Software Metrics and Project Management

Ashish Kumar Dwivedi

Product Metrics

- By its nature, engineering is a quantitative discipline.
- Product metrics help software engineers gain insight into the design and construction of the software they build by focusing on specific, measurable attributes of software engineering work products.
- Product metrics that are computed from data collected from the requirements and design models, source code, and test cases.
- Define the metrics to assess the quality of a product.

Measure, Metrics and Indicators

- A *measure* provides a quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of a product or process.
- *Measurement* is the act of determining a measure.
- *Metric* is a quantitative measure of the degree to which a system, component, or process possesses a given attribute.
- E.g., the average number of errors found per unit test.
- A software engineer collects measures and develops metrics so that indicators will be obtained.
- An *indicator* is a metric or combination of metrics that provides insight into the software process, a software project, or the product itself.
- An indicator provides insight that enables the project manager or software engineers to adjust the process, the project, or the product to make things better.

Metrics for the Requirement Model

- Function-Based Metrics (Function Point):
  - *FP metric* can be used effectively as a means for measuring the functionality delivered by a system
  - Using historical data, the FP metric can then be used to:
    - (1) estimate the cost or effort required to design, code, and test the software
    - (2) predict the number of errors that will be encountered during testing; and
    - (3) forecast the number of components and/or the number of projected source lines in the implemented system.
  - **Number of external inputs(EIs)**
  - **Number of external outputs (EOs)**
  - **Number of external inquiries (EQs).** An *external inquiry* is defined as an online input that results in the generation of some immediate software response in the form of an online output
  - **Number of internal logical files (ILFs).**
  - **Number of external interface files (EIFs)**

Calculate Function point

$$FP = \text{count total} \times [0.65 + 0.01 \times \Sigma (F_i)]$$

where count total is the sum of all FP entries obtained from Figure

FIGURE 23.1

Computing function points

Information Domain Value	Count		Weighting factor: Simple	Average	Complex		
External Inputs (EIs)		×	3	4	6	=	
External Outputs (EOs)		×	4	5	7	=	
External Inquiries (EQs)		×	3	4	6	=	
Internal Logical Files (ILFs)		×	7	10	15	=	
External Interface Files (EIFs)		×	5	7	10	=	
Count total →							

- Each of the following questions is answered using a scale that ranges from 0 (not important or applicable) to 5 (absolutely essential).
- The constant values in Equation (23.1) and the weighting factors that are applied to information domain counts are determined empirically.

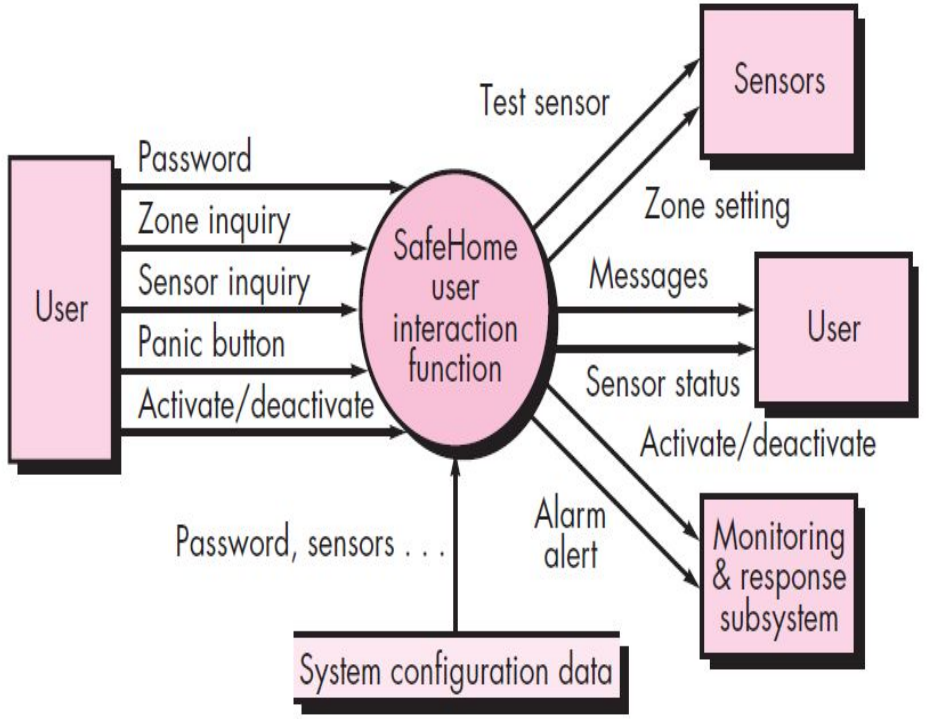
Value adjustment factors are used to provide an indication of problem complexity.

