measured by counting the **program header, declarations, executable, and non-executable lines in the source program.**
Comments lines, blank lines are usually not considered during the LOC measurement.
LOC is generally counted in kilo (thousand) line of codes (KLOC) per person-month (PM).
Size estimation is performed by decomposing a problem into modules and each module into sub modules.
The overall project size is the sum of the sizes of all modules in the project.

Example of LOC

```
/* This program finds a substring in a given
string */

#include<stdio.h>
#include<conio.h>
#include <string.h>

void main(int argc,char * argv[])
{       char  *temp;
  int start, end ;
  clrscr();

if(argc==4)
{
start - atoi(argv[2]);
end = atoi(argv[3]);
strncpy(temp,argv[1]+start,end);
temp[end]='\0';
     printf("entered string  : %s\n ",argv[1]);
     printf("\nsubstring : %s ",temp);
      }
     else
     printf("incorrect arrgument ");
     getch();
     }
```

LOC = LOC counted as 17.

Figure 4.1: A program to find a substring in a string

Function Point Analysis: Methods

FP-based estimations are based on the following five information domain values and their complexities in a particular project.

Number of inputs; each user input
Number of outputs; reports, screens, error messages.
Number of inquiries;
Number of internal logical files; logically related data
Number of external interfaces; transmission of data from other interfaces

The value of each of these five information domains is collected and a subjective evaluation is performed to categorize them as simple, average, and complex.
There are certain weights assigned at each complexity level to the information domains.

The values of each of these are rated on a five point scale.
0-not significant
1-incidential
2-moderate
3-average
4-significant
5-highly essential
Example:
Compute the FP value for the grade calculation of students. Assume that it is an average complexity size project. The information domain values are as follows: number of inputs = 13, number of outputs = 4, number of inquiries = 2, number of external files = 5, number of interfaces = 2. The total value of complexity adjustment attributes is 13.

Information domain	Weights		
	Simple	Average	Complex
Number of inputs	3	4	6
Number of outputs	4	5	7
Number of inquiries	3	4	6
Number of internal logical files	7	10	15
Number of external interfaces	5	7	10

Table 4.1: Information domains and their weights

Function Point Analysis: Methods

- *1. Calculate the unadjusted function point (UFP):* It is calculated by simply counting the value of each information domain and multiplying it by an appropriate weight at its complexity level.

- *2. Compute the complexity adjustment attributes (CAA):* The CAAs are complexity attributes that can vary from project to project. They are computed using the following relationship:

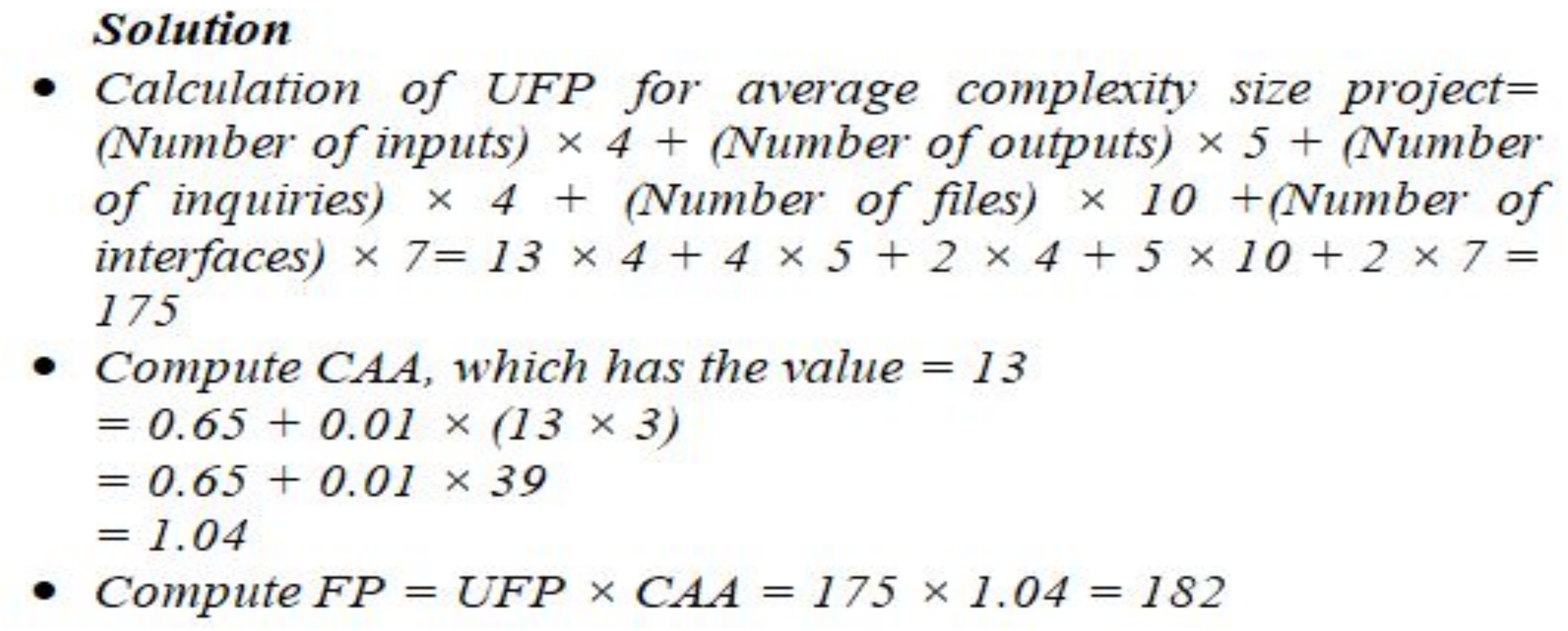
CAA = [0.65 + 0.01 × Σ CAAi], where CAA is the complexity adjustment attributes.

- *3. Then, compute FP = UFP × CAA.*

- *The CAA value is based on 14 general system characteristics that rate the general functionality of the application being counted.*

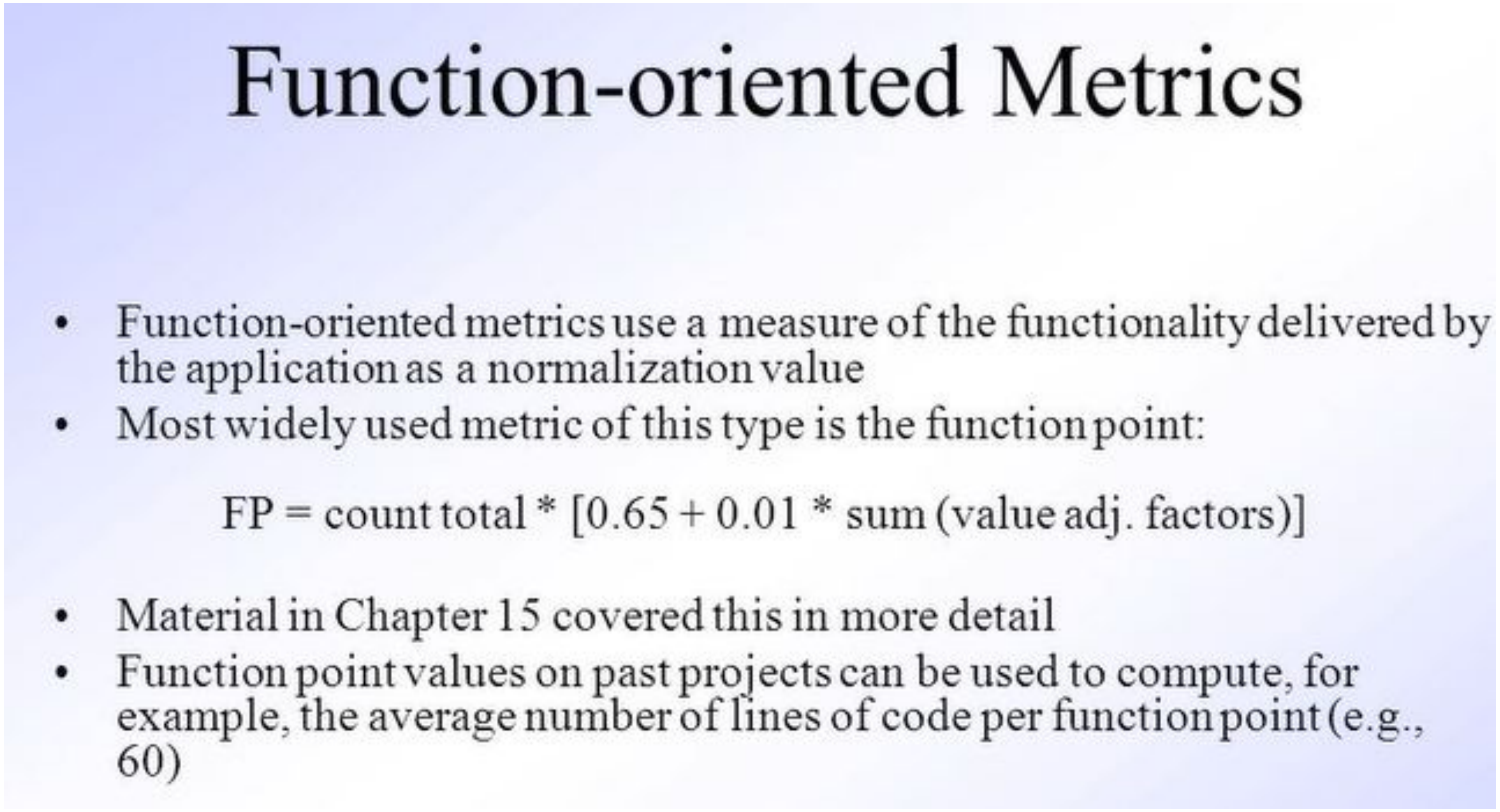
Examples:

*Compute the FP value for the **grade calculation** of students. Assume that it is an average complexity size project. The information domain values are as follows: number of inputs = 13, number of outputs = 4, number of inquiries = 2, number of external files = 5, number of interfaces = 2. The total value of complexity adjustment attributes is 13.*



Solution

- *Calculation of UFP for average complexity size project= (Number of inputs) × 4 + (Number of outputs) × 5 + (Number of inquiries) × 4 + (Number of files) × 10 +(Number of interfaces) × 7= 13 × 4 + 4 × 5 + 2 × 4 + 5 × 10 + 2 × 7 = 175*
- *Compute CAA, which has the value = 13*
  = 0.65 + 0.01 × (13 × 3)
  = 0.65 + 0.01 × 39
  = 1.04
- *Compute FP = UFP × CAA = 175 × 1.04 = 182*

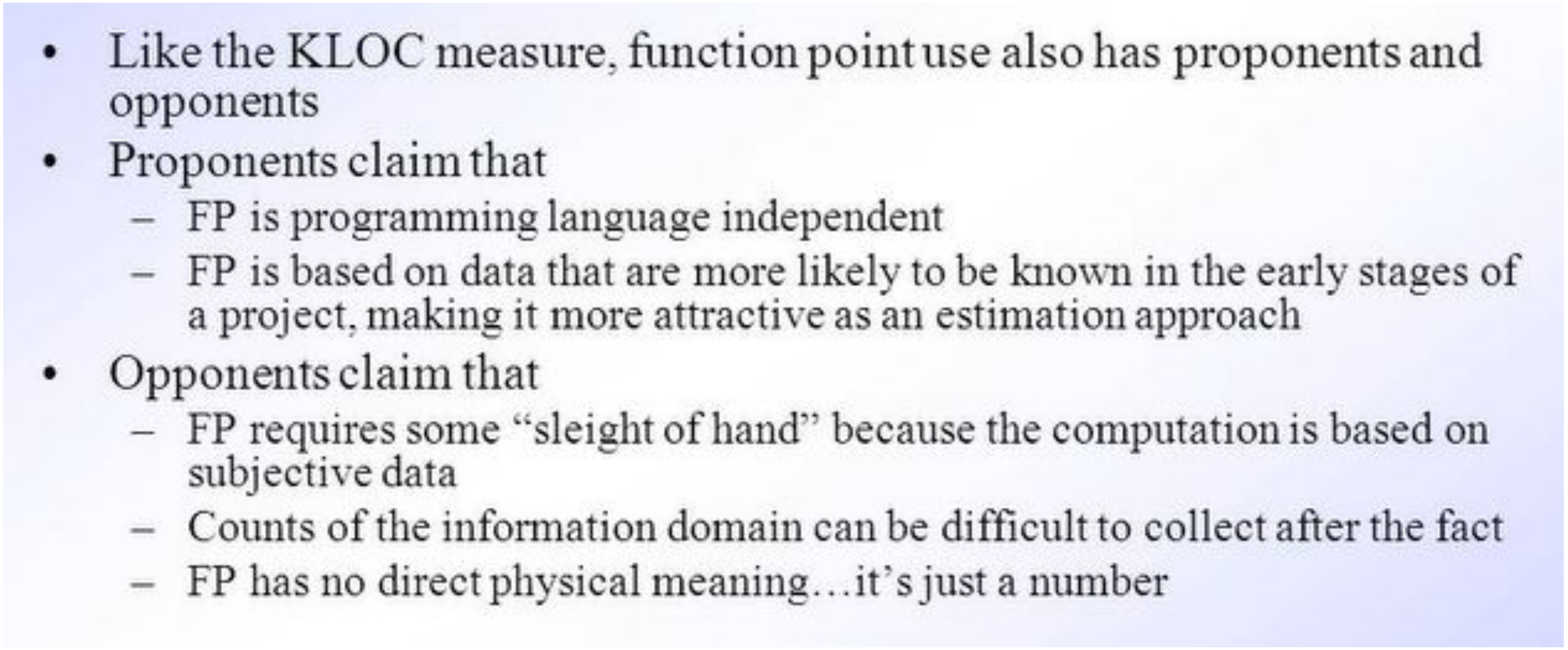


Function-oriented Metrics

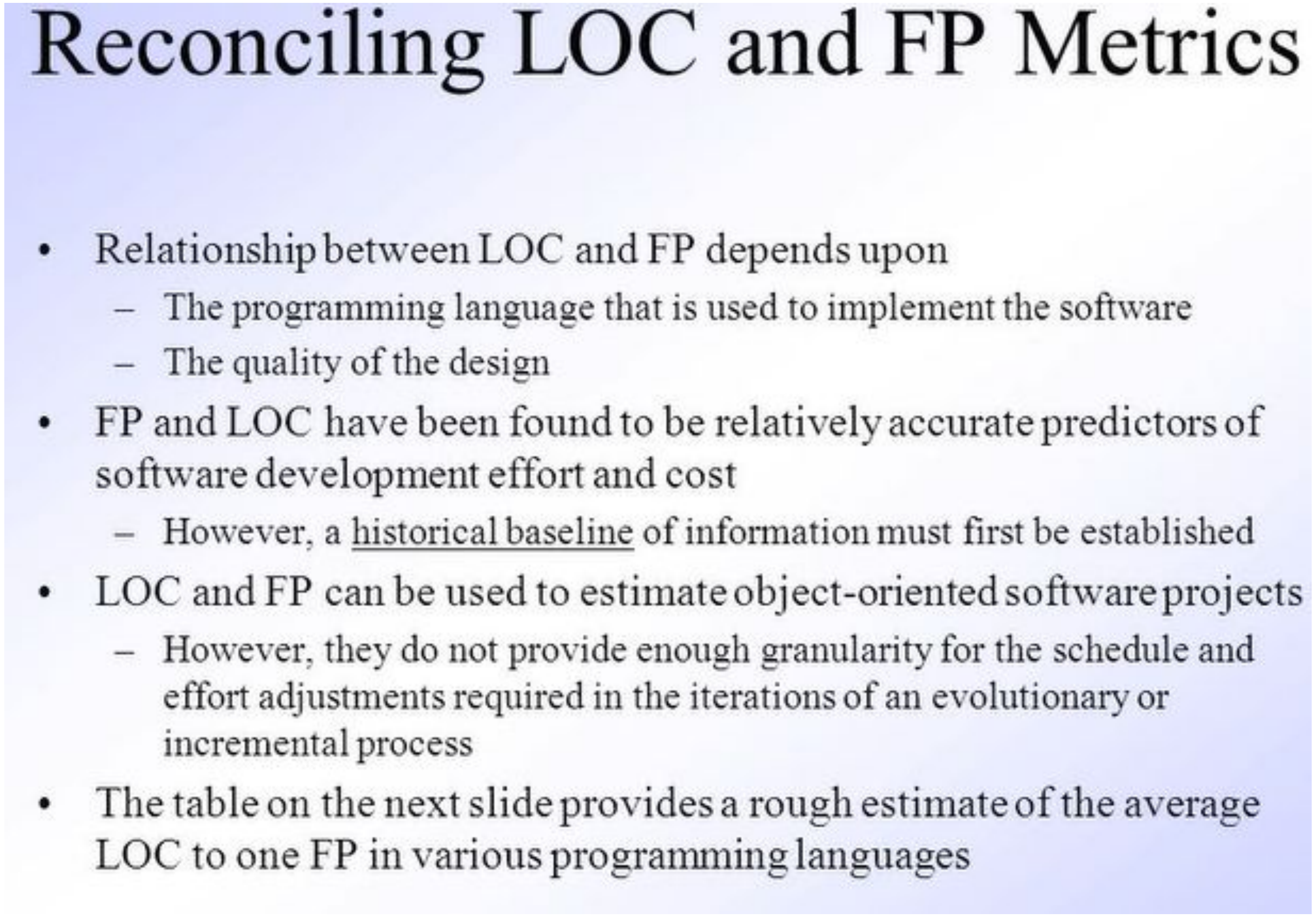
- Function-oriented metrics use a measure of the functionality delivered by the application as a normalization value
- Most widely used metric of this type is the function point:

$$FP = \text{count total} * [0.65 + 0.01 * \text{sum (value adj. factors)}]$$

- Material in Chapter 15 covered this in more detail
- Function point values on past projects can be used to compute, for example, the average number of lines of code per function point (e.g., 60)

