Chapter 3 - Software Project Management - Estimation, Scheduling and Metrics

White diamond - aggregation (a part of)

Cost Estimation Techniques

- Expert Judgement
- Past Experience
  - Build up a databank of past projects and their cost
- Top down
  - Break the problem up into smaller problems and estimate these
- Function Point analysis
  - Uses the requirement specification to assess inputs, outputs, file accesses, user interactions and interfaces and calculates the size based on these
- Algorithmic Cost Modelling
  - COnstructive Cost MOdelling (COCOMO), Barry Boehm 1981, has been an influential approach

Function Point Analysis (FPA)

Getting an idea of complexity from the quantity of components.

The following *system components* are considered. Number of:

- external inputs (eg, input files of transactions)
- external outputs (eg output files of reports, messages)
- user interactions / enquiries (eg menu selection, queries)
- internal or logical files used by the system
- number of external or interface files shared with other applicatoins

In each case, depending on the number of *field types* associated with each component, and the variety of *file types* that these elements refer to, these components are rated as *simple, average or complex*.

A weight is associated with each of the ratings, *simple average or complex*. The number of external inputs is multiplied by the selected weighting for that system component likewise for external outputs and the other system components.

$$UFP = \Sigma\ Input(w_i) + \Sigma\ Output(w_o) + \Sigma\ Enquiry(w_e) + \Sigma\ Logical\ File(w_l) + \Sigma\ Interface\ File(w_{if})$$

COMPONENT	COMPLEXITY LEVEL		
	simple	average	complex
Input (I)	×3	×4	×6
Output (O)	×4	×5	×7
Enquiry (E)	×3	×4	×6
Internal (Logical) File (L)	×7	×10	×15
Interface File (IF)	×5	×7	×10

The weighted total are added to give the **Unadjusted Function Points.**

UFP is then adjusted to take account of the type of application

- This adjustment is made by multiplying UFP by a **technical complexity factor**.
- As preparation for calculating TCF, fourteen **General System Characteristics** are scored for *Degree of Influence* from 0 to 5 (no influence to strong influence)
    1. Data communications
    2. Distributed functions
    3. Performance
    4. Heavily used configuration
    5. Transaction Rate
    6. On-line data entry
    7. End-user efficiency
    8. On-line update
    9. Complex Processing
    10. Reusability
    11. Installation ease
    12. Operational ease
    13. Multiple sites
    14. Facilitate change

- > TCF is then calculated as `TCF = 0.65 + 0.01 x DI` where DI is the total *Degree of Influence* from the 14 scored characteristics

Function points (FP) are calculated as `FP = UFP * TCF`. FPs can be used to estimate Lines of Code (LOC), assuming that the average number of LOC per FP for a given language can be calculated.

But there are some difficulties:

- FPs, and particularly the scores given to the General System Characteristics, are very subjective. They cannot be counted automatically and depend on the analyst's assessment.
- There are only 3 complexity levels for weighting the functionality of the main system components
- The approach has to be calibrated or adjusted for different programming tasks and environments

COCOMO

3 Classes of project are recognised

1. **Simple** (or organic): small teams, familiar environment, well understood applications, no difficult non-functional requirements
2. **Moderate** (or semi-detached): project team may have experience mixture, system may have more significant non-functional constraints, organisation may have less familiarity with application.
3. **Embedded**: HW/SQ systems, tight constraints, unusual for team to have deep application experience.

`COCOMO: E = a x KDSI`^{b}`; D = 2.5 x E`^{c}

- E = Effort in Person-months
- a, b, c = Constants based on project class & historical data
- D = development time in months
- KDSI = Thousands of delivered source instructions (~Lines of code)

The multipliers and exponents for basic COCOMO's formulae change according to the class of the project.

	Variables		
Project Class	a	b	c
Organic	2.4	1.05	0.38
Semi-detached	3.0	1.12	0.35
Embedded	3.6	1.20	0.32

Intermediate COCOMO

Intermediate COCOMO takes the basic COCOMO formula as its starting point. The value *a* is equal to 3.2, 3.0, 2.8, for organic, semi-detached and embedded projects respectively.