The F_i (i = 1 to 14) are *value adjustment factors* (VAF) based on responses to the following questions [Lon02]:

1. Does the system require reliable backup and recovery?
2. Are specialized data communications required to transfer information to or from the application?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require online data entry?
7. Does the online data entry require the input transaction to be built over multiple screens or operations?
8. Are the ILFs updated online?
9. Are the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?

Example

- Three external inputs—**password, panic button**, and **activate/deactivate**—are shown in the figure along with two external inquiries—**zone inquiry** and **sensor inquiry**.
- One ILF (**system configuration file**) is shown.
- Two external outputs (**messages** and **sensor status**) and four EIFs (**test sensor, zone setting, activate/deactivate,** and **alarm alert**) are also present.

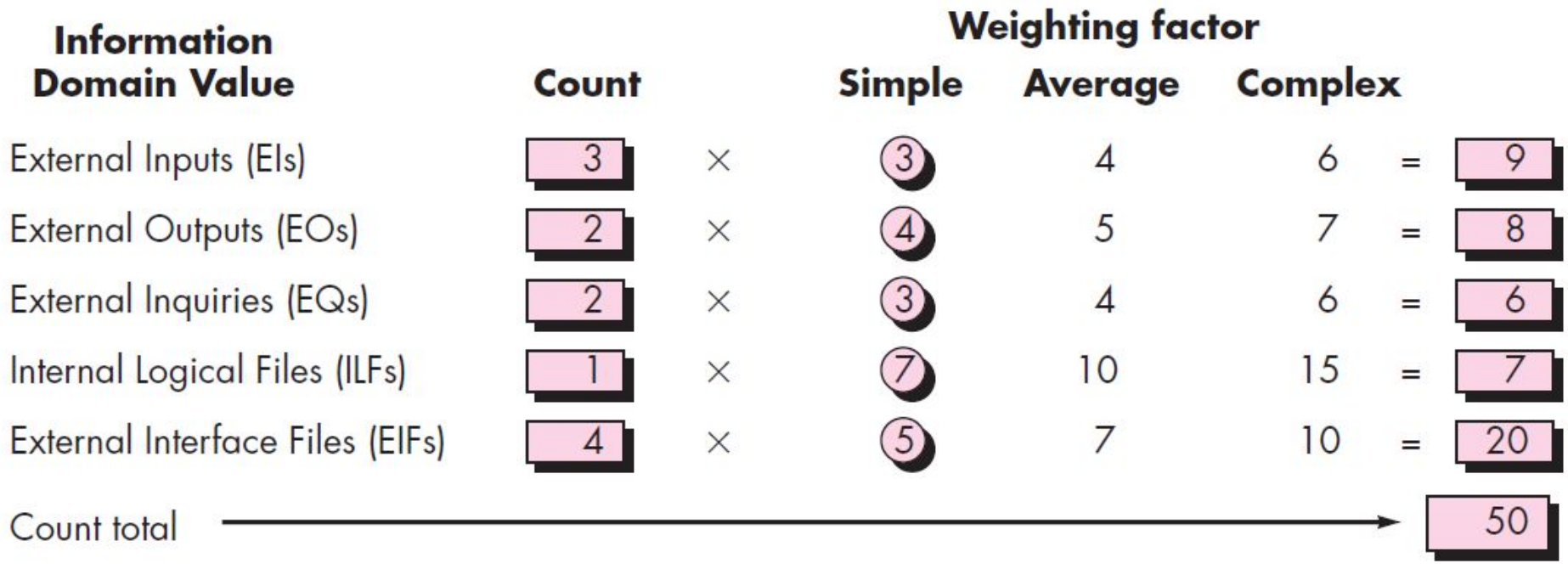


The count total shown in Figure 23.3 must be adjusted using Equation (23.1). For the purposes of this example, we assume that $\Sigma(F_i)$ is 46 (a moderately complex product). Therefore,

$$FP = 50 \times [0.65 + (0.01 \times 46)] = 56$$

FIGURE 23.3

Computing function points



Information Domain Value	Count		Weighting factor: Simple	Average	Complex		
External Inputs (EIs)	3	×	3	4	6	=	9
External Outputs (EOs)	2	×	4	5	7	=	8
External Inquiries (EQs)	2	×	3	4	6	=	6
Internal Logical Files (ILFs)	1	×	7	10	15	=	7
External Interface Files (EIFs)	4	×	5	7	10	=	20
Count total							50

