Zaki Ahmed

Prof. Narayanasami

CS 3354.007

26 February 2025

Homework 4

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Design for change** | **Separation of concerns** | **Information hiding** | **High cohesion** | **Low coupling** | **Keep it simple & stupid** |
| **N-tier** | **X** | **X** | **X** | **X** |  | **X** |
| **Client Server** | **X** |  | **X** |  |  |  |
| **Main program & subroutine** |  | **X** |  | **X** | **X** |  |
| **Event- driven** | **X** |  |  |  | **X** |  |
| **Persistence framework** | **X** | **X** | **X** | **X** |  | **X** |

**N-Tier** achieves Design for Change and Information Hiding due to its inherent multi-layered approach; lower layers typically cannot interact with upper layers, and each upper layer hides the layer beneath it. N-Tier also manages to achieve Separation of Concerns as a result of dedicating each layer to a specific concern. This ends up being a catalyst for High Cohesion and Keeping It Simple and Stupid since each lower layer is directly dependent upon the requests from an upper layer and there typically being a one-to-one correspondence between use cases and controller objects.

**Client Server** reaches for Design for Change and Information Hiding due to how features added to the server tend to have minimal effect on the user and because the implementation details of the server are hidden from the user.

**Main Program & Subroutines** manages Separation of Concerns through its defining usage of subroutines. The main program will call upon trees of subroutines, with each being delegated to a specific task. As a result of being wholly reliant on subroutines, this architecture usually has Low Coupling as each routine stays independent from another. Finally, this style also achieves High Cohesion because certain subroutines will depend on another’s output as its input.

**Event-Driven** necessitates Design for Change and Low Coupling in its implementation due to how nearly every stage within it is state driven. Each component is distinct and must be designed and tested in isolation from every other, such that each part’s integrity is guaranteed, hence the Low Coupling. This fundamental design also results in room for changing a system’s design since each component is independent.

**Persistence Framework** realizes Information Hiding since it occludes implementation details of the database from the user. This framework also achieves Design for Change since changes made to the database system have minimal bearing on the business objects. It follows that there is a clear Separation of Concerns from this; business objects and how they are designed are very distinct from the database itself. There is a need for High Cohesion from these specifications since each user request must communicate with the DB Manager and that must execute any following requests through its internal implementations to efficiently retrieve/store data from the user. Keep It Simple and Stupid essentially follows as a result of the aforementioned specifications; to keep such a complex system efficiently running, objects within it necessarily should be simple.