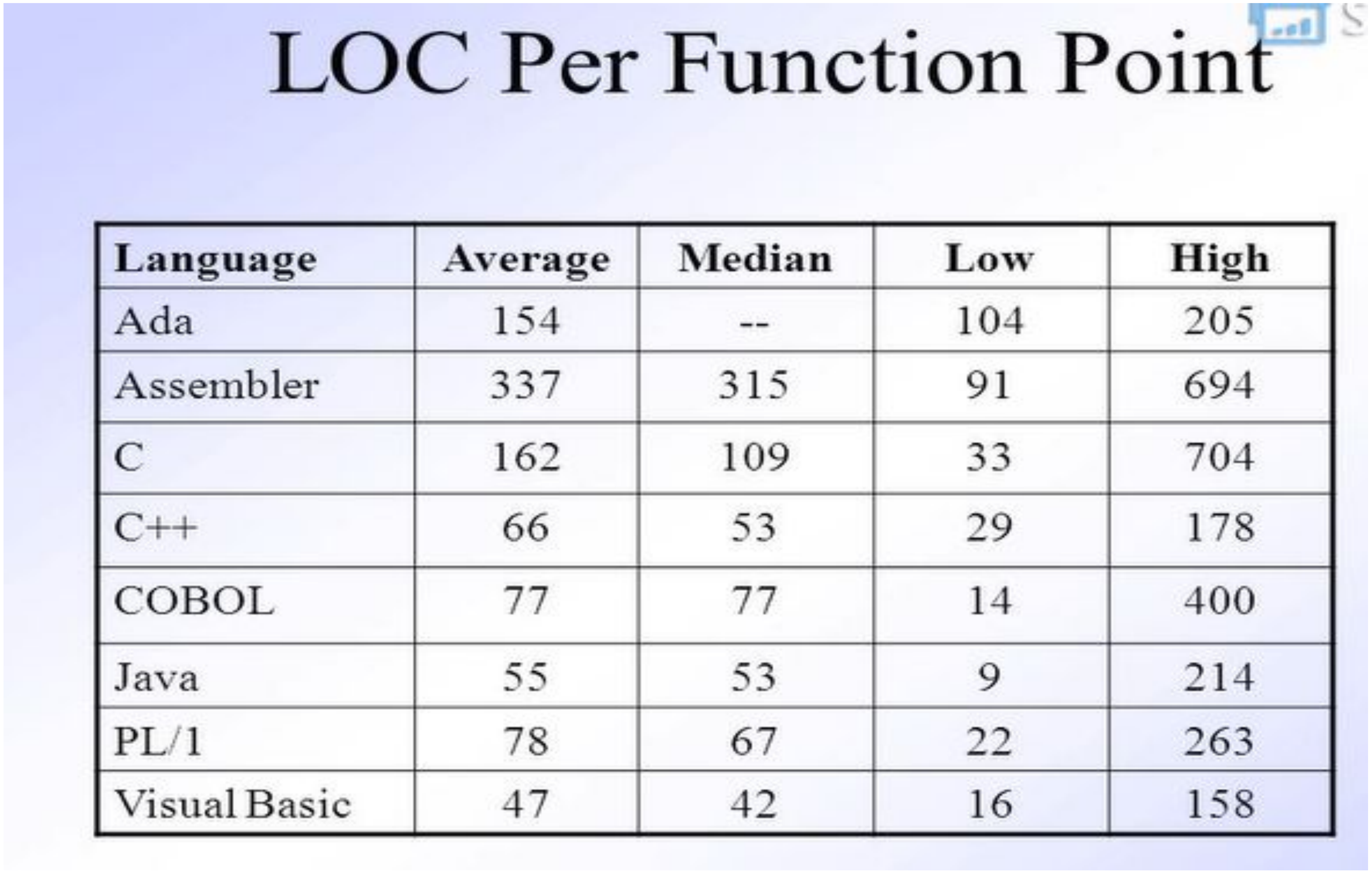


- Like the KLOC measure, function point use also has proponents and opponents
- Proponents claim that
  - FP is programming language independent
  - FP is based on data that are more likely to be known in the early stages of a project, making it more attractive as an estimation approach
- Opponents claim that
  - FP requires some "sleight of hand" because the computation is based on subjective data
  - Counts of the information domain can be difficult to collect after the fact
  - FP has no direct physical meaning…it's just a number



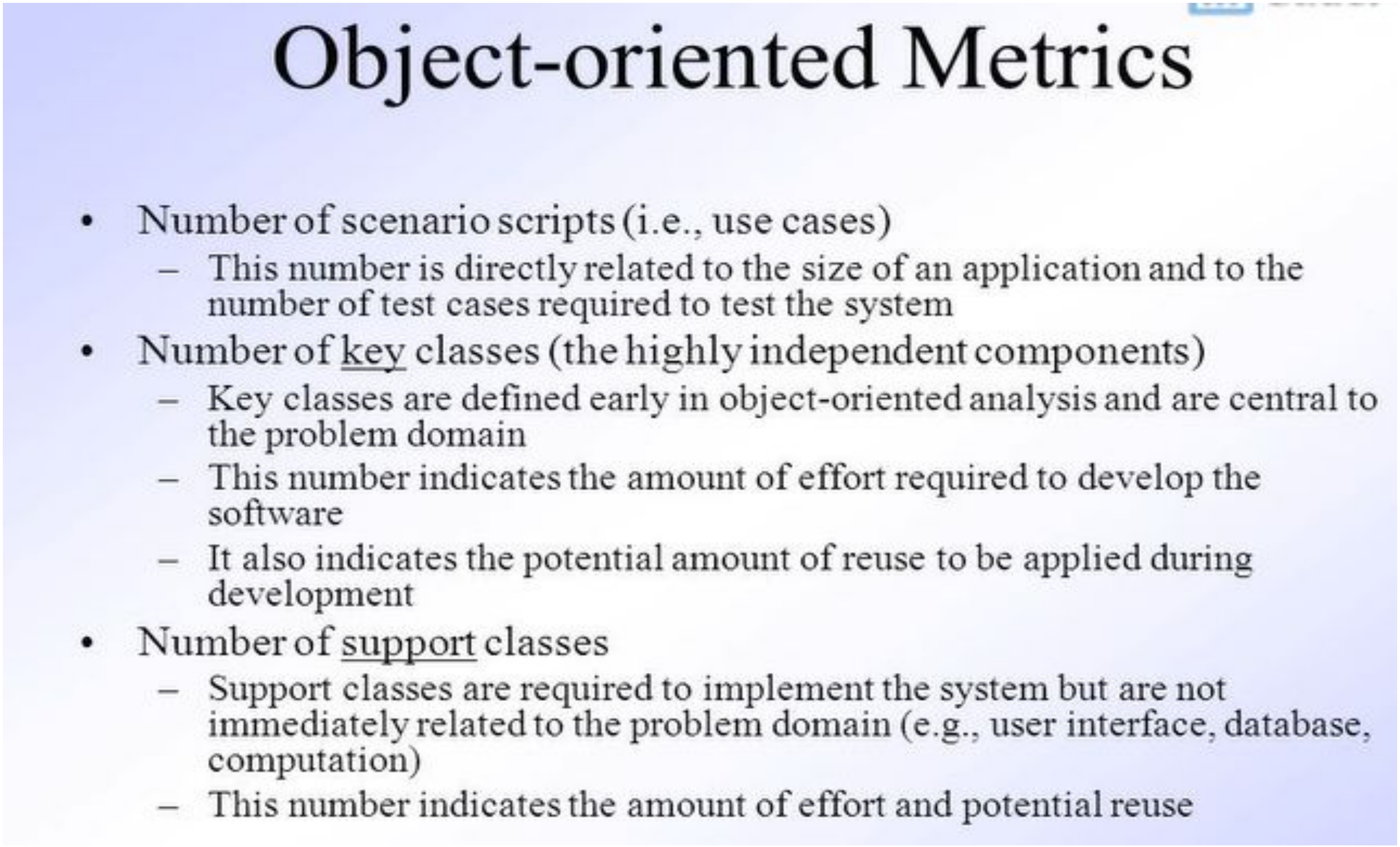
Reconciling LOC and FP Metrics

- Relationship between LOC and FP depends upon
  - The programming language that is used to implement the software
  - The quality of the design
- FP and LOC have been found to be relatively accurate predictors of software development effort and cost
  - However, a historical baseline of information must first be established
- LOC and FP can be used to estimate object-oriented software projects
  - However, they do not provide enough granularity for the schedule and effort adjustments required in the iterations of an evolutionary or incremental process
- The table on the next slide provides a rough estimate of the average LOC to one FP in various programming languages