Design Metrics: Class-oriented metrics- The CK metrics suite

- Chidamber and Kemerer have proposed one of the most widely referenced sets of OO software metrics.
- In the *CK metrics suite,* the authors have proposed six class-based design metrics for OO systems.
- **Depth of the inheritance tree (DIT)**
- **Number of children (NOC)**
- **Coupling between object classes (CBO)**
- **Response for a class (RFC)**
- **Lack of cohesion in methods (LCOM)**
- **Weighted methods per class (WMC).** Assume that *n* methods of complexity *c*1, *c*2, . . . , *cn* are defined for a class **C**.

Metrics for source code

Quote:

"The human brain follows a more rigid set of rules [for developing algorithms] than it has been aware of."

Maurice Halstead

Halstead's theory of "software science" [Hal77] proposed the first analytical "laws" for computer software.[14] Halstead assigned quantitative laws to the development of computer software, using a set of primitive measures that may be derived after code is generated or estimated once design is complete. The measures are:

n_1 = number of distinct operators that appear in a program
n_2 = number of distinct operands that appear in a program
N_1 = total number of operator occurrences
N_2 = total number of operand occurrences

Halstead uses these primitive measures to develop expressions for the overall program length, potential minimum volume for an algorithm, the actual volume (number of bits required to specify a program), the program level (a measure of software complexity), the language level (a constant for a given language), and other features such as development effort, development time, and even the projected number of faults in the software.

Halstead shows that length N can be estimated

$$N = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

and program volume may be defined

$$V = N \log_2 (n_1 + n_2)$$

It should be noted that V will vary with programming language and represents the volume of information (in bits) required to specify a program.

Theoretically, a minimum volume must exist for a particular algorithm. Halstead defines a volume ratio L as the ratio of volume of the most compact form of a program to the volume of the actual program. In actuality, L must always be less than 1. In terms of primitive measures, the volume ratio may be expressed as

$$L = \frac{2}{n_1} \times \frac{n_2}{N_2}$$

Halstead's work is amenable to experimental verification and a large body of research has been conducted to investigate software science. A discussion of this work is beyond the scope of this book. For further information, see [Zus90], [Fen91], and [Zus97].

Operators include all flow of control constructs, conditionals, and math operations. Operands encompass all program variables and constants.

Halstead Metrics Applied to Testing

Testing effort can be estimated using metrics derived from Halstead measures (Section 23.5). Using the definitions for program volume V and program level PL, Halstead effort e can be computed as

$$PL = \frac{1}{(n_1/2) \times (N_2/n_2)} \tag{23.2a}$$

$$e = \frac{V}{PL} \tag{23.2b}$$

Metrics for design model

- Architectural Design Metrics:
- These metrics are “black box” in the sense that they do not require any knowledge of the inner workings of a particular software component.
- Card and Glass [Car90] define three software design complexity measures: structural complexity, data complexity, and system complexity.

For hierarchical architectures (e.g., call-and-return architectures), *structural complexity* of a module *i* is defined in the following manner:

$$S(i) = f^2_{\text{out}}(i)$$

where $f_{\text{out}}(i)$ is the fan-out[6] of module *i*.

6 *Fan-out* is defined as the number of modules immediately subordinate to module *i*; that is, the number of modules that are directly invoked by module *i*.

Data complexity provides an indication of the complexity in the internal interface for a module *i* and is defined as

$$D(i) = \frac{v(i)}{f_{\text{out}}(i) + 1}$$

where $v(i)$ is the number of input and output variables that are passed to and from module *i*.

Finally, *system complexity* is defined as the sum of structural and data complexity, specified as

$$C(i) = S(i) + D(i)$$

As each of these complexity values increases, the overall architectural complexity of the system also increases. This leads to a greater likelihood that integration and testing effort will also increase.

Project Management

- Project management involves the planning, monitoring, and control of the people, process, and events that occur as software evolves from a preliminary concept to full operational deployment.
- A software engineer manages her day-to-day activities, planning, monitoring, and controlling technical tasks.
- Project managers plan, monitor, and control the work of a team of software engineers.
- Senior managers coordinate the interface between the business and software professionals.

