Session 7

Software Maintenance and Process Scheduling.

Ernest Cachia
Department of Computer Science

Session Aims

The main aim of this session is to outline the maintenance process in software engineering, to explain its various parts, to provide a scientific framework for system evolution, and to present a form of measurement of the maintenance effort. To end, two methods used for project activity scheduling will be explained.

- Introduce the ideas behind system maintenance in terms of overall system development
- Lehman's Laws of system evolution
- Maintenance measurement
- Activity scheduling methods

Ernest Cachia
Department of Computer Science

Session Contents

- Maintenance and system evolution
- Maintenance metrics
- System Complexity metrics
- Scheduling models

Ernest Cachia
Department of Computer Science

Common Views on Maintenance

Some basic misconceptions of maintenance:

- Can be considered after solution delivery
- Is something secondary to (and not as important as) development
- Can be handled by less-competent developers
- Not that important to clients
- Not that costly
- Might never be needed anyway

Ernest Cachia
Department of Computer Science

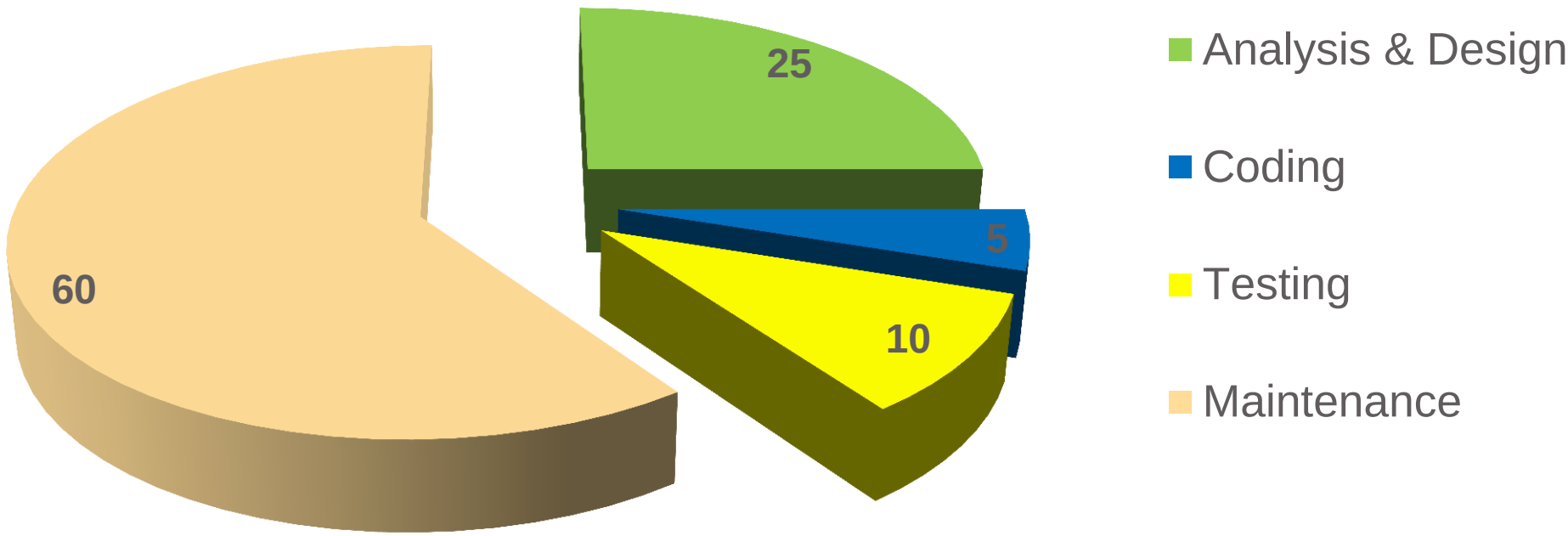
The Truth Be Told…

The truth about maintenance in the modern system development process:

- Must be a driving factor in the way a solution is built
- Is actually a mini development cycle in its own right
- The people who build the solution should be the ones who maintain it
- Is often the clinching issue of many software development contracts
- Should not be costly – however, if neglected can be even more costly than the solution itself
- Is critical for the continued usefulness, and survival, of the system

Ernest Cachia
Department of Computer Science

Maintenance in Development



All values in chart are approximated from various sources and rounded.

Ernest Cachia
Department of Computer Science

Reasons for High Maintenance Costs

- Reputation as being “second class development” amongst software developers
- The widespread presence of legacy systems
- Innovation brings new errors with it
- Gradual degradation of long-standing and often-maintained systems (this will be better explained in the part dealing with Lehman’s Laws)
- Inaccurate and un-matching documentation

Ernest Cachia
Department of Computer Science

Highlighting the Importance of Maintenance

Barry Boehm proposes the following stances (with some personal adaptation):

- Link solution objectives to organisational goals
- Link software maintenance rewards to organisational performance
- Make software members of operational teams take turns at maintenance – create no distinction of roles
- Allow adequate budget and a good degree of independence within teams handling maintenance
- Involve maintenance staff early in the software process and during all stages of development.

Ernest Cachia
Department of Computer Science

Types of Maintenance

- Perfective

  Bringing solution “up-to-scratch” with any minor changes in requirements as well as improving its external quality attributes

