### **1. Client Handles Both Business Logic and Traversal — Violates Single Responsibility Principle**

Right now, our Client class is doing two very different jobs at the same time:

* It’s responsible for **sending messages** to users, which is the core business logic.
* But it also has to **know how to go through** the users inside each social network, and each network uses a different type of collection — like a stack for Facebook, a vector for Instagram, or a map for TikTok.

This mix of responsibilities makes the code harder to maintain. Imagine if the way we store users inside a social network changes — for example, Facebook switches from a stack to a vector. We’d have to change the Client code too, even though its job should just be about sending messages.

In other words, Client is doing too much, and that violates a key design principle called the **Single Responsibility Principle** — a class should have only one reason to change.

### **2. Client Depends on Specific Collection Types — Violates Open/Closed Principle**

Another problem is that Client must know the exact container type each social network uses. Because of this:

* The code for going through the users is different in every method — one uses a while loop popping from a stack, another loops over a vector, and another iterates over a map.
* This means **the traversal logic is duplicated** and scattered across the code.
* More importantly, if we want to add any new feature that needs to go through the users — say, counting active users, or sending notifications — we’ll have to write the traversal code all over again for each network.

This tightly couples Client to these implementation details and makes it very hard to add new operations without touching the existing code. This goes against the **Open/Closed Principle**, which says that software entities should be open for extension but closed for modification.

### **Summary**

So in short:

* **Client mixes business logic with traversal logic, making it fragile and hard to maintain.**
* **Client depends heavily on internal data structures of social networks, leading to duplicated code and low flexibility.**

To fix this, we can introduce a common interface for iterating over users (an iterator). This way, Client doesn’t need to know how the users are stored — it just asks the iterator for the next user, and focuses only on sending messages. This will make the system easier to extend, maintain, and reuse.