Lesson 7: Simplification through abstraction and refinement

This book teaches how to write programs in an object oriented way. There are many ways in which computers can be programmed to solve problems. The object oriented way is just one of many. The object oriented approach makes it (relatively) easy to think about the problems we want to solve in terms that make sense to people, rather than in ways which make sense to computers. This is not to say that object oriented approaches can solve more or different problems than other approaches: the Church-Turing thesis holds that this isn't possible. Rather, our object oriented solutions will likely have a more direct connection with the original problems than other solutions would have: object orientation allows us to model the problem domain fairly transparently. This makes our programs easier to express and to understand than they would be if written in other ways.

An important aspect of any high level programming language is that it insulates the programmer from the details of the underlying hardware. A high level programming language thus abstracts away the details of the hardware, allowing a programmer to express a solution to a problem in simpler terms.

The word "abstract" strikes fear into many people because there is a perception that things which are abstract are difficult. Where this comes from we don't know for certain, but whatever the root of this perception, the process of abstraction does not make things more difficult, but rather it makes things simpler! Abstraction allows you to ignore irrelevant details. This means that the more abstract something is the simpler it is.

In fact, abstraction is a technique that you probably use every day. It is a very human way of dealing with complexity. For example, think about how you might describe how the human body works. You would probably not start by describing how it works at the cellular level. Instead, you would start be describing major systems of the body (e.g. the digestive system, the cardiovascular system, the nervous system, and others). Your first approximation of a model of the human body might therefore be something like this:

- Digestive system
- Cardiovascular system
- Nervous system
- etc.

At this high level you might describe some of the ways in which these systems interact with each other. For example, the nervous system indicates hunger when blood sugar levels fall too low. When a person eats, their digestive system converts food into nutrients which are picked up by the bloodstream, which causes the nervous system to signal that the body is satisfied.

Each system in this high-level view can be refined all the way down to the cellular level. By refining from the abstract (simplified) view to the concrete (detailed) view, the details of each subsystem are presented in context, and independently of the details of the other subsystems. This makes the whole system easier to make sense of. We will want to do something similar when understanding and building computer programs.

As another example of this process of iterative refinement, consider what you do when you write an essay or a report. You have probably been taught that a good way to help you develop and organize an essay is to start with an outline of the essay, and refine it in several steps until the various sections of the essay can be written.

As an example, consider using this approach to organize an essay about the effect of technology on democracy. A top-level outline might look as follows:

- I. Introduction
- II. What is democracy?
- III. What is technology?
- IV. How does technology interact with democracy?
- V. Effects of technology/democracy interaction
- VI. Conclusions

This top-level organization is not very detailed, but it does two very important things. First, it highlights the main components of the essay (e.g. it tells us that both democracy and technology need to be defined for the purposes of this essay). Second, it gives an order for the sections of the essay. For example, it shows that both democracy and technology must be defined before their interaction can be discussed.

This outline needs to be refined before the essay can be written. Each section can be refined, independently of the others. As an example, after some more refinement the essay outline might well look as follows:

- I. Introduction
- II. What is democracy?
  - A. One person, one vote
  - B. Secret ballot
  - C. Freedom of expression
- III. What is technology?
  - A. Encryption
  - B. Audit trails
  - C. Electronic eavesdropping
  - D. User interfaces
  - E. The Internet and the WWW
- IV. How does technology interact with democracy?
  - A. On-line voting
  - B. Verifying identity
  - C. Privacy
  - D. Anonymity
- V. Effects of technology/democracy interaction
  - A. Greater voter participation in elections
  - B. Is the secret ballot secret?
    - i. Vote buying
  - C. Election integrity
    - i. Losing votes

