Software development: Key phases

1. Requirements
2. Architecture
3. Design
4. Implementation
5. Validation  // May lead to debugging
   - Manual code reviews
   - Automated tools such as unit testing frameworks (e.g., JUnit), model checking (e.g., Java Pathfinder)

Three in a Row game app (Version 1)

- src/
  - ThreeInARowGame
  - ThreeInARowBlock
- test/
  - TestExample (2)

Issues:
- Simple architecture
- Poor design
- Violates best programming practices
- Minimal unit testing

Row game app (Version 2)

- src/
  - controller/
    - RowGameController
    - RowGameRulesStrategy
  - model/
    - RowBlockModel
    - RowGameModel
  - view/
    - RowGameGUI
    - RowGameBoardView
    - RowGameStatusView
    - RowGameView
- test/
  - TestExample (14)

Goals:
- MVC architecture
- OO design: OO design principles and OO design patterns
- Satisfies programming best practices
- More extensive testing
Software architecture vs. design

**Architecture (what components are developed?)**
- Considers the system as a whole:
  - High-level view of the overall system.
  - What components exist?
  - What type of storage, database, communication, etc?

**Design (how are the components developed?)**
- Considers individual components:
  - Data representation
  - Interfaces, Class hierarchies
  - ...

---

**Architecture pattern:**

**MVC (Model View Controller)**

Separates data representation (Model), visualization (View), and client interaction (Controller)

---

**Architecture pattern:**

**PAC (Presentation Abstraction Control)**

Separates data representation (Abstraction), visualization (Presentation), and client interaction (Control)

---

**Refactoring:**

**MVC architecture**

Code refactoring

From Wikipedia, the free encyclopedia

"Refactor" redirects here. For the use of "refactoring" on Wikipedia, see Wikipedia:Refactoring talk pages. This article is about a behavior-preserving change. It is not to be confused with Rewrite (programming).

In computer programming and software design, code refactoring is the process of restructuring existing computer code—changing the factoring—without changing the external behavior. Refactoring is intended to improve the design, structure, and/or implementation of the software (its non-functional attributes), while preserving its functionality. Potential advantages of refactoring may include improved code readability and reduced complexity; these can improve the source code's maintainability and create a simpler, cleaner, or more expressive internal architecture or object model to improve extensibility. Another potential goal for refactoring is improved performance; software engineers face an ongoing challenge to write programs that perform faster or use less memory.


---

https://en.wikipedia.org/wiki/Presentation%E2%80%93abstraction%E2%80%93control
**Architecture pattern:**

PAC (Presentation Abstraction Control)

- **Client**
  - sees
  - uses
  - updates

- **Presentation**
  - updates
  - Abstraction
  - sets
  - gets

Separates data representation (Abstraction), visualization (Presentation), and client interaction (Control)

[https://en.wikipedia.org/wiki/Presentation%E2%80%93abstraction%E2%80%93control](https://en.wikipedia.org/wiki/Presentation%E2%80%93abstraction%E2%80%93control)

**Which architecture?**

**MVC:**

- **View**
  - updates
  - Client

- **Controller**
  - manipulates
  - Model

**PAC:**

- **Presentation**
  - updates
  - Client

- **Control**
  - sets
  - Abstraction

- **Abstraction**
  - gets

**Design patterns:**

Observer and Strategy

- **Client**
  - sees
  - View
  - Controller

- **View**
  - uses
  - Model

- **Controller**
  - manipulates

Separates data representation (Model), visualization (View), and client interaction (Controller)

**RowGameController:**

Supporting different row game rules

- Different row game rules:
  - Three in a row
  - Tic tac toe

- Two possible solutions:
  - Strategy design pattern
  - Template method design pattern
Strategy design pattern: RowGameController

```
<<interface>>
Median
+median(a:double[]):double

StrategyMedian
+sortStrategy:Sorter
+median(a:double[]):double
+setSorter(s:Sorter)

"median" delegates the sorting of the array to a "sortStrategy"
```