- Adaptive

  Changes brought about by technology and/or working environment changes

- Corrective

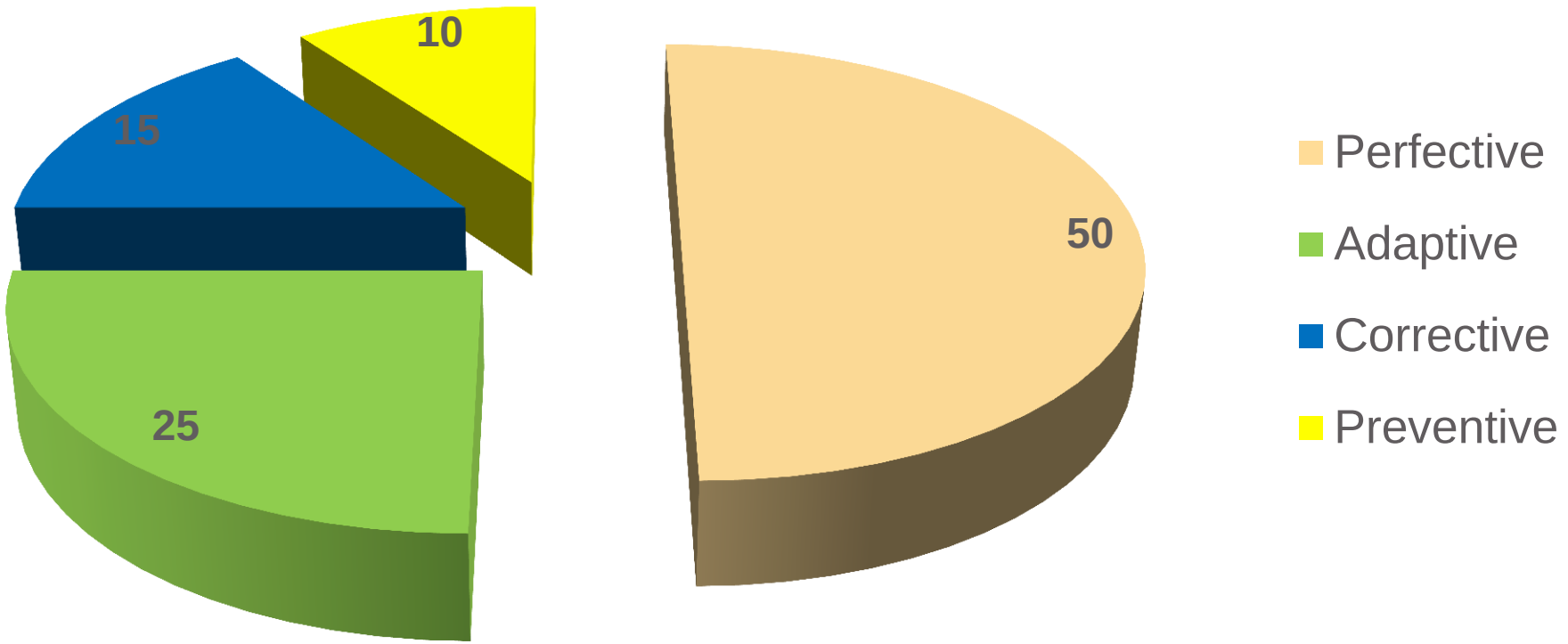
  Carrying out repairs in any development phase of the system

- Preventive

  Making the solution easier to maintain and understand

Ernest Cachia
Department of Computer Science

Maintenance Categories

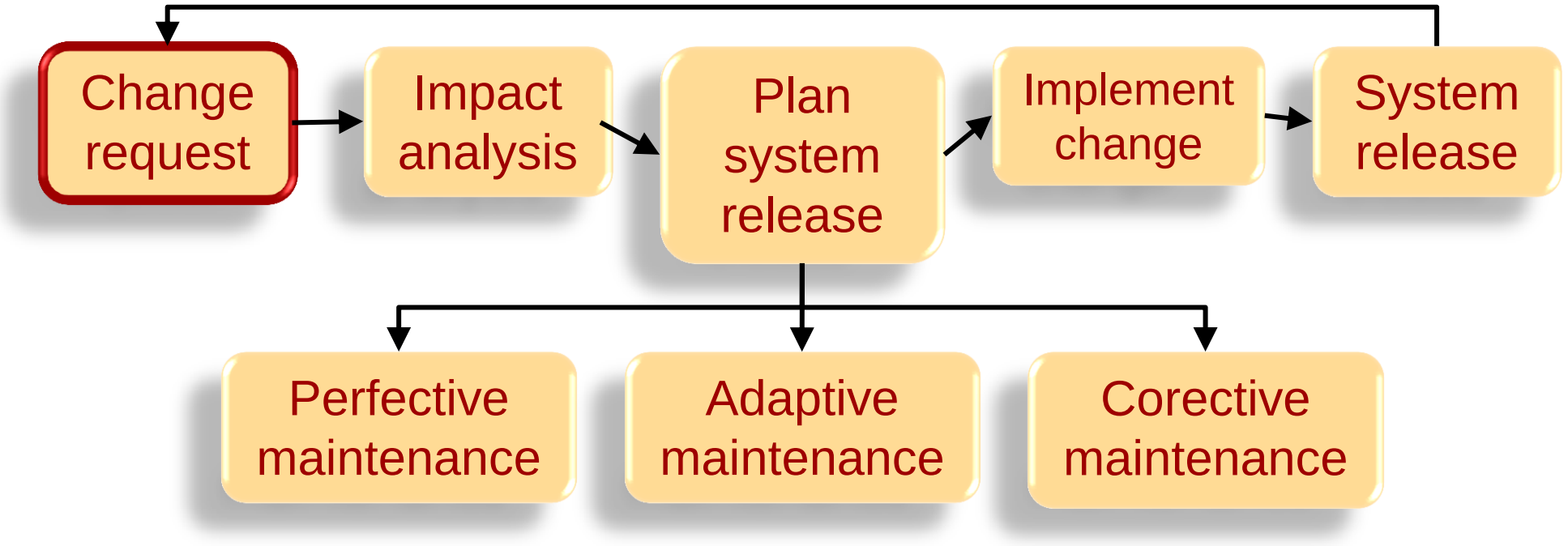


All values in chart are approximated from various surveys and rounded.

Ernest Cachia
Department of Computer Science

A Maintenance Process Example

A maintenance process which uses the different types of maintenance is the following:



Regression Testing

When parts of a system are changed, one must ensure that the unchanged parts work as they did before. This is called regression testing, and is made up of the following steps:

- Prepare a general purpose set of test cases (TCs) for the existing system.
- Run the TCs on the existing version and save the results.
- **Make program modifications.**
- Now run the same TCs on the modified and save the results.
- Compare both sets of results (i.e. from existing and modified).

RESULTS SHOULD BE IDENTICAL

Ernest Cachia
Department of Computer Science

Lehman's Laws of System Evolution

Meir Manny Lehman (while Professor at Imperial College, University of London), together with colleagues, proposed a set of distinct behavioural patterns governing software system evolution. These patterns came to be known as Lehman's Laws.

Lehman's Laws are 8 in all. However only 5 are widely accepted, and of these usually only the first 2 are most commonly quoted. These are the following:

1) Continuing change
   Software must continually evolve, or grow useless.
2) Increasing complexity
   The structure of evolving software tends to degrade.

Ernest Cachia
Department of Computer Science

Maintenance Cost

Factors effecting maintenance costs are subdivided into:

- Technical
- Non-technical

Ernest Cachia
Department of Computer Science

Technical Factors

Technical factors effecting maintenance cost:

- Module independence (maintainability)
- Programming language (understand-ability)
- Programming style (understand-ability)
- Program validation and verification (i.e. correction avoidance)
- Documentation (understand-ability)
- Configuration management (i.e. structured evolution)

Ernest Cachia
Department of Computer Science

Non-Technical Factors

Non-technical factors effecting maintenance cost:

- Application domain familiarity (i.e. clear comprehension)
- Staff stability (i.e. the builders are the maintainers)
- Program age (i.e. structure degradation)
- External environment (i.e. real-word dependence)
- Hardware stability (i.e. technology advancement)

Ernest Cachia
Department of Computer Science

Maintenance Cost Estimation

Annual Change Traffic (ACT) is the fraction (%) of a software product's source instructions which undergo change during a (typical) year either through addition or modification (taken from Ian Sommerville)

Annual Maintenance Effort (AME) is calculated as follows:

$$AME = ACT \times PM$$

Where PM represents the estimated or actual development effort in person (or programmer)-months for the whole system

After this, use AME as effort input to the Intermediate COCOMO-1 method.

