Modularized coding:

There are so many coding principles out there most of them explaining how to write a high quality code that will ease in maintenance and expansion. But few of them were useful to me to do actual debugging. I know that you can use debugger and logging statements to figure out where the problem exists, but these will just tell you the problems that happen due to coding logic errors, not feature level errors. What I mean to say is that you can almost never cut of an integrated feature from your code without going back to the pre-existing code. All the features that are implemented should be standalone applications which are integrated via a spring process and thus can be equally easily removed on causing troubles. This is also what MC using tries to achieve by removing user authentication, content management and mapping requests code to annotations rather than building up if/else. A block of code should be as atomic as it can be. All other conditions of it being executed or not should be handled in separate entities. Whether a piece of code gets executed or not depends on if the state machine snapshot matches the requirements. This might be an advanced form of polymorphism where compiler is burdened with selecting the order and execution of pieces of code. For every state that is possible there would be a different block to execute. The advantage of this approach is that we separate out the flow logic and execution logic. It will help in debugging as any of the state machine snapshots can be remapped to different case, thus controlling featiures on higher level. It might also increase reuse as every operation is designed to be a complete entity in itself.