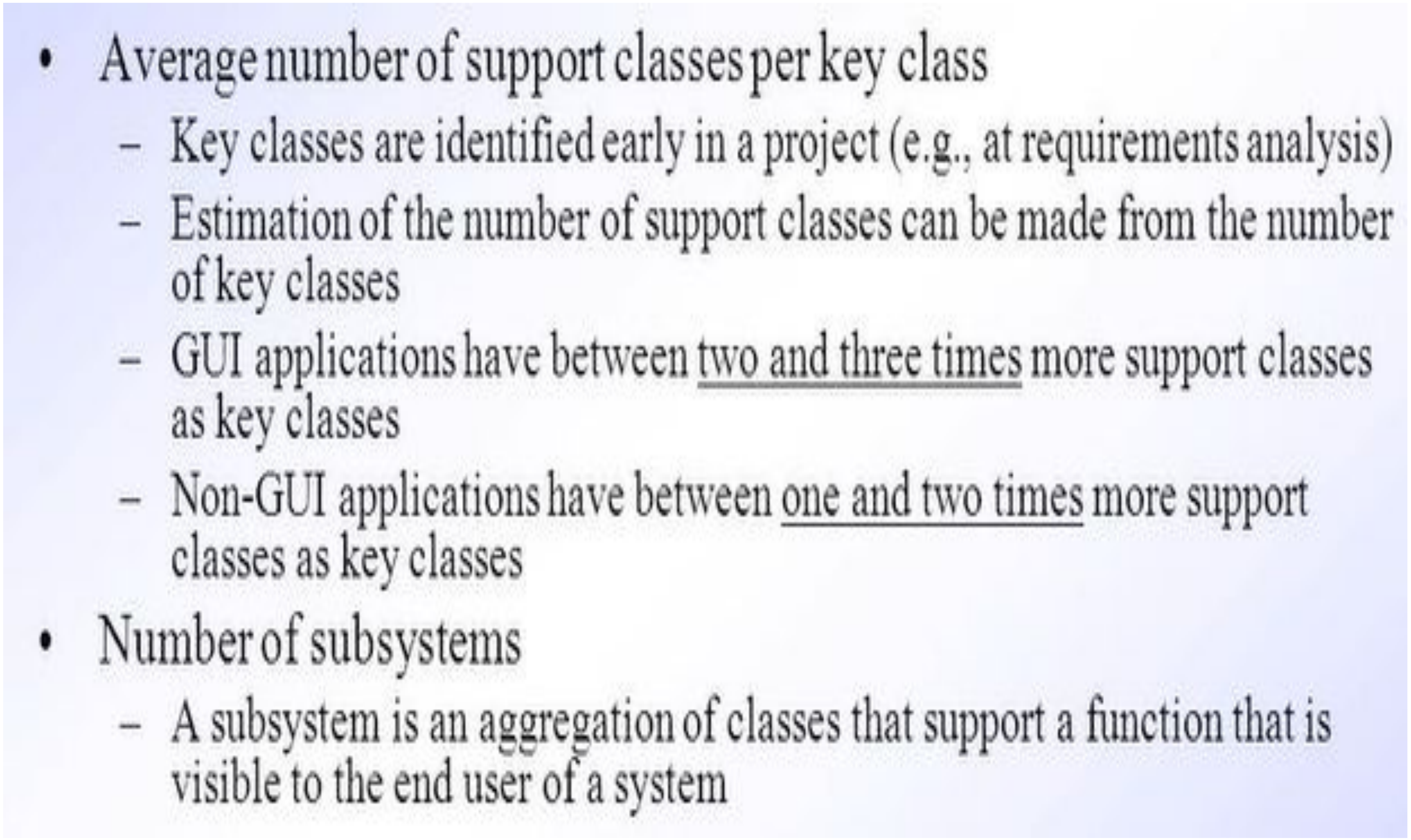
LOC Per Function Point

Language	Average	Median	Low	High
Ada	154	--	104	205
Assembler	337	315	91	694
C	162	109	33	704
C++	66	53	29	178
COBOL	77	77	14	400
Java	55	53	9	214
PL/1	78	67	22	263
Visual Basic	47	42	16	158

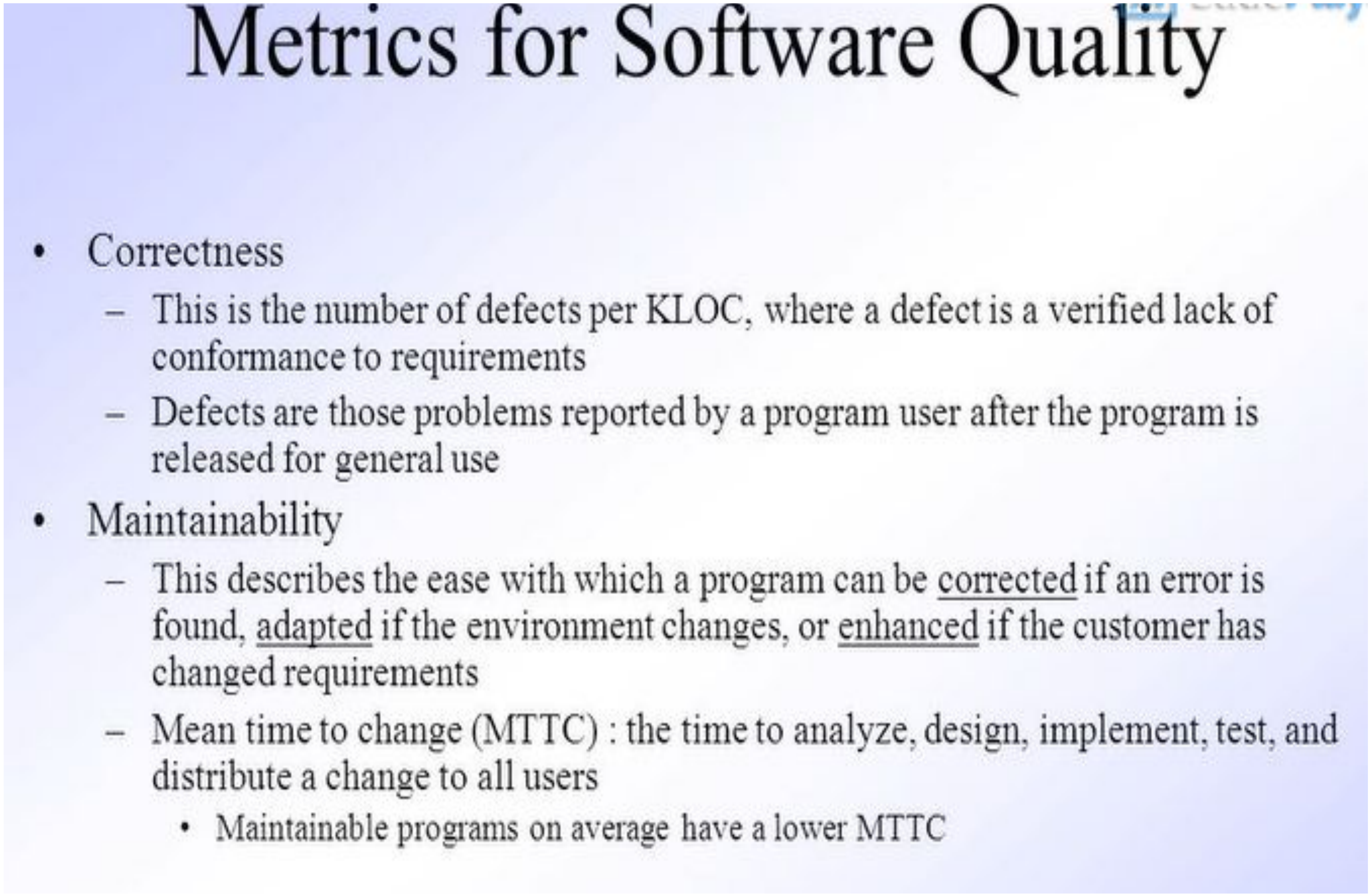


Object-oriented Metrics

- Number of scenario scripts (i.e., use cases)
  - This number is directly related to the size of an application and to the number of test cases required to test the system
- Number of key classes (the highly independent components)
  - Key classes are defined early in object-oriented analysis and are central to the problem domain
  - This number indicates the amount of effort required to develop the software
  - It also indicates the potential amount of reuse to be applied during development
- Number of support classes
  - Support classes are required to implement the system but are not immediately related to the problem domain (e.g., user interface, database, computation)
  - This number indicates the amount of effort and potential reuse

