SOFTWARE ENGINEERING

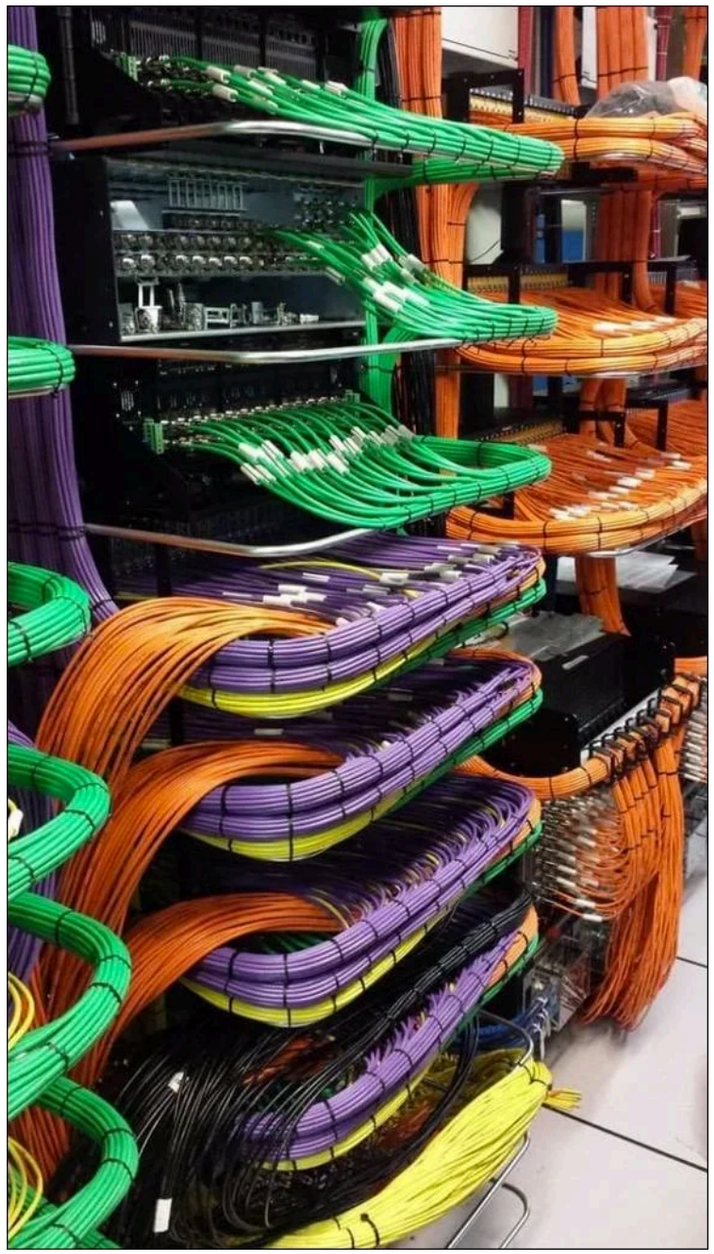
Class 10

amrut@iiit.ac.in

JOURNEY OF AN APPLICATION

APPLICATION JOURNEY

At the beginning… it’s beautiful!

