Session 4

Software Engineering Measurement and Fundamental Estimation Techniques.

Ernest Cachia
Department of Computer Science

Session Aims

The main aim of this session is to introduce you to the notion of measurement of software production as well as a few fundamental estimation techniques.

- To drive home the crucial necessity of measurement
- To create appreciation for the notion of software product measurement
- To appreciate the idea of internal and external metrics
- To present students with an overview of some fundamental estimation techniques

Ernest Cachia
Department of Computer Science

Session Contents

- Basic Measurement Theory
- Software Measurement
- Software Cost and Effort Estimation Approaches

The Meaning of Quality

In general:

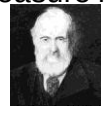
Measurement is the process by which numbers or symbols are assigned to attributes of entities in the world according to clearly defined rules.

On the importance of measurement:

"To measure is to know."

"If you can not measure it, you cannot improve it."

(Sir William Thomson, Lord Kelvin)



Importance of Measurement

- Measurement is what turns an “art” into a “science”
- Measurement is crucial to the progress of all sciences, even Software Solutions.
- Scientific progress is made through
  - Observations and generalisations
  - Data and measurements
  - Derivation of theories
  - Confirmation or refutation of these theories

ALL THE ABOVE FOUR POINTS ARE BASED ON SOME FORM OF MEASUSREMENT

Uses of Measurement

- Measurement helps us understand
  - Makes the current activity visible
  - Measures established guidelines
- Measurement allows us control
  - Predict outcomes and change processes
- Measurement encourages us improve
  - When we hold our product up to some form of measuring stick, we can establish quality targets and aim to improve

Levels of Measurement

Various scales of measurements exist:

- **Nominal Scale**
  By identifier: e.g. British, South African, Lebanese, etc
- **Ordinal Scale**
  By category & rank: e.g. 1st Class, 2nd Class, 3rd Class, etc
- **Interval Scale**
  By relative position on a scale: e.g. Football teams' position in a league table
- **Ratio Scale**
  Relative to an absolute (axiomatic) value on an interval scale: e.g. probability of failure

Measures, Metrics and Indicators

- **Measure** – An appraisal or ascertainment by comparing to a standard (e.g. Joe's body temperature is 37.2° Celsius).
- **Metric** – A quantitative measure of the degree to which an element corresponds to a given attribute (e.g. 2 errors were discovered by customers in 18 months. This is more meaningful than simply saying that 2 errors were found).
- **Indicator** – A device, variable or metric that can indicate whether a particular state or goal has been achieved. Usually used to draw someone's attention to something (e.g. A half-mast flag indicates that someone has died, a red light is a warning, etc)

Examples of Software Measures & Metrics

Measurements which relate to a software product, and/or a software engineering process, for example:

- the size of a program in lines of code;
- the number of reported errors;
- the number of person-days required;
- the "Gunning Fog Index" of a product manual, and others

Adequate measures are essential to the production of a cost effective, timely and quality product.

The 3 "P"s of Software Measurement

With regards to software, one can measure:

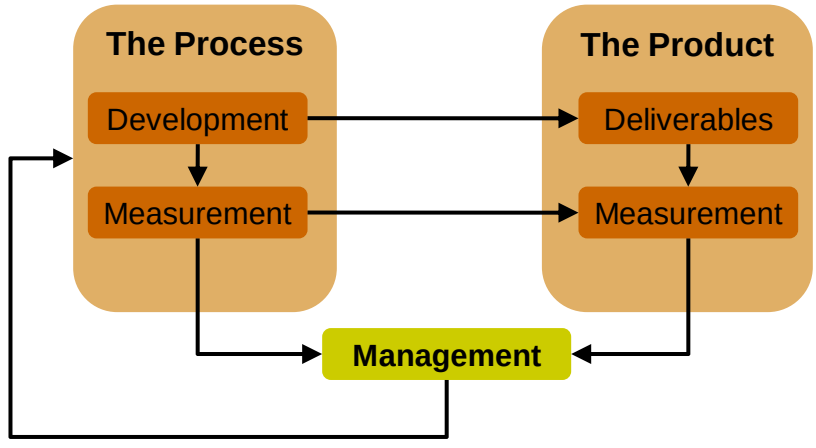
- **P**roduct
- **P**rocess
- **P**eople