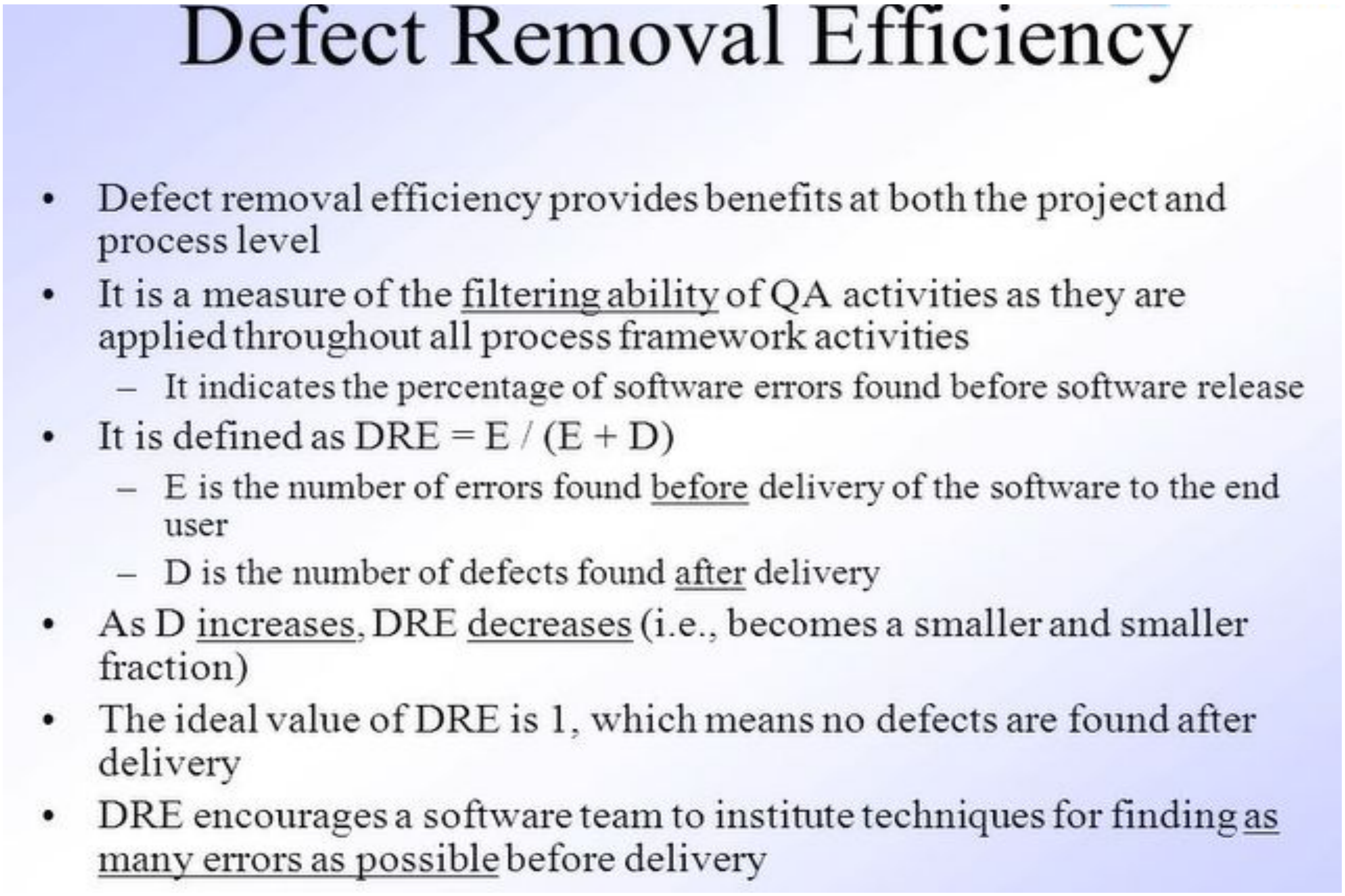


- Average number of support classes per key class
  - Key classes are identified early in a project (e.g., at requirements analysis)
  - Estimation of the number of support classes can be made from the number of key classes
  - GUI applications have between two and three times more support classes as key classes
  - Non-GUI applications have between one and two times more support classes as key classes
- Number of subsystems
  - A subsystem is an aggregation of classes that support a function that is visible to the end user of a system



Metrics for Software Quality

- Correctness
  - This is the number of defects per KLOC, where a defect is a verified lack of conformance to requirements
  - Defects are those problems reported by a program user after the program is released for general use
- Maintainability
  - This describes the ease with which a program can be corrected if an error is found, adapted if the environment changes, or enhanced if the customer has changed requirements
  - Mean time to change (MTTC) : the time to analyze, design, implement, test, and distribute a change to all users
    - Maintainable programs on average have a lower MTTC



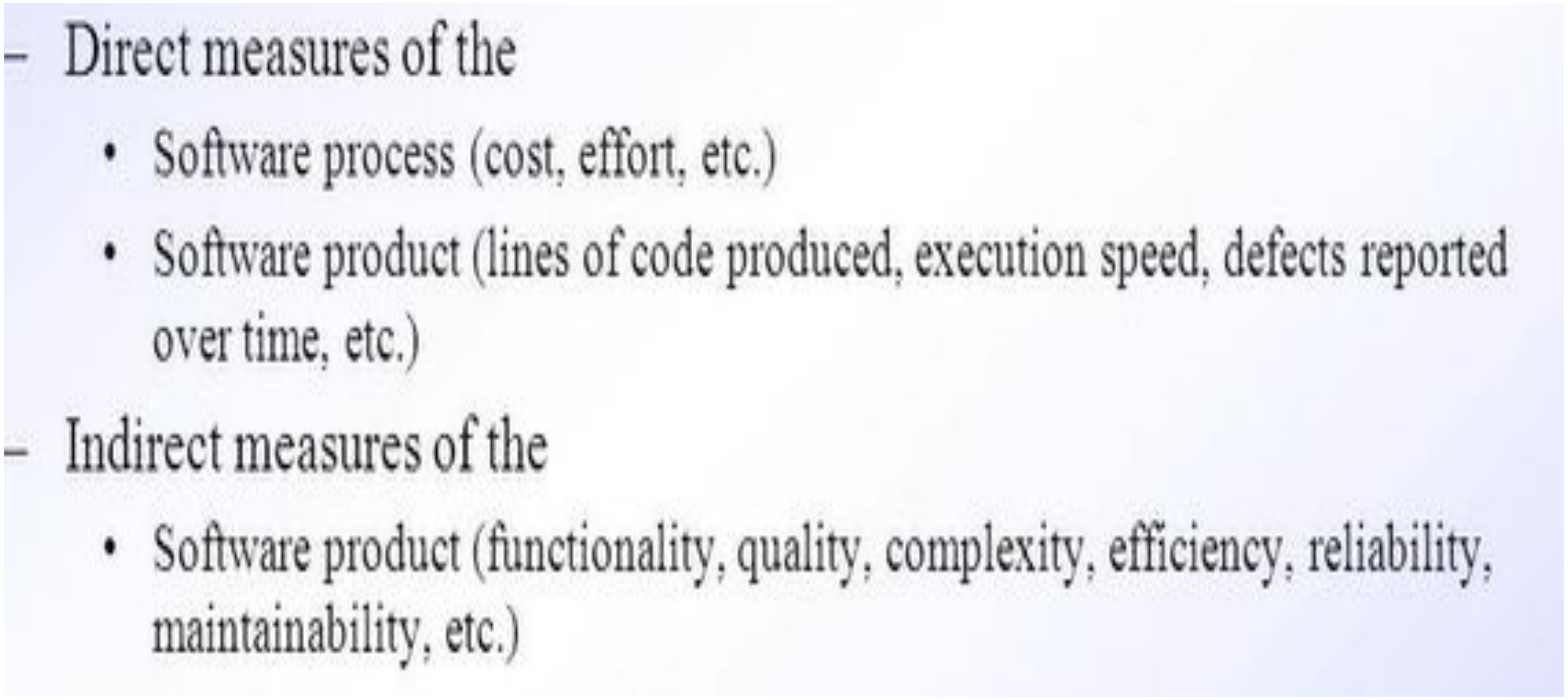
Defect Removal Efficiency

- Defect removal efficiency provides benefits at both the project and process level
- It is a measure of the filtering ability of QA activities as they are applied throughout all process framework activities
  - It indicates the percentage of software errors found before software release
- It is defined as DRE = E / (E + D)
  - E is the number of errors found before delivery of the software to the end user
  - D is the number of defects found after delivery
- As D increases, DRE decreases (i.e., becomes a smaller and smaller fraction)
- The ideal value of DRE is 1, which means no defects are found after delivery
- DRE encourages a software team to institute techniques for finding as many errors as possible before delivery

Overview of Previous Class

Software Measurement

- Direct Measurement
- Indirect Measurement



- Direct measures of the
  - Software process (cost, effort, etc.)
  - Software product (lines of code produced, execution speed, defects reported over time, etc.)
- Indirect measures of the
  - Software product (functionality, quality, complexity, efficiency, reliability, maintainability, etc.)

