Implementation av det som saknades i programmet gjorde jag med Open-Closed- principen i baktanke. Principen säger att kod som kan innefatta klasser, moduler eller metoder, ska vara öppna för förlängning men samtidigt vara stängda för ändring. Min kod uppfyller detta genom att flera krypteringar (checkboxes) kan läggas till för att omvandla vald text. Applikationen i sin helhet bryter dock mot principen i kontrollerklassen eftersom modifieringar av existerande kod sker med bl.a. nya checkboxar.

Vidare använde jag designmönstret Decorator i lösningen där dekorationer läggs på en vanlig text. Dekorationerna är alltså krypteringar av inkommande text. Implementation av detta mönster görs genom att dynamiskt lägga till ansvar till ett objekt och är ett flexibelt alternativ till sub-klassning. I efterhand inser jag att min lösning blev aningen redundant i och med användning av två abstrakta klasser. Detta är något jag skulle gjort annorlunda om jag gjort uppgiften på nytt.

Jag anser att Decorator passar bra som mönster i denna applikation. Till skillnad från Strategy-mönstret som ändrar objekt på djupet, så ändrar Decorator bara ytan, eller skinnet av objektet, vilket är vad som ska hända i applikationen En känd negativ faktor med Decorator som mönster är att samtliga metoder i ett dekorerat interface behöver implementeras i dekorations-klassen. Detta blev dock inte till ett problem eller motgång i min lösning då applikationen bara har en sådan. Det faktum att applikationen bryter mot Open-Closed- principen vid det nämnda tillfället anser jag vara godtagbart då fulländade implementationer av en sådan princip inte alltid går att framställa.