Hector Mojica, Cassel Sloan, Guilherme Menezes
CSCI 360
Team Project 5: Engineering Design Principles
4 February 2007

1. Listing all alternatives and their design principle attributes:

Alternative A

Small modules – good, because each module has at most three components, and most have 1
Information hiding – good, everything shows what is needed
Least privilege – good, everything uses only what is needed
Coupling – good, they only communicate the necessary
Cohesion – good, all of the components competently relate to themselves
Simplicity – good, few modules, simple roles
Design with reuse – bad, because it doesn't reuse anything from anywhere
Design for reuse – bad, there is little to reuse; the architecture is relevant only to itself
Beauty – good, because it is highly simple and competently shows the composition of the EggTimer, and meets all specifications

Alternative B

Small modules – good, not a lot of components within them
Information hiding - good
Least privilege - good
Coupling – good
Cohesion – good
Simplicity – good, contains logical inheritance, few modules, and encourages use of already established and understood components
Design with reuse – good, uses an existent and reliable library class
Design for reuse – bad, because new components can still not be reused, despite using established components
Beauty – good, because it is still simple, designed with reuse and logical inheritance class, and equally powerful to the Alternative A

Alternative C

Small modules – good
Information hiding – good, considering that observers and seconds are hidden from outside
Least privilege – good, other modules only receive what is need
Coupling - good, still only a few amounts of associations and calls
Cohesion – bad, EggTimer inherits from two classes that have very little to do with each other
Simplicity – bad, over exaggerates simple tasks

Design with reuse – good, because it uses established classes and patterns
Design for reuse – good, using established classes will allow for parts to be used interchangeably; also, using the patterns will allow for future implementation
Beauty – bad, not very simple, despite good reuse, over-explaining can kill a simple architecture. It does seem to be more powerful (with option to add observers and displays) and suitable to later versions, but does not justify

2. The distinguishing design principles were simplicity, cohesion, design with/for reuse, and beauty. These principles varied the most among the three alternative designs.

3. Descriptions of alternatives:

Alternative A

It is very simple, but has little reuse and does not take advantage of existing classes.

Alternative B

This alternative is also very simple and understandable, is somewhat reusable, and contains good cohesion. It also utilizes existing classes.

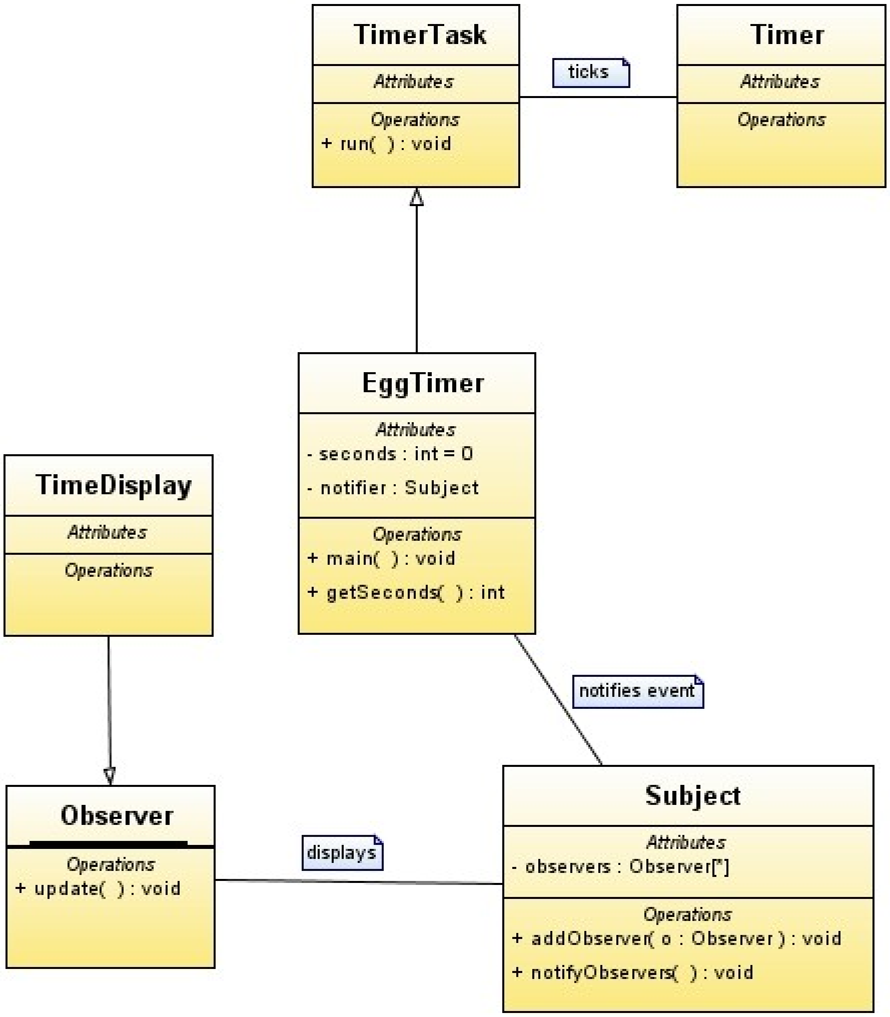
Alternative C

This alternative is more powerful and reusable, but contains very bad cohesion and simplicity.

4. The perceived best alternative would be B. Alternative B is very simple and cohesive, as well as reusable. However, one will need more details for later implementation;

5. The hierarchy relationship between "Subject" and "Eggtimer" gives the latter a new responsibility: to notify all observers of events that occurred. This reduces the cohesion of "Eggtimer", since it has more than one responsibility (it is at the same time a "TimeTask" and a "Subject"). A way to increase the cohesion of the classes would be to model "Subject" as a component of "Eggtimer". In this design approach, we create a notifier object in "Eggtimer" that is responsible for notifications, which keeps "Eggtimer" with a high cohesion, with a equally powerful design.

New Design



Assessment

1. Compared to our last meeting, our team performed much better. This was due to much more cooperative discussion, as well as more performance within less time (and less Netbeans).
2. Our team did indeed achieve the learning objectives; this activity forced us to discuss the design principles in detail, including if inheritance was coupling, the relationship between coupling and cohesion, the level of detail UML could give, and other things.