Ernest Cachia
Department of Computer Science

Maintenance Effort Estimation Example

Let us assume that a 90pm were required to develop a system. Furthermore, it is estimated that the annual change traffic (ACT) is 15% (i.e. approx. 15% of code will change in the course of a year)

Therefore, the annual maintenance effort (AME):

$$AME = 0.15 * 90pm = 13.5pm$$

Two possible problems to this approach (Sommerville)**:**

1) What would the ACT value for new systems be?
2) Are all COCOMO development attributes applicable to maintenance?

Ernest Cachia
Department of Computer Science

Modularity

Definition: “One of a set of separate parts which, when combined, form a complete whole” (Cambridge on-line dictionary)

In may classifications, this is a recurring factor influencing system maintenance.

Modularity influences system complexity which directly effects system maintainability

The metrics used to measure system complexity are:

- Coupling (defined as 5 levels of coupling)
- Cohesion (defined as 7 levels of cohesion)

[These were covered in the first year of the Software Engineering stream]

Ernest Cachia
Department of Computer Science

Activity “CSA3170-D”

In the context of modular systems development, read up and understand why and how coupling and cohesion effect system maintainability. Name and briefly outline all five levels of coupling and all seven levels of cohesion. One short paragraph for each level is enough.

For your information (mainly to remind you):

The 5 levels of coupling are:

Context; Common; Control; Stamp; Data

The 7 levels of cohesion are:

Coincidental; Logical; Temporal; Procedural; Communicational; Sequential; Functional

Ernest Cachia
Department of Computer Science

Project Scheduling

Definition: “A list of planned activities or things to be done showing the times or dates when they are intended to happen or be done” (Cambridge on-line dictionary)

A software project is made up of activities, and these must happen according to plan – i.e. scheduled.

Schedulable components:

- Activities
- Resources (including the human variety)
- Time (durations and deadlines)
- Products (intermediate and final)

Ernest Cachia
Department of Computer Science

Activity On Arrow Diagrams

We need to be able to clearly model activities to be able to schedule them. One approach is to use an Activity On Arrow (AOA) style diagram.

A prime example of such (AOA) diagrams is the Project Evaluation and Review Technique (or PERT) chart.

- Diagram components (symbols)
  - Nodes (drawn as circles)
  - Links (drawn as directed arcs)
- Symbol meanings
  - Nodes: Start/Stop events (points)
  - Links: Activities

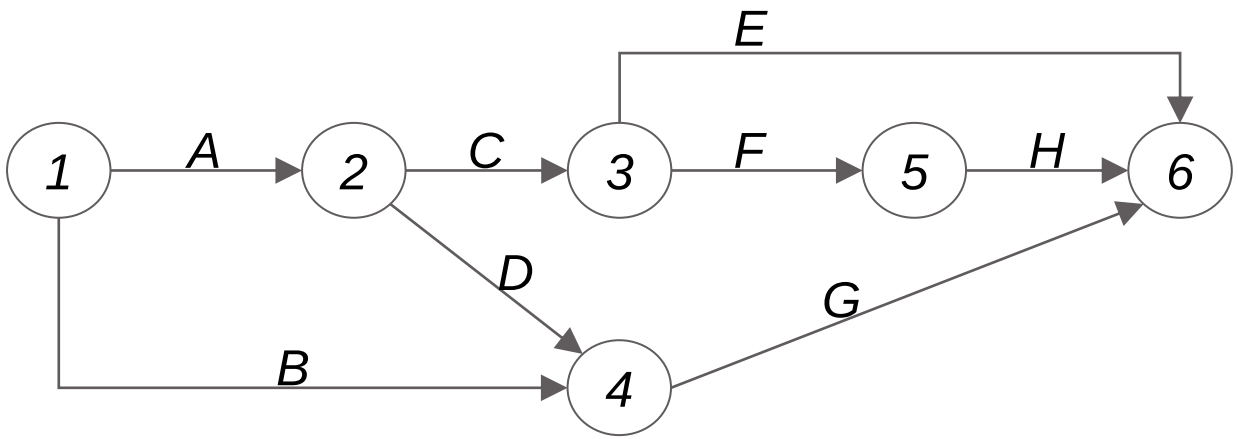
Ernest Cachia
Department of Computer Science

AOA Chart Construction Rules

- Must contain only one start and one end node
- A link has duration (optionally shown)
- A node has no duration (simply start/stop point)
- Time flows from left to right
- Nodes are numbered sequentially
- Loops are not allowed (by concept)
- “Dangles” are not allowed (except in the case of the one and only end node)

Ernest Cachia
Department of Computer Science

AOA Chart Example *(1/3)*

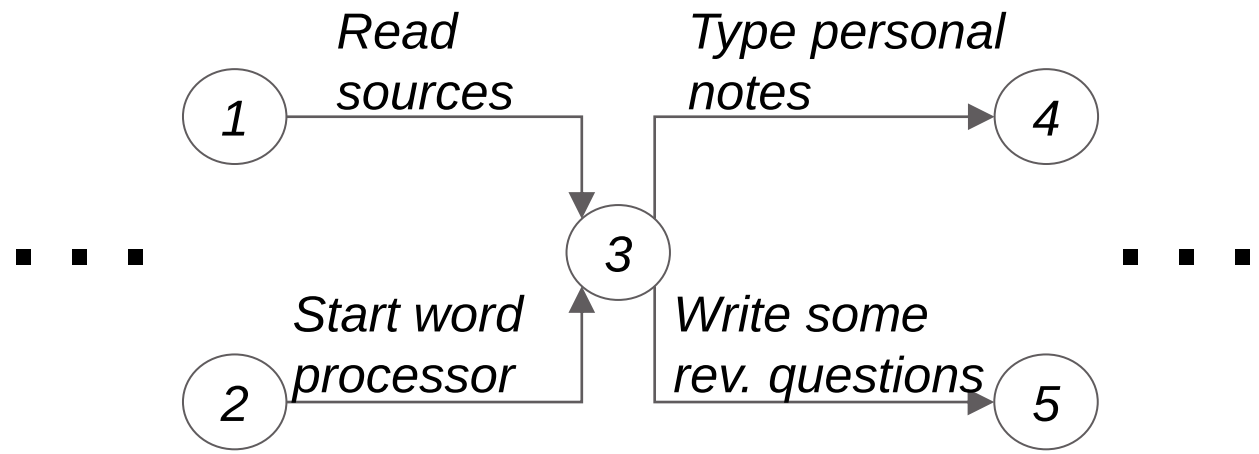


Explanation:

The above project (or part of) consists of eight activities (“A”~“H”). The duration of each activity is not indicated. The project starts at node one and ends at node six. The derived duration of activity “A” is the time difference between node two and node one; the derived duration of activity “B” is the time difference between node four and node 1; and so on.

