METRICI SOFTWARE

- capture notions of size and complexity

Size metrics

- Ideally, we want to define a set of attributes for software size analogous to human height and weight. That is, we want them to be fundamental so that each attribute captures a key aspect of software size
  - Linii de cod Length (LOC)
  - Functionality
  - Complexitate

LOC

- The most commonly used size indicator.
- Issues:

– How to define LOC?

– How to know LOC before any coding?

– How to interpret LOC? (is 250 more productive than 200?)

- What about different programming languages?

Lines of Code Calculation

•What can be consideredas a line?

• statement

• data definition

• comments?

• Line ≠ debug code?

• Different languages? Assembly and Java

Scaling factor

Beta Distribution Measurement

• Ask developers for the following LOC values:

• Optimistic (O)

• Pessimistic (P)

• Realistic (R)

LOC = (O+P+4*R)/6

• E.g., R=250, O=200, P=400 => LOC=266

The Blitz

• Based on historical observation of previous

projects.

LOC = Processes x prgs/processes x avg. prg size

• Process = class, db table, …

• E.g., 20 classes, 1 program/class, 50 Java LOC/prg

=> LOC = 20*1*50 = 1000

Functionality

- Many software engineers argue that length is misleading and that the amount of functionality in an inherent product paints a better picture of product size
- In particular, those who generate effort and duration estimates from early development often prefer to estimate functionality rather than physical size

Function points (FP)

- Uses the functionality of the software as a measure
- Analogy: For a given house, we can say how many square meters it has (LOC) or we can say how many bedrooms and bathrooms it has (FP).
- The idea was first put forward by Allan Albrecht of IBM in 1979.
- Derived using an empirical relationship based on countable (direct) measures of software's information domain (i.e., things in the external behavior that will require processing)
- A function point count is a measure of the amount of functionality in a product
- Can be estimated early in project before a lot is known

Information domain values

- Number of external inputs (EI)
  - Originates from a user or is transmitted from another application and provides distinct application-oriented data or control information
- Number of external outputs (EO)
  - Derived within the application and provides information to the user
- Number of external inquiries (EQ)
  - Interactive inputs requiring a response
- Number of internal logical files (ILF)
  - A logical grouping of data that resides within the application's boundary and is maintained via external inputs
- Number of external interface files (EIF)
  - A logical grouping of data that resides external to the application but provides data that may be of use to the application

Computing function points: step 1

- Establish count for information domain values
  - Number of external inputs
  - Number of external outputs
  - Number of external inquiries
  - Number of internal logical files
  - Number of external interface files

Computing function points: step 2

- Associate complexity value with each count
  - Simple, Average, Complex
  - Determined based on organisational experience based criteria. Nonetheless, the determination of complexity is somewhat subjective
  - Compute count total

- Computing function points: step 2 (S14)

Computing function points: step 3

- Calculate complexity adjustment values (Fi, where iÎ[1..14])
    - Does the system require reliable backup and recovery?
    - Are data communications required?
    - Are there distributed processing functions?
    - Is performance critical?
    - Will the system run in an existing, heavily utilised operating environment?
    - Does the system require on-line data entry?
    - Does the on-line data entry require the input transaction to be built over multiple screens or operations?
    - Are the master files updated on-line?
    - Are the inputs, outputs, files or inquiries complex?
    - Is the internal processing complex?
    - Is the code designed to be reusable?
    - Are conversion and installation included in the design?
    - Is the system designed for multiple installations in different organisations?
    - Is the application designed to facilitate change and ease of use by the user?
- Answer each question using a scale of 0 (N/A) to 5 (absolutely essential)
- Sum the 14 complexity adjustment values SumFi

  Computing function points: step 4

- Calculate ec

ecuatie (s18)

- 0.65 and 0.01 are empirically derived constants

Function point (FP): Advantages

- Programming language independent
- Used readily countable characteristics that are determined early in the software process
- Does not “penalize” inventive (short) implementations that use fewer LOC than other more clumsy versions

Comparing LOC and FP

- LOC and FP (S20)
- LOC and FP-based metrics have been found to be relatively accurate predictors of software development effort and cost
- Typical metrics
  - $ per LOC, $ per FP
  - LOC per person-month, FP per person-month
  - errors per KLOC, errors per FP
  - pages of documentation per KLOC, pages of documentation per FP