Software Project Measurement

A software project is made up of all three “P”s. In this context, measurement…

- Is an integral part of Quality Assurance (QA)
- Is an important aspect of project planning
- Does not take into account other (non-S/W) system components and related estimations
- Has no fixed SDLC positioning – i.e. can happen in any development phase(s)
- Is a “refine-able” effort – i.e. can be recalculated at various levels of abstraction
- Mainly, but not exclusively, targets *estimation*

Relationship between Process & Product Metrics



The Relation Between External & Internal Metrics

$$F_{ex} = m_1 c_1 + m_2 c_2 + \ldots + m_n c_n$$

Where:

- F_{ex} is a meaningful value of an external factor;
- M_i is a constant representing the relative importance of internal attribute *i* (i.e. the weighting);
- C_i is the value of internal attribute *i.*

Example in Determining an External Quality Factor

Internal values:

- Code size = 100LOC
- Number of unique variables = 30 vars
- Number of unique operators = 10 ops
- Cyclomatic number = 6 (no units)

External value to gauge effort to build (weights 1-5):

If emphasis is on size:

Effort = 100x5 + 30x3 + 10x1 + 6x1 = **606** units

If emphasis is on complexity:

Effort = 100x1 + 30x3 + 10x3 + 6x5 = **250** units

So… What does this mean?

Interpreting the Previous Example

In the previous example, when importance was given to the size of the system, more effort was estimated to be needed. This because there was considerable size (100LOC). However, when importance was put mainly on the complexity of the system, not as much effort was estimated to be needed. This because the system does not exhibit high complexity (only 6).

Examples of Measurable Internal Quality Factors *(1/2)*

- Access audit

  The ease with which software and data can be checked for compliance with standards or other requirements.
- Access control

  The provisions for control and protection of the software and data.
- Completeness

  The degree to which a full implementation of the required functionality has been achieved.
- Consistency

  The use of uniform design and implementation techniques and notations throughout a project.

Examples of Measurable Internal Quality Factors *(2/2)*

- Generality
  The breadth of the potential application of software components.
- Operability
  The ease of operation of the software.
- Simplicity
  The ease with which the software can be understood.
- Traceability
  The ability to link software components to requirements.
- Training
  The ease with which new users can be made to use the system.

Activity "CSA3170-4"

For this activity, you are to look up, report and comment on the relationship that McCall's proposed exists between 23 internal quality criteria and 11 external quality factors. You should represent these in the form of a table, and give a two-sentence explanation wherever there is a relationship.

You are also asked to look up how to calculate McCabe's Cyclomatic Complexity value for a system (or piece of code). Explain how this is done.

Most Common Estimation Points

- The Cost
- The Effort

Costs to Estimate

- Software development (given highest priority)
- Hardware usage
- Development staff
- Support staff
- Premises, communication and services
- Training
- Travel

Software Development Cost Estimation Approaches

- Expert judgement
- By analogy
- Algorithmic Cost Modelling
- Parkinson's Law
- Pricing "to win"

(Compiled from a combination of Sommerville & Van Vliet)

All the above approaches can be carried out as either:

- Top-down estimation, or
- Bottom-up estimation

Expert Judgement Approach

- Delphi method
  - Moderator mediated redistribution of estimates amongst experts until agreement is reached
- ORP method
  - Multiple estimates from experts as "optimistic", "real" and "pessimistic". These are then used in various relationships as required, e.g. average, (o+4r+p)/6, etc.

Estimation by Analogy

- Based on past completed similar projects
- Can be adopted in full or partially
- *Pros:*
  - Minimal effort
  - Rooted in practice
- *Cons:*
  - Tends towards the pessimistic
  - "Technology shift" prone

Algorithmic Cost Modelling *(1/2)*

- Use of empirically obtained data in set formulae which relate this data to specific s/w metrics such as size or function points
- One of the most commonly used approaches due to its integration of both the practical and theoretical estimation aspects

The COnstructive COst MOdel (COCOMO) is the most widespread example using this approach. However, other models such as Walston-Felix, DeMarco, and Putnam do exist.