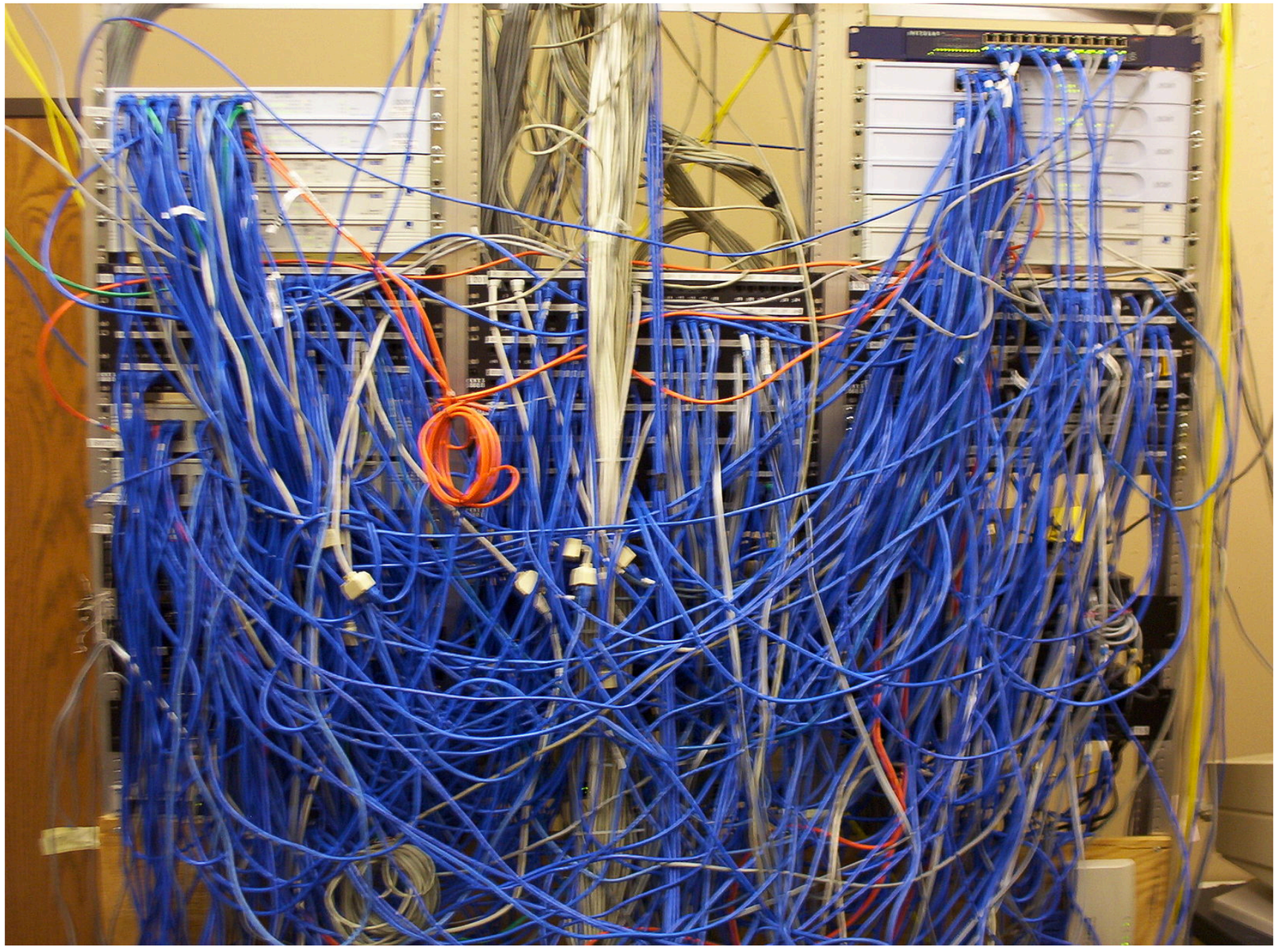


APPLICATION JOURNEY

but then.. things changed..

