Software Architecture and Engineering

Silvan Mosberger

March 22, 2017

Contents

1 Modularity

1.1 Coupling

1.1.1 Data Coupling

Representation Exposure: Try not to expose internal representation, because this couples thes code with the client and is difficult to change during maintenance.

1. Approach 1: Restricting Access to Data

   Force narrow interface, hide implementation, no references to subobjects (leaking), no storing of arguments as subobjects (capturing).

   Facade pattern: A single, simplified interface without hiding detais completely.

2. Approach 2: Making SHared Data Immutable

   Most problems don't occur like this, but copies can lead to run-time/memory overhead and sometimes mutation is still needed.

   Flyweight pattern: Basically memoization, a factory for objects checks if one exists for a key and returns it or creates a new one otherwise.

Java uses this for strings, multiple "Hello" strings are shared between all clients that requested it.

3. Approach 3: Avoid Sharing Data

   Abstract the relevant parts to not depend on each other and avoid sharing data.

4. Architectual Style

   (a) Pipe-and-Filter

   Basically reactive programming, just work with filters, splits, maps, pipes, etc. Very functional, much wow. Also Unix pipes.

   Fusion: Fusion reduces communication cost at the expense of parallelism

   Fission: Fission is profitable if the benefits of parallelization outweigh the overhead introduced by fission

   Advantages: Reusability, ease of maintenance, potential for parallellism

   Disadvantages: Sharing is expensive/limiting, difficult to design incremental filters, "not appropriate for user interaction" (lol no), "error handling is hard" (only when you don't do it right), "smallest denominator on data transmission" (ever heard of types?).

1.1.2 Procedural Coupling

Modules are coupled by called methods of each other. Changing either one of them implies changing another one.

1. Approach 1: Moving Code

   Fix the dependency by moving code to the module that it should be in.

2. Approach 2: Event-Based Style

   Components generate events, others can listen to specific events. Generators don't know who's affected. E.g. IDE's, user interfaces, web sites. Observers and Observables.

   Advantages: Good reuse, add/remove/replace components with minimum effect on other components

   Disadvantages: Loss of control (what happens on this event, etc.), ensuring correctness is difficult because it depends on context

    (a) Model-View-Controller :/

3. Approach 3: Restricting Calls

   Enforce restricted access to calls of different modules. Example: Layered architecture (can only call methods of next layer).

    (a) Three-Tier Architecture

        Presentation Tier (UI), Logic Tier (functionality), Data Tier (persistent storage). Can exchange any tier.

        Advantages: Increasing levels of abstraction on higher layers, simple maintenance (low coupling), reusability of layer implementation

        Disadvantages: Performance since there are many levels of indirection.

1.1.3 Class Coupling

Inheritance couples classes together and changes to the superclass can cause problems in the subclass.