Software Quality (OOSC – Chapter 1)

Lecture Handout

Introduction

Software engineering is fundamentally about quality. Quality is multi-dimensional: performance, ease of use, adaptability, and correctness all matter. We distinguish between external qualities, visible to users, and internal qualities, visible to developers. External qualities depend on strong internal design.

External vs. Internal Factors

External qualities include speed, usability, and reliability. Internal qualities include modularity, readability, and structure. Though users don't see internal qualities, external ones cannot be achieved without them. Like a house, the wiring (internal) enables the lights (external).

Correctness

Correctness is the foundation: if a system is not correct, nothing else matters. Correctness is usually pursued through a layered approach—each layer relies on the correctness of the lower layers. While many rely on testing, correctness can be built in with assertions, typing, and specifications.

Robustness

Robustness complements correctness. It is about how a system behaves when faced with abnormal conditions such as invalid input or failed sensors. Robust software does not crash but degrades gracefully, providing clear error messages or fallback behaviors.

Extendibility & Reusability

Extendibility is the ease of adapting software to change. Change is inevitable—laws change, businesses evolve, scientific understanding grows. Reusability is the ability of software components to serve multiple applications. Both benefit from simplicity and decentralization of design.

Compatibility & Efficiency

Compatibility is critical: software must work well with other systems. Incompatibility caused failures such as the AMR Confirm system, wasting millions. Efficiency must be balanced with correctness and maintainability. Over-optimizing can lead to fragile code; ignoring efficiency can lead to unusable systems.

Other Quality Factors

Other factors include portability across platforms, ease of use for novices and experts, functionality without overloading the system with features, and timeliness—software must be delivered when needed, not years too late.

Documentation

Documentation is not a separate quality factor but flows from others. Ease of use requires external documentation, extendibility requires internal documentation, and reusability requires module documentation. Self-documenting code and automated tools reduce the burden.

Tradeoffs & Key Concerns

Quality factors often conflict, such as efficiency versus portability. Tradeoffs are inevitable but must be explicit. Correctness cannot be compromised. Key concerns are reliability (correctness + robustness) and modularity (extendibility + reusability). Object-oriented techniques directly improve these.

Software Maintenance

Maintenance consumes about 70% of software costs. Noble maintenance includes adapting to new requirements; less noble maintenance is fixing bugs. Data format changes like Y2K show the costs of poor design. Abstract data types reduce the impact of such changes by isolating data representation.

OOP and Change

Object-oriented programming and design help answer: what happens when our application changes? OOP anticipates change with modularity, abstraction, and encapsulation. Well-defined responsibilities in objects localize change and reduce ripple effects.

Avoiding Rigidity & Fragility

OOP addresses rigidity, fragility, immobility, and viscosity. Rigidity resists change, fragility causes cascading bugs, immobility prevents reuse, and viscosity pressures shortcuts. OOP principles—inheritance, polymorphism, modularity—combat these issues, producing resilient systems.

Dependencies and Productivity

Dependencies, if unmanaged, kill productivity. Tightly coupled modules spread changes and introduce bugs. OOP reduces unnecessary dependencies with encapsulation, modularity, and abstraction, enabling flexible and maintainable systems.

Dependency Tangle vs. Modular Architecture

A dependency tangle occurs when modules connect directly to each other, causing rigidity, fragility, immobility, and viscosity. A modular architecture has modules connect through a shared service, reducing dependencies and improving flexibility and maintainability.

Conclusion

Software quality is multi-faceted. Correctness and robustness together define reliability. Reusability and extendibility together define modularity. These are where software engineering struggles most, and where object-oriented methods bring the greatest benefits.