CS1007 lecture #21 notes

thu 29 nov 2001

http://www.cs.columbia.edu/˜sklar/cs1007

today:
- news
- Software Engineering (read ch 10)
- Command line interface and menu processing
- Search

news.

- my office hours
  - this week only: Thu office hours from 4.15pm-5.00pm (instead of 12.30pm)
  - sign up for office hours outside my office door
- final exam
  - official exam: Tue Dec 18 9am-12noon
  - makeup exam: Thu Dec 20 1pm-4pm
  - you MUST sign up for the makeup exam — sign up sheet is being passed around in class today
- homework #7 has been posted, due Wed Dec 5
  there will be NO extensions!

Software Engineering.

In class, we discussed the "software life cycle", below:

$\boxed{\text{development}} \rightarrow \text{release} \rightarrow \boxed{\text{use}} \leftrightarrow \boxed{\text{maintenance}} \rightarrow \boxed{\text{retirement}}$

There are several process models:

- build-and-fix model
- waterfall model
- iterative model
- evolutionary model

build-and-fix model.

This is the oldest model and probably what you've been doing when you write your home-work.

$\boxed{\text{write program}} \rightarrow \boxed{\text{modify program}} \leftrightarrow \text{release}$

waterfall model.

This model was developed as software evolved into large projects, involving many lines of code, many files and many programmers working together on the same large project.

$$\begin{tikzcd}
\boxed{\text{establish requirements}} \arrow[dr] & & & & \\
& \boxed{\text{create design}} \arrow[dr] & & & \\
& & \boxed{\text{implement code}} \arrow[dr] & & \\
& & & \boxed{\text{test system}} \arrow[dr] & \\
& & & & \text{release}
\end{tikzcd}$$

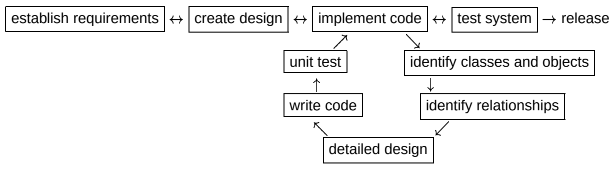
iterative model.

This model was developed after it was recognized that the waterfall model was unrealistic. In actuality, each step can be (and usually is) revisited. This is especially common in large companies, where multiple people are working on the same project and the people who, for example, "establish requirements" are not the same people who "create design" or "implement code" or "test system".

$\boxed{\text{establish requirements}} \leftrightarrow \boxed{\text{create design}} \leftrightarrow \boxed{\text{implement code}} \leftrightarrow \boxed{\text{test system}} \rightarrow \text{release}$

evolutionary model.

This model evolved, again from companies where large software projects are developed and maintained, particularly after the introduction of the “object-oriented” way of thinking. This model emphasizes *modularity* and allows for software re-use as well as testing of individual modules to make sure that each piece is robust and correct before it is added to the whole.



command-line interface and menu processing.

In class, we developed a flow chart to handle typical database processing operations.

These are:

- add data
- edit data
- remove data
- print data

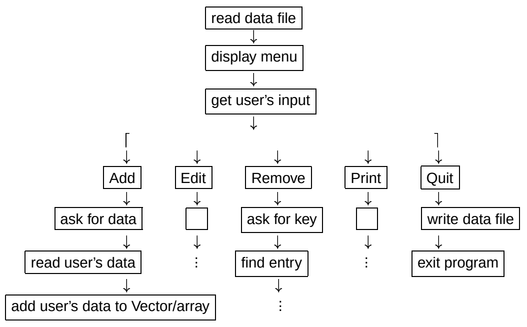
Typically, a program begins by reading the data from a data file – for your homework, this will be a text data file – into variables inside the program (e.g., Vectors or arrays).

Then the program will manipulate the data in the Vector(s)/array(s), until the user is ready to quit.

Finally, the program will write the manipulated data back to the data file.

menu processing flow chart.

We developed this flow chart partially in class...



search.

How do we find an entry to remove?

We need to *search* the data.

We will talk about two search algorithms:

- linear search
- binary search

The idea is that the data contains many fields (i.e., columns in a spreadsheet). One of the fields is a *key* – identifying each row in the spreadsheet or each entry in the Vector/array in which it is stored.

In our application, you will read the key from the user and then you will have to *search* the Vector/array for a match with the user's key and return the index into the Vector/array of the entry that matches.

linear search.

Here is an example of a linear search, using the `Inventory` and `InvItem` classes from the previous lecture. Here we'll use the modification that the inventory `items` are stored in a `Vector` rather than an array (as in the book). Recall that it was suggested in last class that you rewrite the book's example using a Vector, as an exercise.

The key is the `name` field in the `InvItem` class.

The method returns the index of the argument `key` in the `items Vector`, or -1 if the key is not found.

```
public int findItem( String key ) {
  int i = 0;
  boolean matchFound = false;
  while (( ! matchFound ) && ( i < items.size() )) {
    String itemName = ((InvItem)items.elementAt(i)).getName();
    if ( key.compareTo( itemName ) == 0 ) {
      matchFound = true;
    }
    else {
      i++;
    }
  } // end of while
  if ( matchFound ) {
    return( i );
  }
  else {
    return( -1 );
  }
} // end of findItem() method
```