Ernest Cachia
Department of Computer Science

AOA Chart Example *(2/3)*

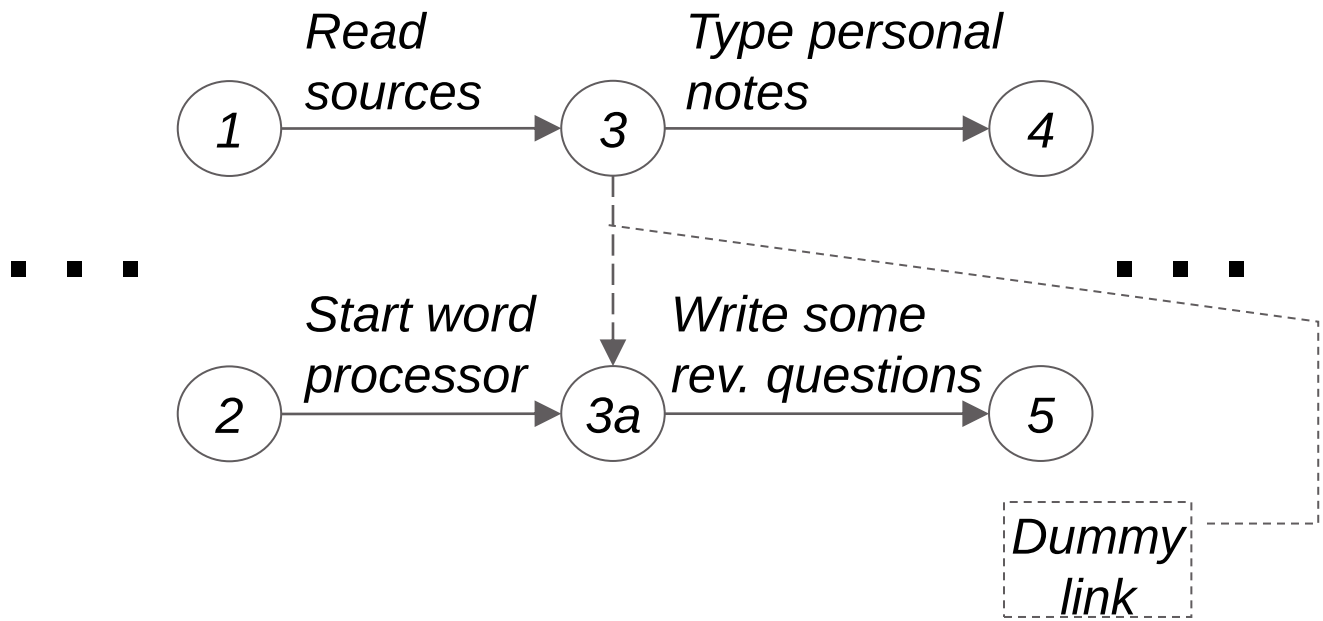


Explanation:

There are four activities in all. A student reads from various sources and starts a word-processor to then type in some personal notes and furthermore, manually writes some questions on paper to remember to ask the lecturer. IN PRACTICE reading and writing questions can proceed separately from starting the word processor to type in some personal notes. ***Therefore…***

Ernest Cachia
Department of Computer Science

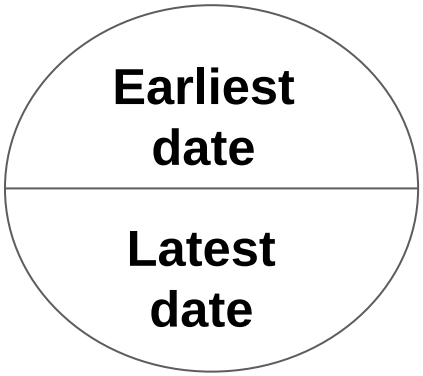
AOA Chart Example *(3/3)*



Please note, that a “dummy link” has zero duration time and uses absolutely no resources.

Ernest Cachia
Department of Computer Science

PERT Chart Nodes



PERT Chart (milestone) node

Activity ID and duration →

PERT Chart activity

Ernest Cachia
Department of Computer Science

PERT Chart Example *(1/2)*

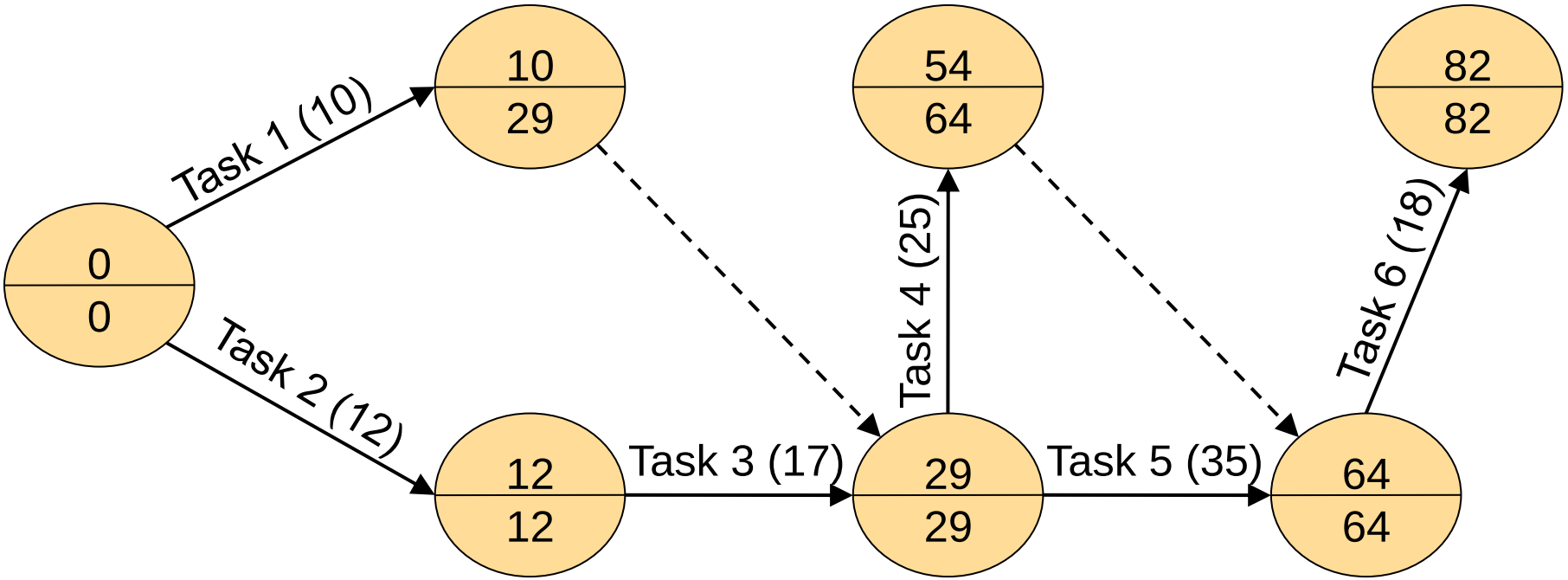
Let us take the table below, representing various activities in a hypothetical project, as an example.