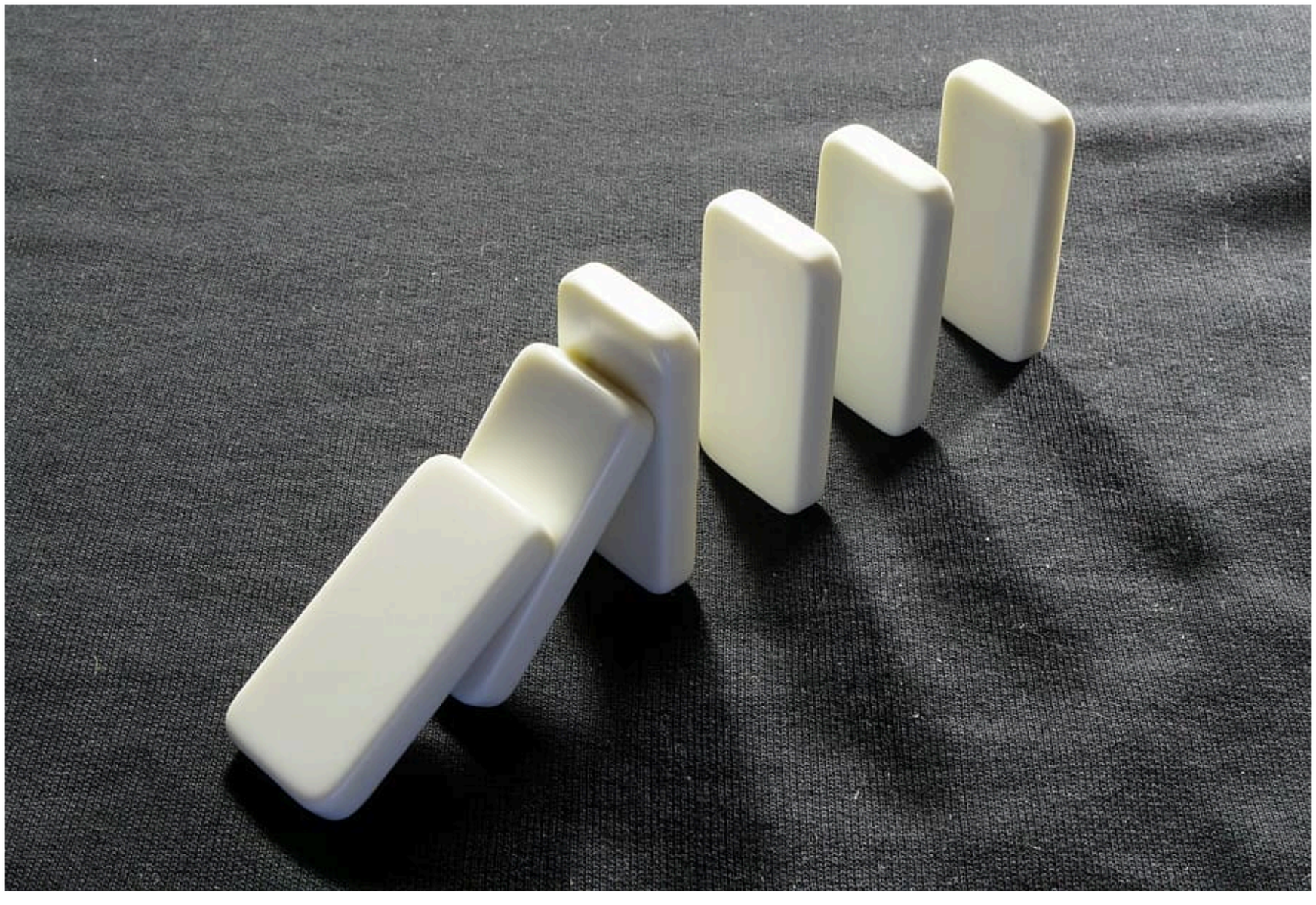
APPLICATION JOURNEY

.. and now.. it’s ugly!



APPLICATION JOURNEY

Application is ***Rigid***!

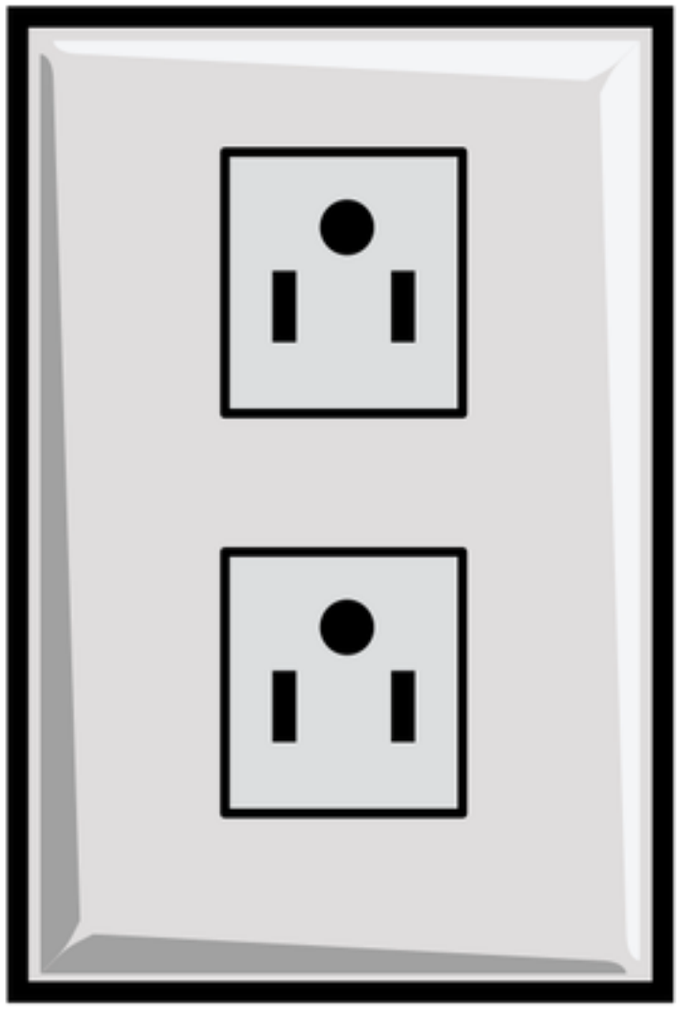


APPLICATION JOURNEY

Application is ***Fragile***!

APPLICATION JOURNEY

Application is ***Immobile***!



APPLICATION JOURNEY

Application is ***Viscous***!

APPLICATION JOURNEY

Application is ***Rotting***!

APPLICATION JOURNEY

Application ***Design*** is ***Rotting***!

