CS 50:
Software Design and Implementation

Cohesion and coupling

Agenda

1. Coupling and cohesion
2. Activity

Question

Why break programs up into routines (functions)?

Routines should be well named:

- Name should be a strong verb followed by object (*printCalendar()*)
- Name should describe its return value (*numberOfNonzeros()*)
- a *boolean function* name should sound like a question (*isInternalURL()*)

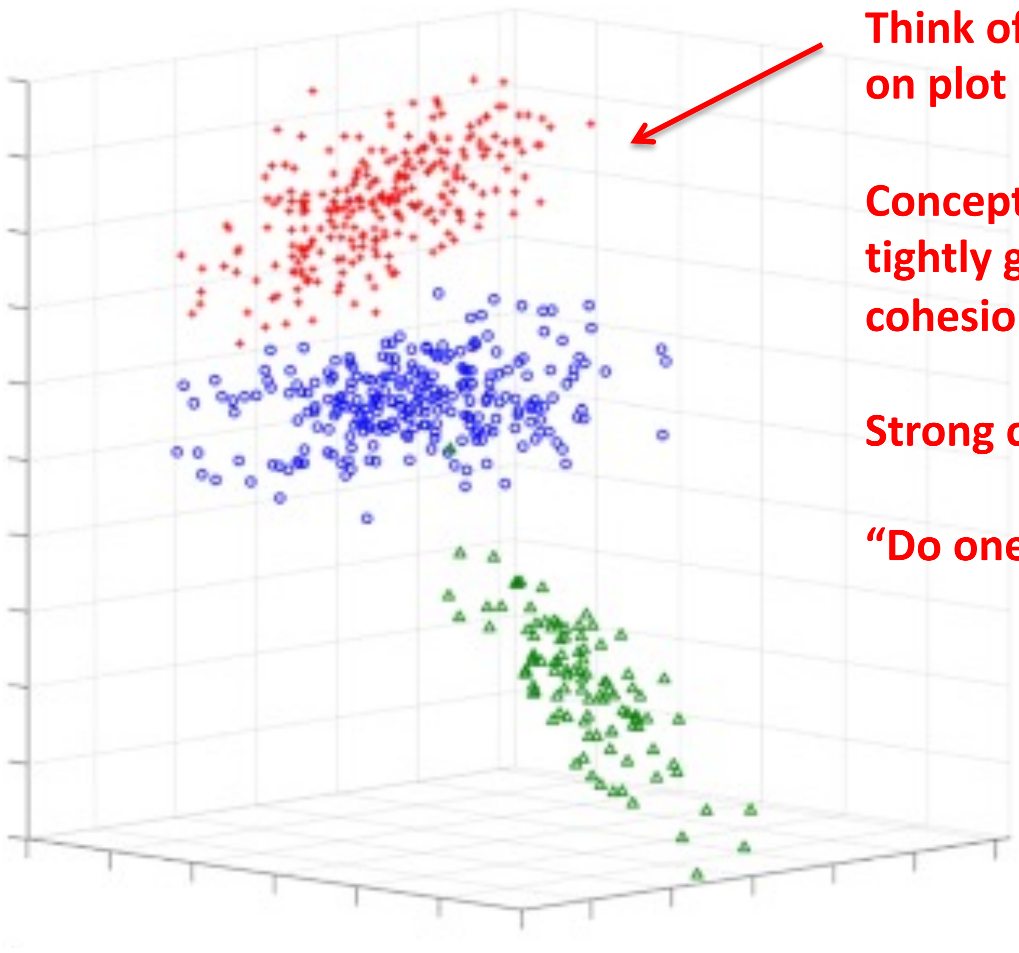
A good routine name:

- avoids nondescriptive verbs (like *do*, *perform*)
- describes everything the routine does
- is as long as necessary
- follows naming conventions!