Activity	Duration (units)	Dependencies
Task 1	10	
Task 2	12	
Task 3	17	Task 2
Task 4	25	Tasks 1 & 3
Task 5	35	Tasks 1 & 3
Task 6	18	Tasks 4 & 5

A PERT chart model of this sequence of activities is shown on the next slide.

Ernest Cachia
Department of Computer Science

PERT Chart Example *(2/2)*

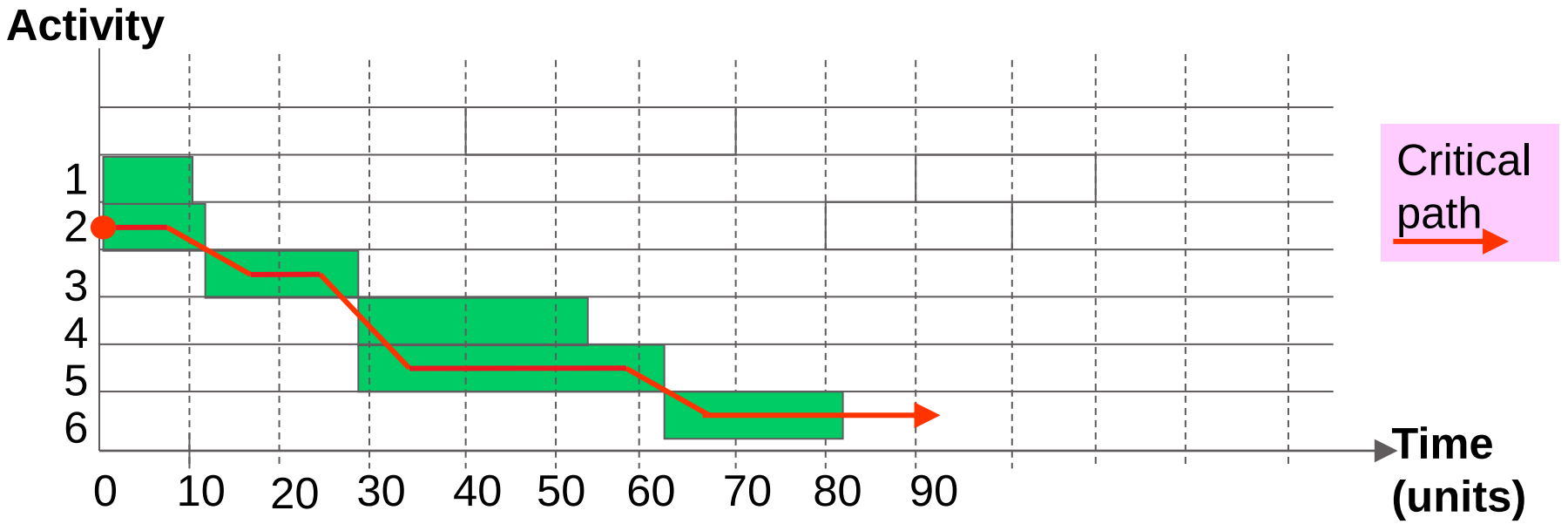


The “critical path” is the one that contains activities that would cause project delay on the whole had they to be delayed themselves. **In this example: Tasks 2, 3, 5, and 6.**

Ernest Cachia
Department of Computer Science

Gantt Chart Example

Gantt charts are a form of bar chart published by Henry Laurence Gantt (an American mechanical engineer) in 1910.



Ernest Cachia
Department of Computer Science

Summary (Session 7)

- An introduction to software system maintenance
- Types of maintenance
- Software evolution through two of Lehman's Laws
- Maintenance measurement and regression testing
- Coupling and cohesion as complexity/maintainability metrics
- An introduction to scheduling
- Scheduling through PERT and Gantt charts

Ernest Cachia
Department of Computer Science

Barry W. Boehm

Dr. Barry Boehm served within the U.S. Department of Defense (DoD) from 1989 to 1992 as director of the DARPA Information Science and Technology Office and as director of the DDR&E Software and Computer Technology Office. He worked at TRW from 1973 to 1989, culminating as chief scientist of the Defense Systems Group, and at the Rand Corporation from 1959 to 1973, culminating as head of the Information Sciences Department. He entered the software field at General Dynamics in 1955.

His current research interests involve recasting software engineering into a value-based framework, including processes, methods, and tools for value-based software definition, architecting, development, validation, and evolution. His contributions to the field include the Constructive Cost Model (COCOMO), the Spiral Model of the software process, and the Theory W (win-win) approach to software management and requirements determination. He has received the ACM Distinguished Research Award in Software Engineering and the IEEE Harlan Mills Award, and an honorary ScD in Computer Science from the University of Massachusetts. He is a Fellow of the primary professional societies in computing (ACM), aerospace (AIAA), electronics (IEEE), and systems engineering (INCOSE), and a member of the U.S. National Academy of Engineering.

Back to originating slide

Ernest Cachia
Department of Computer Science

Session 8

Software Quality Assurance and Related Measurements.

Ernest Cachia
Department of Computer Science

Session Aims

The main aim of this session is to explain Function Point calculation and use, and to introduce the student to some basic ways in which software quality can be measured from a statistical point of view in terms of defects and from a probabilistic point of view in terms of reliability

- Explain Function Points and calculation and measurement based upon them
- Introduce the fundamentals of Statistical Quality Assurance
- Present the fundamentals of defect classification and modelling
- Offer some basic reliability and availability calculation techniques

Ernest Cachia
Department of Computer Science

Session Contents

- Function Points
- Statistical Quality Assurance
- Error Modelling
- System Reliability and Availability

Ernest Cachia
Department of Computer Science

Function Points

Some basic concepts of Function Points*

- Function Points (FPs) are an attribute of a system based on its internal functions
- Function Points are sometimes preferred to Lines of Code (LOC) as a base measure
- Closer to user perspective of the system - Its function "size" rather than its coding size
- LOC can be misleading when, amongst other things, language generations are crossed

** Developed by Allan Albrecht working at IBM in 1979*

Ernest Cachia
Department of Computer Science

Function Point Determination

The next four slides will show you the sequence of how to extract FPs and determine their count, for a particular system. Following this, a practical example will be presented.

