Aufgabe 1

a) Verringerung des Berechnungsaufwands einer Funktion

- Wartungsphase

b) Zuordnung von verfügbaren Ressourcen zu Aufgabenpaketen

- Planungsphase

c) Testen der Rückgaben einer Funktion

- Implementierungsphase

d) Erstellen von Klassendiagrammen

- Entwurfsphase

e) Erstellen des Lastenhefts

- Anforderungsphase

f) Kompilieren und Erstellen eines Build des Gesamtsystems

- Integrationsphase

g) Konvertieren und Übertragen der gespeicherten Daten in eine andere Datenbank

- Rückzugsphase

h) Erstellen eines Pflichthefts

- Spezifikationsphase

Aufgabe 2

- MineSweeper
  - Logische Bindung, da hier die Komponenten des Programmes logisch zusammengefasst werden
- GameView
  - Funktionale Bindung, da die Klasse nur public update besitzt
- GameView.FieldButton
  - Prozedural Bindung, da `refreshView` zwar nach `update` aufgerufen wird, aber auch ohne `update` aufgerufen werden kann
- GameBoard
  - Kommunikatorische Bindung, da alle Methoden darum zentriert sind mit dem Spielfeld zu interagieren
- GameBoard.Field
  - Kommunikatorische Bindung, da alle Methoden darauf abzielen mit dem Feld zu interagieren
- FieldController
  - Funktionale Bindung, da die ganze Klasse mehr oder weniger nur für die Funktion mouseClicked existiert
  - Hat nur eine public Methode

von/zu	MineSweeper	GameView	GameView.FieldButton	GameBoard	GameBoard.Field	FieldController
MineSweeper	-	Stempelkopplung (Konstruktor)		Datenkopplung (Konstruktor)		
GameView		-		Datenkopplung (z. B. add Aufruf)	Stempelkopplung (Konstruktor)	
GameView.FieldButton			-		Datenkopplung (z. B. addObserver)	Stempelkopplung (Konstruktor)
GameBoard				-	Datenkopplung (Konstruktor)	
GameBoard.Field				Steuerkopplung (fieldRevealed)	-	
FieldController					Datenkopplung (reveal)	-

Aufgabe 4

Beispiel 1

Liskov Substitution Principle verletzt:

```
public class Animal {
    public void noise() {
        System.out.println("Animal noise");
    }
}

public class Dog extends Animal {
    @Override
    public void bark() {
        System.out.println("Dog bark");
    }
}

public class Cat extends Animal {
    @Override
    public void meow() {
        System.out.println("Cat meow");
    }
}
```

Fixed:

```
public class Animal {
    public void noise() {
        System.out.println("Animal noise");
    }
}

public class Dog extends Animal {
    @Override
    public void noise() {
        System.out.println("Dog bark");
    }
}

public class Cat extends Animal {
    @Override
    public void noise() {
        System.out.println("Cat meow");
    }
}
```

Beispiel 2

Interface Segregation Principle/Single Responsibility Principles verletzt:

```
public interface LivingThing {
    public boolean isAnimal();
    public int activeSeason();
    public void noise();
    public void action();
}
```

Fixed:

```
public interface Plant {
    public boolean isAlive();
    public int activeSeason();
}

public interface Animal {
    public void noise();
    public void action();
}
```