Structure Chart

Structure Chart

- A structure chart represents the software architecture. The various modules making up the system, the module dependency (i.e., which module calls which other modules), and the parameters that are passed among the different modules.

- The structure chart representation can be easily implemented using some programming language.

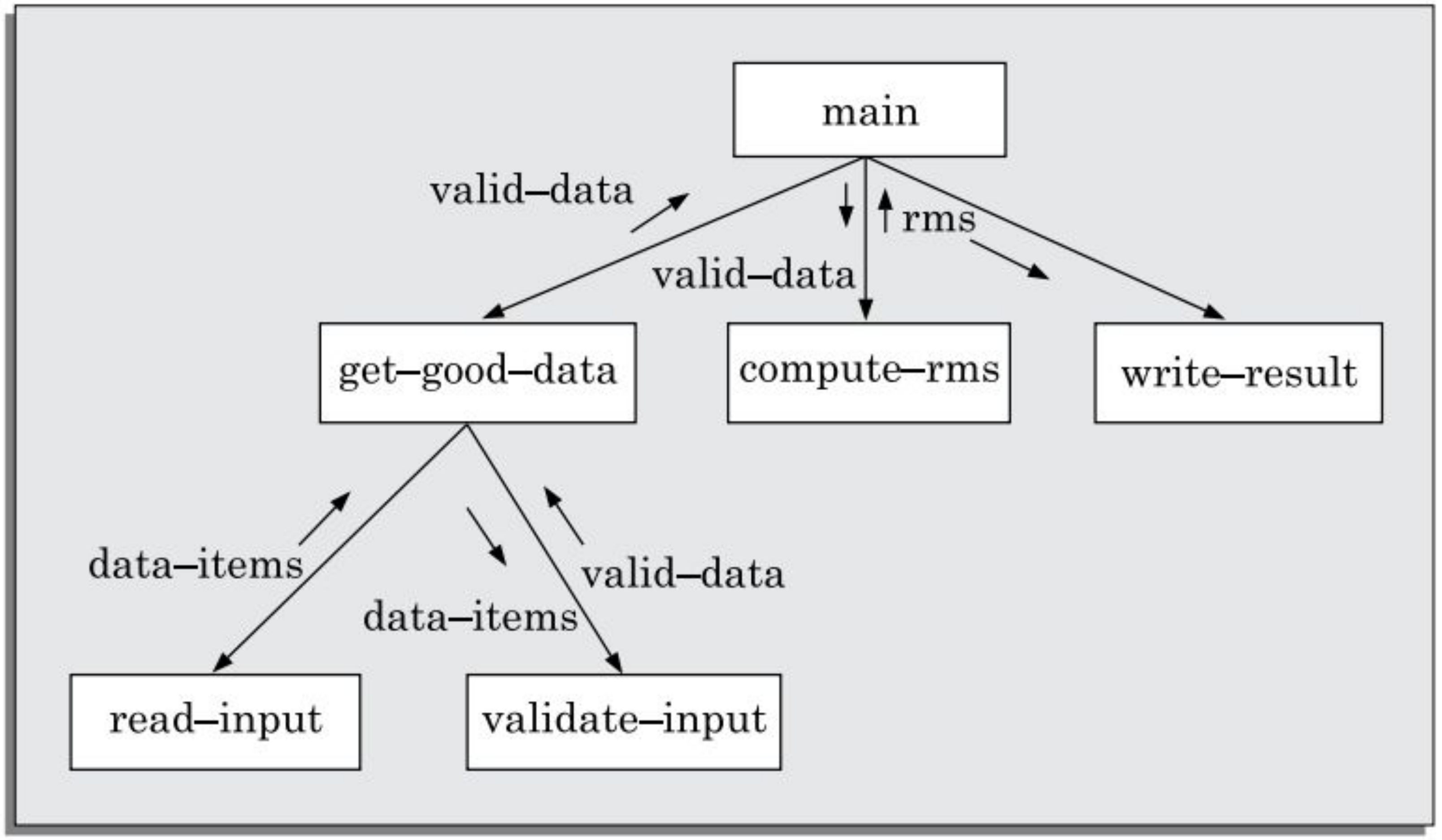
- Since the main focus in a structure chart representation is on module structure of a software and the interaction among the different modules, the procedural aspects (e.g., how a particular functionality is achieved) are not represented.

Structure Chart

The basic building blocks using which structure charts are designed are as following:

- **Rectangular boxes:** A rectangular box represents a module. Usually, every rectangular box is annotated with the name of the module it represents.
- **Module invocation arrows:** An arrow connecting two modules implies that during program execution control is passed from one module to the other in the direction of the connecting arrow. However, just by looking at the structure chart, we cannot say whether a modules calls another module just once or many times. Also, just by looking at the structure chart, we cannot tell the order in which the different modules are invoked.
- **Data flow arrows:** These are small arrows appearing alongside the module invocation arrows. The data flow arrows are annotated with the corresponding data name. Data flow arrows represent the fact that the named data passes from one module to the other in the direction of the arrow.
- **Library modules:** A library module is usually represented by a rectangle with double edges. Libraries comprise the frequently called *modules*. Usually, when a module is invoked by many other modules, it is made into a library module.
- **Selection:** The diamond symbol represents the fact that one module of several modules connected with the diamond symbol is invoked depending on the outcome of the condition attached with the diamond symbol.
- **Repetition:** A loop around the control flow arrows denotes that the respective modules are invoked repeatedly.