Ernest Cachia
Department of Computer Science

Function Point Calculation *(1/4)*

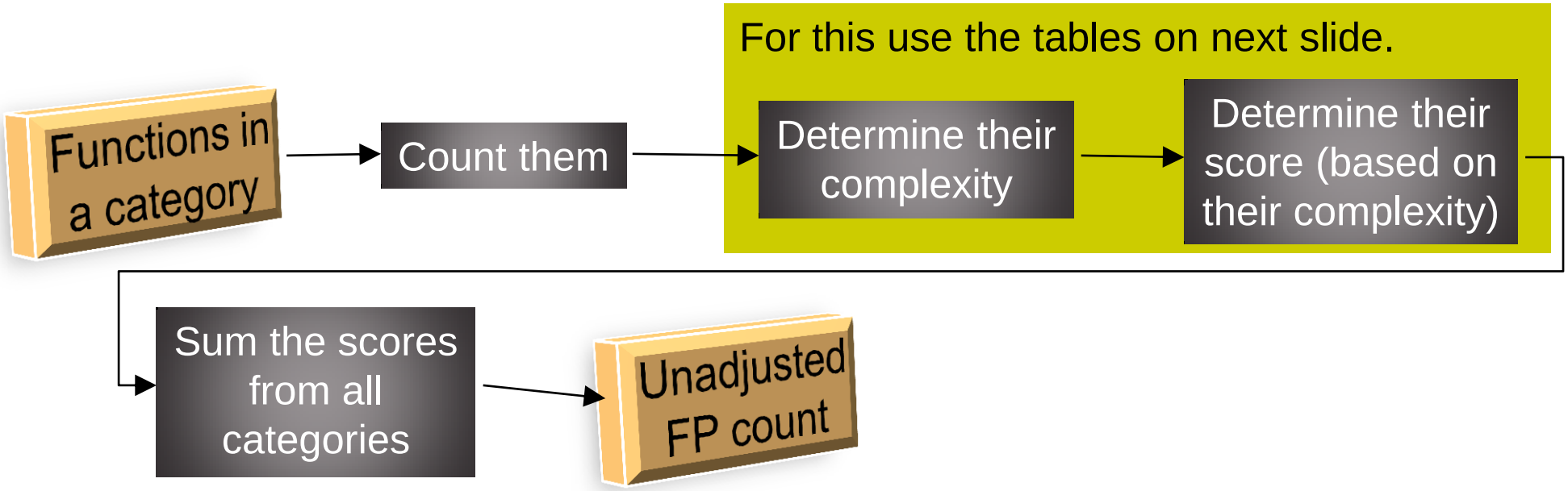
Isolate the basic function types in a system's specification and decide in which of the following categories each function would fit. Use Albrecht's 5-type categorisation, as follows:

1. *External inputs*
   Distinct data used by system (e.g. data structures, file names, etc.)
2. *External outputs*
   e.g. reports, normal/error messages, menu screens, etc.
3. *Enquiries*
   Interactive inputs (these require an immediate response)
4. *External files*
   Files shared with other systems
5. *Internal files*
   Files not visible outside the system

Ernest Cachia
Department of Computer Science

Function Point Calculation *(2/4)*

- These basic function types are then counted and weighted according to their complexity *[The criteria for this may vary between organisations and if taken as an absolute value can be subjective. However, if they are consistently applied organisation-wide, then they are very useful]*
- The following top-level process is used to compute an initial FP count value:



Ernest Cachia
Department of Computer Science

Function Point Calculation *(3/4)*

Function type	Determinants								
	# of files eff.			*# of rec. eff.*			*# of fields eff.*		
	1	*2*	*3*	*1*	*2*	*3*	*1*	*2*	*3*
Input	0-1	2	≥3	-		-	1-4	5-15	≥16
Output	0-1	2-3	4	-		-	1-5	6-19	≥20
Internal file	-	-	-	1	2-5	≥6	1-19	20-50	≥51
External file	-	-	-	1	2-5	≥6	1-19	20-50	≥51
Query	Use the greater of either its input or output part								

① Grade = 1 + 2 = 3

Grade	Complexity level
2-3	Simple
4	Average
5-6	Complex

② = Simple

Function type	Score		
	Simple	*Average*	*Complex*
Input	3	4	6
Output	4	5	7
Internal file	7	10	15
External file	5	7	10
Query	3	4	6

③ Score = **4** (Sum of all scores = UFC)

UFC means "Unadjusted FP Count"

Ernest Cachia
Department of Computer Science

Function Point Calculation *(4/4)*

Once the unadjusted function count (UFC) has been calculated, the following relationship can be applied:

$$FP = UFC \times [0.65 + 0.01 \times \Sigma F_i]$$

F_i is known as the complexity factor. It is obtained by grading in a range of "0" to "5" ("0" meaning "least relevant", "5" meaning "extremely relevant") a list of 14 characteristics called "General System Characteristics" (GSCs), and then taking *the sum* of all the characteristic grades. Click here to see these 14 characteristics.

The constants used in the above formula are derived empirically.

Ernest Cachia
Department of Computer Science

Function Point Example *(1/6)*

System Description

5	*Inputs*
2	*Outputs*
1	*Internal files*
1	*Queries*
X	*External files*

The system will enable new customers to be added and deleted from a customer database. The system must also support paying in and withdrawal transactions, and will display a warning message if a borrower has an excessive overdraft. Customers should be able to query their account balance via apposite terminals. A report of overdrawn customers can be requested.

System description taken from
"Foundations of Software Measurement",
by M. Shepperd

Ernest Cachia
Department of Computer Science

Function Point Example *(2/6)*

Function Categorisation

Inputs:

- Add customer
- Delete customer
- Pay-in
- Withdraw
- Request overdrawn customers report

Outputs:

- Warning message
- Report of overdrawn customers

Queries:

- Request account balance

Internal files:

- Customer database

External files:

<none>

Ernest Cachia
Department of Computer Science

Function Point Example *(3/6)*

Given (or deduced) Values

Assumptions for this example:

- A customer record can contain up to 20 separate fields;
- Only one customer file will be used;
- Every customer entity data will be distributed over three separate tables.

Ernest Cachia
Department of Computer Science

Function Point Example *(4/6)*

Tabulate Function Complexities

Function type	*# files*	*# recs.*	*# fields*	*Complexity*	*Score*
Add	1	-	20	average	4
Delete	1	-	8	simple	3
pay-in	1	-	2	simple	3
withdraw	1	-	2	simple	3
report request	1	-	0	simple	3
warning message	1	-	5	simple	4
overdrawn report	1	-	12	simple	4
balance query	1	-	6	simple	3
customer file	-	3	25	average	10
				UFC:	***37***

Ernest Cachia
Department of Computer Science

Function Point Example *(5/6)*

Refine the UFC

According to:

$FP = UFC(0.65 + (0.01 \cdot \sum F_i))$

Where $\sum F_i$ is the sum of the resulting replies to the 14 Complexity Adjustment Values.

