SOFTWARE LAYERS AND MEASUREMENT

Jean-Marc Desharnais, Software Engineering Laboratory in Applied Metrics
Denis St-Pierre, DSA Consulting Inc.
Serge Oligny, Laboratoire de recherche en gestion des logiciels
Alain Abran, Laboratoire de recherche en gestion des logiciels

ABSTRACT

Systems rarely run alone. They are usually part of a complex system of software layers (e.g. database managers, network drivers, operation systems and device drivers). Software layers constitute a specific way of grouping functionalities on a level of abstraction.

When measuring the functionality of a system, practitioners usually consider one type of layer: user application, or the highest-level layer. They consider the other layers as technical. This approach might work with Management Information Systems, where there is often no business need to consider layers other than the highest-level one. This is because the other layers are usually already developed (e.g. Windows, UNIX, printer drivers). However, this is often not the case for real-time and embedded systems. Embedded system development projects involve developing or modifying operating systems, drivers and user applications as well. Not considering software layers can result in misleading measurements, as measuring only the highest-level layer may lead to misrepresentation of the size of a project or application.

This paper covers the definition of software layers and how to identify them, and by extension the identification of peer systems: systems residing on the same layer.

1. INTRODUCTION

A key aspect of software functional size measurement is the establishment of what is considered part of the software and what is considered part of the operating environment of the software. The reality is that software is bounded by I/O and storage hardware. Examples of I/O are: mouse, keyboard, printer, screen and sensor. Examples of storage are: hard disk, diskette and RAM memory. The front end is related to I/O and the back end to storage.

Figure 1 illustrates the generic flow of data from a functional perspective.

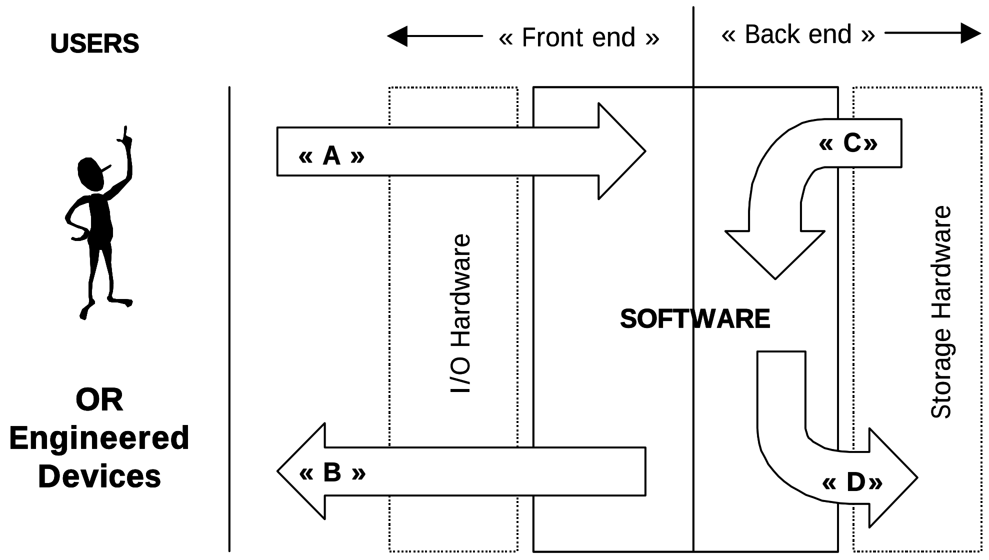

Figure 1 - Generic flow of data through software from a functional perspective

From a functional perspective, it can be observed that:

- The generic flow of data passes across many distinct pieces of software from its origin (users) to its destination (storage device or back to the users).
- There is a hierarchical relationship among many of the pieces depicted. This relationship is controlled from the centre.

From there it is possible to make a distinction between the physical and functional aspects of the software. The reality becomes more complex when we realize that there is more than one category of software.

Software usually included within the scope of an organization can be categorized based on the type of services provided, as illustrated in Figure 2 below.

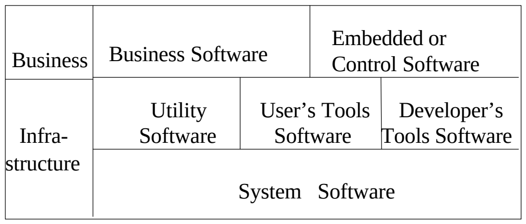

Figure 2 Categories of software according to the service provided (Morris, 1998)

More detailed descriptions of these types of services are provided in Appendix A.

