Object-Oriented Programming

Software Quality (OOSC – Chapter 1)

Introduction

- What makes software 'good'?
- Quality is multi-dimensional
- External vs. Internal qualities

External vs. Internal Factors

- External: speed, usability, adaptability
- Internal: modularity, readability, structure
- External depends on internal discipline

Correctness

- Prime quality: meets specification
- Layered correctness approach
- Design for correctness: typing, assertions

Robustness

- Behavior outside the specification
- Handle abnormal cases gracefully
- Graceful degradation

Extendibility & Reusability

- Software must adapt to change
- Principles: simplicity & decentralization
- Reusability prevents duplication

Compatibility & Efficiency

- Compatibility: smooth integration
- Case: AMR Confirm failure
- Efficiency balanced with other goals

Other Quality Factors

- Portability across platforms
- Ease of use for all users
- Functionality vs. creeping featurism
- Timeliness: deliver on time

Documentation

- Not a separate factor
- External: ease of use
- Internal: extendibility
- Module interface: reusability

Tradeoffs & Key Concerns

- Qualities often conflict
- Correctness is non-negotiable
- Core qualities: reliability & modularity

Software Maintenance

- 70% of lifecycle cost
- Noble: modifications
- Less noble: debugging
- Abstract data types reduce costs

OOP and Change

- Anticipates inevitable change
- Supports adaptability
- Design for evolving requirements

z

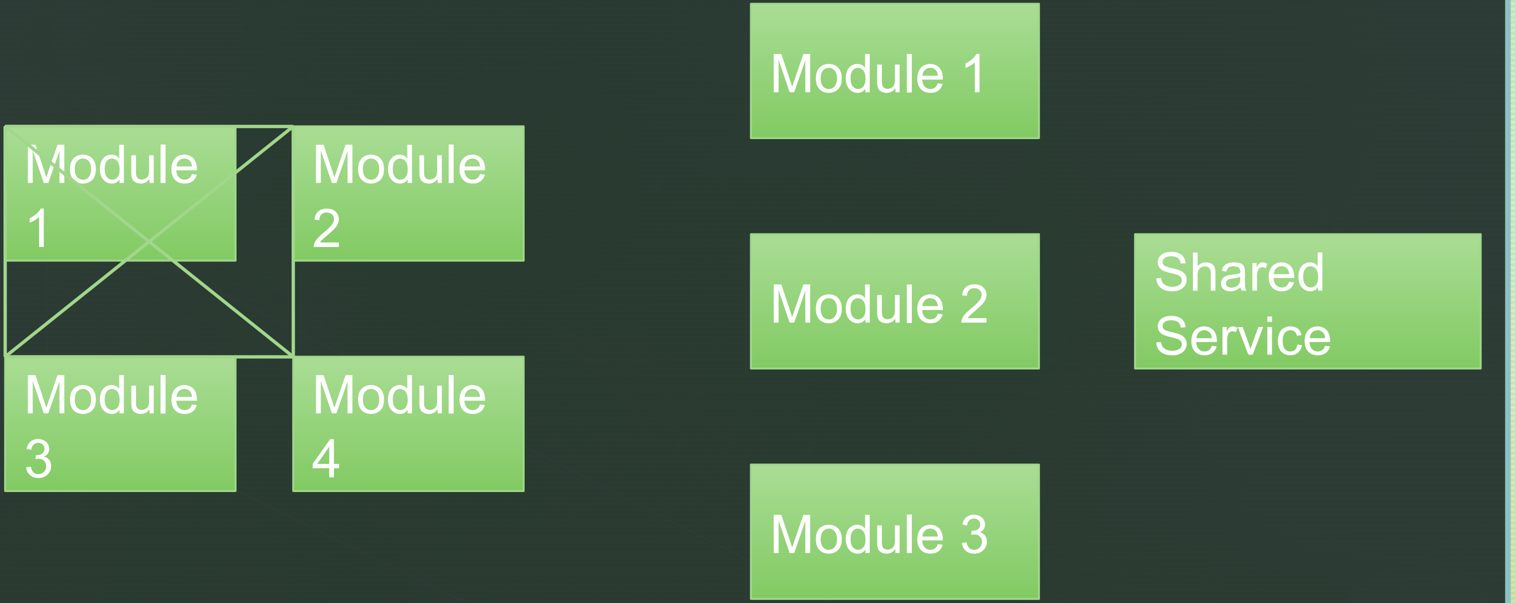
Avoiding Rigidity & Fragility

- Rigidity: hard to change
- Fragility: changes introduce bugs
- Immobility: cannot reuse code
- Viscosity: shortcuts encouraged

Dependencies and Productivity

- Dependencies kill productivity
- Cause rigidity, fragility, immobility, viscosity
- OOP reduces coupling

Dependency Tangle vs. Modular Architecture



Module 1

Module 2

Module 3

Module 4

Module 1

Module 2

Module 3

Shared Service