

How to complete a B.Sc. Computer Science project successfully

The main aim of this lecture is to help you organise your work in such a way as to avoid getting to the end of the time you have available to work on your project, only to find that you have nothing working!

Simple model of project:

Acknowledgements

The ideas I present come largely from Bob Dickerson and Patrick Quick, reinforced by my experiences of students who have not done as well in their projects as they might.

Some code taken from SQL for Web Nerds
by Philip Greenspun

<http://philip.greenspun.com/sql/>

Method

- list requirements
- design your data
- sketch interface
- enumerate transactions
- build prototype

The assumption here is that your project involves building some kind of application, system or program.

Make a List of Requirements

Make a list of requirements, ordered more or less in increasing difficulty.

[This is not the same as the list of objectives for your project, nor the tasks you need to perform, though the lists must clearly be related in some way]

Now draw a line part-way down your list of requirements. The line should be at the point where the requirements above the line fit comfortably inside your head.

Don't make the list too long

If you move the line further down than that you risk that the design stages will take too long, you will have little to show by the time of the IPR, and when you finally finish designing you will realise that what you have is beyond your capabilities to implement.

If you move the line down successfully, you may have a better project; but if you are that good you can probably repeat the whole exercise with a bigger set of objectives after you have completed it (quickly) with the smaller set.

A shorter subset is safer.

Design a system to address the requirements

Next, you have three tasks, corresponding to three facts:

- most systems store data,
- most present data on the screen,
- and most are required to manipulate the data in some way:

First, design your data

Draw an ER model that would support your requirements. If you struggle to do this, you may have to move your line closer to the top of your list of requirements. If your line is already at the top of the list, you need to simplify your requirements and split them up into smaller chunks.

You could also normalise any paper forms or other documents relating to the project that you have available, and use these to check your ER model.

Implement your model

So far you have been working on paper; now implement the model on the computer as a database. You may want to use MS Access as a rapid prototyping tool, or you may type in a file of CREATE TABLE commands that you can use with Oracle or mySQL or some other RDBMS. Make sure that you enforce referential integrity and any other integrity constraints that you can (or find a citation that justifies not doing so).

Test your model

Type in some test data. Make up your test data without looking at the database, so you do not constrain the data to that which you know will work. For example, type in several orders, products, customers, etc., making sure (for example) that a customer can order the same product on more than one order.

If you use a GUI to type in the test data, see if you can save it to a script that can be run again later if you have to re-create the database.

At the end of this process you should have a database designed and implemented that supports a subset of your requirements.

Design the User Interface

The second stage is to design your user interface. Start with sketching out a storyboard on paper. Once it is as good as you think you can make it, you should consider implementing it as a mock-up on the computer [I think this is called a non-functional prototype]. You can create your forms/screens/pages using some kind of GUI builder, with HTML (by hand, or using a tool like Dreamweaver), or using VB. Ideally these can be linked in the same way as they will be in the real system, but even if you print them out you can explain how the interface will look.

Cross-check

You can check the user interface against the requirements: does it allow the user to do everything that is needed. And you can check the user interface and the database design against each other: does the database have the data that the user interface requires, and does the user interface give suitable access to all the data.

Transactions

The third stage is to list the operations that the system needs to perform. Initially these can be listed on paper, but if you have a database, you can translate them into SQL statements. Again you can cross-check: does the database support all the transactions, is there a way to invoke each transaction from the user interface, does each operation in the user interface have a transaction defined.

Design Complete

You now have a design. But more than that you have a design that is tested, you have evidence that you have a design, and you have evidence that it is tested.

Implementation

Implementation is now a matter of arranging that when an operation is invoked on the user interface, the correct SQL command is executed on the database. This can be quite hard if you have never done it before, and is the point at which I have seen numerous students come to grief in the past.

So we want to get to this point by December, if not before.

Easing the implementation

Given that implementation is hard, how do we make it easier? By implementing something very small: a “hello world” project. In parallel with the design work (or before or after if you prefer), take your very first requirement, or even a part of that requirement. For example, list all customers, or list all modules, or list all parts, ... Implement just that in your chosen implementation language. Doing so proves that you have all the required software installed and working, and you have the basic idea of how to connect your programming language to your database, how to implement a user interface in that language, and how to make it all work.

If you can do:

```
select * from users;
```

then it is only a bit harder to do:

```
select user_id, count(*) as how_many  
from bboard  
where posting_time + 60 > sysdate  
group by user_id  
having count(*) >= 30  
order by how_many desc;
```

or

```
insert into comments_audit
(comment_id, user_id, ip_address,
audit_entry_time, modified_date, content)
select comment_id,
       user_id,
       '128.16.173.52',
       sysdate, modified_date,
       content from comments where
comment_id = 836;
```

OK, quite a lot harder, but the difficulty is local, and no longer a show-stopper.

What next?

After the “hello world” project, implement the first set of requirements, then revisit the requirements list, redesign the system to incorporate some more of the requirements, implement them, and repeat.

Overall Structure of the Method