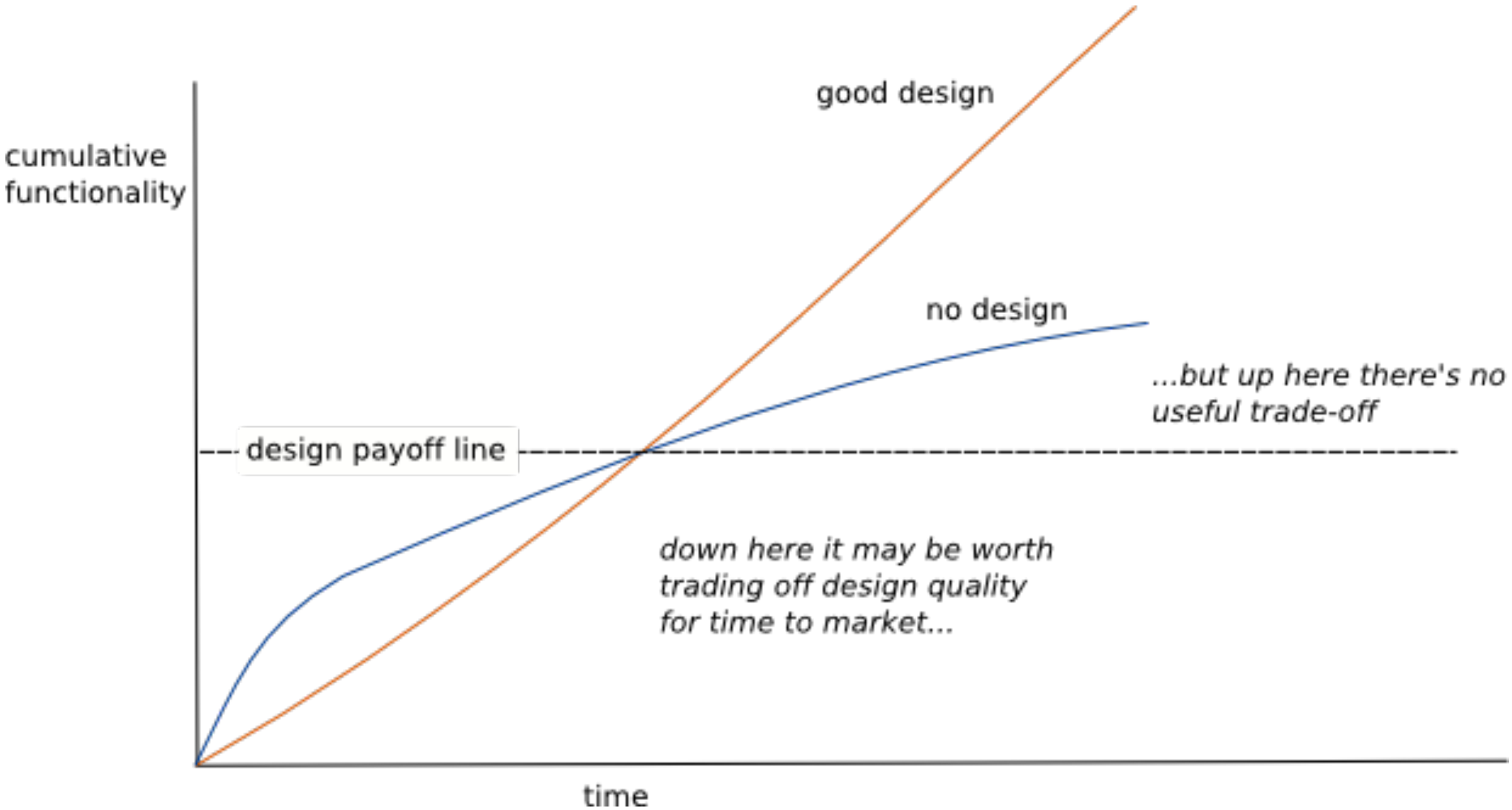
APPLICATION JOURNEY

Good Design is the solution!

GOOD DESIGN AND DESIGN PRINCIPLES

GOOD DESIGN

Is it worth spending time doing design?



GOOD DESIGN

Good Design ***resists design rot***
when things change.

GOOD DESIGN

Design Principles are ***strategies*** used to create Good Design.

SOLID

SOLID

- S - Single Responsibility Principle
- O - Open/Close Principle
- L - Liskov Substitution Principle
- I - Interface Segregation Principle
- D - Dependency Inversion Principle

Single Responsibility Principle

A class should have one, and only one, reason to change.

Open Closed Principle

You should be able to extend a classes' behavior, without modifying it.

or..

A class should be open for extension and closed for modification.

Liskov Substitution Principle

Subclasses should be substitutable by their base classes.

Interface Segregation Principle

Clients should not be forced to depend upon interfaces that they do not use.

Dependency Inversion Principle

High level modules should not depend on low level modules. Both should depend upon abstractions.

or..

Abstractions should not depend upon details. Details should depend upon abstractions.

REFACTORING EXAMPLE

Design log parsing application which reads unstructured logs from a file and prints structured information.

Unstructured logs

09/02/2020 05:22 PM Session Started

09/02/2020 05:25 PM Session Ended

...

