Design Paradigm Choice and Justification

For our chess program, we chose a component-level design paradigm. This was because it is intuitive to abstract the different parts of our program (both back end and front end) as components with unique functions that interact with each other in various ways. For example, in our chess program we could have different views of the chess board (within the UI) depending on which player's turn it is. Both views could be their own components. Once a player interacts with the board UI by making a legal move, the view changes to accommodate the other player's perspective. So these two UI components are connected by a bit of logic. Both of these views could be located under a "views" component, which in turn is located under a "UI" component. This is another strength of the component-level design paradigm: decomposition and modularization. Particularly, it may be useful to separate components by whether they are directly associated with UI or internal logic (the implementation, such as the game rules). Another useful categorization of components is whether they are event providers or consumers. Modularization, in turn, allows for high cohesion within a more high-level component (like UI) while still maintaining the ability to couple components between UI and internal logic when needed. Finally, the component-level paradigm allows our team to separate concerns/distribute tasks more effectively. Particularly, the distinction between interface and implementation is again useful, because it allows several members of our team to work on one of these areas or the other while also being able to see how the subcomponents they are implementing in one area affect subcomponents in the other.