Software Engineering Midterm

2025-26 Sem I

All the questions are for 10 marks each.
Thoroughly prove and justify all your answers.

1. A defense contractor is building a real-time missile guidance system that must meet strict safety certifications. The project involves multiple vendors, frequent changes in requirements due to evolving threat models, and has high penalties for failure.
    1. If the team strictly followed the **Waterfall model**, what risks and benefits would arise in terms of verification, stakeholder alignment, and certification timelines?
    2. If the team adopted a **Spiral model**, how would risk management and cost of iterations differ from Waterfall in this context?
    3. If the team attempted to use **Scrum/Agile**, what challenges would they face in managing safety certification, and how could Agile principles be adapted (or hybridized with other models) to balance flexibility with regulatory compliance?

    Finally, propose a ***hybrid lifecycle strategy*** for this project, justifying why it might outperform using a single model exclusively.

2. i. What is the output of this code? Justify your answer.
   ii. Why are weak pointers being used here?
   iii. What does the *w.lock*() function do? Why is it being used here?
   iv. What does *b.reset*() do? What are its implications?

```
struct Node {

    int id;
    vector<weak_ptr<Node>> kids;
    shared_ptr<string> cache;
    unique_ptr<int> weight;

  Node(int i, shared_ptr<string> c, int w)
            : id(i), cache(move(c)), weight(make_unique<int>(w)) {}

  void link(const shared_ptr<Node>& child) { kids.push_back(child); }

  void dump() const {
       cout << *cache << " #" << id << "(" << *weight << ")\n";
       for (auto &w : kids) if (auto s = w.lock()) cout << "  -> " << s->id << "\n";
  }
};
```

```
int main() {

    auto cache = make_shared<string>("payload");
    auto a = make_shared<Node>(1, cache, 10);
    auto b = make_shared<Node>(2, cache, 20);
    auto c = make_shared<Node>(3, cache, 30);

    a->link(b); b->link(c); c->link(a);
    a->dump(); b->dump(); c->dump();

     b.reset();

     cout << "After dropping b:\n";
     a->dump(); c->dump();
}
```

3. You are building a payments client library used by multiple microservices.
   Requirements:

   1. The **routing logic** for choosing a payment gateway (Stripe, Razorpay, in-house) must be **swappable at runtime** based on the configuration — avoid a long list of `if/else` statements.
   2. The library must support a **"sandbox (dry-run)" mode** where calls are **intercepted**: real gateways are not invoked; requests/responses are logged and validated, then a dummy result is returned. The logic of the caller is not changed.
   3. Keep the public API stable (`processPayment(req): Result`).

Unit tests should be able to inject fake routing and sandbox behavior easily.

What design patterns should be used here? Justify your answer.

4. We wish to create a browser-based image editor, whose functionality is like Microsoft Paint. It needs to have the following features.
   - I. State needs to be maintained on both the browser and the back-end server.
   - II. It needs to work on both slow and fast networks regardless of the bandwidth.
   - III. Undo and redo functionalities need to be supported.
   - IV. For a single server-side backend, it should be possible to support multiple browser-based frontends. This means that the change made in one browser should be visible on the rest of the browser-based clients.
   - V. Switch between different server-client networks.

   Which patterns should be used to realize such functionalities? Justify your answer.

5. Consider two programs *P* and *Q*. Consider the four complexity metrics SLOC, cyclomatic complexity, Halstead effort and Data flow complexity.

   i. Assume *Q* is a permutation of *P* (some statements are reordered). Which of the four aforementioned complexity metrics change? In other words, which complexity metric is dependent on the order of statements? Justify your answer.

   ii. Consider the following Weyuker property: There are finitely many programs of complexity **c**. Which of the four aforementioned metrics satisfy it? Why?

   iii. Let CC be the cyclomatic complexity metric. Show an example where:
   CC(P) + CC(Q) < CC (P;Q)
   `;' is the concatenation operator. Prove your answer.