REFACTORING

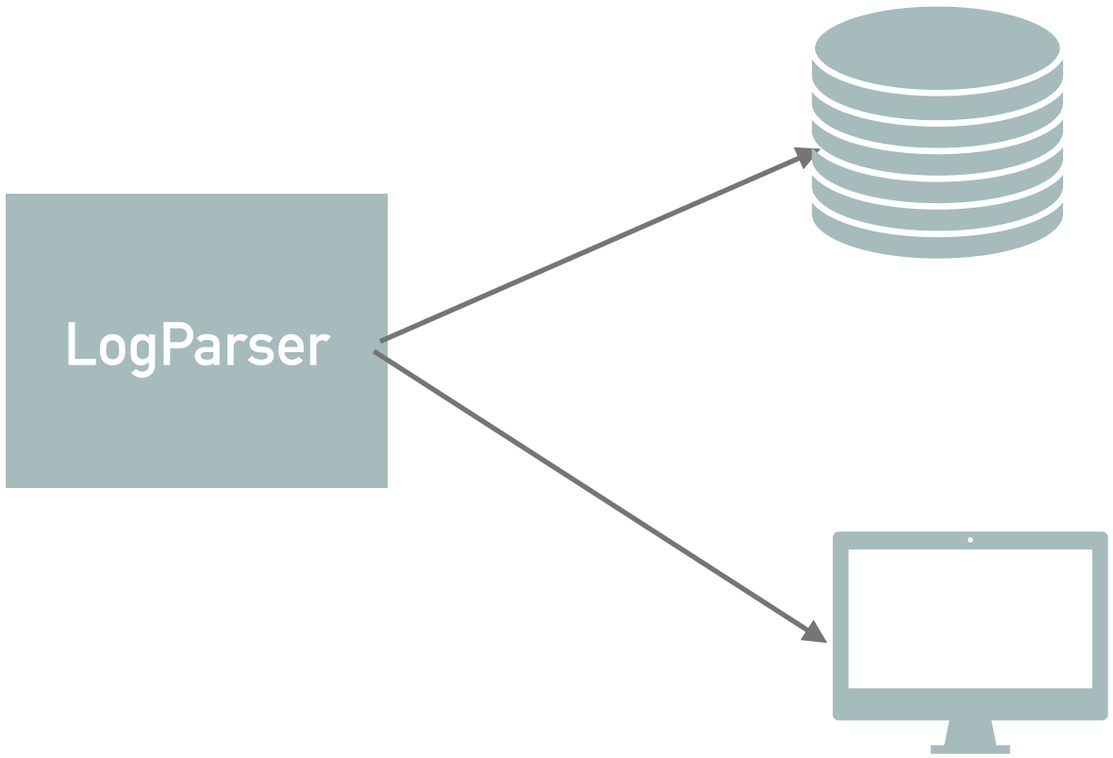
Unstructured logs

09/02/2020 05:22 PM Session Started

09/02/2020 05:25 PM Session Ended

...

REFACTORING



REFACTORING

```
public void readAndParseAndSave() throws IOException {
    List<String> logs = Files.readAllLines(Paths.get("unstructured.log"));
    String logEntryPattern =
            "([\\d]{2}/[\\d]{2}/[\\d]{4}) ([\\d]{2}:[\\d]{2} [AP]M) (.*)";
    Pattern p = Pattern.compile(logEntryPattern);
    logs.forEach(line -> {
        Matcher matcher = p.matcher(line);
        if(matcher.matches()) {
            System.out.printf("Date: %s, Time: %s, Log: %s%n",
                    matcher.group(1), matcher.group(2), matcher.group(3));
        }
    });
}
```

REFACTORING

```
public void readAndParseAndSave() throws IOException {
    List<String> logs = Files.readAllLines(Paths.get("unstructured.log"));
    String logEntryPattern =
            "([\\d]{2}/[\\d]{2}/[\\d]{4}) ([\\d]{2}:[\\d]{2} [AP]M) (.*)";
    Pattern p = Pattern.compile(logEntryPattern);
    logs.forEach(line -> {
        Matcher matcher = p.matcher(line);
        if(matcher.matches()) {
            System.out.printf("Date: %s, Time: %s, Log: %s%n",
                    matcher.group(1), matcher.group(2), matcher.group(3));
        }
    });
}
```

REFACTORING

```
public void readAndParseAndSave() throws IOException {
    List<String> logs = Files.readAllLines(Paths.get("unstructured.log"));
    String logEntryPattern =
            "([\\d]{2}/[\\d]{2}/[\\d]{4}) ([\\d]{2}:[\\d]{2} [AP]M) (.*)";
    Pattern p = Pattern.compile(logEntryPattern);
    logs.forEach(line -> {
        Matcher matcher = p.matcher(line);
        if(matcher.matches()) {
            System.out.printf("Date: %s, Time: %s, Log: %s%n",
                    matcher.group(1), matcher.group(2), matcher.group(3));
        }
    });
}
```

REFACTORING

```
public void readAndParseAndSave() throws IOException {
    List<String> logs = Files.readAllLines(Paths.get("unstructured.log"));
    String logEntryPattern =
            "([\\d]{2}/[\\d]{2}/[\\d]{4}) ([\\d]{2}:[\\d]{2} [AP]M) (.*)";
    Pattern p = Pattern.compile(logEntryPattern);
    logs.forEach(line -> {
        Matcher matcher = p.matcher(line);
        if(matcher.matches()) {
            System.out.printf("Date: %s, Time: %s, Log: %s%n",
                    matcher.group(1), matcher.group(2), matcher.group(3));
        }
    });
}
```

REFACTORING

```
public void readAndParseAndSave() throws IOException {
    List<String> logs = Files.readAllLines(Paths.get("unstructured.log"));
    String logEntryPattern =
            "([\\d]{2}/[\\d]{2}/[\\d]{4}) ([\\d]{2}:[\\d]{2} [AP]M) (.*)";
    Pattern p = Pattern.compile(logEntryPattern);
    logs.forEach(line -> {
        Matcher matcher = p.matcher(line);
        if(matcher.matches()) {
            System.out.printf("Date: %s, Time: %s, Log: %s%n",
                    matcher.group(1), matcher.group(2), matcher.group(3));
        }
    });
}
```