Cohesion describes how closely operations in a routine are related

Cohesion

Conceptual



Think of lines of code as dots on plot

Conceptually lines of code are tightly grouped with strong cohesion

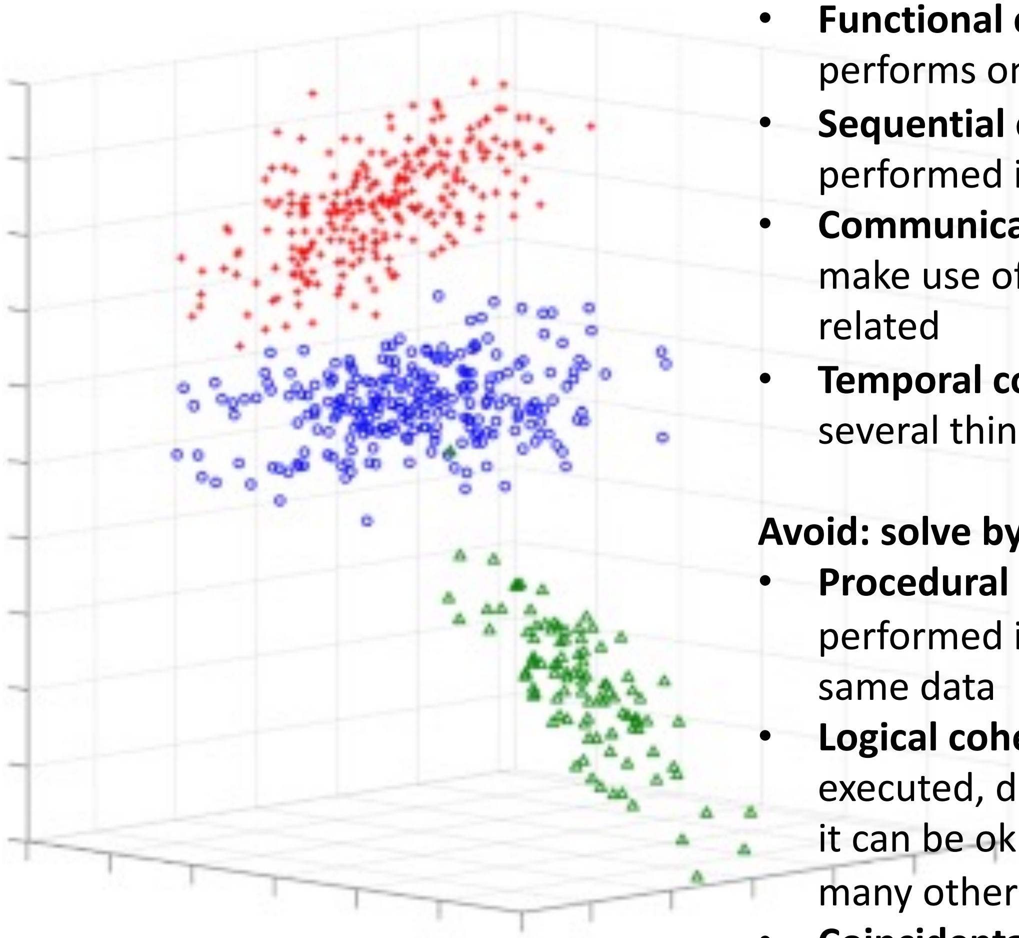
Strong cohesion is desired

"Do one thing, and do it well"

Good routines have strong cohesion

Cohesion

Conceptual



Acceptable cohesion:

- **Functional cohesion** (strongest and best kind): performs one and only one operation
- **Sequential cohesion**: contains operations that must be performed in a sequential order
- **Communicational cohesion**: contains operations that make use of the same data, but are not otherwise related
- **Temporal cohesion**: contains operations that do several things, because all are done at the same time

