**DD TO BE ANALYZED**

SECTION 2

Regarding the most important design principles, in our opinion the document tries to keep the level of abstraction high, using interfaces to provide services and giving a non-strict description of the three-tier architecture. On the other hand, the object diagram is too much specific, since it defines all methods and attributes of the components, resulting also a bit difficult to understand. Besides this, the main weakness of the document is that there is no mention of Qualities of Service and how these have affected the design process and the architecture, also considering their relevance (24/7 availability, encryption of all the data, etc). While all the services are split in different components (applying correctly the modularity principle), there are too many dependencies between them (which means, high level of coupling). The introduction of subsystem could have been a positive solution (see the strengths and weaknesses of section 6)

OK:

* Different tier
* Sequence diagram and object diagram
* Interfaces 🡪 keep the level of abstraction ad high as possible
* Design for testability

KO:

* QoS
* Design for portability 🡪 only android? (coerente però)
* Coupling 🡪 many components but too much dependencies??? 🡪 subsystem???
* Design defensively ?? // design for reusability
* Object diagram good but too much specific

SECTION 3,4,5

In our opinion the main strength of these parts is the use of pseudo code in describing algorithm, which gives freedom to programmers: unfortunately, the algorithm in section 3.5 (Check to find slot for break in the schedule) is wrong: instead of determining if there is enough time for a break in the given schedule, it checks if, for a break of given start time and end time, there are no overlapping meeting. Another positive side of the document is the accuracy in coupling requirements and application components in section 5 [Requirements traceability]

OK:

* Traceability
* The use of pseudo code [added by pon]

KO:

* Pseudo-code

SECTION 6

In our opinion the strength of this part is the way the plan for implementation and testing is elaborated: it correctly avoids presenting test cases, while it states that components are implemented in different time periods depending on their relevance and testing issues (with good motivation of choices), integration tests start as soon as possible, the bottom up strategy is coherent with the architecture. The only weakness of the solution presented is that this strategy is not presented in detail in the final diagrams: it does not show the phases of successive integrations to reach the final test of the entire application from the single integrations of components.

OK:

* Good implementation and testing plan (is really a plan and it is a good order?) 🡪 subsystem very good

KO:

* Diagram? 🡪 non fa vedere poi come si sale nell’integrazioni
* Integration plan for model?