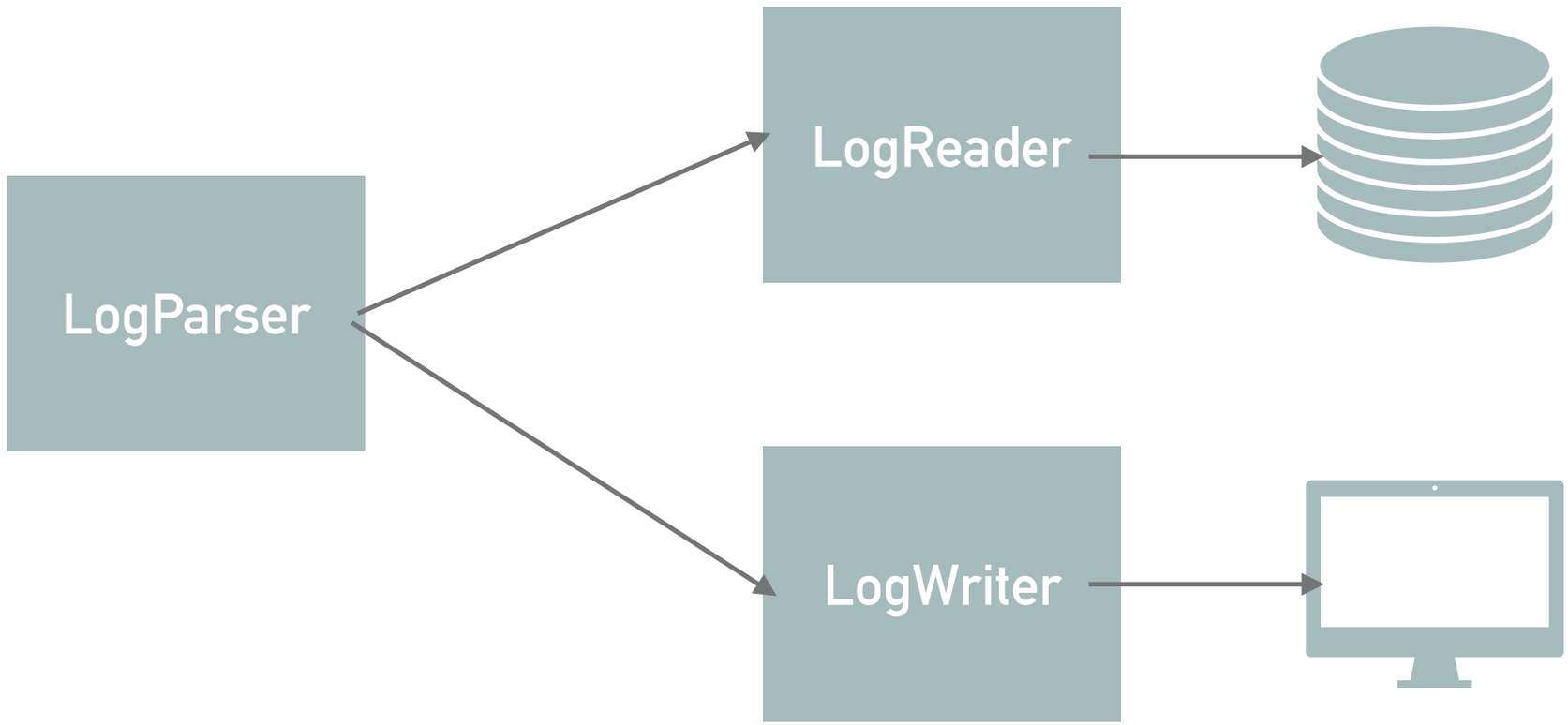
Are we done?

REFACTORING

```
public void readAndParseAndSave() throws IOException {
    List<String> logs = Files.readAllLines(Paths.get("unstructured.log"));
    String logEntryPattern =
            "([\\d]{2}/[\\d]{2}/[\\d]{4}) ([\\d]{2}:[\\d]{2} [AP]M) (.*)";
    Pattern p = Pattern.compile(logEntryPattern);
    logs.forEach(line -> {
        Matcher matcher = p.matcher(line);
        if(matcher.matches()) {
            System.out.printf("Date: %s, Time: %s, Log: %s%n",
                    matcher.group(1), matcher.group(2), matcher.group(3));
        }
    });
}
```

Does this have one responsibility?

REFACTORING



REFACTORING

```
public final class Log {
    String date;
    String time;
    String logMessage;
}
```

REFACTORING

```
public final class LogReader {
    public List<String> readLog() throws IOException {
        List<String> logs =
            Files.readAllLines(Paths.get("unstructured.log"));
        return logs;
    }
}
```

REFACTORING

```
public final class LogWriter {
    public void writeLog(final Log log) {
        System.out.printf("Date: %s, Time: %s, Log: %s%n",
                log.date, log.time, log.logMessage);
    }
}
```

REFACTORING

```
public final class LogParser {

    private final LogWriter logWriter;
    private final LogReader logReader;

    public LogParser() {
        this.logReader = new LogReader();
        this.logWriter = new LogWriter();
    }

    … … …
}
```

REFACTORING

```
public void parseLog() throws IOException {
    List<String> unstructuredLogs = this.logReader.readLog();
    String logEntryPattern =
          "([\\d]{2}/[\\d]{2}/[\\d]{4}) ([\\d]{2}:[\\d]{2} [AP]M) (.*)";
    Pattern p = Pattern.compile(logEntryPattern);
    unstructuredLogs.forEach(line -> {
        Matcher matcher = p.matcher(line);
        if(matcher.matches()) {
            Log log = Log.builder()
                        .date(matcher.group(1).time(matcher.group(2))
                        .logMessage(matcher.group(3)).build();
            this.logWriter.write(log);
        }
    });
}
```