Template method design pattern: RowGameController

```
<<interface>>
Sorter
+sort(array:double[]):

AbstractMedian
(abstract)
+median(a:double[]):double
# sort(a:double[])

SimpleMedian
# sort(a:double[])

Should the median method be final?
- The template method (median) implements the algorithm but leaves the sorting of the array undefined.
- The concrete subclass only needs to implement the actual sorting.
```

Design patterns: Observer and Strategy

```
Separates data representation (Model), visualization (View), and client interaction (Controller)
```

Observer pattern

```
Observable
(abstract)
# observers:Set<Observer>
+ register(o:Observer)
+ unregister(o:Observer)
+ stateChanged()

MyObservable
- state:State
+ getState():State
+ setState(state:State)

Observer
+ update()

public void stateChanged() {
    for (Observer o : observers) {
        o.update();
    }
}

MyObserver
- state:State
+ update()

// For the setState method, use the stateChanged method
```
Observer pattern: Reset JButton and RowGameController

Learn by example

- [https://docs.oracle.com/en/java/javase/14/docs/api/java.desktop/javax/swing/JButton.html](https://docs.oracle.com/en/java/javase/14/docs/api/java.desktop/javax/swing/JButton.html)

- [https://docs.oracle.com/en/java/javase/14/docs/api/java.desktop/javax/swing/AbstractButton.html](https://docs.oracle.com/en/java/javase/14/docs/api/java.desktop/javax/swing/AbstractButton.html)

- [https://docs.oracle.com/en/java/javase/14/docs/api/java.desktop/java/awt/event/ActionListener.html](https://docs.oracle.com/en/java/javase/14/docs/api/java.desktop/java/awt/event/ActionListener.html)

Observer pattern: Model and its View(s)

// For the setState method, use the stateChanged method
Learn from user documentation

- [https://docs.oracle.com/en/java/javase/14/docs/api/java.desktop/java/beans/PropertyChangeSupport.html](https://docs.oracle.com/en/java/javase/14/docs/api/java.desktop/java/beans/PropertyChangeSupport.html) (In our case, we'll be using another design pattern called Adapter.)

- [https://docs.oracle.com/en/java/javase/14/docs/api/java.desktop/java/beans/PropertyChangeListener.html](https://docs.oracle.com/en/java/javase/14/docs/api/java.desktop/java/beans/PropertyChangeListener.html)

Manual code review:
Proposed fixes

- See the Homework 1 solution
- Your own

Automated testing:
JUnit test suite

- At least 14 test cases (including the two test cases given to you)
- For each package model, view, and controller, at least one test cases
- For each of the row game rules (Three in a Row and Tic Tac Toe), at least the following test cases:
  - Illegal move (should not change the game board)
  - Legal move (should change the game board)
  - One of the players win
  - The two players tie
  - Reset

JUnit test case: Input/Output pairs

```java
@Test(expected = IllegalArgumentException.class)
public void testNewBlockViolatesPrecondition()
{
    RowBlockModel block = new RowBlockModel(null);
}
```
JUnit test case: Class invariants

```java
@Test
public void testNewGame() {
    RowGameModel gameModel = new RowGameModel();
    // Check for the expected initial configuration
    // TODO: For all blocks,
    assertEquals("1", gameModel.player);
    assertEquals(9, gameModel.movesLeft);
}
```

Software Engineering: Applied skills

- Architecture pattern: MVC, PAC
- OO design patterns: Observer, Strategy (or Template method)
- Code review: OO design principles and programming best practices
- JUnit testing framework
- Javadoc API documentation tool
- Git version control system

Homework 2

- **Goal:** Re-design, re-implement, and test the Row game app

- **Deliverables:** MVC architecture pattern, Observer design pattern, Strategy (or template method) design pattern, Code review proposed fixes, Git log messages

- **Due:** Friday October 23, 2020, 9 PM EDT

[https://people.cs.umass.edu/~hconboy/class/2020Fall/CS5520/hw2.pdf](https://people.cs.umass.edu/~hconboy/class/2020Fall/CS5520/hw2.pdf)