ii. Phantom votes
iii. Tampering with election results

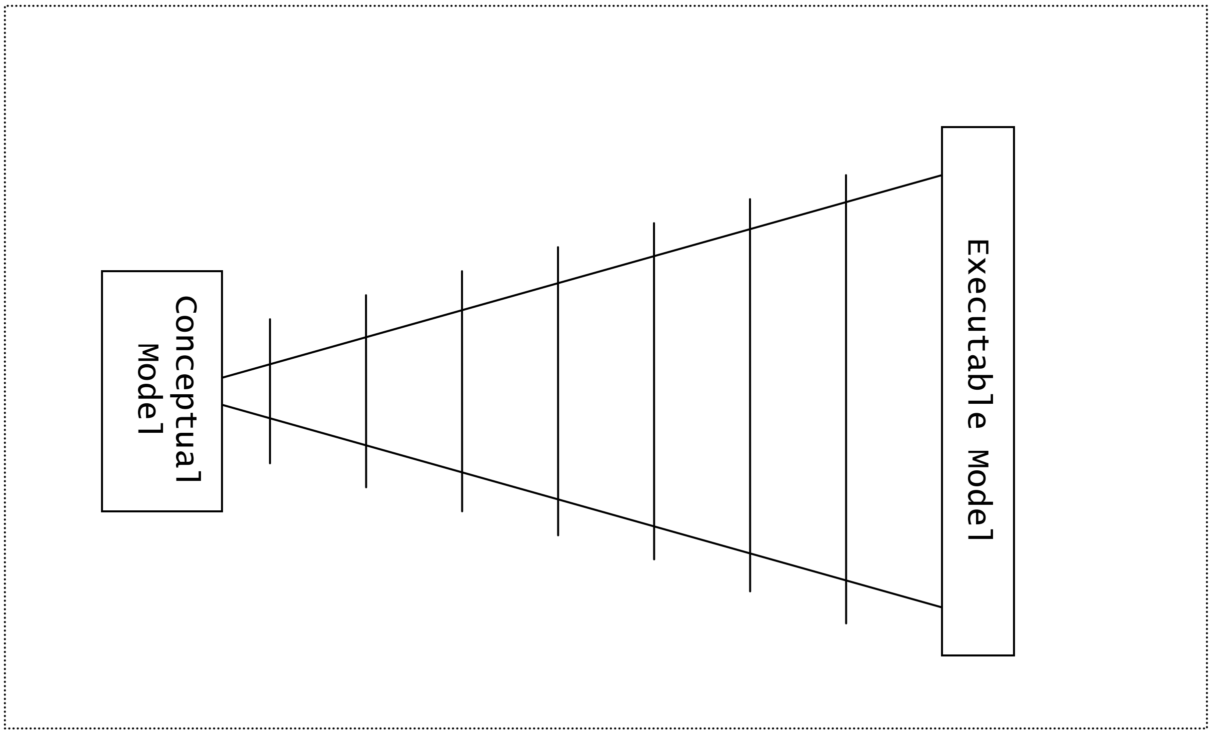
VI. Conclusions

This outline clearly contains more detail. Refining it further may well force a re-ordering of topics. Eventually the essay can be written. A well-written essay introduces ideas in a logical order and draws well-supported conclusions. A well-written essay clearly doesn't appear out of thin air – it is carefully constructed step by step.

Of course, there are other ways to write essays. This outlining technique may not be helpful for all kinds of writing. When writing fiction, other ways of iteratively refining the structure of a piece of writing may be helpful. Major plot points can be outlined and ordered while characters can be listed and developed in parallel. The point is that having some process to follow is crucial to being able to solve large problems.

We can follow a similar process to write computer programs. The outline for a computer program is bit different from the outline for an essay. To help emphasize this we call it a model rather than an outline. The first model that we produce we call a conceptual model, because it mirrors how we conceptualize the problem. As we refine the conceptual model it becomes closer and closer to what we call an executable model (since it is expressed in a precise enough way that it can be executed by a computer).

The following diagram is intended to show the progression from an abstract conceptual model to a concrete executable model. The model "fans out" to indicate that there is more detail in the executable model than there is in the conceptual model.



What is a conceptual model? The conceptual model is a high-level outline of the main components of the solution to our problem. How do we arrive at this conceptual model? The conceptual model is a model of the problem domain. We therefore arrive at a conceptual model by constructing a model of the problem domain. When solving any sort of problem the first thing one should do is figure out the domain of the problem. The domain is the set of things which are relevant in some way to the problem.

Example 1

Suppose your problem is that you want to invite some friends for lunch. You need to figure out who to invite and what to serve them. What is the domain of this problem? The problem domain presumably includes the set of your friends, and their food likes/dislikes, their food allergies, what food you have in the house, and so on.

Things which are not relevant to solving the problem do not belong in the domain. For instance, things like the age of the roof of your house, or your friends' hair colors, are not relevant and therefore do not belong in the problem domain.

Example 2

Consider the problem of modeling a human being for the purpose of simulating the effect of medications on them. It is probably relevant to this problem to include in the model the weight, age, height and gender of the human being, and so these characteristics should be included in the domain. The person's hair style, clothes and musical tastes are irrelevant (for the purpose indicated), and so don't need to be included in the domain.

Part of the art of being a good problem-solver is being able to pick out what the domain of a problem really is. In many so-called "textbook" examples the domain of a problem is given to you, or is very easy to pick out. In real-world examples, this is not always the case.

When faced with a novel problem to solve, think about what the domain should be. Do not worry about getting it exactly right the first time: problem solving is an iterative process. A process is iterative if it is repeated many times. In the problem solving context we continually refine our ideas and approaches as we learn more about a problem. In other words, since problem solving is a process of repeatedly refining a potential solution, it is not critical that our first attempt is 100% correct.[6]

[6] You can think of a problem solving process a little bit like golf, if you're familiar with the game. In golf you try to hit a golf ball from a starting position into a little hole. The hole is typically several hundred yards away from the starting position. The ideal is to hit a hole-in-one: to get the ball into the hole with one hit. This almost never happens, and is mostly pure luck when it does happen. The reality is to keep moving the ball closer to the hole with each hit. Not every hit moves the ball directly towards the hole; sometimes the ball goes from the starting position towards the hole in a bit of a zig-zag pattern. Problem solving in general is often like this. Rarely can you go directly from a problem to a solution in one step – it is as unlikely as a hole-in-one. More typically solving a problem is a multi-step process, which doesn't always move you directly toward the solution.

The process of getting the golf ball into the hole is an example is a program. If the ball is already in the hole, we're done. If not, hit the ball towards the hole, and repeat. An important question to ask it whether this program, when followed, will terminate (i.e. can we guarantee that the ball eventually gets into the hole?) Can you think of a condition which, if imposed, will guarantee termination?

It is interesting to note that this is also the way that the scientific method works: scientific knowledge progresses via a process of iterative refinement through experimentation (data gathering) and data analysis (examination of the experimental results). You can think of determining the domain of a problem as an experiment. You hypothesize what you believe is a good domain, and then test it (try to solve the original problem). If the domain is not good enough, the experimental results force you to revise your hypothesis.