Algorithmic Cost Modelling *(2/2)*

Problems of algorithmic cost modelling

(as reported in Sommerville and in Van Vliet, with some adaptation)

- Could sometimes use values which are difficult to obtain
- These values can be subjective
- Assumes a sequential SDLC
- Do not cater for advances in development and project reasoning
- Do not support product maintenance

Parkinson's Law

States that:

"Work expands to fill the time available"

Therefore…

1. Check your available resources
2. Cost your available resources
3. Assume their max usage
4. Estimate the project cost accordingly

Pricing “to win”

- In other words, providing a unjustified (blind) estimate solely based on either other competitors’ estimates or on the client’s perceived or real available budget, or on both.
- Invariably results in products of inferior quality (or, very often, no product at all)

It is worth noting at this point, that in the understanding of modern business communities and understanding, this approach can constitute unethical (even prosecutable) behaviour.

Summary (Session 4)

- Some basic theory of measurement
- Internal and external measurement and their relationship
- Examples of internal qualities that can be directly measured
- Activity “SEM6” to compare McCall’s and ISO2126 quality classifications
- Estimation approaches (expert judgement, analogy, algorithmic, Parkinson’s, and “pricing to win”)

The Gunning Fog Index

The **Fog Index** is a readability test designed to show how easy or difficult a text is to read. It uses the following formula:

```
Reading Level (Grade) = (Average No. of words in
sentences + Percentage of words of three or more
syllables) x 0.4
```

The resulting number is your Gunning Fog Index.

The Gunning Fog Index gives the number of years of education that your reader hypothetically needs to understand the paragraph or text. The Gunning Fog Index formula implies that short sentences written in plain English achieve a better score than long sentences written in complicated language.

(Reproduced from http://www.usingenglish.com)

Back to calling slide

Session 5

COCOMO – An Algorithmic Estimation Model

Session Aims

The main aim of this session is to introduce you to a modern and widely used cost and effort estimation method known as COnstructive COst MOdel, or COCOMO. A brief overview of Object Points is also included.

- To cement the link between the theoretical principles in session 6 with a practical method
- To impart some appreciation of the use that COCOMO can be put to
- To clearly define the various levels and structure of COCOMO
- To appreciate the measurement improvement that Object Points offer

Session Contents

- COCOMO-1
- COCOMO-2
- Object Points
- Just a mention of Function Points

COCOMO

COCOMO (which stands for COnstructive COst MOdel) is a cost and effort estimation technique. It involves a set of relationships based on:

- The type of system being developed
- The level of abstraction at which it is being developed
- The importance attributed to various quality factors in that system

Boehm's COCOMO

- First generation COCOMO
  (aka COCOMO-1)
- Second generation COCOMO
  (aka COCOMO-2)

Many tools supporting various estimation models of COCOMO-1 and -2. However, an application called "Costar" (now in v7.0) is the only one that supports all COCOMO-1 and -2 models.

Levels of COCOMO-1

- Basic
  - Based on project type *(see next slide)*
- Intermediate
  - Introduces refining (called “improving”) factors based on four project-related aspects
- Detailed
  - Uses changing refining factors depending on the stage of development as well as system structural modularity

COCOMO-1 (Basic Level) Project Types

An estimation of “crude” effort

Boehm’s project class types:

- Organic
  (i.e. of a “familiar” nature, uses small teams)
- Semi-detached
  (i.e. contains some novelty, uses larger teams)
- Embedded
  (i.e. part of a highly sophisticated system)

Basic COCOMO-1 Formulae

Organic

$E_{crude} = 2.4(KDSI)^{1.05}$

Semi-detached

$E_{crude} = 3(KDSI)^{1.12}$

Embedded

$E_{crude} = 3.6(KDSI)^{1.2}$

KDSI stands for "Thousands of Delivered Source Instructions"

Intermediate COCOMO-1

Introduces four COCOMO aspects and attribute sets as follows:

- Product attributes
  Determined by the type of s/w in development
- Computer attributes
  Determined by the h/w on which the developed s/w is to run
- Personnel attributes
  Determined by the expertise and skill of the people developing the s/w
- Project attributes
  Determined by project-wide tools, techniques and schedules

(source: Boehm)

Values for Intermediate COCOMO-1 Attribute Sets

For your perusal, these 15 values are available as a series of additional slides on-line from the web-site.

