1 Design Techniques

1.1 Documentation

Readability

- Documentation: Explain design, architecture, etc.
- Source Code Comments:
  - API documentation: Packages, interfaces, classes, methods, etc.
  - Line comments: Small clarifications of the code
- Source Code

Code Specific Aspects

- Coding Style Conventions: Naming, formatting, bracket placement, etc.
- Restrictions on Usage of Language Features
- Naming of Identifiers: Coherent and descriptive naming
- Meaningful Comments

1.1.1 Comments

Comments can be seperated in two different categories:

API Comments

- **Intent:** Document API usage
- **Audience:** Third-Party developers using the code as a framework or library

Used to describe in detail when method can be called and how it behaves

- Restrictions on parameters besides types: non-negative, not null, etc.
- Side effects on object state
- Thrown exceptions (when and why)
- What is returned

Statement Level Comments (Line Comments)

- **Intent:** Describe implementation details, code structure
- **Audience:** Developers on the same code base

Should be concise and used sparingly

- Clear, well-written code is mostly self explanatory
- Might indicate poorly written or overly complex code **Refactor**

1.2 Refactoring

Refactoring describes the process of restructuring code without changing its external behaviour.

This can be done in order to:

- Prepare for addition of new features (just preparing, not actually adding)
  - Reduce code duplication
  - Avoid nested conditionals
- Improve design (Cohesion, etc.)

- Increase comprehensibility
  - Choose a clear naming convention
  - Choose meaningful names
  - Simplify convoluted logic
- Improve maintainability

1.2.1 Refactoring Techniques

Extract Method

A method should be extracted if one method does multiple things or similar functionality is realized within one method or across multiple methods.
The method extraction should be done like this:

1. Create a new method (target)
2. Copy extracted code to target
3. Identify local variables used in extracted code
   - 3.1. Variables only used in extracted code can be declared as local variables
   - 3.2. Extracted code modifies exactly one outside variable: Check if target method can be query
   - 3.3. If more than one outside variable is modified: Extract Method is not possible
   - 3.4. Pass undeclared variables in target as method parameters
4. Replace extracted code with call to target

This unfortunately sometimes reduces cohesion as target method does not necessarily accesses all the same variables as the source method. This indicates that the class has to many responsibilities. Indicates that usage of **Move Method** is needed.

Move Method should be only be used if:

- Source method does **not** use features of the source class
- Source method does **not** override a method or is overriden by a subclass

Move Method

1. Create new method in target class
2. Copy source code to target method and adjust it to work there
3. Determine how to reference target object from source
4. Turn source method into delegating method
5. If not needed: Remove source method and accessors in target class

If a class has **two or more independent responsibilities** and no other class can handle these responsibilities, the **Extract Class** technique should be used.

Extract Class

1. Create new class
2. Link new class from old class (e.g. attributes), may require a new accessor
3. Apply refactoring **Move Method** and **Move Field**
4. Review and reduce class interfaces (unnecessary accessors, etc.)

1.3 Design Patterns

A design pattern describes

- a problem that reoccurs regularly in the domain

- the core of a solution to this problem, such that one can reuse the solution in other contexts (might not be exactly the same)

Template Method Pattern

Implements an algorithm in a manner that allows adaptation to different implementations.
- Define skeleton algorithm, but defer implementation of some concrete parts to subclasses
- Often used in frameworks and APIs

Some benefits:
- Separation of variant and invariant parts
- Avoidance of unnecessary code duplication
- Control of subclass extensions

Design Pattern Template

1. - Name: Short mnemonic to extend the design vocabulary
   - Intent: Goals and reasons why to use the pattern

---

2. - Motivation: States problem situation
   - Applicability: Context in which the pattern can be used

---

3. - Structure: Static structure of the pattern (UML Class diagram)
   - Participants: Which classes are involved
   - Collaborations: How the classes interact
   - Implementation: How to implement the pattern

---

4. - Consequences: Gains and trade-offs

---

5. - Known Uses: Examples of using the pattern
   - Related Patterns: References to and discussion of related patterns