## Software Architecture: Choice and Justification

We chose to structure our chess program around an event-driven architecture. This is because our chess game is mainly constituted of a UI and internal logic which handles events that result in changes to the UI. So the UI and internal logic can naturally be treated as event producers and consumers, respectively. This architecture also gels with our chosen design paradigm (component-level design) because within the event-driven architecture each component can still maintain their distinct functions, be modularized and categorized, and be arranged in a way which still maintains every advantage of component-level design. The separation of components into producers and consumers just adds another way to modularize the components, and the event bus provides a more structured way of representing connections between UI and implementation. An example of how this could work is when a player clicks a square to move a piece to. This generates an event which, based on the connections defined in the event bus (which is not necessarily a program but is a model of what each event does), is handled by the appropriate component of logic (a consumer). This consumer, in turn, affects the UI in different ways, depending on the selected piece, square, and legality of the move. It could notify the player that an illegal option was chosen, or it could show the selected piece making one of the legal moves and change the view for the second player. Defining these connections and their possible outcomes is necessary for our project, and the event-driven architecture facilitates this nicely.