This descriptive approach is convenient for distinguishing different types of functionality, but there are some gray zones when it is used to measure software functional size. For this reason, it is necessary to propose formalized types of functionality for the purpose of measuring the functional size of software. This will be considered later in the article.

2. THE LAYER DESCRIPTION APPROACH

Considering the flow of the data and the categories of software, software can be described as shown graphically in Figure 3.

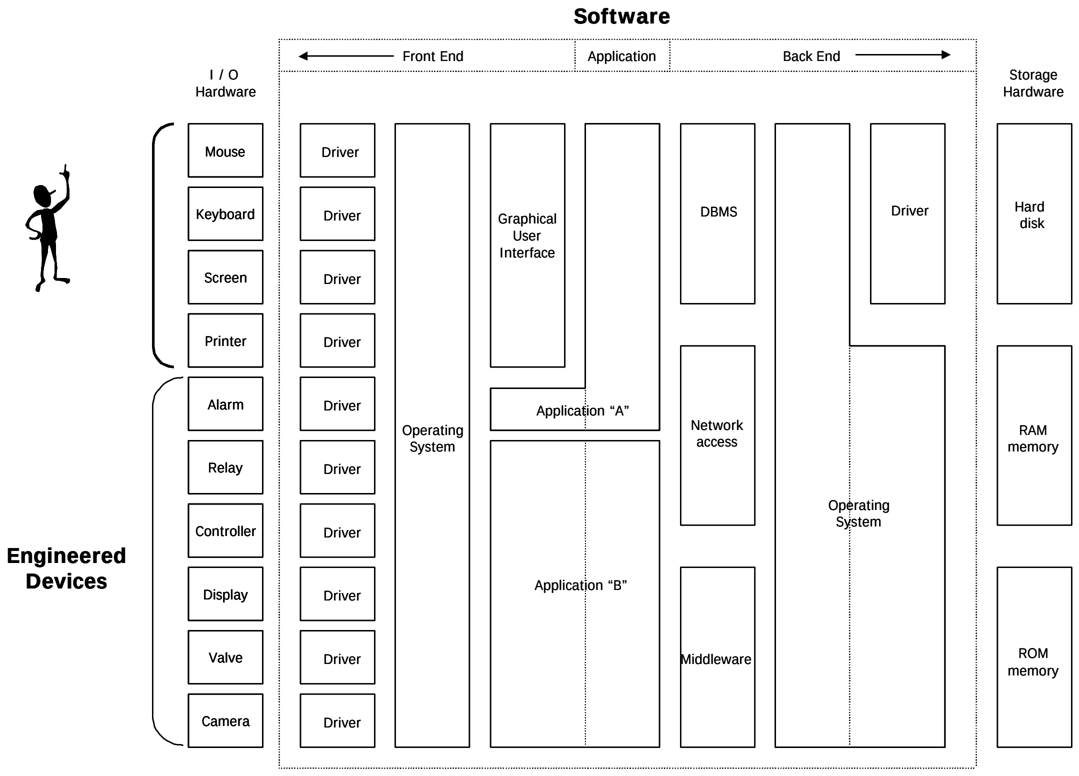

Figure 3 - Software in its environment

While all the pieces exchange data, they will not necessarily operate at the same "level". For the first time in Functional Size Measurement, the Full Function Points software context model has recognized this general configuration by providing rules to identify different layers of software. Each piece of software on each layer is defined by an enclosing boundary, and users are identified in relation to this boundary.

A layer is a functional partition of the software environment where all the functional processes included perform at the same level of abstraction and usually exhibit a high degree of cohesion.

In other words, software layers are a specific way of grouping functionalities at the same level of abstraction. Each layer is a world unto itself (Rules, 1998).

From the perspective of a given layer, software delivers functionality to a specific category of users[1]. When information is exchanged between a pair of layers, one of them is considered a client of the services provided and the other the subordinate. Each layer is thus the 'user' of another layer in the hierarchy. Software layers in fact represent different levels of abstraction.

In Figure 3, the "client" or "application" layer is the one delivering functionality to the software end-users. Software within any layer may deliver functionality to peer systems within the same layer (Application "A" to Application "B" in Figure 3), however their primary users are pieces of software lying in the layer directly linked to them in the hierarchy:

- Applications "A" and "B" deliver functionality to end-users. End-users are either humans (often the case for business applications) or equipment (often the case for embedded or real-time software).
- Subordinate layers deliver functionality directly to support their users: Graphical User Interface with Application "A", for instance. Also, a part of the Operating System is subordinate to Applications "A" and "B". DBMS, Network ACCESS and Middleware, for example are subordinate to Applications "A" and "B".
- Parts of the Operating System are subordinate to DBMS, Network ACCESS and the Middleware. Drivers are subordinate to the Operating System.

