# SOFTWARE DESIGN (SWD)

Instructor: Dr. Hany H. Ammar

Dept. of Computer Science and Electrical Engineering, WVU

# OUTLINE OF SOFTWARE DESIGN

- Introduction to Software Design (SWD) and the SW Design Description (SDD) document
- Software Design Criteria
- Software Design Methodologies
- Structured Design (SD) Using ICASE

# Introduction to SWD, and The SDD Document

- The SWD phase in the DOD standard MIL-STD-498 (Chapter 2, section 3) focuses on developing the physical model for each computer software configuration item (CSCI)
- The output of this phase is the the SWD document (See Table 4.1, section 4.1 of the notes)

# Introduction to SWD, and The SDD Document

The SDD consists of the following sections

1. Scope.
2. Referenced documents
3. CSCI-wide design decisions
4. CSCI architectural design
   - 4.1 CSCI components
   - 4.2 Concept of execution
   - 4.3 Interface design
5. CSCI detailed design
6. Requirements traceability

# Introduction to SWD, and The SDD Document

- The SWD starts in the first section by identifying the scope of the CSCI, presenting a system overview, and a document overview
- The second section lists the number, title, revision, date, and source of all documents referenced in this specification.
- In the third section, the CSCI-wide design decisions are described as follows:

  a. Design decisions regarding inputs the CSCI will accept and outputs it will produce,

  b. Design decisions on CSCI behavior in response to each input or condition,

# Introduction to SWD, and The SDD Document

- In section 4, the CSCI architectural design is specified. The section is divided into the following subsections described below,
- Section 4.1 Identifies the  (CSCs), software components or units that make up the CSCI
- Section 4.2 describes the concept of execution among the software components, i.e.,  flow of execution control, data flow, dynamically controlled sequencing, concurrent execution, etc.

# Introduction to SWD, and The SDD Document

- Section 4.3 describes the Interface design and it includes both interfaces among the software units and their interfaces with external entities, and it consists of two parts as follows:

  1. Section 4.3.1depicts the interfaces identification and diagrams (interfaces identifiers name, number, and version; fixed or to be modified, etc.)

  2. This part consists of several sections (starting from section 4.3.2). Each section identifies a particular interface, with data type, format, range, Priority, timing, frequency, volume, Security and privacy constraints

# Introduction to SWD, and The SDD Document

- Section 5 specifies the detailed design of each CSC including the following
- The components structure in software units, components design decisions, design constraints, The programming language to be used, concept of execution of its units, and internal interfaces between units
- The task of Requirements traceability needed for verification and validation is documented in section 6 of the SDD.

# Software Design Criteria

- The established criteria for software design quality help the designer in developing and assessing the quality of the software design architecture
- This include:
  1. Modular design (coupling, and cohesion),
  2. Information hiding,
  3. Design complexity,
  4. Testability, and
  5. Reusability

# Software Design Criteria
# Modular design

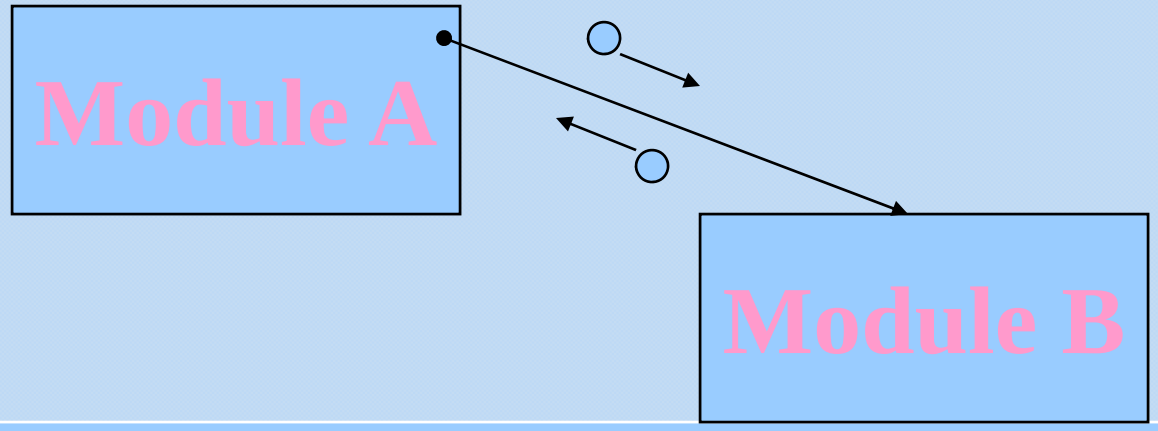
- Modular Design: software is decomposed of software modules such that a change in one module has minimal effect on other modules
- A software module is a program unit consisting of procedures, functions, and data structures that
  - can be separately compiled, and
  - independently callable unit of code

