1. Think of this: you design a piece of software to have minimal downtime (for updates, fixing bugs, etc.), but you also want to make the most secure software available to you (because who wouldn't, right?) At some point, these needs are going to clash. To be secure, software needs to be able to hold its own against intrusions. To make this same piece of software more secure, it usually has to go down to either plug a hole somewhere or to update is resiliency to malicious attacks. This downtime, also, most likely scales to how many security actions need to be taken on the development side. There will be an equilibrium point, for sure, but there will always have to be a trade-off in which requirement gets more focus at any given time.

2. I like the idea of using a blackboard architecture pattern for a music selling and distributing Internet service. The actual architecture would need to be multi-layered to deal with the vast sorts of interactions between users and the system itself. The blackboard layer would hold all the songs that are available to send out to users and some sort of algorithms for determining what to show to people to expand what they buy and have sent to them. The knowledge sources for this architecture would hold each individual person's account, including their user information (payments, settings, etc.), that would tailor their experience to them specifically. The control component for this system would take care of loading a specific user's data to their interface and allow access to their account and all it's functionality based on their choices and purchases in the system.

3. Activity model of ATM Withdrawal

Card Reader	--------------->	*Get User Info*	--------------->	Account Balance	--------------->	*Show Balance*
						↓
Dispense Amount Specified	<---------------	Cash Dispense Motor	<---------------	*Check Available Funds*	<---------------	Input Withdrawal Amount
↓						
Show New Balance						