Risk management

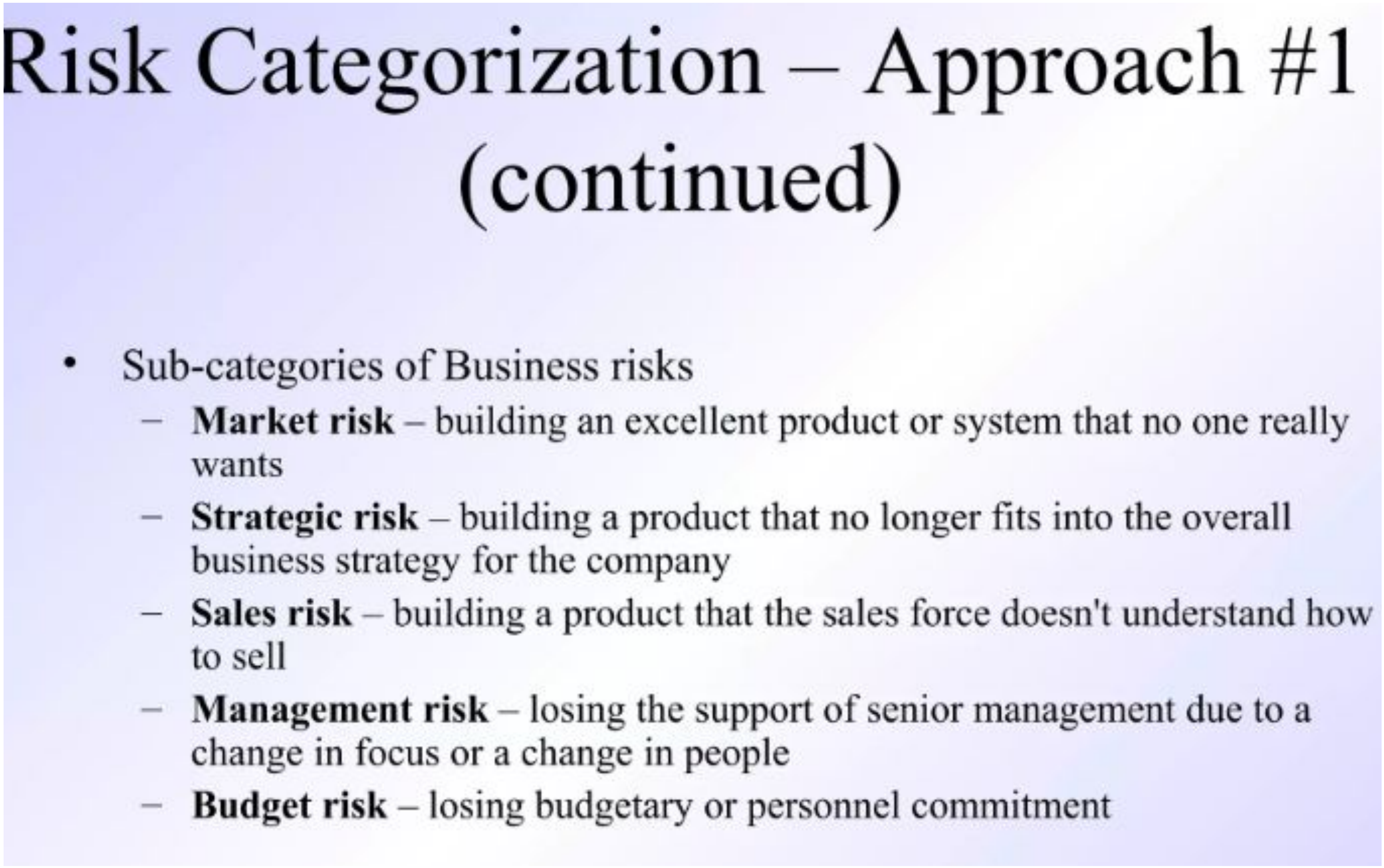
- Reactive Vs proactive risk strategies
- Software risks
- Risk Identification
- Risk Projection
- Risk Refinement
- RMMM
- RMMM plan

Definition of Risk

- A risk is a potential problem – it might happen and it might not
- Conceptual definition of risk
  - Risk concerns future happenings
  - Risk involves change in mind, opinion, actions, places, etc.
  - Risk involves choice and the uncertainty that choice entails
- Two characteristics of risk
  - Uncertainty – the risk may or may not happen, that is, there are no 100% risks (those, instead, are called constraints)
  - Loss – the risk becomes a reality and unwanted consequences or losses occur

Risk Categorization – Approach #1

- Project risks
  - They threaten the <u>project plan</u>
  - If they become real, it is likely that the <u>project schedule</u> will slip and that costs will increase
- Technical risks
  - They threaten the <u>quality</u> and <u>timeliness</u> of the software to be produced
  - If they become real, <u>implementation</u> may become difficult or impossible
- Business risks
  - They threaten the <u>viability</u> of the software to be built
  - If they become real, they <u>jeopardize</u> the project or the product



Risk Categorization – Approach #1 (continued)

- Sub-categories of Business risks
  - **Market risk** – building an excellent product or system that no one really wants
  - **Strategic risk** – building a product that no longer fits into the overall business strategy for the company
  - **Sales risk** – building a product that the sales force doesn't understand how to sell
  - **Management risk** – losing the support of senior management due to a change in focus or a change in people
  - **Budget risk** – losing budgetary or personnel commitment

Risk Categorization – Approach #2

- Known risks
  - Those risks that can be uncovered after careful evaluation of the project plan, the business and technical environment in which the project is being developed, and other reliable information sources (e.g., unrealistic delivery date)
- Predictable risks
  - Those risks that are extrapolated from past project experience (e.g., past turnover)
- Unpredictable risks
  - Those risks that can and do occur, but are extremely difficult to identify in advance

Reactive risk management tries to reduce the damage of potential threats and speed an organization’s recovery from them, but assumes that those threats will happen eventually.

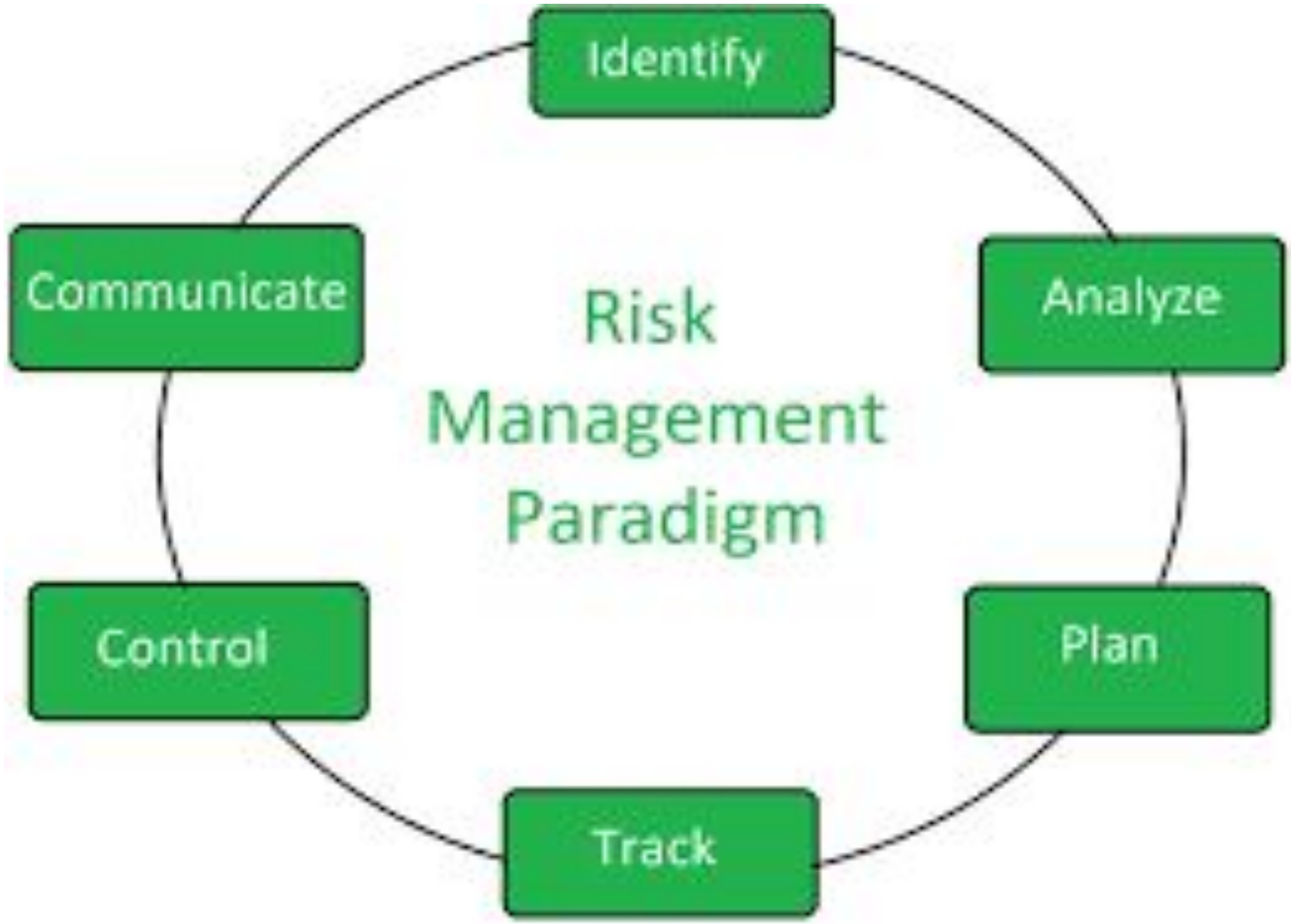
Proactive Risk Management means that you identify risks before they happen and figure out ways to avoid or alleviate the risk.