Example of what to expect

Product Attributes

- Required reliability
  - Range: VL-L-N-H-VH
  - Values: 0.75/0.88/1/1.15/1.4
- Database size
  - Range: L-N-H-VH
  - Values: 0.94/1/1.08/1.16
- Product complexity
  - Range: VL-L-N-H-VH-EH
  - Values: 0.7/0.85/1/1.15/1.3/1.65

…etc…

Intermediate COCOMO-1 Formula

$$E_{improved} = E_{crude} * m1 * m2 * m3 * \ldots * m15$$

m_i are the 15 COCOMO aspects as detailed in the separate slides (as indicated in the previous slide).

Detailed COCOMO-1

This is the use/application of different and more targeted COCOMO attribute value tables (like the generic one discussed in Intermediate COCOMO-1) at the various stages of an SDLC.

Please note, that examples of this level of COCOMO-1 are too length to consider in lectures. Furthermore, their mechanism is simply an extension of the (already covered) Intermediate COCOMO-1 technique.

COCOMO-1 Example

Consider the situation where the basic COCOMO model predicts an effort of 20 programmer-months to develop a software system. The effort multipliers for the intermediate COCOMO model all have nominal values except for:

- ACAP (analyst capability) which is 1.19;
- PCAP (programmer capability) which is 1.17.

A proposal was made to allocate more experienced staff to this project (to reduce ACAP and PCAP to nominal values). Is this proposal justifiable given that the costs of using inexperienced staff amount to €1000 per person per month, and the cost of more experienced staff is €250 per person per month higher?

Solution:

Option 1 – Use less experienced staff (i.e. pay for their "handicap")

$E_{improved}$ = 20pm * 1.19 * 1.17 = 27.85pm (where "pm" is person-months)

Cost = 27.85 * €1000 = €27850

Option 2 – Use more experienced staff (i.e. pay them more)

$E_{improved}$ = 20pm (doesn't change, all other attributes have nominal values, i.e. 1)

Cost = 20 * €1250 = €25000

Therefore, the second option would be favoured.

Time Estimation Using COCOMO-1

TDEV is to be taken as project development time. Effort (*E*) is calculated as previously described

Organic:

$TDEV = 2.5(E)^{0.38}$

Semi-detached:

$TDEV = 2.5(E)^{0.35}$

Embedded:

$TDEV = 2.5(E)^{0.32}$

Time Calculation Example

In the previous COCOMO-1 worked example, the option chosen was the one which would take 20pm. Now, assuming that this is a "semi-detached" solution (see slide 7 for meaning), the formula TDEV = $2.5(E)^{0.35}$ would apply.

Therefore:

$$TDEV = 2.5x20^{0.35} = \mathbf{7.13pm}$$

Meaning that it would take approximately 7 months and 4 days of a person's time to build.

COCOMO-2

Its difference from COCOM-1 is:

- Its higher level of abstraction, i.e. it moves away from KDSI
- Uses Object Points and Function Points instead of KDSI
- Considers CASE (Computer Aided Software Engineering) tool usage, reuse practices and iterative development models, amongst other issues

Stages of COCOMO-2

- Early Prototyping

  The initial development stages when the system is not tied to any particular development paradigm
- Early Design

  The initial system structuring stages when system starts to assume basic issues about the way it will be written. It includes a further refinement by using **7 composite cost drivers** *(see next slide)*
- Post Architecture

  This deals with the actual system code and therefore uses KDSI as in COCOMO-1. It includes a further refinement by using **17 productivity attributes** *(see next slide)*

Composite Cost Drivers & Post-Arch. Prod. Attrib. for COCOMO-2

As in the case of the COCOMO-1 Attribute Sets, these COCOMO-2 Post-Architecture Productivity Attributes are also available on-line as a series of additional slides on the web-site.