[1] Users: Human beings, other software systems or engineered devices which interact with the measured software

A software layer is usually perceived as a collection of interacting pieces, as shown previously. Users may have little knowledge of the software layers, which is not a reason to avoid measuring them. In fact, in many cases considering layers is necessary in order to measure a set of functionalities adequately. Using software layers, it is possible to adequately measure at any layer provided the functionality is interpreted on its own terms, with the layer above or at the same level being its users. Keep in mind that a user can be any person and/or thing communicating or interacting with the software at any time.

From the strict functional measurement point of view, not considering software layers can lead to misleading measurement because systems at different layers delivers different types of functionality. For example a word processor and the operating system on which it runs, deliver very different types of functionality (Figure 4 below). Word processors deal with words, and operating systems deal with records of data (to make a long story short). When an operating system executes a command coming from a word processor (e.g.: save a file), it does not know that the records of data contain words. For the operating system, it is just a long string of characters. These characters may come from many types of systems including word processors.

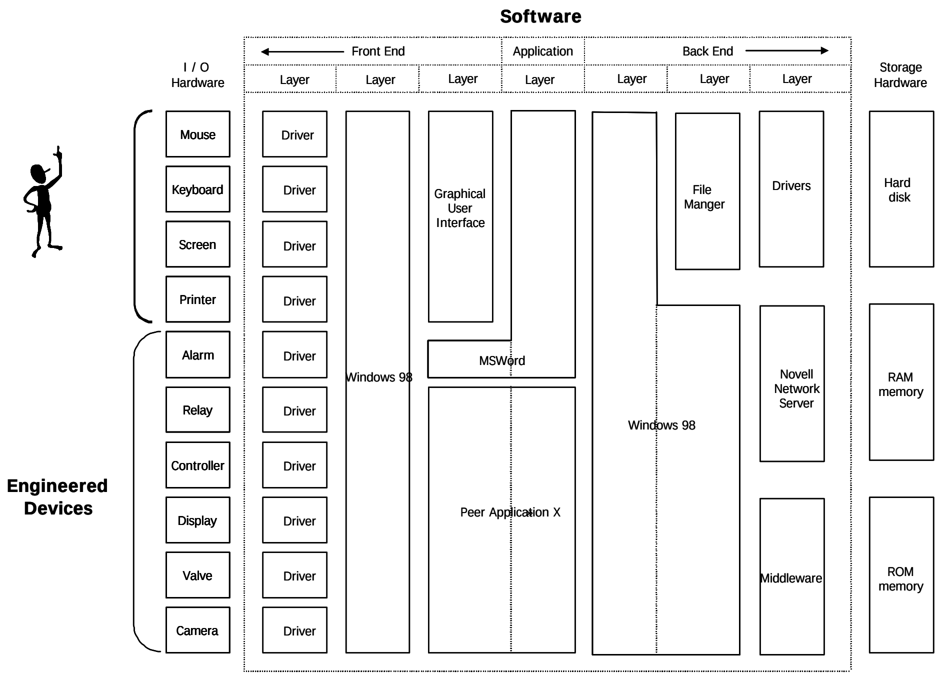

Figure 4 – Examples of software with different types of functionality

Using Figure 3, it is possible to construct Figure 4 and see application "A" as MS Word[2] and application "B" as MS EXCEL[3]. They are both considered as client software. The Operating System could be MS Windows 98[4] (using OS on the left side of Figure 4), which is considered as software subordinate that is to the previous software. Network access could be the Novell Network Server[5], which is a client system when passing through MS Windows 98 (right side of Figure 4). MS Windows 98 is then the client software of the Seagate Drivers[6] (subordinate), which permit the access to the hard disk.

Table 2 summarizes these relationships.

[2] Microsoft Corporation ®
[3] Microsoft Corporation ®
[4] Microsoft Corporation ®
[5] © 1998 Novell, Inc. All Rights Reserved
[6] Seagate Corporation ®

Software Layers	System Examples	User Examples
System	MS Word and MS EXCEL	Users of MS Word and MS EXCEL (human users)
Operating System	MS Windows 98	MS Word and MS EXCEL
Network	Novell Network Server	MS Windows 98
Drivers	Seagate Hard Disk Drivers	Novell Network Server

Table 2: Layers and Examples

As mentioned previously, each layer is a world unto itself. The way the inputs are generated is irrelevant from the layer perspective. The only requirement is that the input be dealt with appropriately when it is received.