Proactive vs Reactive risks

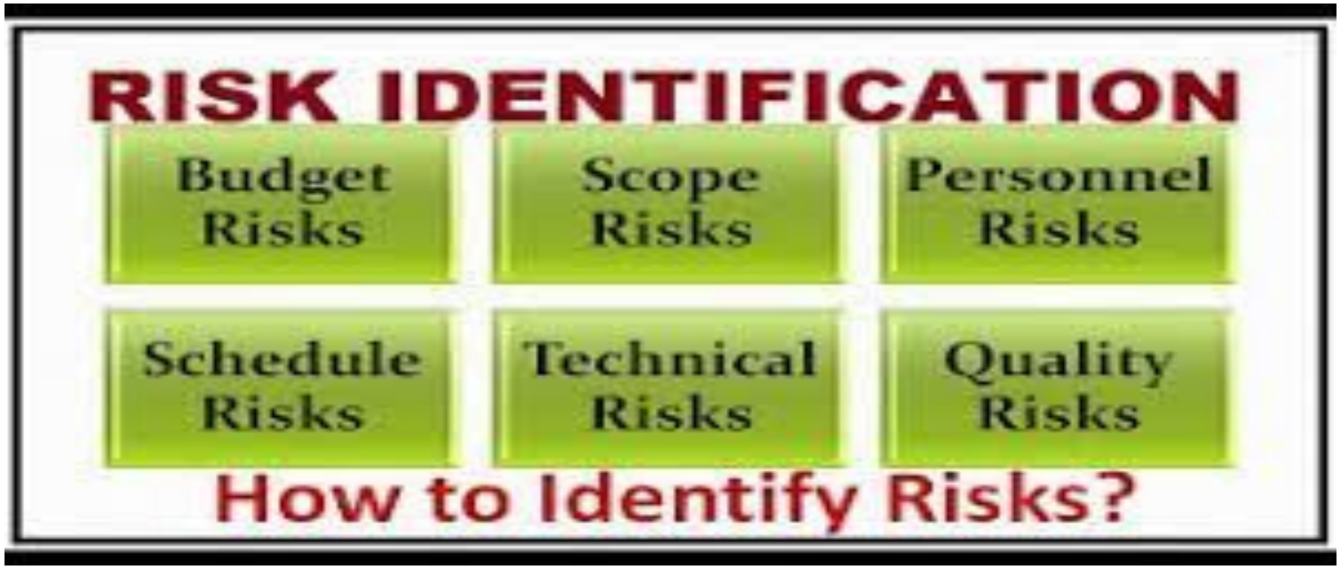
PROACTIVE RISK MANAGEMENT	REACTIVE RISK MANAGEMENT
How do we make sure nothing bad happens? Most work is done pre-project. Risks are mitigated and action plan for future risks is created.	Something bad happened. What do we do now? Following the action plan and dealing with new risks asap.

Steps for Risk Management

1) <u>Identify</u> possible risks; recognize what can go wrong
2) <u>Analyze</u> each risk to estimate the <u>probability</u> that it will occur and the <u>impact</u> (i.e., damage) that it will do if it does occur
3) <u>Rank</u> the risks by probability and impact
   - Impact may be negligible, marginal, critical, and catastrophic
4) <u>Develop</u> a contingency plan to manage those risks having <u>high probability</u> and <u>high impact</u>


Identify
Analyze
Plan
Track
Control
Communicate
Risk Management Paradigm

Risk Identification



Background

- Risk identification is a systematic attempt to specify threats to the project plan
- By identifying known and predictable risks, the project manager takes a first step toward avoiding them when possible and controlling them when necessary
- Generic risks
  - Risks that are a potential threat to every software project
- Product-specific risks
  - Risks that can be identified only by those a with a clear understanding of the technology, the people, and the environment that is specific to the software that is to be built
  - This requires examination of the project plan and the statement of scope
  - "What special characteristics of this product may threaten our project plan?"

Risk Item Checklist

- Used as one way to identify risks
- Focuses on known and predictable risks in specific subcategories (see next slide)
- Can be organized in several ways
  - A <u>list</u> of characteristics relevant to each risk subcategory
  - <u>Questionnaire</u> that leads to an estimate on the impact of each risk
  - A <u>list</u> containing a set of risk component and drivers and their probability of occurrence

Known and Predictable Risk Categories

- **Product size** – risks associated with overall size of the software to be built
- **Business impact** – risks associated with constraints imposed by management or the marketplace
- **Customer characteristics** – risks associated with sophistication of the customer and the developer's ability to communicate with the customer in a timely manner
- **Process definition** – risks associated with the degree to which the software process has been defined and is followed
- **Development environment** – risks associated with availability and quality of the tools to be used to build the project
- **Technology to be built** – risks associated with complexity of the system to be built and the "newness" of the technology in the system
- **Staff size and experience** – risks associated with overall technical and project experience of the software engineers who will do the work

Questionnaire on Project Risk

(Questions are ordered by their relative importance to project success)

1) Have top software and customer managers formally committed to support the project?
2) Are end-users enthusiastically committed to the project and the system/product to be built?
3) Are requirements fully understood by the software engineering team and its customers?
4) Have customers been involved fully in the definition of requirements?
5) Do end-users have realistic expectations?
6) Is the project scope stable?

7) Does the software engineering team have the right mix of skills?
8) Are project requirements stable?
9) Does the project team have experience with the technology to be implemented?
10) Is the number of people on the project team adequate to do the job?
11) Do all customer/user constituencies agree on the importance of the project and on the requirements for the system/product to be built?

- The project manager identifies the risk drivers that affect the following risk components
  - **Performance risk** - the degree of uncertainty that the product will meet its requirements and be fit for its intended use
  - **Cost risk** - the degree of uncertainty that the project budget will be maintained
  - **Support risk** - the degree of uncertainty that the resultant software will be easy to correct, adapt, and enhance
  - **Schedule risk** - the degree of uncertainty that the project schedule will be maintained and that the product will be delivered on time
- The impact of each risk driver on the risk component is divided into one of four impact levels
  - Negligible, marginal, critical, and catastrophic
- Risk drivers can be assessed as impossible, improbable, probable, and frequent

Risk Projection (Estimation)

Background

- Risk projection (or estimation) attempts to rate each risk in two ways
  - The probability that the risk is real
  - The consequence of the problems associated with the risk, should it occur
- The project planner, managers, and technical staff perform four risk projection steps (see next slide)
- The intent of these steps is to consider risks in a manner that leads to prioritization
- Be prioritizing risks, the software team can allocate limited resources where they will have the most impact

Risk Projection/Estimation Steps

1) Establish a scale that reflects the <u>perceived likelihood</u> of a risk (e.g., 1-low, 10-high)
2) Delineate the <u>consequences</u> of the risk
3) Estimate the <u>impact</u> of the risk on the project and product
4) Note the <u>overall accuracy</u> of the risk projection so that there will be no misunderstandings

Contents of a Risk Table

- A risk table provides a project manager with a simple technique for risk projection
- It consists of five columns
  - Risk Summary – short description of the risk
  - Risk Category – one of seven risk categories (slide 12)
  - Probability – estimation of risk occurrence based on group input
  - Impact – (1) catastrophic (2) critical (3) marginal (4) negligible
  - RMMM – Pointer to a paragraph in the Risk Mitigation, Monitoring, and Management Plan

Risk Summary	Risk Category	Probability	Impact (1-4)	RMMM

Developing a Risk Table

- List all risks in the first column (by way of the help of the risk item checklists)
- Mark the category of each risk
- Estimate the probability of each risk occurring
- Assess the impact of each risk based on an averaging of the four risk components to determine an overall impact value (See next slide)
- Sort the rows by probability and impact in descending order
- Draw a horizontal cutoff line in the table that indicates the risks that will be given further attention

Assessing Risk Impact

- Three factors affect the consequences that are likely if a risk does occur
  - **Its nature** – This indicates the problems that are likely if the risk occurs
  - **Its scope** – This combines the severity of the risk (how serious was it) with its overall distribution (how much was affected)
  - **Its timing** – This considers when and for how long the impact will be felt
- The overall risk exposure formula is RE = P x C
  - P = the probability of occurrence for a risk
  - C = the cost to the project should the risk actually occur
- Example
  - P = 80% probability that 18 of 60 software components will have to be developed
  - C = Total cost of developing 18 components is $25,000
  - RE = .80 x $25,000 = $20,000