COCOMO-2: Early Prototyping Stage

Deals with the calculation of effort in terms of "person-months" (PM)

$$PM = (NOP * (1 – reuse\%/100)) / PROD$$

NOP: Number of Object Points *(see next slide)*

reuse%: percentage of reused (i.e. not developed) code

PROD: A productivity value *(see slide #53)*

Object Points

- Try to overcome the intrinsic problems of measuring in terms of Lines Of Code (LOC)
- Looks a system from an "available component" point of view
- Objects (i.e. system components) taken into account are:
  - Screens
  - Reports
  - 3rd Generation Language (3GL) modules

Object Point Counts

	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL module			10

Object Point Example

A system offers a user one form of average sophistication from which to input data and 2 different simple output forms. The system can provide 2 types of report. One report is very intricate, the other is straightforward. The system also allows the user to configure its settings through a separate basic interface. The system carries out 3 distinct functions and each function uses 2 software modules. What would the Object Point count of such a system be?

OP_{scr} = (1scr x 2) + (2scr x 1) + (1scr x 1) = 5

OP_{rep} = (1rep x 8) + (1rep x 2) = 10

OP_{mod} = (6mod x 10) = 60

$OP_{sys} = OP_{scr} + OP_{rep} + OP_{mod}$ = 75 object points

COCOMO-2 Productivity Values

As suggested by Boehm in Ian Sommerville's Software Engineering textbook

- Developer experience and capability
- CASE maturity and capability

Range: VL | L | N | H | VH

PROD = 4 | 7 | 13 | 25 | 50

(back to original COCOMO-2 slide)

COCOMO-2: Early Design Stage

Estimation on the basis of automated and manual code production can now be carried out

$$PM = PM_{man} + PM_{auto}$$

PM_{auto}: Is tool-dependent and is calculated independently

$$PM_{man} = \alpha * size^{\beta} * m$$

$\boldsymbol{\alpha}$ is organisation-dependent (Boehm suggests 2.5. This is only loosely empirical and partially anecdotal)

$\boldsymbol{\beta}$ is a coefficient relating effort to project size. It is not fixed (as in COCOMO-1) and varies between 1.1 and 1.24

m is a multiplier based on the 7 composite cost drivers

size is usually calculated as FPs (next slide) and then converted to KDSI

Just a Mention of Function points

Function Points (FPs) will be tackled in a later lecture. However, they will be introduced at this stage:

- They are a further elaboration of Object Point concepts
- They are a measure intended to gauge function rather than size or components
- They are more relevant to user needs (i.e. externally visible)
- They require good insight into the system to be accurate
- They are quite a popular measurement basis

COCOMO-2: Post-Architecture Stage

Same formula as was used in the Early Design stage. However, the 17 productivity attributes are used instead of the 7 composite cost drivers.

At this stage KDSI is adjusted taking into account:

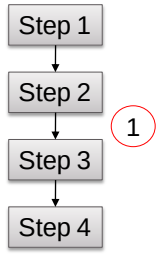
- Code reuse
- Requirements volatility

An Algorithmic Complexity Measurement

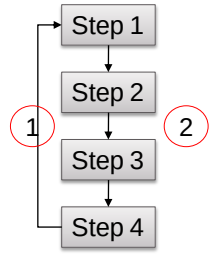
"McCabe's Cyclomatic Complexity" value

- Is relatively simple to calculate
- Is indicative of intra-modular logic
- Is based on code control graphs

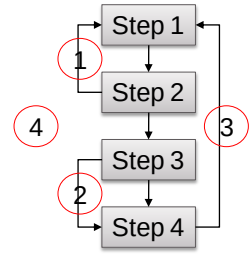
Which of these three algorithms is more complex? **THE THIRD ONE!**



Cyclo. = 1



Cyclo. = 2



Cyclo. = 4

Ernest Cachia
Department of Computer Science

Activity "CSA3170-5"

Look up and describe two different types of metrics:

1) Halstead's "Software Science" metrics (or suite of measurements)
2) The "Chidamber-Kemerer " (CK) metrics.

You should only write what they are, what they are used for, and give one simple example of each.

Ernest Cachia
Department of Computer Science

Summary (Session 5)

- An introduction of COCOMO as an estimation method/tool.
- The two generations of COCOMO and the various levels in each generation
- Examples in Intermediate COCOMO-1 (effort, cost, and time)
- COCOMO-2 and Object Points
- Introduction to the notion of Function Points
- McCabe's Cyclomatic Complexity estimation together with an example

Ernest Cachia
Department of Computer Science