REFACTORING

```
public void parseLog() throws IOException {
    List<String> unstructuredLogs = this.logReader.readLog();
    String logEntryPattern =
          "([\\d]{2}/[\\d]{2}/[\\d]{4}) ([\\d]{2}:[\\d]{2} [AP]M) (.*)";
    Pattern p = Pattern.compile(logEntryPattern);
    unstructuredLogs.forEach(line -> {
        Matcher matcher = p.matcher(line);
        if(matcher.matches()) {
            Log log = Log.builder()
                        .date(matcher.group(1).time(matcher.group(2))
                        .logMessage(matcher.group(3)).build();
            this.logWriter.write(log);
        }
    });
}
```

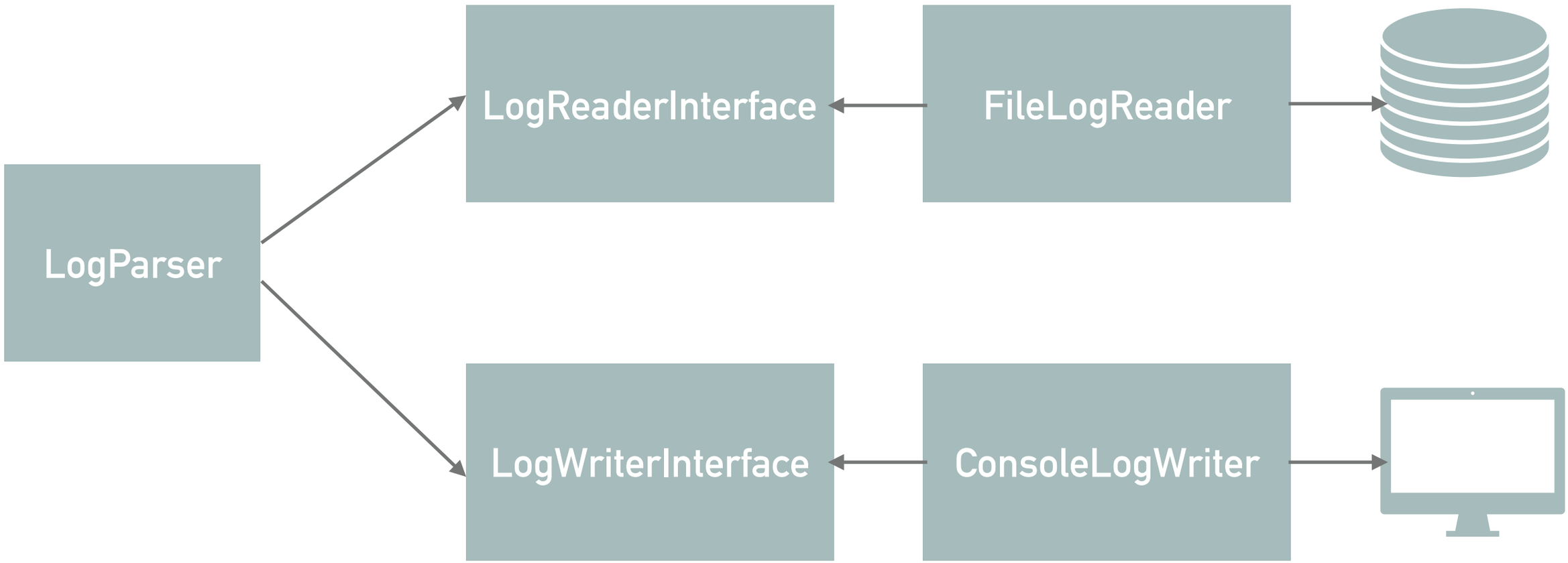
Are we done?