Risk Mitigation, Monitoring, and Management

Background

- An effective strategy for dealing with risk must consider <u>three</u> issues

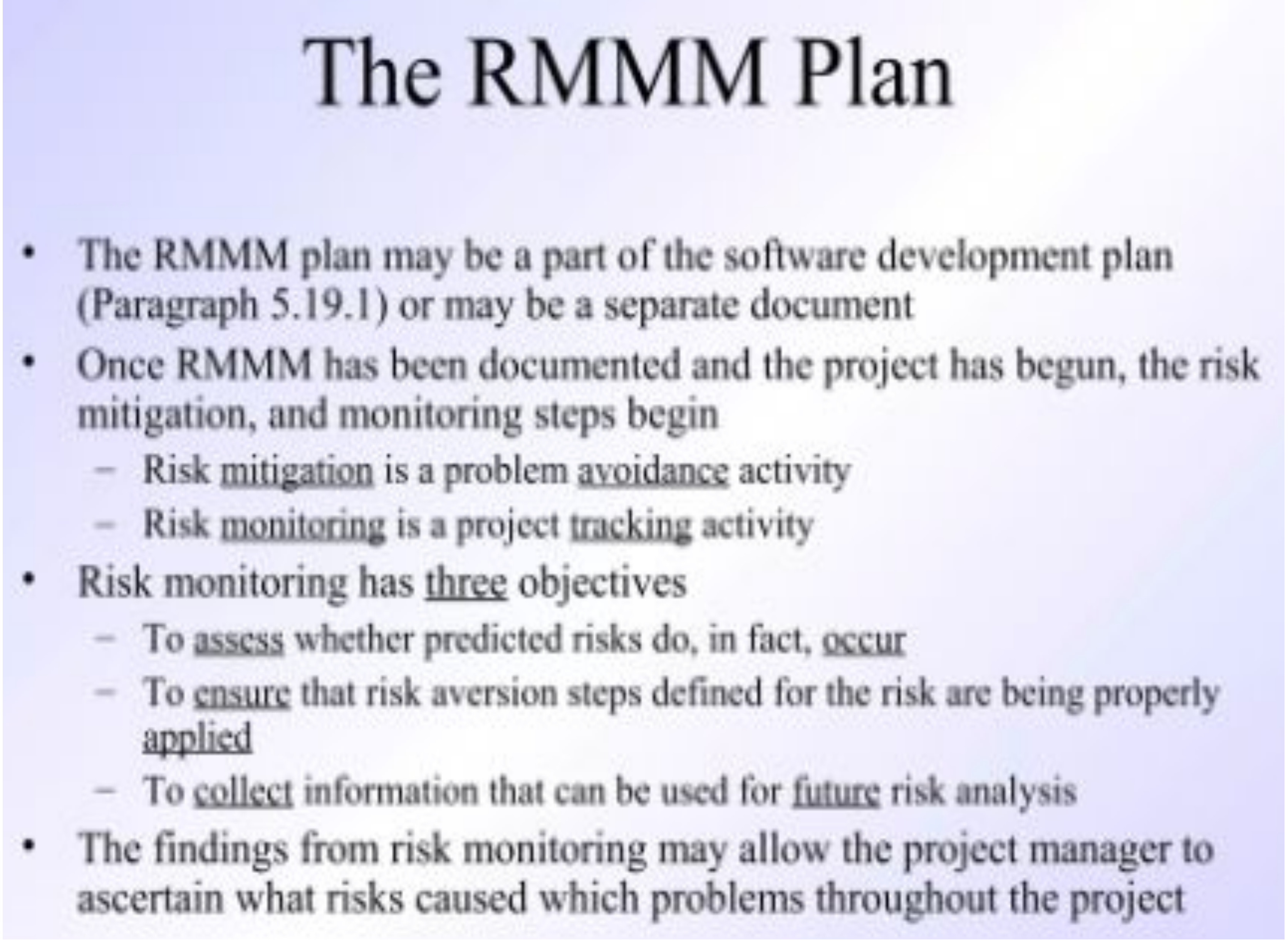
  (Note: these are not mutually exclusive)
  - Risk mitigation (i.e., avoidance)
  - Risk monitoring
  - Risk management and contingency planning
- <u>Risk mitigation</u> (avoidance) is the primary strategy and is achieved through a plan
  - Example: Risk of high staff turnover (see next slide)

Strategy for Reducing Staff Turnover

- <u>Meet</u> with current staff to <u>determine causes</u> for turnover (e.g., poor working conditions, low pay, competitive job market)
- <u>Mitigate</u> those causes that are under our control before the project starts
- Once the project commences, <u>assume</u> turnover will occur and <u>develop</u> techniques to ensure continuity when people leave
- <u>Organize</u> project teams so that information about each development activity is <u>widely dispersed</u>
- <u>Define</u> documentation standards and <u>establish</u> mechanisms to ensure that documents are developed in a timely manner
- <u>Conduct</u> peer reviews of all work (so that more than one person is "up to speed")
- <u>Assign</u> a backup staff member for every critical technologist

- During risk monitoring, the project manager monitors factors that may provide an indication of whether a risk is becoming more or less likely
- Risk management and contingency planning assume that mitigation efforts have failed and that the risk has become a reality
- RMMM steps incur additional project cost
  - Large projects may have identified 30 – 40 risks
- Risk is not limited to the software project itself
  - Risks can occur after the software has been delivered to the user

- Software safety and hazard analysis
  - These are <u>software quality assurance</u> activities that focus on the <u>identification</u> and <u>assessment</u> of potential hazards that may affect software negatively and cause an entire system to fail
  - If hazards can be <u>identified early</u> in the software process, software design features can be specified that will either <u>eliminate</u> or <u>control</u> potential hazards



The RMMM Plan

- The RMMM plan may be a part of the software development plan (Paragraph 5.19.1) or may be a separate document
- Once RMMM has been documented and the project has begun, the risk mitigation, and monitoring steps begin
  - Risk mitigation is a problem avoidance activity
  - Risk monitoring is a project tracking activity
- Risk monitoring has three objectives
  - To assess whether predicted risks do, in fact, occur
  - To ensure that risk aversion steps defined for the risk are being properly applied
  - To collect information that can be used for future risk analysis
- The findings from risk monitoring may allow the project manager to ascertain what risks caused which problems throughout the project

Seven Principles of Risk Management

- **Maintain a global perspective**
  - View software risks within the context of a system and the business problem that is is intended to solve
- **Take a forward-looking view**
  - Think about risks that may arise in the future; establish contingency plans
- **Encourage open communication**
  - Encourage all stakeholders and users to point out risks at any time
- **Integrate risk management**
  - Integrate the consideration of risk into the software process
- **Emphasize a continuous process of risk management**
  - Modify identified risks as more becomes known and add new risks as better insight is achieved
- **Develop a shared product vision**
  - A shared vision by all stakeholders facilitates better risk identification and assessment
- **Encourage teamwork when managing risk**
  - Pool the skills and experience of all stakeholders when conducting risk management activities

Software quality is the degree of conformance to explicit or implicit requirements and expectations.

Software Quality Concept

- The quality of a software product is defined in terms of its characteristics or attributes.
- The values of attributes vary from product to product.
- A product can be of good quality or bad quality. Weaker values of attributes define a bad-quality product whereas higher values of attributes define a good-quality product.
- The aim of software development organizations is to produce a high-quality product.
- The external quality of a product does not always ensure the internal quality.
- *It is analogous to the quality of a building.*

Software Quality Concept

- A good quality product aims to satisfy the customer requirements, developed under the guidelines of software quality standards.
- Cost and schedule also play important roles in defining the quality of software.
- The process of development can affect the product quality.
- A standard process provides a systematic approach to development and leads to a high-quality product.
- Quality of software can be defined in terms of following points.
    - *satisfies customer requirements*
    - *possesses higher values of its characteristics*
    - *has sound internal design along with external design*
    - *is developed within the budget and cost*
    - *follows the development standards*

Software Quality Assurance (SQA)

- “assurance” means “guarantee”. So, software quality assurance (SQA) guarantees high quality of software.
- IEEE defines *quality assurance as a planned and systematic pattern of all actions necessary to produce adequate confidence that the item or product conforms to established technical requirements* .
- Quality assurance activities are different from development and maintenance activities.
- Software quality assurance is a preventive approach of quality.
- Its goal is to provide and implement activities within the quality system that assist in getting confidence to ensure that the work product will fulfill the requirements of quality.

SQA Activities

- The quality assurance activities are performed at two levels, namely, the activities performed by the software development team and the activities performed by the quality assurance team.
- The software development team focuses on applying technical methods and tools in achieving product quality.
- Following methods are performed by the development team to ensure product quality:
  - *Applying technical methods and tools*
  - *Conducting verification and validation*
  - *Performing measurements*
  - *Performing testing*

SQA Activities

- The SQA team performs the following activities:
  - *Prepares SQA plan*
  - *Participates in process execution for development*
  - *Reviews software engineering activities to verify compliance with the defined software process*
  - *Software auditing*
  - *Record keeping*
  - *Reporting*

SQA Plan

- ❖ The following activities are included in the SQA plan:
  - *Purpose, scope, and objectives of the SQA plan*
  - *Documents referenced in the plan*
  - *Design of organizational structure for performing and reporting quality tasks*
  - *Documents to be prepared and verified*
  - *Standards, practices, and conventions to be used*
  - *Tools, techniques, and methodologies*
  - *Reviews and audits to be conducted*
  - *Configuration management planning*

SQA Plan

- *Practices and procedures for reporting, tracking, and resolving software problems*
- *Maintaining SQA records*
- *Code control*
- *Media control and backup*
- *Supplier control*

SOFTWARE REVIEW

COST IMPACT OF SOFTWARE DEFECTS:

The cost of defects can be measured by the impact of the defects and when we find them. Earlier the defect is found lesser is the cost of defect. For example if error is found in the requirement specifications then it is somewhat cheap to fix it. The correction to the requirement specification can be done and then it can be re-issued. In the same way when defect or error is found in the design then the design can be corrected and it can be re-issued. But if the error is not caught in the specifications and is not found till the user acceptance then the cost to fix those errors or defects will be way too expensive.

If the error is made and the consequent defect is detected in the **requirements phase** then it is relatively cheap to fix it.

Similarly if an error is made and the consequent defect is found in the **design phase** then the design can be corrected and reissued with relatively little expense.

ISO 9000 Standard

- It is based on eight quality management principles that senior management applies for organizational improvement.

  - *Customer focus:* As organizations depend on their customers, they should understand current and future customers.
  - *Leadership:* Leadership establishes unity of purpose and direction of the organization. Organizations should create and maintain an internal environment in which people can become fully involved in achieving the organization's objectives.
  - *Involvement of people:* People at all levels are the assets of an organization and their full involvement enables their abilities to be used for the organization's benefits.
  - *Process approach:* The desired result is achieved more efficiently when activities and related resources are managed as a process.

ISO 9000 Standard

- *System approach to management:* Identifying, understanding, and managing interrelated processes as a system contributes to the organization's effectiveness and efficiency in achieving its objectives.
- *Continual improvement:* Continual improvement of the organization's overall performance should be a permanent objective of the organization.
- *Factual approach to decision making:* Effective decisions are based on the analysis of data and information.
- *Mutually beneficial supplier relationships:* An organization and its suppliers are interdependent and a mutually beneficial relationship enhances the ability of both to create value.