Four P’s

- **People** must be organized to perform software work effectively.
- Communication with the customer and other stakeholders must occur so that **product** scope and requirements are understood.
- A **process** that is appropriate for the people and the product should be selected.
- The **project** must be planned by estimating effort and calendar time to accomplish work tasks: defining work products, establishing quality checkpoints, and identifying mechanisms to monitor and control work defined by the plan.
- A project plan is produced as management activities commence.

Stakeholders and Team leades

- Senior Manager
- Project Manager
- Customer
- Vendor
- End users

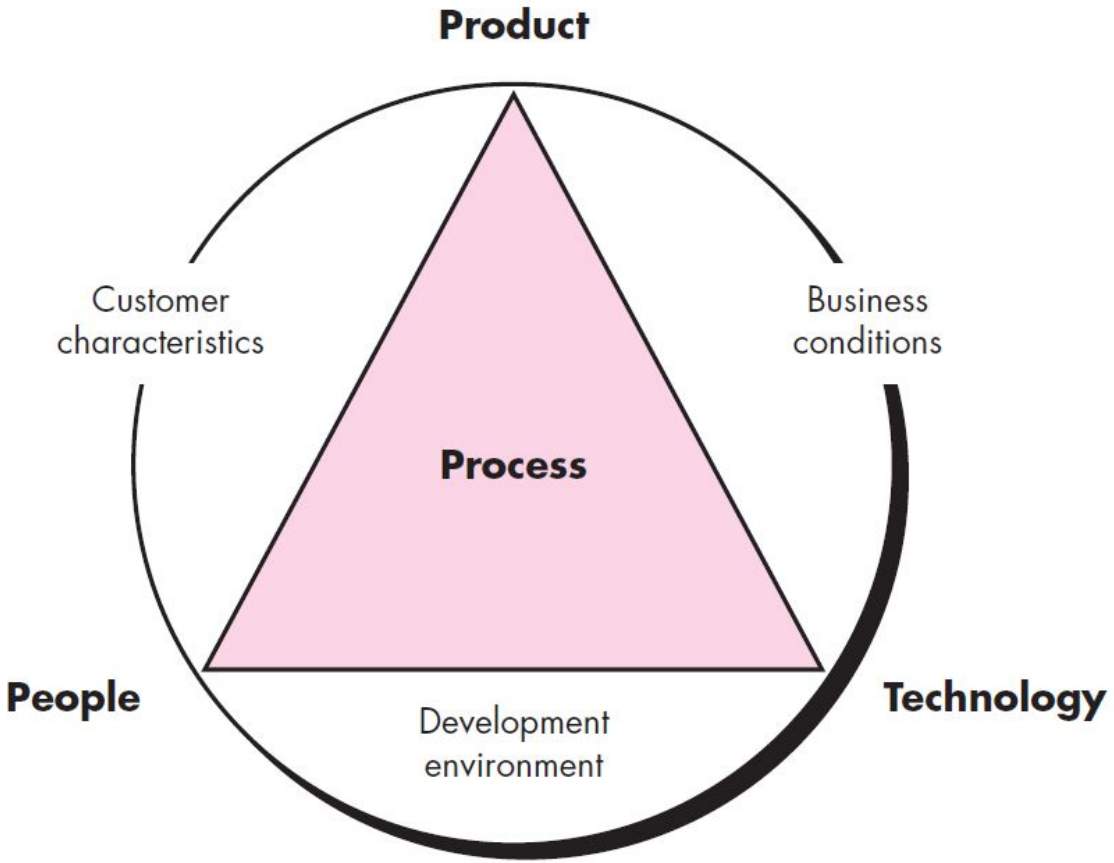
W^5HH Principles

- *Why is the system being developed*
- *What will be done*
- *When will it be done*
- *Who is responsible for a function*
- *Where are they located organizationally*
- *How will the job be done technically and managerially*
- *How much of each resource is needed*

Process and Project Metrics

Process Metrics and Software Process Improvement

- Determinants for software quality and organizational effectiveness



Software Measurements

Size-oriented metrics

Project	LOC	Effort	$(000)	Pp. doc.	Errors	Defects	People
alpha	12,100	24	168	365	134	29	3
beta	27,200	62	440	1224	321	86	5
gamma	20,200	43	314	1050	256	64	6
•	•	•	•	•	•		
•	•	•	•	•	•		
•	•	•	•	•	•		

Project resources