How to develop complex software

You need some kind of complexity to make things simple.

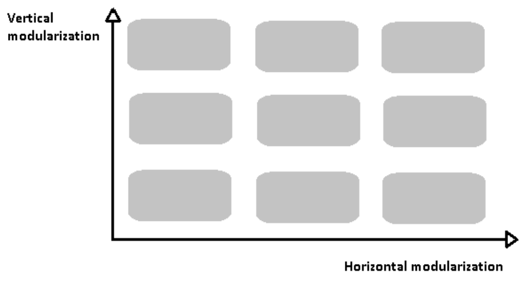
There is no limit to the complexity you can manage, as long as you keep it separated in many tiny black boxes. You can then assemble boxes at different levels and build extremely complex structures.

A mason doesn't have to worry about building a brick or a window, he just assembles them. A developer assembles software more and more complex, relying on blocks of functionality.

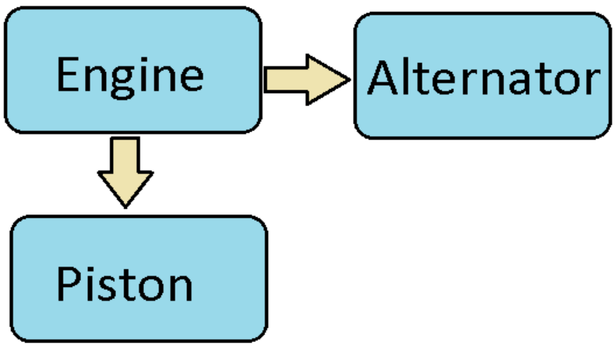
This means building funtionality on the top of other functionality: this is called abstraction.

However, it's also possible to abstract too much. The criteria is that a lower entity is considered as such, only when it requires a different type of input, being inherently different. If you find youself passing the same parameters to different levels, then think about integrating those layers into one. Then, if needed, break horizontally instead of vertically.

Your code should be both modularized vertically and horizontally, into small, loose coupled boxes.



Let's take a car, for example. The work of the engine is built on the top of the pistons: this is vertical modularization. The separation is justified by the fact that the input of the engine is different then the input of a single piston.
However, the output of the engine is processed by the alternator, as they work at the same level of abstraction: this is horizontal modularization.



In the case of vertical modularization, the calls are nested:

engine.run(...) calls piston.move(...)

In the case of horizontal modularization, the first call must terminate, before a second call can start:

engine.run(...) terminates and the output is passed on to alternator.convert(...)

Don't confuse the "call diagram" above with a class diagram. Each call can appear at different levels of abstraction, but you can only have two types of calls: in sequence or nested. This can be represented as an "horizontal-vertical matrix". As long as each cell is a separate and indipendent block, belonging to an indipendent module, your code will stay simple and will be easy to maintain. Modularization is nothing else that the common denominator of concepts like loose coupling, separation of concern, choesion and encapsulation. But modularization is the ultimate goal to manage complexity.

It's widely known that, in proportion, simple projects have fewer bugs, than bigger projects. That happens because the latter haven't been sufficiently modularized into small pieces.

Your mind is unable to understand thousands of lines of spaghetti code, at the same time. However, it can understand a simple module, that has limited consequences on other parts of the software.

Create sub-projects, create libraries, use project modules, packages and well defined classes to keep functionality separated and indipended. Do that both vertically and horizontally. Refactor spaghetti code by spotting related parts and extracting modules. Do all that constantly, and you will have bug free, mainteinable code.