**Critique\_2**

**By Anvesh Tummala (16173144)**

**Critique on “An Introduction to Software Architecture”:**

This paper is about how architectural design impact software design. To present this view author gave us few architectural patterns, their context, constraints, benefits, drawbacks on the system design.

These are few interesting points that I agree, disagree and doubt in the paper.

* In saying “pipe and filter systems are not good at handling interactive applications because of their transformational character”, I don’t feel much justification from paper. When we look at a filter, it takes the input and gives us the incremental output which seems a best fit for interactive applications. But I don’t understand how transformational character is effecting the incremental updates of output.
* I agree that in object oriented design, objects should know identity of other objects for interaction. It effects the changeability of system. But we can simplify the problem using abstract interfaces by defining interface types for the objects. In future even the identity of invoking object changes, there is no need of changing the interface types that we use for accessing that object. But it does not possible to have zero knowledge of identity of other objects, not even interfaces.
* In event driven design, list of events maintained by system is becoming centralized. If the central component which is maintaining the subscription of all events is corrupted, whole system fails.
* Layered system reduces the complexity of a problem by level of abstraction. But, the paper haven’ discussed the choice of top-down or bottom-up approach to look for abstraction and also modularity of levels. These two factors have impact on the production time and system performance respectively which is obvious.
* The differences between blackboard and event driven design is primarily how events are triggered, in blackboard system central state is the cause for event triggering whereas in event driven design, components gives the list of events it makes. There is no need of list of events to be maintained in blackboard. In both cases the subscribers, knowledge systems are invoked implicitly. In event driven the data transfer is a problem but in blackboard we have central data structure. In both cases there is a central point of failure (list of events registered in system, central data).
* One of the interesting aspect is each pattern has its own constraints that effects the shape of systems architecture. As an example, when we are using a dashboard design it is more likely to take a star topology. This means you can understand the structure of the system well before it is being developed.
* From the KWIC case study the main essence is we can build any system in more than one architectural styles, but the choice of styles will be better made when we understand the constraints, context, and benefits of architectures as this allows better choice of design alternatives.

**Critique on “On the Criteria To Be Used in Decomposing Systems into Modules”:**

This paper highlights the effectiveness of modularization is dependent upon the criteria used in dividing the systems into modules. To illustrate this, the author has given an example of decomposing KWIC system in conventional -*the sequence of steps to perform* and nonconventional -*information hiding* criteria’s.

Even though both systems are identical after assembly of modules there are some major differences in the benefits expected from modularization.

* Managerial – development time should be shortened. The interfaces in the first method are complex formats and table organizations. The decision on these table structure needs joint effort from several development groups, hinders independent development and shows bad effect on development time. Whereas in second method the interfaces are more abstract and simple, that means development of modules can be started earlier just by knowing function names, type of parameters.
* Product flexibility – it should allow drastic changes to one module without affecting others. In the first method, the change in the format of the line storage effects all modules as they all depend on the format of the storage. Whereas in second method the knowledge of the format of line storage is hidden from all the other modules. So, any changes in the format of the line storage has no effect on other modules. I like this difference that comes from the beauty of choosing the criteria “*information hiding*” while modularization in second method.
* Comprehensibility – it should be possible to study the system one module at a time. I obey with the author that first system is only comprehensible as a whole, as every module needs some understanding of other modules due to dependence on the constraints caused by other modules (like table structure). But he don’t explained about second module. I believe second module is not completely comprehensible as there are still dependence on other modules but its comprehensibility is better than first method because of the abstract interfaces.

“Module is considered as a responsibility assignment rather than a subprogram”, this is best seem in the second method over first method because all the modules are more independent, not depending on the decisions from other modules, this makes them more responsible assignments.

I am really fascinated by the above observations, it is just the choice of criteria “*information hiding*” that gives more benefits of modularization in second method. But, I don’t understand is it always information hiding a good criteria or just for this specific example or are there any other criteria’s we can choose. I believe it depends on the system we are developing. The author might have given some other criteria’s that are best fit for specific systems to give broader understanding of choice of criteria.

In conclusion, the author want to emphasize that each module should be divided based on knowledge of deign decisions (secret decisions has to be made, make it a module) it hides from other modules also the interfaces should reveal little about its internal implementation.