Session 5

Software Maintenance and Process Scheduling

In this session we look at some fundamental concepts behind software maintenance which highlight its importance as a major component of software development. We then move on to process control through the use of task scheduling and its modelling.

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

Common Views on Maintenance

Some basic *mis*conceptions 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

The Truth Be Told…

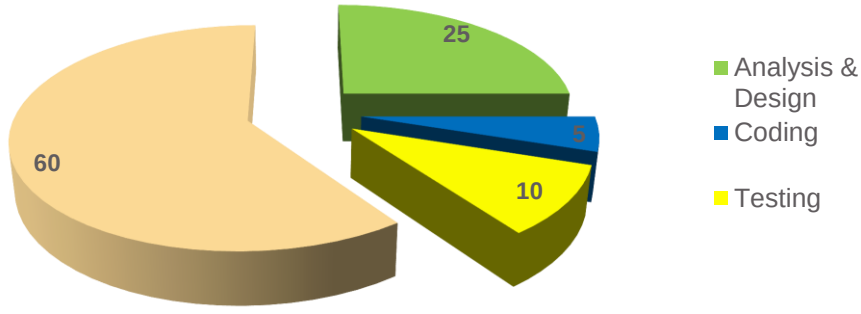
The truth about maintenance in the modern system development process:

- 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

Software Development Effort *(as a percentage)*



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

Highlighting the Importance of Maintenance

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

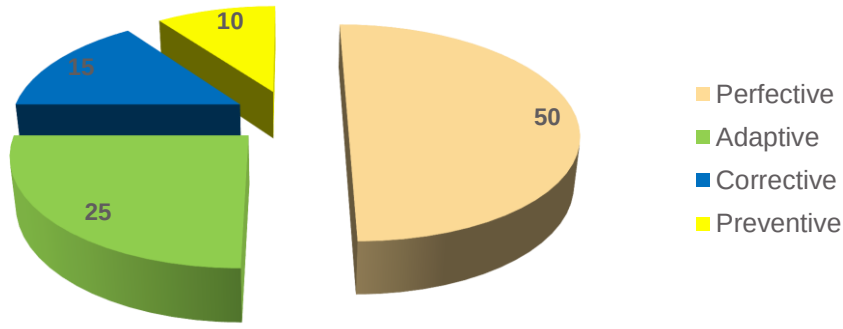
- Link solution objectives to organisational goals
- Link software maintenance rewards to organisational performance
- Make software members of operational teams take turns at maintenance – create no distinction of roles
- Allow adequate budget and a good degree of independence within teams handling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

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

Maintenance Categories

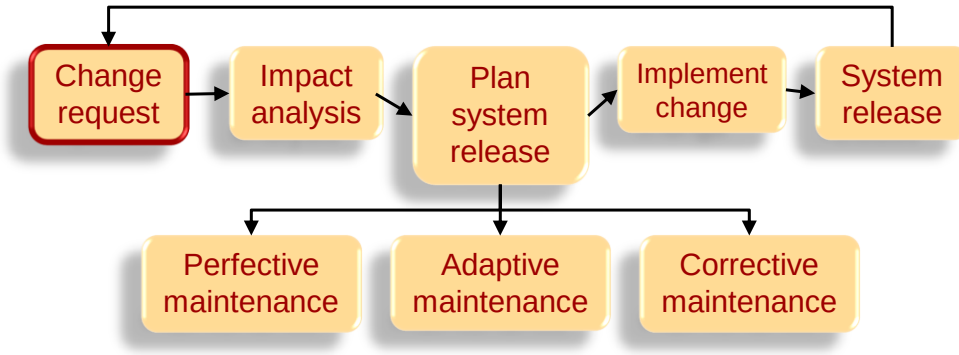
Maintenance by Type *(as a percentage)*



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

A Maintenance Process Example

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



Taken from Ian Sommerville

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).

THE RESULTS SHOULD BE IDENTICAL.

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.

Maintenance Cost

Technical factors effecting maintenance cost

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

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)*

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

Beside the point: **After this, AME can be used as effort input to the Intermediate COCOMO-1 method.**

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):

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

A possible problem to this approach *(Sommerville)***:**

What would the ACT value for new systems be?

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

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)

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.

