Architectural Styles and Non-Functional Requirements

- Jan Bosch. *Design and Use of Software Architectures.* Addison-Wesley, May 19, 2000.

Performance

- *That attribute of a computer system that characterizes the timeliness of the service delivered by the system* - SEI

- Measures
  - Response time, throughput, capacity, utilization

- Devices
  - Caching
  - Concurrency
  - Memory management

Maintainability

- Extent to which enhancements can be readily added to a system
  - Also called evolvability, flexibility, adaptability

- Measures
  - Coupling
  - Cohesion

- Devices
  - Encapsulation
  - Published interfaces
  - Subclassing
  - Indirection
  - Wrapping

Reliability

- Likelihood of failure in a given time period; continuity of service

- Measures
    - Mean Time To Failure (MTTF)

- Devices
    - Redundancy, fault tolerance, recovery blocks

Safety

- Extent to which system protects against injury, loss of life or property damage; absence of catastrophic consequences

- Measures
  - Interaction complexity, time coupling, fault-tree analysis

- Devices
  - Hardware interlocks
  - Fault containment

Security

- Extent to which system protects against unauthorized intrusion; confidentiality

- Measures
  – Levels (confidential, top secret); formal proof

- Devices
  – Authentification/authorization
  – Security kernels
  – Encryption
  – Auditing and logging
  – Access control

Pipe and Filter

- **Performance:** concurrency and buffering enhances throughput, but context switches can slow things down

- **Maintainability:** independent components improves reuse, but requirements changes can effect multiple components

- **Reliability:** reduced reliability due to "weakest link" (serial dependencies); that is, redundancy is antithetical

- **Safety:** reduced by multiple dependencies but verification may be enhanced because all output comes from a single source

- **Security:** simplicity increases opportunities for authentification, encryption and implementation of security levels

Layering

- **Performance:** response to external events must be passed up and down the layers; increased context swapping

- **Maintainability:** stable protocols lead to well-defined and reusable components; it may be possible to replace an entire layer

- **Reliability:** because an event may be "handled" in multiple layers, reliability is reduced; however, higher layers may have the oversight to provide redundancy

- **Safety:** easy to insert safety-monitoring layers

- **Security:** security layers can be added to intercept and evaluate external events before they can compromise a system

Blackboard

- **Performance:** lack of well-defined control flows may lead to redundant, administrative behavior (polling of repository)

- **Maintainability:** independent components enhance flexibility but changes to a common control paradigm or data format may be pervasive

- **Reliability:** independence of components can increase resilience; no overall definition of system behavior makes identification of problem situations difficult

- **Safety:** blackboard can promote spreading of bad data

- **Security:** access control enhanced because of common data storage; but dynamic addition of new components may reduce confidence

Object Orientation

- **Performance:** small objects lead to multiple context switches

- **Maintainability:** independent components can localize changes; but objects store references to each other increasing dependencies

- **Reliability:** decentralized control reduces opportunity for oversight; but encapsulation can reduce vulnerability to unintended interactions

- **Safety:** correspondence between real-world entities and objects improves intentionality and accountability

- **Security:** fragmentation (negative) and encapsulation (positive); explicit user interface objects can reduce vulnerability

Implicit Invocation

- **Performance:** bookkeeping and indirection; extra communications

- **Maintainability:** increased reuse due to independence

- **Reliability:** broadcast enables system-wide handling; increased interaction complexity

- **Safety:** interaction complexity

- **Security:** fragmentation (negative) and encapsulation (positive)

Summary

- **Performance:** concurrency from independent components can improve throughput; but distributed responsibility can lead to multiple context swaps

- **Maintainability:** flexibility in the face of requirements change; that is, what kinds of changes can affect multiple components; how easy is it to plug and play components

- **Reliability:** isolation of problems in single components; opportunities for redundancy and for oversight

- **Safety:** complexity, isolation

- **Security:** limited interfaces

© 2007, Spencer Rugaber