REFACTORING

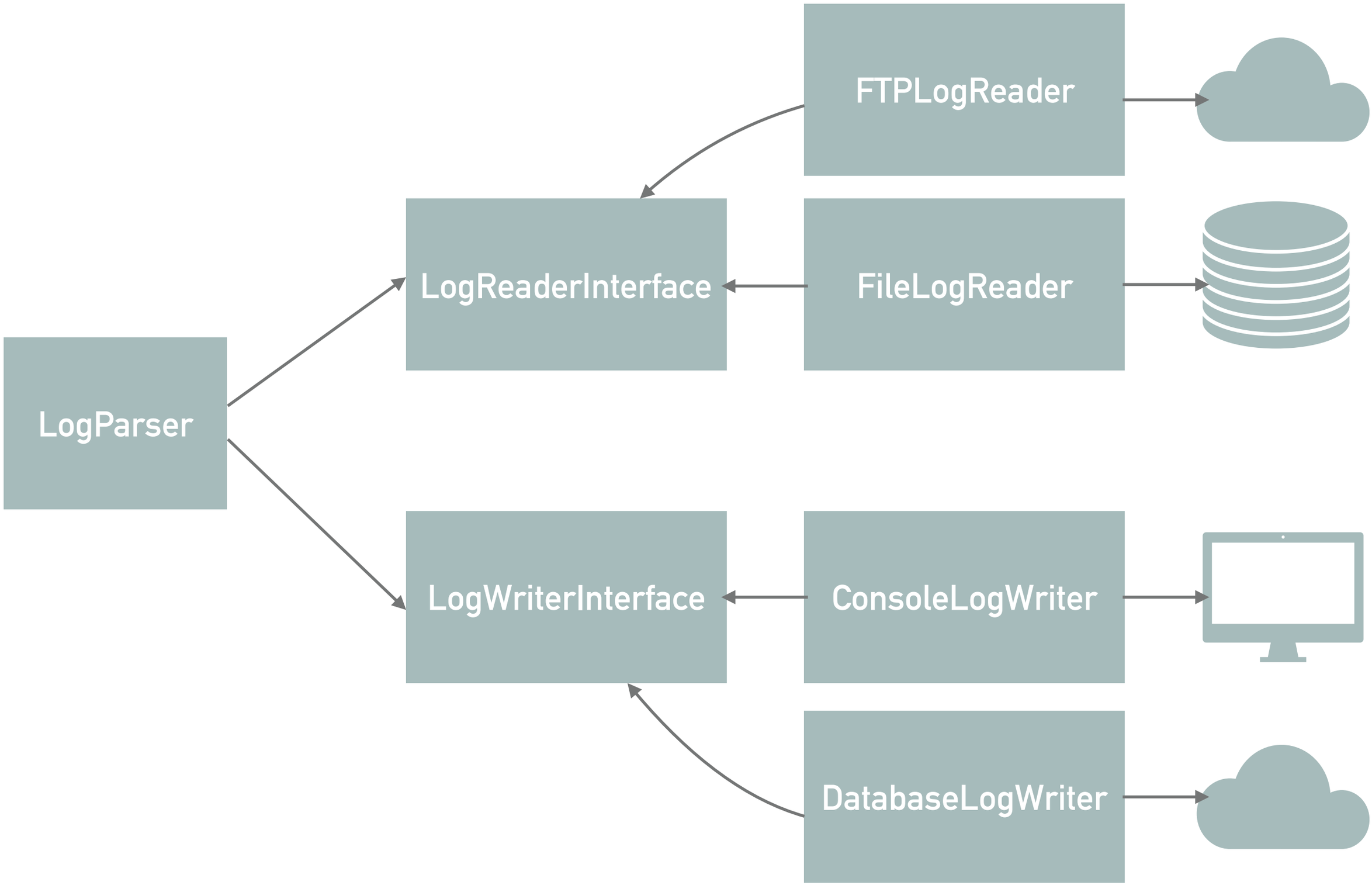
```
public void parseLog() throws IOException {
    List<String> unstructuredLogs = this.logReader.readLog();
    String logEntryPattern =
          "([\\d]{2}/[\\d]{2}/[\\d]{4}) ([\\d]{2}:[\\d]{2} [AP]M) (.*)";
    Pattern p = Pattern.compile(logEntryPattern);
    unstructuredLogs.forEach(line -> {
        Matcher matcher = p.matcher(line);
        if(matcher.matches()) {
            Log log = Log.builder()
                        .date(matcher.group(1).time(matcher.group(2))
                        .logMessage(matcher.group(3)).build();
            this.logWriter.write(log);
        }
    });
}
```

Is it easy to extend?

REFACTORING



REFACTORING



REFACTORING

```
public interface LogReaderInterface {
    List<String> readLog() throws IOException;
}

public interface LogWriterInterface {
    void write(Log log);
}
```

REFACTORING

```
public final class FileLogReader implements LogReaderInterface {

    @Override
    public List<String> readLog() throws IOException {
        List<String> logs =
                Files.readAllLines(Paths.get("unstructured.log"));
        return logs;
    }
}
```

REFACTORING

```
public final class ConsoleLogWriter
                implements LogWriterInterface {

    @Override
    public void write(Log log) {
        System.out.printf("Date: %s, Time: %s, Log: %s%n",
                log.date, log.time, log.logMessage);
    }
}
```

REFACTORING

```
public final class LogParser {
    private final LogReaderInterface logReader;
    private final LogWriterInterface logWriter;
    private String logEntryPattern;

    public LogParser(
        final LogReaderInterface reader,
        final LogWriterInterface writer,
        final String pattern) {
      this.logReader = reader;
      this.logWriter = writer;
      this.logEntryPattern = pattern;
    }
}
```

REFACTORING

```
public void parseLog() throws IOException {
    List<String> unstructuredLogs = this.logReader.readLog();
    Pattern p = Pattern.compile(this.logEntryPattern);
    unstructuredLogs.forEach(line -> {
        Matcher matcher = p.matcher(line);
        if(matcher.matches()) {
            Log log = Log.builder()
                        .date(matcher.group(1))
                        .time(matcher.group(2))
                        .logMessage(matcher.group(3)).build();
            this.logWriter.write(log);
        }
    });
}
```

REFERENCE

- Design Principles and Design Patterns
- Design Stamina Hypothesis
- Principles of OOD