A layer perceives the layer below it as a set of primitives. Each layer "sees" the layers below, but cannot see the layers above. Similarly, what happens to data output from a layer is irrelevant to the producer once that output has been dispatched.

The internal operation of one layer does not need to be “known” by any other layer. Indeed, it is preferable that internal details be protected from outside alteration. Making one layer dependent on the internal organization of another is highly undesirable. Such dependency restricts the ability to maintain (i.e. change or enhance) the layer on which the other of depends. This is the basic principle of information hiding. Ideally, organizations should to be able to entirely re-construct a server layer without affecting the clients that use its primitives, so long as the software retains the same interface. For this reason, organizations should prevent a client layer from directly using the services of any subordinate layer except the one immediately below it.

3. MEASUREMENT BASIS

3.1. Identification Principles

Definition of a Layer
A layer is a functional partition of the software environment where all included functional processes should ideally exhibit a high degree of cohesion and perform at the same level of abstraction.

Layer identification is an iterative process. Once identified, each candidate layer must comply with the following principle based on the fact that there are two categories of layers: client layers and subordinate layers. The client layer uses the functionalities of a (or many) subordinate layer(s) to perform its own functionalities. The subordinate layer provides functionalities to one (or many) client layer(s).

Principles for the identification of a Layer[7]

a) A subordinate layer is perceived as a technical implementation from the perspective of the client layer.
b) A subordinate layer could perform without assistance from a client layer. If the client layer is not performing properly, the subordinate layer is not affected.
c) A subordinate layer is independent of the technology used by a client layer.
d) A subordinate layer provides services to client layers.
e) A client layer may not perform fully if a subordinate layer is not performing properly.
f) A client layer does not necessarily use all the functionality supplied by a subordinate layer.
g) A subordinate layer can be perceived as a client layer from the perspective of a third subordinate layer.
h) All layers deliver functionality.

Data might be perceived differently from one layer to another.

Because, in practice specific software could be either client or subordinate, depending on its

[7] The term “principle” corresponds to “rules” in the IFPUG context.

relationship with the other software, it might not easy to make a clear distinction between different layers.

3.2. Identifying Layer Boundaries

If, as in some circumstances, it proves difficult to distinguish between two interacting layers, it might be helpful to conceive of the two layers as two sets of functional processes exhibiting a low degree of coupling between them and a high degree of cohesion within each set, as illustrated by Figure 6.

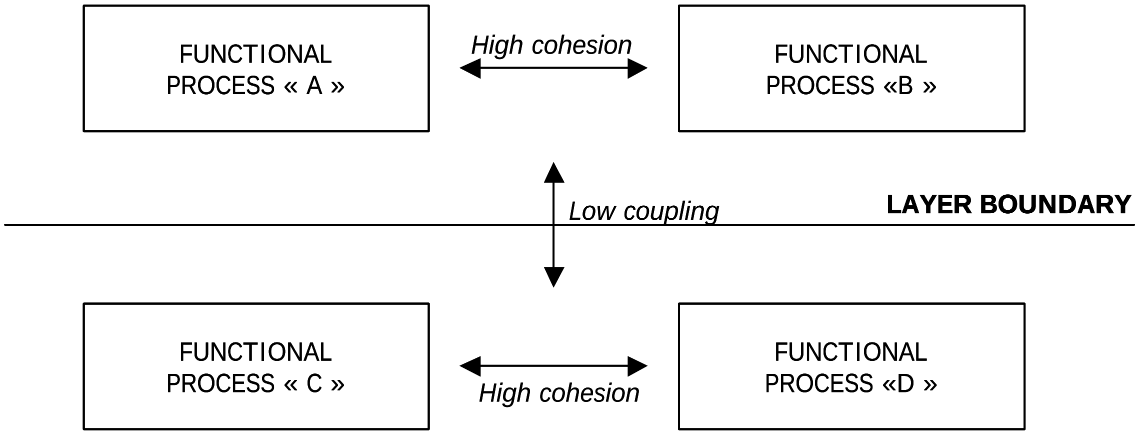

Figure 6 – Cohesion and coupling between two interacting software layers

For this purpose, we took the concepts of coupling and cohesion from Pressman[8] and Sommerville[9], and adapted them to distinguish between two interacting layers:

DEFINITION – Coupling
Coupling is a measure of the interconnection among functional processes between two sets of functional processes. Coupling depends on the interface complexity between functional processes, the point at which entry or reference is made to a functional process, and what data pass across the interface. Coupling is measured on an ordinal scale (from low to high coupling): • **no** direct coupling • **data** coupling • **stamp** coupling • **control** coupling • external coupling • **common** coupling • **content** coupling

[8] Adapted from Sommerville I., 1995.
[9] Adapted from Pressman R.S., 1995.