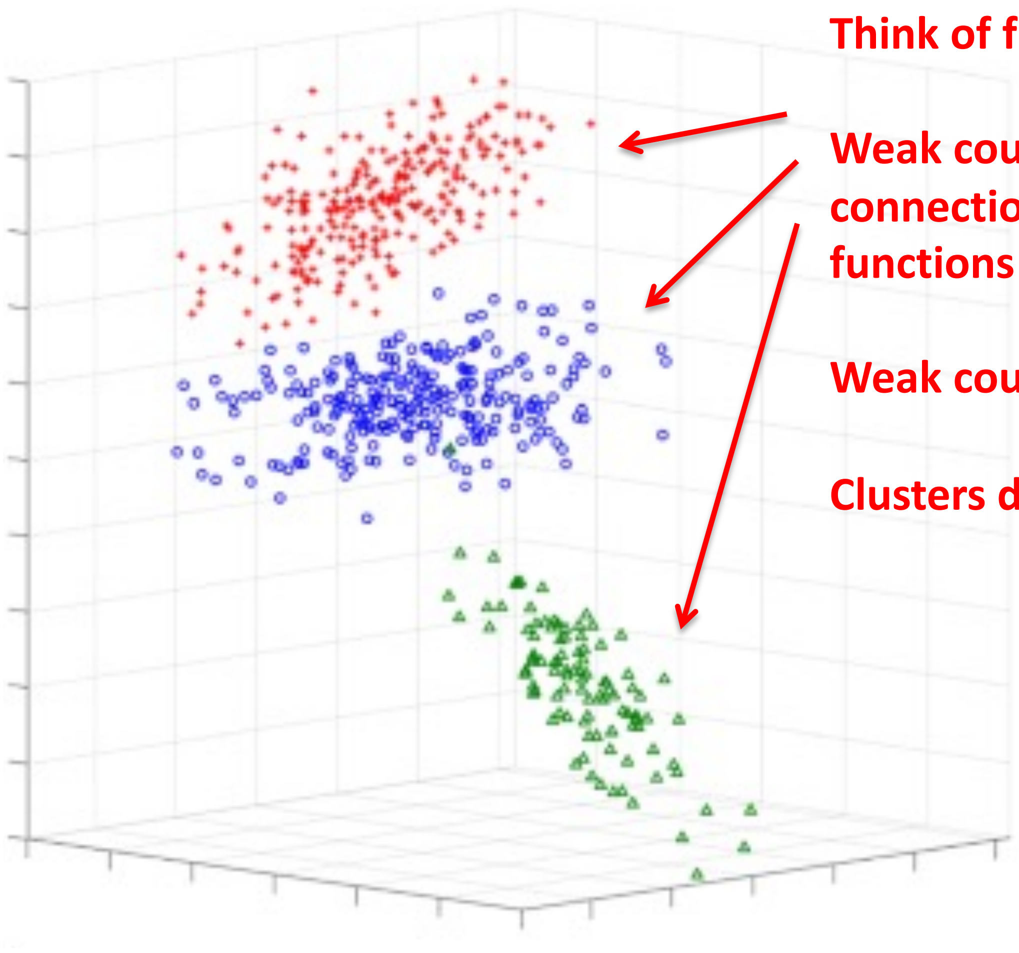
Avoid: solve by breaking routine into multiple routines:

- **Procedural cohesion**: contains operations that must be performed in a sequential order, but don't share the same data
- **Logical cohesion**: several things in a routine, only one executed, depending on a flag parameter. (Exception - it can be ok if using a switch statement to call one of many other (cohesive) functions.)
- **Coincidental cohesion**: no apparent reason for things to be together in a routine!

Coupling is strength of connection between routines; it is a complement to cohesion