Now assuming that $\sum F_i = 30$, this would yield:

$FP = 37(0.65 + (0.01 \times 30)) = 35.15$

i.e. 35.15 function points.

Ernest Cachia
Department of Computer Science

Function Point Example *(6/6)*

Example of how one could use FP counts

Pascal program: 4500 LOC; C# program: 1200 LOC

Pascal programmer takes 6 months

C# programmer takes 2 months

Productivity (LOC):

PP = 4500/6 = 750 LOC/month ✓

CP = 1200/2 = 600 LOC/month

Productivity (FP):

PP = 35.15/6 = 5.86 FP/month

CP = 35.15/2 = 17.58 FP/month ✓

Ernest Cachia
Department of Computer Science

Statistical Quality Assurance

A quantitative way to qualitative evaluation

The Main Steps Involved

1. Categorise data on s/w defects
2. Define the underlying causes of s/w defects
3. Use the “Pareto principle” (aka “The 20/80 Rule”) to condense the defect causes
4. Implement corrective measures on causes

Ernest Cachia
Department of Computer Science

The Pareto Principle

80% of effects are attributable to 20% of the causes

- Not something to do with software development only
- Has been around for quite a while now

  Since 1906, introduced by Italian economist (Vilfredo Pareto) as a mathematical formula in a study of wealth distribution in Italian society
- Has proven its validity in many domains on many occasions since its inception

Vilfredo Pareto
(1848 – 1923)

Ernest Cachia
Department of Computer Science

The Causes of Software Defects

- Incomplete or erroneous spec. (IES)
- Misinterpretation of customer comm.
- Intentional deviation from spec.
- Violation of programming standards.
- Error in data representation. (EDR)
- Inconsistent module interface.
- Error in design logic. (EDL)
- Incomplete or erroneous testing.
- Incomplete or inaccurate documentation.
- Programming language translation of design error. (PLT)
- Ambiguous or inconsistent HCI.
- Miscellaneous. *(a form of catch-all)*

Ernest Cachia
Department of Computer Science

Corrective Measure Examples

- IES - Improve specification techniques, introduce new methods, upgrade personnel, etc.
- EDR - Adopt automated data design tools, impose stringent data modelling and reviews, etc.
- PLT - Use more visibility, check design phase output, enforce strict translation techniques, etc.
- EDL – Reinforce good requirements understanding, ensure personnel quality, adopt widespread design techniques, etc.

Ernest Cachia
Department of Computer Science

The “Error Index”

- The “Error Index” (EI) is an arbitrary value by which to quantify the quality, in terms of errors in code, of software development.
- The EI is determined for a software product as a whole and is derived from an error index at every development phase known as a “Phase Index” (PI).

Ernest Cachia
Department of Computer Science

EI Calculation Procedure

Calculate the PI for a given phase:

$$PI_i = w_s(S_i/E_i) + w_m(M_i/E_i) + w_t(T_i/E_i)$$

Where: S_i: number of serious errors,
M_i: number of moderate errors,
T_i: number of trivial errors,
E_i: total errors uncovered in i^{th} step of the process
$w_s / w_m / w_t$: The weighting given to each type of error (i.e. serious, moderate, or trivial)

Unless stated otherwise, it is recommended that:

$w_s = 10$; $w_m = 3$; $w_t = 1$

The final error index is the sum of all the phase indices weighted according to their sequence in the SE process.

$$EI = \sum(i \times PI_i)/PS_i = (PI_1 + 2PI_2 + \ldots + iPI_i)/PS$$

Where: PS_i is product size at i^{th} step *(depending on the phase reached, could be feature list, requirements, design units, LOC, specification units, FPs, etc.)*

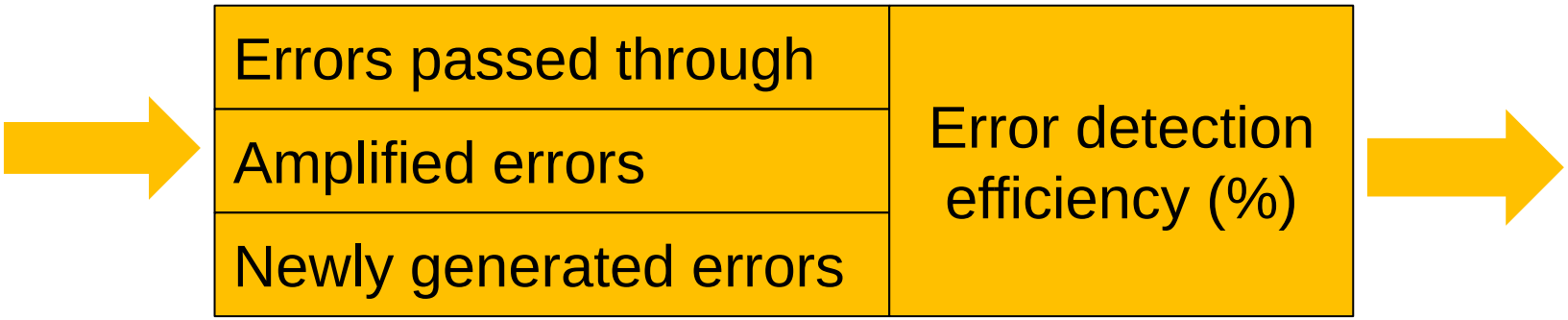
An example follows after the next slide…

Ernest Cachia
Department of Computer Science

Defect Amplification

This is when a defect in a development phase is not detected and therefore "amplifies" its negative effect on the product in subsequent phases. This effect can be modelled using what is called a "Defect Amplification Model" (DAM) *– Developed by IBM in 1981.*

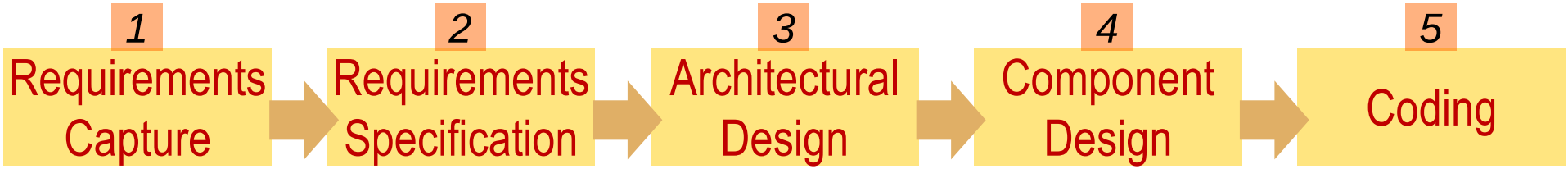
DAM charts are built from chains of nodes like the one shown here:



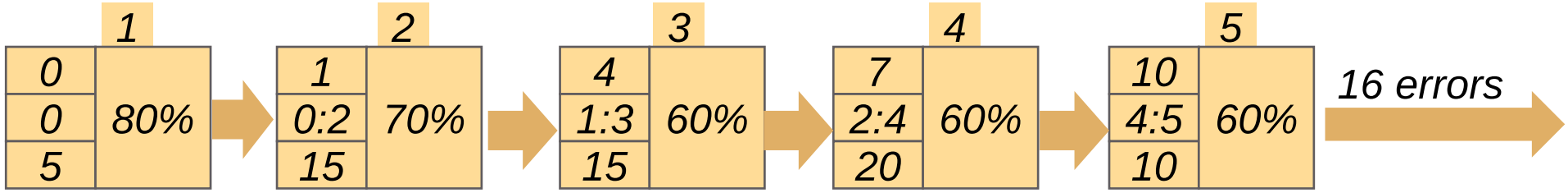
Ernest Cachia
Department of Computer Science

EI Calculation example *(1/3)*

For the sake of this example, let the following development process be assumed.



Assuming 10% serious, 50% moderate and 40% trivial errors after every phase. Also assuming the following error-flow chart...



Error counts in the DAM chart are rounded.

…will result in the following… *(see next slide)*

Ernest Cachia
Department of Computer Science

EI Calculation example *(2/3)*

Error breakdown after each phase (from previous DAM chart and using error severity percentages from previous slide):

From 1: Requirements capture

1 error: 0 serious / 1 moderate / 0 trivial

From 2: Requirements specification

4.8 errors: 0.48 serious / 2.4 moderate / 1.92 trivial

From 3: Architectural design

8.8 errors: 0.88 serious / 4.4 moderate / 3.52 trivial

From 4: Component design

14 errors: 1.4 serious / 7 moderate / 5.6 trivial

From 5: Coding

16 errors: 1.6 serious / 8 moderate / 6.4 trivial

Ernest Cachia
Department of Computer Science

EI Calculation example *(3/3)*

Now apply the relationships from slide 21 to compute EI from the various PIs:

$$PI_1 = 10(0/1) + 3(1/1) + 1(0/1) = 3$$

$$PI_2 = 10(0.48/4.8) + 3(2.4/4.8) + 1(1.92/4.8) = 2.9$$

$$PI_3 = 10(0.88/8.8) + 3(4.4/8.8) + 1(3.52/8.8) = 2.9$$

$$PI_4 = 10(1.4/14) + 3(7/14) + 1(5.6/14) = 2.9$$

$$PI_5 = 10(1.6/16) + 3(8/16) + 1(6.4/16) = 2.9$$

Now if we had to assume a product size (PS) of 50 KLOC:

$$EI = (3 + 2 \times 2.9 + 3 \times 2.9 + 4 \times 2.9 + 5 \times 2.9)/100 = 43.6/50$$

$$= \mathbf{0.872}\ KLOC^{-1}$$ *(i.e. statistically this many errors per thousand LOC)*

Ernest Cachia
Department of Computer Science

Reliability

Meaning:

An attribute of any system that consistently produces the same results, preferably meeting or exceeding its specifications. - *The Free On-line Dictionary of Computing*. Retrieved April 26, 2008, from Dictionary.com website.

The probability of failure free operation in a specified environment for a specified time. – *John D. Musa*

Practically speaking, assume that program “P” has a reliability value of 0.95 during 10 hours of operation. This would mean that, if “P” is executed 100 times and each time it is executed it runs for 10 hours, then “P” is likely to fail in some way 5 times (i.e. 0.05 times per execution).

Ernest Cachia
Department of Computer Science

Availability

Closely related to reliability

Meaning:
The degree to which a system suffers degradation or interruption in its service to the customer as a consequence of failures of one or more of its parts. - *The Free On-line Dictionary of Computing*. Retrieved April 26, 2008, from Dictionary.com website.

Practically speaking, assume that program “P” is likely to encounter some sort of failure on average once every 100 hours of operation. This would mean that if “P” is executed 10 times each time running for 5 hours, then the chances of finding the system available throughout the executions is 99.5% (i.e. 99.95% per execution).

Ernest Cachia
Department of Computer Science

The Meaning of Reliability-Related Measures

- MTTF (Mean Time To Failure)

  The average time taken from the start of observation to successive failures

- MTTR (Mean Time To Repair)

  The average time between successive successful repair actions

- MTBF (Mean Time Between Failures)

  The average time between successive failures

Ernest Cachia
Department of Computer Science

Basic Reliability and Availability Metrics

MTBF = MTTF + MTTR

(i.e. the time it works well + the time taken to fix it)

Availability = MTTF/(MTTF + MTTR) X 100%

Reliability = (MTTF + MTTR)/(MTTF + MTTR + 1)

or... = MTBF/(MTBF + 1)

Ernest Cachia
Department of Computer Science

Combining Reliability

This is a study of the way reliability changes when components (or systems) are brought to work together. There are two ways in which components can be combined:

Serial combination

In this case reliability decreases: $R(t) = \prod_{i=1}^{N} R_i(t)$

e.g. $R_{sys1} = 0.95$; $R_{sys2} = 0.88 \Rightarrow R_{sys} = 0.95 \times 0.88 = 0.836$

Parallel (redundant) combination

In this case reliability increases: $R(t) = 1 - \prod_{i=1}^{N} [1 - R_i(t)]$

e.g. $R_{sys1} = 0.95$; $R_{sys2} = 0.88 \Rightarrow R_{sys} = 1 - [(1 - 0.95) \times (1 - 0.88)] = 0.994$

Ernest Cachia
Department of Computer Science

Activity “CSA3170-E”

Look up and describe two variations on the original FP technique.

1) The “Mark II” FPs
2) The “3D” FPs

You should only write what they are, what prompted their inception, how they differ from the original FP technique, and what they are mainly used for.

Ernest Cachia
Department of Computer Science

Summary (session 8)

- An introduction to Function Points (FPs)
- How to calculate FPs
- A practical example in determining the FP count of a system
- Statistical Quality Analysis (SQA) and error measurement
- Introduction to what reliability and availability are
- Some basic metrics to estimate reliability and availability

Ernest Cachia
Department of Computer Science

General System Characteristics

1. Data Communications
2. Distributed Data Processing
3. Performance
4. Heavily Used Configuration
5. Transaction Rate
6. Online Data Entry
7. End-User Efficiency
8. Online Update
9. Complex Processing
10. Reusability
11. Installation Ease
12. Operational Ease
13. Multiple Sites
14. Facilitate Change

Allocate a number between 0 and 5 to each of the 14 characteristics shown on the left. “0” means not important or relevant to the system, and “5” means very important or critical for the system.

Back to originating slide

Ernest Cachia
Department of Computer Science