# Software Design Criteria
# Modular design

- The development of a modular design structure is guided by the need for satisfying design criteria regarding the *coupling* of pairs of modules, and the *cohesion*, complexity, and reusability of individual components in each module
- coupling is a measure of level of the interactions between two modules

# Software Design Criteria
# Modular design

- Modules interact by passing parameters through a call statement, or by sharing common data and file structures. The goal of modular design is to reduce the level of coupling between modules



# Software Design Criteria
# Modular design

- The goal of modular design is to reduce the level of coupling between modules and increase the cohesion of each module
- The following forms of module ***coupling*** are listed from the lowest to the highest levels of coupling

    1. ***Data coupling*:** communication between modules is accomplished through well-defined parameter lists consisting of data information items

# Software Design Criteria
# Modular design

2. ***Stamp coupling*:** communication between modules is accomplished through well-defined data structure parameter lists where only parts of the data structures are used in the target module

3. ***Control coupling*:** a module controls the flow of control or the logic of another module. This is accomplished by passing control information items as arguments in the argument list.

# Software Design Criteria
# Modular design

4. ***Common coupling*:** modules share common or global data or file structures. This is the strongest form of coupling both modules depend on the details of the common structure

5. ***Content coupling*:** A module is allowed to access or modify the contents of another, e.g. modify its local or private data items. This the strongest form of coupling

# Software Design Criteria
# Modular design

- In modular design, modules should be developed as consisting of strongly cohesive components
- ***Cohesion*** is a measure of the internal relatedness of the components of a module.
- The following forms of cohesion are described from the highest strength to the lowest levels of cohesion

  1. ***Functional cohesion:*** is achieved when the components of a module cooperate in performing exactly one function, e.g., POLL_SENSORS, GENERATE_ALARM, etc.

# Software Design Criteria
# Modular design

2. ***Communicational cohesion:*** is achieved when software units or components sharing a common information or data structure are grouped in one module

3. ***Procedural cohesion:*** is the form of cohesion obtained when software components are grouped in a module to perform a series of functions following a certain procedure specified by the application requirements

# Software Design Criteria
# Modular design

4. ***Temporal cohesion:*** this form of cohesion is found when a module is composed of components or functions which are required to be activated during the same time interval. Examples are functions required to be activated for a particular input event, or during the same state of operation

# Software Design Criteria
# Modular design

5. ***Logical cohesion:*** refers to modules designed using functions who are logically related, such as input/output functions, communication type functions (such as send and receive),

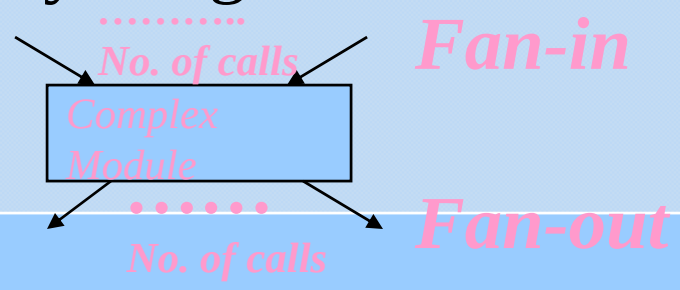
6. ***Coincidental cohesion:*** is found when several unrelated functions are grouped in one module to decrease the total number of modules and increase the module size. This is the lowest level of cohesion

# Software Design Criteria Information hiding

- Design decisions that are likely to change in the future should be identified and modules should be designed in such a way that those design decisions are hidden from other modules
- the concept of ***information hiding*** is to hide the implementation details of shared information and processing items by specifying modules called ***information hiding modules***
- Module Interfaces are created to allow other modules accessibility to shared items without having visibility to its internal implementations

# Software Design Criteria
# Design Complexity

- ***Complexity*** is another design criteria used in the process of decomposition and refinement. A module should be simple enough to be regarded as a single unit for purposes of verification and modification
- a measure of complexity for a given module is proportional to the number of other modules calling this module (termed as fan-in), and the number of modules called by the given module (termed as fan-out)



# Software Design Criteria
# Design for Testability

- software modules must be designed to enhance or facilitate ***testability***, i.e., the ease of developing a set of test cases and test drivers
- Avoid modules with a large numbers of input/outputs, no inputs/outputs, whose interfaces pass unnecessary data, and
- Avoid a process whose functionality is spread over several unrelated modules (difficult to trace the functional requirements to test cases of modules

# Software Design Criteria
# Design for Reuse

- Reusability is now considered as an important design criteria after the emergence of software repositories which provide means of classifying, cataloging, and retrieving ***software components***
- Both domain specific and general reusable modules and ***Design Patterns and frameworks*** are sought out in the design of current systems
- Avoid special assumptions and dependencies (e.g., on specialized components, library functions, operating system functions, etc.)
- Need well defined interfaces (specifying provided and required services)