Coupling

Conceptual



Think of functions like clusters

Weak coupling has loose connections between functions

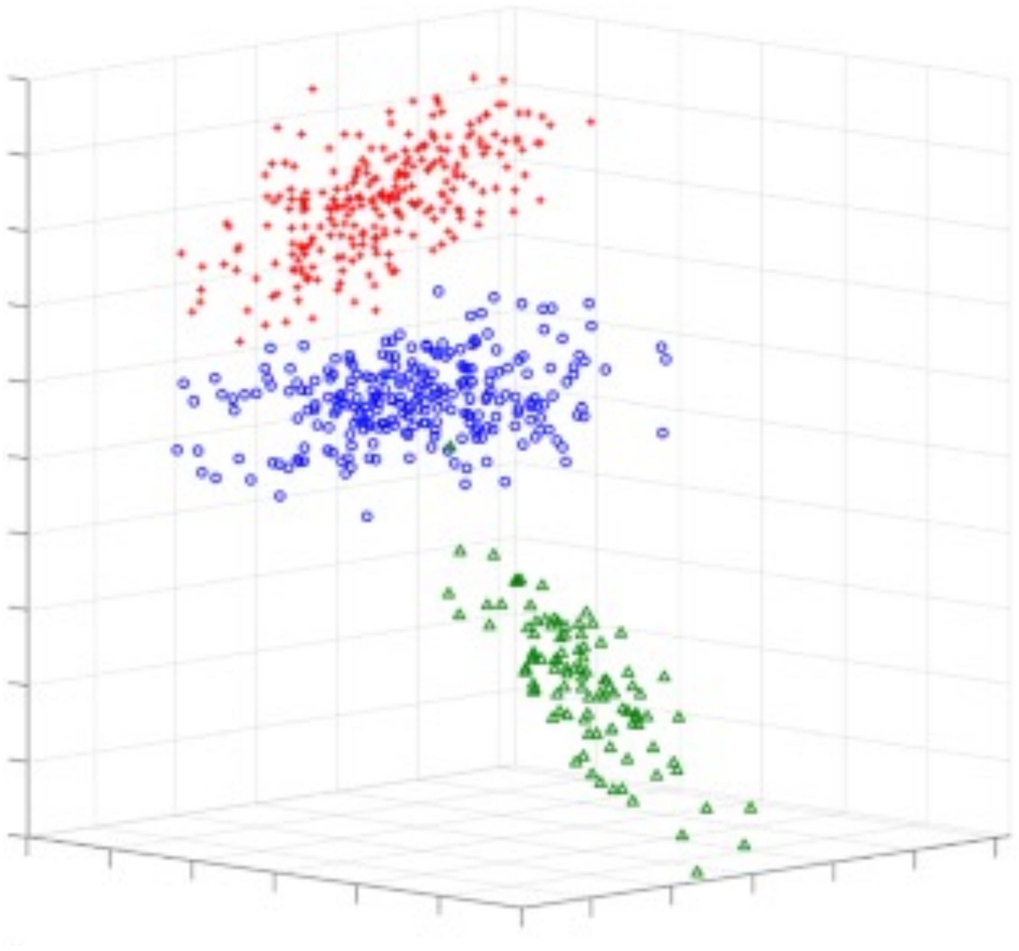
Weak coupling is desired

Clusters do not overlap

Good routines have weak coupling

Coupling

Conceptual



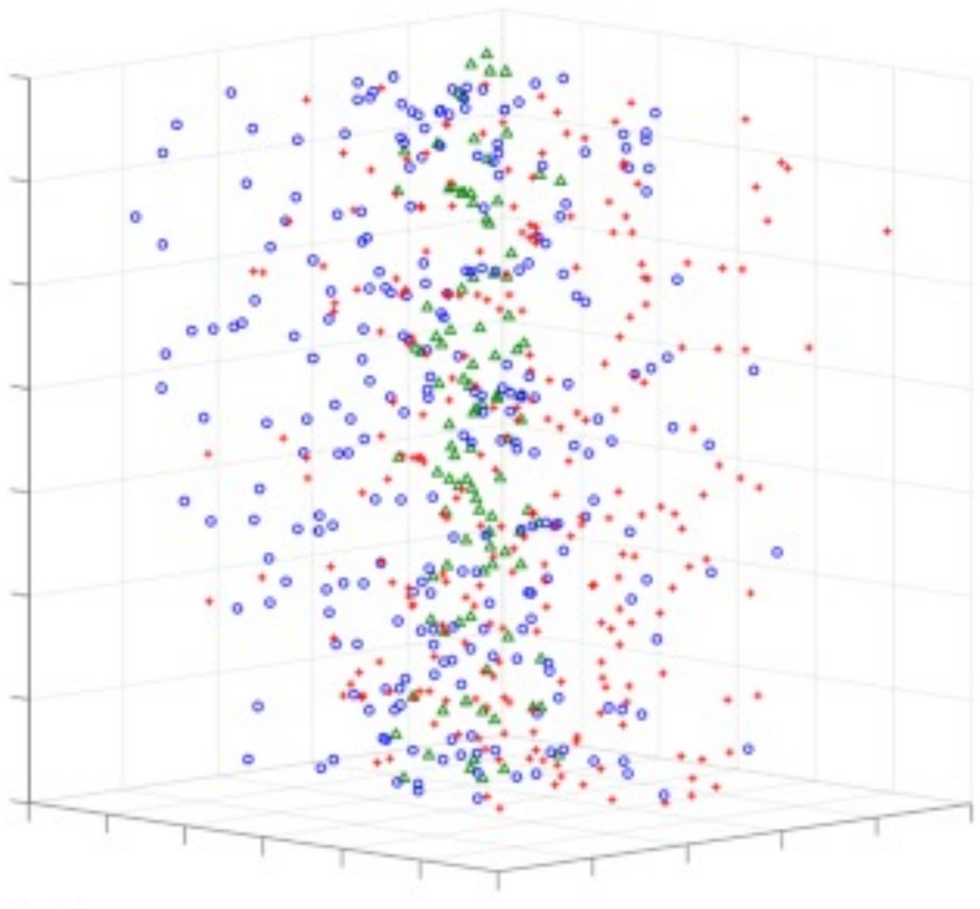
Coupling

- **Simple-data coupling**: the only data passed from one routine to another is through parameters, and is nonstructured
- **Data-structure coupling**: one routine passes a data structure to another; best if it really needs the whole data structure
- **Control coupling**: one routine tells the other what to do
- **Global-data coupling**: two routines use the same global data; may be tolerable if read-only
- **Pathological coupling**: one routine uses the data inside the other routine. (Somewhat hard to do in C and C++.)

Poor cohesion: routines do many things; poor coupling: unneeded connections

Coupling and cohesion

Conceptual



Poor cohesion:

- **Routine do many things that are not closely related**

Poor coupling:

- **Routines have unnecessarily strong connectivity between them**

Example from calendar/datebook application

```c
 * adapted from C++ code used in CS23, January 1996.
 */

// pseudo-code from a calendar/datebook application

typedef struct date date_t;
typedef struct calendar calendar_t;
typedef struct event event_t;

// given one date, compute the next date;
// account for leap years, etc.
date_t* date_next(date_t* day);

// compute the number of days from "this" date until "that" date
int date_ComputeDaysUntil(const date_t* this, const date_t* that);

// return current date
date_t* date_today(void);

// print calendar
void calendar_print1(calendar_t* cal)
{
  // print the month name
  // print the day names
  // for each row, print the dates in that row
}

// print calendar
void calendar_print2(calendar_t* cal, int which, date_t* day)
{
  // if which==1
  //   print the month name
  //   print the day names
  //   for each row, print the dates in that row