Structure Chart

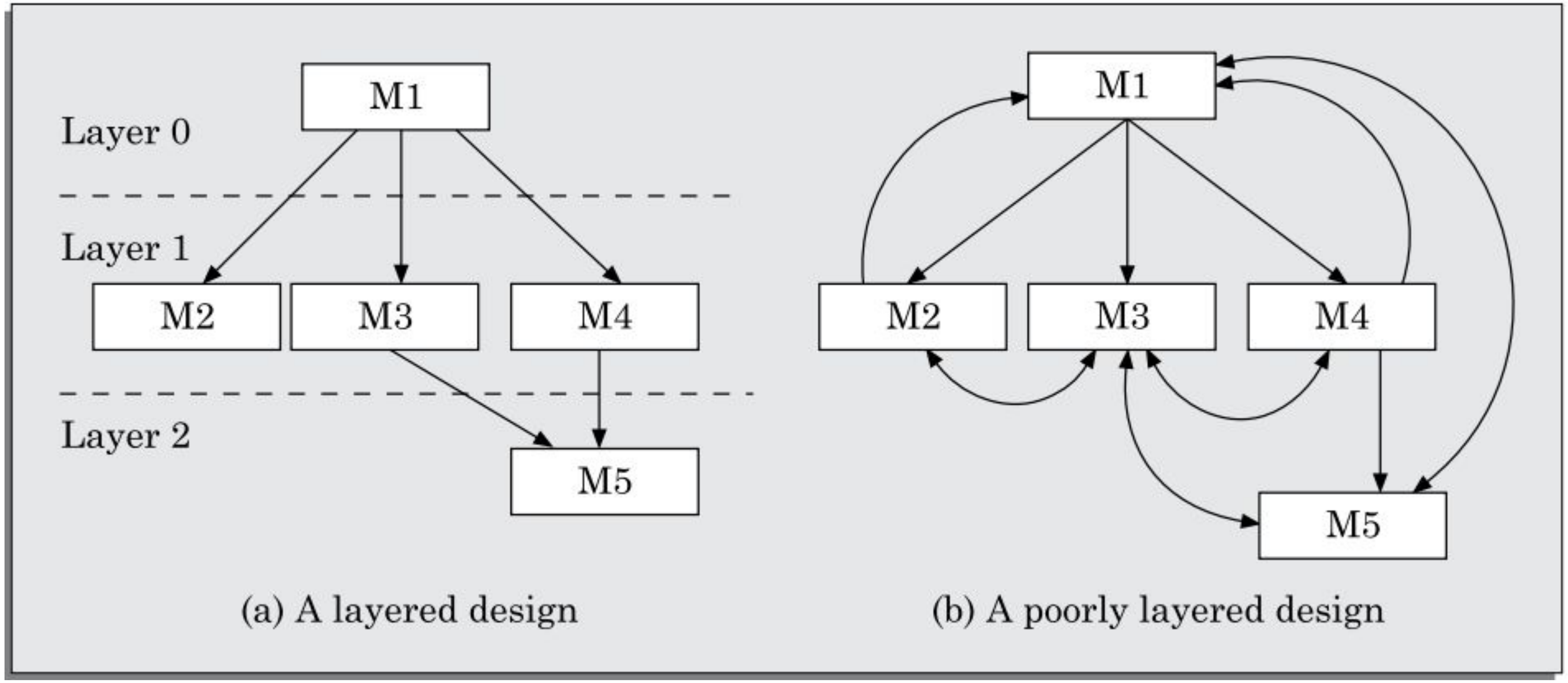


A simple structure chart for the problem of RMS computation.

Structure Chart

- In any structure chart, there should be one and only one module at the top, called the *root.*
- There should be at most one control relationship between any two modules in the structure chart. **This means that if module A invokes module B, module B cannot invoke module A.** The main reason behind this restriction is that we can consider the different modules of a structure chart to be arranged in layers or levels.
- The principle of abstraction does not allow lower-level modules to be aware of the existence of the high-level modules.
- However, it is possible for two higher-level modules to invoke the same lower-level module.

Structure Chart



Examples of properly and poorly layered designs.

Structure Chart

Flowchart VS. Structure Chart

Flow chart is a convenient technique to represent the flow of control in a program. A structure chart differs from a flow chart in three principal ways:

- It is usually difficult to identify the different modules of a program from its flow chart representation.
- Data interchange among different modules is not represented in a flow chart.
- Sequential ordering of tasks that is inherent to a flow chart is suppressed in a structure chart.

Transformation of a DFD Model into Structure Chart

Systematic techniques are available to transform the DFD representation of a problem into a module structure represented by as a structure chart. There are two strategies to guide transformation of a DFD into a structure chart:

- Transform analysis
- Transaction analysis

At each level of transformation, it is important to first determine whether the transform or the transaction analysis is applicable to a particular DFD.

Whether to apply Transform or Transaction Processing

- First, examine the data input to the diagram.
- If all the data flow into the diagram are processed in similar ways (i.e., if all the input data flow arrows are incident on the same bubble in the DFD) then transform analysis is applicable.
- Otherwise, transaction analysis is applicable. Normally, transform analysis is applicable only to very simple processing.
- Transform analysis is normally applicable at the lower levels of a DFD model.
- Each different way in which data is processed corresponds to a separate transaction. Each transaction corresponds to a functionality that lets a user perform a meaningful piece of work using the software.

Transform analysis

Transform analysis identifies the primary functional components (modules) and the input and output data for these components. The first step in transform analysis is to divide the DFD into three types of parts:

- ❑ Input.
- ❑ Processing.
- ❑ Output.

- The input portion in the DFD includes processes that transform input data from physical (e.g., character from terminal) to logical form (e.g., internal tables, lists, etc.). Each input portion is called an *afferent branch*.
- The output portion of a DFD transforms output data from logical form to physical form. Each output portion is called an *efferent branch.*
- The remaining portion of a DFD is called *central transform.*