Complexity metrics

- Numerous metrics have been proposed for measuring program complexity – probably more than for any other program characteristic
- Complexity can be interpreted in different ways, depending on our perspective
  - Problem complexity (also called computational complexity) measures the complexity of the underlying problem
  - Algorithmic complexity reflects the complexity of the algorithm implemented to solve the problem
  - Structural complexity measures the structure of the software used to implement the algorithm
  - Cognitive complexity measures the effort required to understand the software

- Our intuition tells us that the complexity of a program can affect the time it takes to code it, test it, and fix it
- We suspect that complexity can help us to understand when a module is prone to contain faults
- We often assume that we know what complexity is, and we measure complexity without first defining it in the real world
- Many software developers define program complexity as the cyclomatic number proposed by McCabe

Cyclomatic complexity

- Cyclomatic complexity is a software metric that provides a quantitative measure of the logical complexity of a program
- It was proposed by Tom McCabe in 1976
- This number, based on a graph-theoretic concept, counts the number of linearly independent paths through a program (we have seen this in the context of testing with basis path testing)
- McCabe felt that the number of such paths was a key indicator not just of testability but also of complexity

- A number of industry studies have indicated that the higher the cyclomatic complexity of a module is, the higher the probability of errors
- In one experiment, modules with cyclomatic complexity > 10 were shown to have statistically more errors
- If there are lots of modules to test and it is not feasible to test all of them, concentrate on the ones with highest cyclomatic complexity values, they are the ones that are most likely to be error prone

actual cases

- [Grady, 1994] reported a study at Hewlett-Packard where cyclomatic number was computed for each module of 850,000 lines of FORTRAN code
- A close relationship between a module’s cyclomatic complexity number and the number of updates required was discovered

- The study team concluded that 15 should be the maximum cyclomatic number allowed in a module
- The quality assurance procedure for the software in the Channel Tunnel rail requires that a module be rejected if its cyclomatic number is greater than 20, or if it has more than 50 statements [Bennett, 1994]

- The cyclomatic number presents only a partial view of complexity
  - There are many programs that have a large number of decisions but are easy to understand, code, and maintain
  - Relying only on the cyclomatic number to measure actual program complexity can be misleading
- Current state of software metrics
- Even in the case of widely studied metrics (e.g., LOC, Halstead's metrics, McCabe's cyclomatic complexity), not universally agreed what they measure
- Difficult to interpret and compare quoted metric results. For example, if different programming languages are involved, metrics involving LOC values can, if not carefully interpreted lead to incorrect conclusions
  - Do you see any problem(s) with LOC/month as a productivity measure?
    - Could suggest that assembly language programmers are more productive than high-level language programmers
- Despite these problems, if properly used, many software metrics can lead to very useful results

Language

QSM SLOC/FP Data

	Avg	Median	Low	High
ABAP(SAP)	28	18	16	60
ASP	51	54	15	69
ASSEMBLER	119	98	25	320
C	97	99	39	333

ABAP (SAP) * 28 18 16 60

ASP* 51 54 15 69

Assembler * 119 98 25 320

Brio + 14 14 13 16

C * 97 99 39 333

C++ * 50 53 25 80

C# * 54 59 29 70

COBOL * 61 55 23 297

Cognos Impromptu

Scripts + 47 42 30 100

Cross System

Products (CSP) + 20 18 10 38

Cool:Gen/IEF * 32 24 10 82

Datastage 71 65 31 157

Excel * 209 191 131 315

Focus * 43 45 45 45

FoxPro 36 35 34 38

HTML * 34 40 14 48

J2EE * 46 49 15 67

Java * 53 53 14 134

JavaScript * 47 53 31 63

JCL * 62 48 25 221

LINC II 29 30 22 38

Lotus Notes * 23 21 19 40

Natural * 40 34 34 53

.NET * 57 60 53 60

Oracle * 37 40 17 60

PACBASE * 35 32 22 60

Perl * 24 15 15 60

PL/1 * 64 80 16 80

PL/SQL * 37 35 13 60

Powerbuilder * 26 28 7 40

REXX * 77 80 50 80

Sabretalk * 70 66 45 109SAS * 38 37 22 55

Siebel * 59 60 51 60

SLOGAN * 75 75 74 75

SQL * 21 21 13 37

VB.NET * 52 60 26 60

Visual Basic * 42 44 20 60

EX: Find the mean, median, mode, and range for the following list of values:

13, 18, 13, 14, 13, 16, 14, 21, 13

mean: 15
median: 14
mode: 13
range: 8