  // else if which==2
  //   print the month name, day name, and day number
  //   print the events occurring on that day
}

```

Functional cohesion; good name

Functional cohesion; simple data coupling; good name

Functional cohesion; no coupling; good name

Sequential cohesion, perhaps; data structure coupling; terrible name!

Logical cohesion; data structure coupling; terrible name!

Example from calendar/datebook application

```
// print calendar
void calendar_print3(calendar_t* cal)
{
  // print the month name, day name, and day number
  // foreach event occurring on that day
  //   print the time of the event
  //   print the type of event
  //   print the description of the event
}

void DoPrintBook(calendar_t* cal, const char* intro,
        image_t* frontCover, image_t* backCover)
{
  // print the frontCover
  // print the intro
  // foreach month
  //   foreach day of that month
  //     calendar_print2(cal, 2, day);
  // print the backCover
}

calendar_t theCalendar;          // the calendar we use below

void Initialize(string filename, window_t* window)
{
  // initialize theCalendar
  // for each event read from the file
  //   add event to the calendar
  // initialize the window
  // create some buttons on the window
  // current date =  Today()
  // display the current day in window
  // look for any events in the next hour
  //    pop up dialog box for each such event
  // Update(cal, window);
}

void Update(date_t* today, window_t* window)
{
  // check the current time
  // look through theCalendar eventlist for events on date "today"
```

Sequential cohesion, perhaps; data structure coupling; terrible name!

Sequential cohesion; data structure coupling; name should be calendar_print()

Hidden global variable causes invisible coupling below

Sequential cohesion; data structure coupling; global data coupling; incomplete description

Temporal cohesion; data structure and global data coupling; vague name

Agenda

1. Coupling and cohesion
2. Activity