Software engineering

Software engineering requires a disciplined design method to develop, test, and maintain software. Most of the process occurs long before you choose a language, or write any code. The first step is an interview of the principle clients, the ones paying for your services, and their employees, who will use the software you create for them. Your goal is to determine exactly what the client wants the application to do, and how the employees will use it. It is important to get input from both groups. Managers don't always understand exactly how the job gets done by their workers. You, as the designer, want to create a tool which makes the employees more efficient, and the managers more effective.

Once you have refined your notes from your meetings you begin your design. You examine how the data flows through the system. Determining data structures comes later but here is where you start grouping the data into manageable units. Your goal, is to find one or more data structures to contain and manipulate your data. You need to examine how the data is input, processed, and stored or displayed. Which is the most efficient way to store those data structures? Which data structures best fit any sorting or searching you may need to do? Which algorithms best fit your problem?

When you get some idea of the data flow, it is time to start thinking about the flow of control through the program. Is there an initialization module followed by a loop responding to input? Are you querying an external database which requires handshaking? Is the user in control, on a command by command basis, or do they provide batches of commands for the program to perform in sequence? This is your control flow analysis.

Now you have a better idea of how all the parts fit together, with choices of data structures and algorithms which would be best for the project. You are now ready to create a top down design of your application. Each portion of each module is written in pseudo code - careful descriptions of what is done at each step. You want to keep track of both cohesion and coupling. Cohesion is a measure of how each module fulfills one and only one task. Coupling is a measure of the interaction between modules. Limit global variables to as few as possible. Your pseudocode, along with your data flow and control flow charts, should be enough to give you an idea of how the modules fit together to process information. You want to pass as few variables as possible, while keeping each module as simple and straightforward as possible. One task per module is the goal (most cohesive).

When you have finished this step you can think about which language would be the best fit to your problem. Yes, it is wise to hold off on language selection until this point. Don't let your preconceived notions get in the way. Simply because you know one language well does not mean it is the best choice for every problem you encounter. You are free to design the best solution, to any problem, by creating a detailed design before you choose a language. Your goal is to create a tool which is invisible to the user. They should not need to think about how to perform the next step. They are skilled at their job, you are simply enhancing their skills. You want every click, and key stroke, to be automatic and

intuitive. If the user thinks they should be able to do something a certain way, you should accommodate them. This was why you interviewed the workers when you began the project. Your goal is to help them, without getting in their way.

Once you pick your language you decompose each module of your pseudocode into smaller and smaller chunks. When you can write each line in the language of your choice you are ready to code. At this point writing the application is easy, expand your notes into working code.

Part of your design stage is to foresee the maintenance of your application. You need to provide extensive annotation of the code in each comment section. Any fancy coding requires extra commenting. Prepare your test data, and test harnesses, so you can test your application repeatedly throughout its lifetime. Plan for future expansion. How can the code be enhanced to better assist the company? With a little forethought you can make your job easier. Remember, those comments will help you in the future, as well as any other programmer who needs to work on the code. In six months you won't remember all the details, while a new programmer will be thankful for all of the notes you have placed in your source files. Ask the managers and workers to give you sample data which you can use to create test data sets. Review the tests with them after you have a working application.

Meet with your clients often. Determine whether you are going in the right direction by showing them working code often, even if it is only a framework. Limit their change orders at this point to minor details. Any major revisions should be noted but left for later. It is important to keep to your schedule. Adding new features, late in the process, will make finishing the project more difficult and more costly.