DEFINITION – Cohesion
Cohesion is a measure of the interdependence among functional processes belonging to the same set. A cohesive set of functional processes performs a single task and requires little interaction with other sets of functional processes. Cohesion is measured on an ordinal scale (from low to high cohesion): • **Coincidental**: the functional processes in the set are not related, but simply bundled into a single set. • **Logical**: functional processes in the set perform similar tasks, such as input, error handling, and so on. • **Temporal**: all functional processes in the set are activated at one time, such as at start-up or when shutting down. • **Procedural**: the functional processes in the set make up a single control sequence. • **Communicational**: all functional processes in the set operate on the same input data or produce the same output data. • **Sequential**: the output from one functional process in the set serves as input for some other functional process in the same set. • **Functional**: each functional process in the set is necessary for the execution of a single function.

The exact degree of cohesion and coupling required to distinguish between two layers is currently being investigated through field tests.

4. CLOSING REMARKS

Software layers below the highest-level layer provide the infrastructure. Although the users below the first layer are not the end-users, end-users are still able to take advantage of the infrastructure software because, without it, their systems would be inoperable. The infrastructure software delivers functionality indirectly to the end-users via other systems.

The functionality delivered by software within one layer is not the same type of functionality delivered by software within a lower layer. Therefore, functionality delivered by software at different layers should only be combined, or compared, with great care.

When reporting the functional size of software measured on different levels, it is important that systems on different layers be identified and measured separately. They should also be analyzed independently to some extent. For example, if only the cumulative size of all 'layers' is reported, this can give the impression that all the functionality is delivered to the end-user. Therefore, size should be presented with respect to its corresponding layers and classes of users so that people can associate size with the functionality delivered. Once this is done the cumulative size of all 'layers' may be presented with less risk of misinterpretation.

Considering layers is as essential as considering the software boundary. Without this consideration, the measurement results could be misleading and the wrong message sent to managers, technical staff and users.

REFERENCES

COSMIC, Full Function Points Measurement Manual, version 2.0, field-test version, July 31st 1999 editor S. Oligny, UQAM. The public release of this version is due in October 1999.

Morris P., Desharnais J.-M., *Measuring ALL the software, not just what the Business uses*, Proceedings of the IFPUG Conference, Orlando, 1998.

Rules G., *Comments on ISO 14143 Part 5*, Release V1b, 1998.

Sommerville I., "Software Engineering", 5th ed., Addison Wesley, 1995, pp. 218-219.

Pressman R.S., "Software Engineering – A practitioner's approach", 4th ed., McGraw Hill, 1997, pp. 359-361.

APPENDIX A

EXAMPLE – Functionality provided by different types of software

Business Software
This type of software delivers functionality which supports the organization's core business.
The users are primarily human business users, however a small proportion of the functionality may also be delivered to, or triggered by, other Business Applications. This type of software is typically business or commercial (MIS) software and would include Payroll applications, Accounts Receivable or Fleet Management systems.

Embedded or Control Software
This type of software also delivers functionality which supports the organization's core business or products. The users are primarily other software applications embedded in equipment. This type of software typically operates under strict timing conditions and is often referred to as real-time software. Examples would include Equipment Monitoring Systems, Telephone Switching Systems.

Utility Software
This type of software delivers functionality providing the infrastructure to support Business Software. The users are primarily the Business Software itself, which initiates the operation of the utilities, but may include the developers or business administration people as the administrative users. Examples would include backup utilities (to ensure the data reliability of the Business Application) or archiving utilities (to optimize the performance of the Business Application). Other examples are installation and conversion software.

User's Tool Software
This type of software delivers the tooling functionality used by administrative users to create the functionality delivered by Business Software. The users are primarily the Business Software itself, which uses the functionality delivered by the tools to enable them to deliver functionality to the business. Administrative human users of these tools may be either from the Business or from IT. Examples would include Report Generators, Spreadsheets and Word Processors.

Developer's Tool Software
This type of software delivers the tooling functionality used by developers to create the functionality delivered by Business Software. The users are primarily other applications, which are either generated by, or used as input to, tool operation. Human users may also include IT developers as administrative users. Examples would include Code Generators, Testing Software, New Product Generators, etc.

System Software
This type of software enables all the other types of software to operate and deliver their own functionality. The users are primarily other applications with a limited interface to human IT operational staff. Examples would include operating systems, printer drivers, protocol converters and presentation software.