Software Quality Attributes

Software Architecture Perspective

- Software Architectures are usually complex
- We often reduce our detail recognition to an abstraction
  - Simply said we see high-level details only
- **Architectural Styles**
  - Layered style, Call and return
- **Architectural Patterns**
  - Model-View-Controller

Software Architecture Definition from UML

- Architecture is the organizational **structure** and associated **behavior** of a system.
- An architecture can be recursively decomposed into parts that interact through interfaces, relationships that connect parts, and constraints for assembling parts.
- Parts interact through interfaces include classes, components and subsystems

Software Architecture Definition from Bass

- The software architecture of a program or computing system is the **structure** or structures of the system..
- which comprise software elements..
- the externally visible properties of those elements,..
- and the relationships among them.

Quality Attributes (QA)

- There are many architectures to choose from
- To recognize advantages we use QA
  - Comparison criteria
  - Evaluation
  - Benefits
  - Trade-offs

Quality Attributes (QA)

- Runtime / Dynamic
  - How system works to fulfil requirements?
  - How does it deliver results?
  - What is the timing?
  - Are results correct?
  - Does it work when integrated?
- Static
  - How easy is to integrate system?
  - How easy can we test it?
  - How hard is to maintain it?
  - How much does it cost?
  - How to time the planning?

Symbolism essay (writer’s approach)

I write a generic essay

Then revise to add knowledge from symbolism

==

Write a generic program and then add QA

Is this the right approach?

No, know your QA priorities first!

Quality Attributes

(Design review / Code review / pair programming / static code analysis)

Dynamic / Runtime

- Performance
- Security
- Availability / Reliability
- Functionality
- Usability

Static

- Maintainability
- Reusability (Recycling)
- Integrability
- Testability
- Extendability
- Portability
- Modularity
  - Coupling / Cohesion

Dynamic QA

Dynamic / Runtime

- Performance
- Security
- Availability / Reliability
- Functionality
- Usability

Performance

- Responsiveness – time for a transaction/processing time
- Bulk processing – transaction per time

- Latency – delay between request and reaction
- Queue size
- Algorithm complexity
- Arrival rate
- Distribution of service
- Synchronization
- Interaction
  - Profiler (Visual VM)

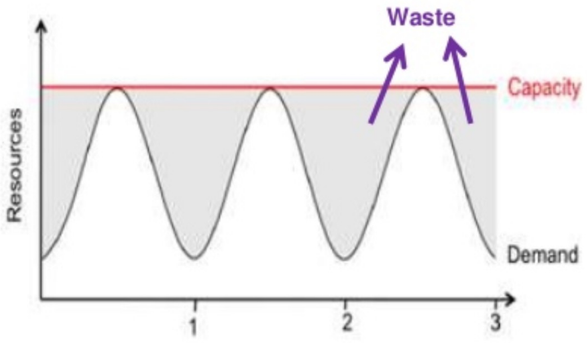
Security

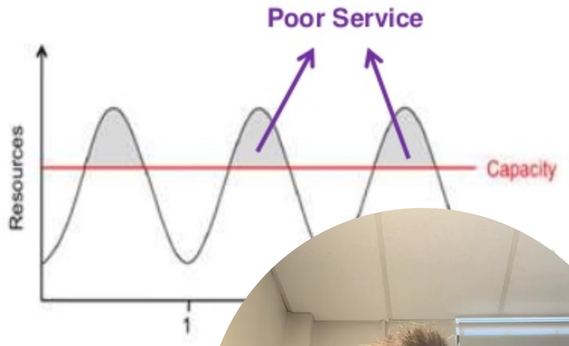
- Authentication – identity of user
- Authorization – rights
- Integrity of the system
- Resistance toward attacks
  - Data modification vs denial of service
- Strategies
  - Authentication service
  - Network monitoring, logging
  - Firewall, proxy
  - Building on top of SSL
  - User roles
  - Low-level security (component interaction, OS)

Availability / Reliability

uptime / (uptime + downtime)
the probability of failure-free **software** operation for a specified period of time in a specified environment

- Resistance towards errors
- Error prevention
- Error reporting / handling
- Failover
- Recovery





- Zabbix, nagios, backups, ticketing system for public

Functionality

- Orthogonality to other QA
- Structure implements functionality

- Does it work as intended?
- Tricky when multiple components involved
  - Each component has given responsibility

- System can be a monolith or distributes structure

Usability

- Learnability
  - Time to learn system
- Efficiency
  - Does it respond with sufficient timing
- Error avoidance
  - Input validation
- Memorability
  - Can user memorize features and use them longer time
- Satisfaction
  - Isn’t better to use Excel?

Static QA

- Maintainability
- Reusability (Recycling)
- Integrability
- Testability
- Extendability
- Modularity
  - Coupling / Cohesion

Maintainability (modifiability)

- Influenced by architecture
- How many components to rewrite

- Classification
  - Adding/changing functionality
  - Removal of not needed stuff
  - Adaptation to a new environment
  - Portability extensions
  - Refactoring to new modules

Portability

- Usability of the same software in different environments
- Platforms / hardware / devices

- Encapsulation of platform-specific decisions
- Indirection
  - Bridge/Adapted/Façade/DAO design patterns

Reusability

- Component reuse somewhere else
  - Low coupling and high cohesion
  - How large is the change?
  - How many components impacted/modified
  - Modifiable system ~ easy reuse

Reusability - sample 1

- System S is built from 100 components S1 – S100.
- New system T should also have 100 components.
- The developer notices that T1 and S1 are identical.
- S1 is reused since the new one would have to be implemented, and tests with it are expensive.
- Similar identity found for ‘n’ others

Reusability - sample 2

- System S is built from 100 components S1 – S100.
- This system is upgraded. However, the change only involved components S100 and S99.
- New components are designed, implemented, and tested.
- The new version of the system is named T.

Reusability - observation

- Reuse and maintainability ~ two sides of the same coin

Integrability

- How hard is it to integrate a component to the system

- Assigning responsibility to components
- Communication protocol, mechanism of interaction?
  - Method calls
  - REST, SOAP, XML
- Usage of standards

Testability

- How hard is it to test the system
- How easily can we deduce states
- Verification of states (encapsulation)
- Documentation