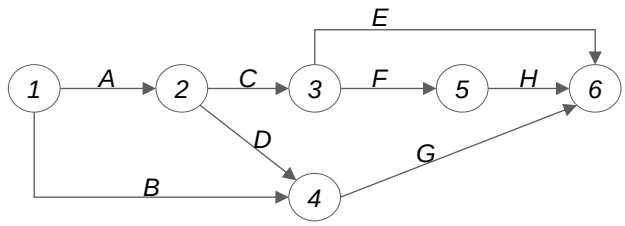
A prime example of such (AOA) diagrams is the Project Evaluation and Review Technique (or PERT) chart.

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

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)

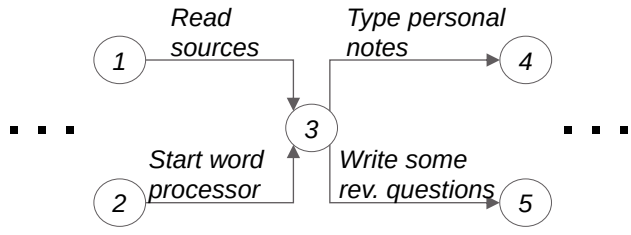
AOA Chart Example *(1/3)*



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

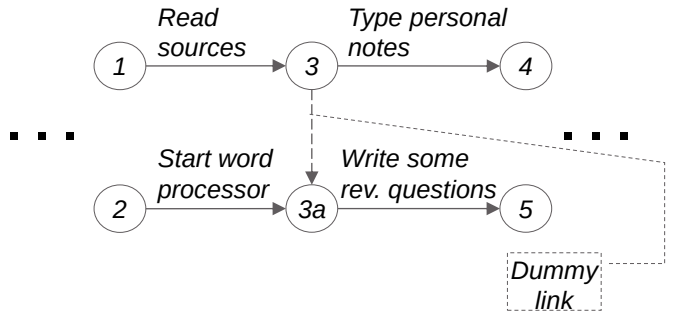
AOA Chart Example *(2/3)*



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…

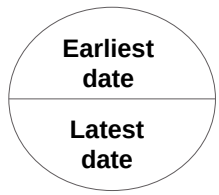
Ernest Cachia
Department of Computer Science

AOA Chart Example *(3/3)*



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

PERT Chart Nodes



PERT Chart (milestone) node

Activity ID and duration →

PERT Chart activity

PERT Chart Example *(1/2)*

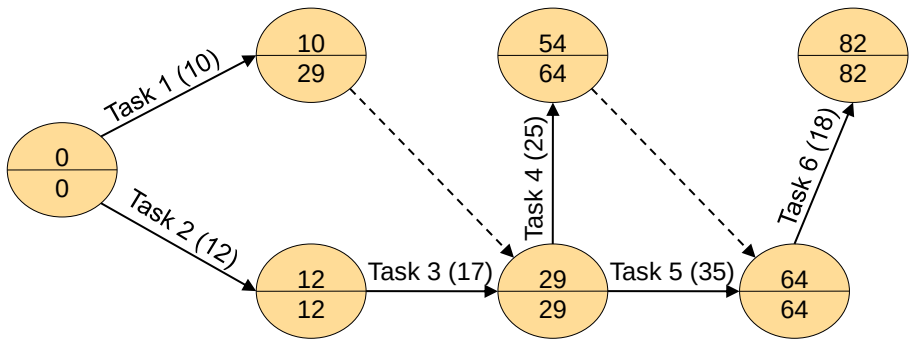
Let us take the table below, representing various activities in a hypothetical project, as an example.

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)*

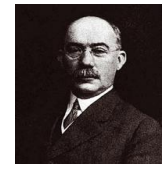


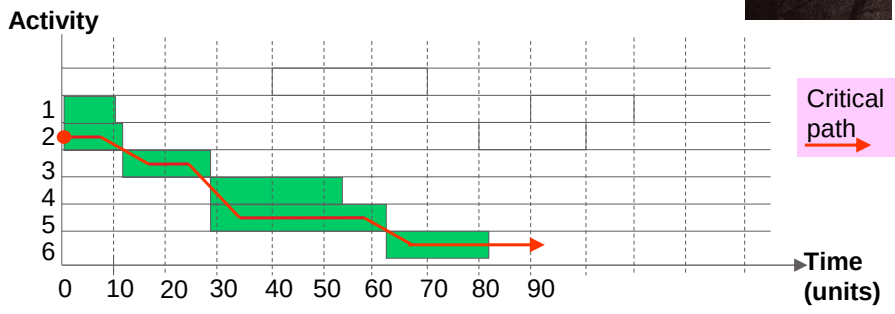
The “critical path” is the one that contains activities that would cause project delay on the whole had they to be delayed themselves.
In this example: Tasks 2, 3, 5, and 6.

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 5)

- 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