Outline

1. Modular Design
   - programming in the large
   - software engineering

2. Good Design in Action
   - choosing between two designs
   - justifying the right choice

MCS 260 Lecture 23
Introduction to Computer Science
Jan Verschelde, 10 July 2023

software engineering
modules in Python

Modular Design
building large software systems

- we have practiced programming in the small

- programming in the large requires modular design

- characteristics of large programs:

    1. size: more than 100,000 lines of code

    2. effort: many teams of programmers

    3. time: program maintenance and evolution

- modular design of programs aims to control the complexity of a program by dividing it into *modules*

- a typical example of a module is a library of functions

Design of Software Systems – Layers or Levels

Layers typical for almost any software system:

1. the *kernel* consists of basic functions
2. the *main operations* apply the kernel
3. the *user interface* defines how the user interacts with the software

An operating system is an example of a large software system.

For another example, the mathematical software SageMath consists of

1. components which focus on a particular area
2. ipython: modification of Python interpreter
3. notebook interface is GUI (Graphical User Interface)

software engineering
modules in Python

Modules
components of software systems

A software system consists of

1. a collection of modules; and
2. the relations between the modules.

Modular design defines the decomposition of a system into modules.

Each module has an **interface** and a **body**:

interface is the set of all elements in a module available
to all users of the module, also called the module's
exported resources

body is what realizes the functionalities of a module,
also called the **implementation**.

A module *imports* resources from another module.
A module *exports* resources via its interface.

Principles of Modular Design

criteria for good software engineering

The software architect designs the system architecture.
The system architecture represents the decomposition of the system
into modules and the intermodule relations.

A first recommendation to design modular software:

- **Information Hiding**
  The interface must be separated from the body.
  Programs that rely on the module via its interface
  do not have to be rewritten as the body changes.

This principle implies that the interface of a module contains
the right kind of information.

Example: Users who need to manipulate polynomials should not be
required to take into account the internal data structures used to
represent the polynomials.

Bottom-Up Design of Programs
principle of low coupling and high cohesion

Modules are implemented by different teams of programmers,
often working over different time periods.
The functionality of a module must be ready for testing and verification
independently of the rest of the program.

A second recommendation to design modular software:

- **Low Coupling and High Cohesion**
  Low coupling means that modules are largely independent
  from each other.

  Functions often used together belong to the same module
  so each module has a high internal cohesion.

A module to manipulate polynomials should collect all the operations
needed in the software system.

Reuse of Modules
standard libraries of components

Software development is an expensive process...

A third recommendation to design modular software:

- **Design for Change**
  For example, use of parameters and constants for data
  that may later change.

For modules to manipulate polynomials, we foresee that different
coefficient fields could be needed.

Object-oriented design is typically bottom up
and leads to reusable software.

We will cover object-oriented programming in Python.

software engineering
modules in Python

making a good design

Consider the modular structure of a program to compose music for a band consisting of guitar, drum, and piano.
Each instrument comes with `in` and `out` functions.
The `in` function takes instructions and simulates the sound.
The `out` function prints instructions for the musician.

We can design the program in two ways:

1. There are two modules: `input` and `output`.
   The `input` module collects all `in` functions of the instrument.
   All `out` functions are in `output`.

2. There is a separate module for each instrument.
   Each module contains the `in` and `out` functions for the instrument.

Which design would be best?
Justify using the principles of good modular design.

software engineering
modules in Python

justifying the right choice

The second design works best.

Justification along the three principles:

1. information hiding,
2. high cohesion and low coupling,
3. design for change.

Information Hiding

The programmer of a module in the first design needs to know the ins and outs of each instrument,
while in the second design, a programmer of a module can focus on one instrument.

In the second design, each module hides the details about its instrument to the other modules.

The Second Principle
high cohesion and low coupling

There is high cohesion in the second design *because* all functionality about one instrument remains in one module,

while the functionality of one instrument is spread out over several modules in the first design.

In the first design there is high coupling *because* the instructions for the input to simulate the sounds will be similar to what will be given to the musicians.

There is low coupling in the second design, *because* the programmer can share conventions in input and output routines, proper for each instrument.

The Third Principle
design for change

If another instrument is added to the band . . .

- In the second design,
  we have to add only another module,
  the existing modules remain the same.

- In the first design,
  we need to change all existing modules.

Exercises

1. Describe the cohesion and coupling for a novel and a textbook. Compare the differences in degrees of cohesion and coupling for both.

2. Modify the modular design of the stack into a module to represent a queue of data. Define in the module `queue_of_data` the operations `enqueue` and `dequeue` to respectively add and remove elements. Define input/output and write a test program.

3. Design a program to search a phone directory. Users can enter a name (or a telephone number) and the program will then search for the corresponding telephone number (or name). Draw your modular design. For each module describe what functions are exported and what is imported. Justify your design, referring to the three key principles of good design.