Additionally Intermediate COCOMO identifies personnel, product, computer and project attributes which affect cost. With intermediate COCOMO, the basic cost is adjusted by attribute multipliers: presently we'll give values to a sample of these multipliers:

1. Product attributes
    - Required software reliability (RELY)
    - Database Size (DATA)
    - Product Complexity (CPLX)
2. Computer Attributes
    - Execution time constraints (TIME)
    - Storage constraints (STOR)
    - Virtual machine volatility (VIRT)
    - Computer turnaround time (TURN)
3. Personnel Attributes
    - Analyst capability (ACAP)
    - Programmer capability (PCAP)
    - Applications experience (AEXP)
    - Virtual machine experience (VEXP)
    - Programming language experience (LEXP)
4. Project Attributes
    - Modern programming practices (MODP)
    - Software tools (TOOL)
    - Required development schedule (SCED)

These are attributes which were found to be significant in one organisation with a limited project history database. Other attributes may be more significant for other projects and other organisations.

Configuration Management

In an on-going project, all products of the software process have to be managed.

- Specifications
- Designs
- Programs
- Test data
- User manuals

Thousands of separate documents are generated for a large software system.

CM Plan

- Defines the types of documents to be managed and a document naming scheme
- Defines who takes responsibility for the CM procedures and creation of baselines
- Defines policies for change control and version management
- Defines the CM records which must be maintained

The configuration database

All CM information should be maintained in a configuration database. This should allow for queries. The CM database should preferably be linked to the software being managed. (eg. Who has a particular system version? What platform is required for a particular version? What versions are affected by a change to component X?)

The change management process

- Request change by completing a change request form
- Analyse change request
- if change is valid then
  - assess how change might be implemented
  - asses change cost
  - submit request to change control board
  - if change is accepted then
    - do this until software quality is adequate
      - make changes to software
      - submit changed software for quality approval
    - create new system version
  - else
    - reject change request
- else
  - reject change request

Derivation history

A record of changes applies to a document or code component. It should record, in outline form, the change made, the rationale for the change, who made the change and when it was implemented.

Versions/variants/releases

Version

An instance of a system which is functionally distinct in some way from other system instances

Variant

An instance of a system which is functionally identical but non-functionally distinct from other instances of a system

Release

An instance of a system which is distributed to users outside of the development team

Software Metrics

> “

> When you can measure what you are speaking about and express it in numbers, you know something about it

Allow processes and products to be assessed. Used as indicators for improvement

Quality Metrics

Defect Removal Efficiency

Measures how good quality assurance is. DRE should be 1 (in an ideal situation)

```
DRE = E/(E+D)
```

E = Number of errors before delivery.
D = Number of defects after delivery.

Defects per kLOC

```
C = #defects/kLOC
```

Integrity

A system's ability to withstand attacks (including accidental ones) on security

```
I = sigma[1 - threat x (1 - security)]
```

Threat = probability than an attack of a particular type will occur at a given time
Security = probability that an attack of that type will be repelled

Design Complexity

'fan-in and fan-out' in a structure chart. High fan-in = high coupling. High fan-out = high coupling (complexity). Length is any measure of program size such as LOC.

$$Complexity = Length * (fan\text{-}in * fan\text{-}out)^2$$

Reliability Metrics

Probability of failure on demand

This is a measure of the likelihood that the system will fail when a service request is made. a PoFOD of 0.001 means that 1 out of every 1000 service requests result in failure. This is relevant for safety-critical or non-stop systems. It is computed by measuring the number of system failures for a given number of system inputs.

Rate of occurrence of failures at time *t*

The mean rate of failures per unit time at time `t`. A RoCOF of 0.02 means 2 failures are likely in each 100 operational time units

Revision #3
Created 1 year ago by Christopher Wilkinson (L2)
Updated 1 year ago by Christopher Wilkinson (L2)