DRY (Don’t Repeat Yourself)

```
public void delete(Course course) {
  try {
     int i=…
     while…
     if{…
     }
  }
  catch (Exception e) {
    System.out.println…
  }
}
```

```
public void delete(Course course) {
  try {
     deleteCourse(aCourse);
  }
  catch (Exception e) {
    logError(e);
  }
}
private void deleteCourse(Course course) throws
Exception {
    deleteCourse(course);
    registry.deleteReference(course.code);
}

private void logError(Exception e) {
    logger.log(e.getMessage());
}
```

Refactoring

```
public class Square {
  public Point topLeft;
  public double side;
}

public class Rectangle {
  public Point topLeft;
  public double height;
  public double width;
}

public class Circle {
  public Point center;
  public double radius;
}
```

```
public class Geometry {
  public final double PI = 3.141592653589793;
  public double area(Object shape) throws
NoSuchShapeException
  {
    if (shape instanceof Square) {
        Square s = (Square)shape;
        return s.side * s.side;
    } else if (shape instanceof Rectangle) {
        Rectangle r = (Rectangle)shape;
        return r.height * r.width;
    } else if (shape instanceof Circle) {
        Circle c = (Circle)shape;
        return PI * c.radius * c.radius;
    }
        throw new NoSuchShapeException();
  }
}
```

```
public class Square implements Shape {
  private Point topLeft;
  private double side;
  public double area() {
    return side*side;
  }
}

public class Rectangle implements Shape {
  private Point topLeft;
  private double height;
  private double width;
  public double area() {
    return height * width;
  }
}

public class Circle implements Shape {
  private Point center;
  private double radius;
  public final double PI = 3.141592653589793;
  public double area() {
    return PI * radius * radius;
  }
}
```

```
public interface Shape()
{
      public double area();
      public double perimeter();
}

public class Geometry
{
      //declare geometry relation with Shape…
      public double area(Shape shape)
      {
           return shape.area();
      }
      public double perimeter(Shape shape)
      {
           return shape.perimeter();
      }
}
```

Data structures makes it easy to add new functions without changing the existing data structures. OO code(using objects), makes it easy to add new classes without changing existing functions.

```
public class RunTaskMutator {
  // common fields
  public void configureRun() { /* ... */ }
  public void updateStartScriptsTask(String
taskStartScriptsName) { /* ... */ }
  // 12 other methods (incl. 2 common methods)
}
```

```
public class RunTaskMutator extends
AbstractExecutionMutator {
  public void configureRun() { /* ... */ }
  // 2 other methods
}
public class StartScriptsMutator extends
AbstractExecutionMutator {
  public void updateStartScriptsTask(String
taskStartScriptsName) { /* ... */ }
  // 8 other methods
}
```