Software Architecture

Quality Attributes

Lecturer: Zhiguo Ding

Phone: 13750985775(660775)

E-mail:dingzhiguo@zjnu.cn

Office: Room 313, Building 21, College of MPI

What are Quality Attributes

- Often know as –ilities
  - Reliability
  - Availability
  - Portability
  - Scalability
  - **Performance** (!)
  - … but much more than this
- Part of a system's NFRs
  - "how" the system achieves its functional requirements

System Quality Attribute

- Performance
- Availability
- Usability
- Security

End User's view

- Maintainability
- Portability
- Reusability
- Testability

Developer's view

Business Community view

- Time To Market
- Cost and Benefits
- Projected life time
- Targeted Market
- Integration with Legacy System
- Rollout Schedule

A list of quality attributes exists in
ISO/IEC 9126-2001 Information Technology – Software Product Quality

Architecture and Quality Attributes

- Achieving quality attributes must be considered throughout design, implementation, and deployment
  - Satisfactory results are a matter of getting ‘the big picture’
- Architecture is critical to the realization of many qualities of interest in a system,
  - these qualities should be designed in and can be evaluated at the architectural level.
- Architecture, by itself, is unable to achieve qualities.
  - It provides the foundation for achieving quality, but this foundation will be to no avail if attention is not paid to the details.
- They influence each-other.

Classes of Quality Attributes

- Qualities of the system.
  - availability, modifiability, performance, security, testability, usability, scalability …
- Business qualities
  - Time to market
  - Cost and benefit
  - Projected lifetime of the system
  - Rollout schedule
- Qualities of the architecture itself
  - Conceptual integrity,
  - Correctness and completeness
  - Buildability

Comments

- Business qualities (such as time to market) are affected by the architecture.
- Iterlinked: Qualities of the architecture itself
  - indirectly affect other qualities, such as modifiability.
  - E.g., conceptual integrity,

System Quality Attributes

- Availability, modifiability, performance, security, testability, usability, scalability …

- Warning: use operational definitions!
  - Architects are often told:
    - "My application must be fast/secure/scale"
    - Far too imprecise to be any use at all
  - Quality attributes (QAs) must be made precise/measurable for a given system design, e.g.
    - *"It must be possible to scale the deployment from an initial 100 geographically dispersed user desktops to 10,000 without an increase in effort/cost for installation and configuration."*

Quality Attribute Specification

- QA’s must be concrete
- But what about testable?
  - Test scalability by installing system on 10K desktops?
- Often careful analysis of a proposed solution is all that is possible
  - “It’s all talk until the code runs”
  - Can you do better?

System Quality Attribute Problems

- Definitions are not operational.
- A focus of discussion is often on which quality a particular aspect belongs to.
  - Is a system failure an aspect of availability, an aspect of security, or an aspect of usability? All three attribute communities would claim ownership of a system failure.

- Each attribute community has developed its own vocabulary.
  - performance community →"events"
  - security community → "attacks"
  - the availability community → "failures"
  - the usability community → "user input."
    - All of these may actually refer to the same occurrence,

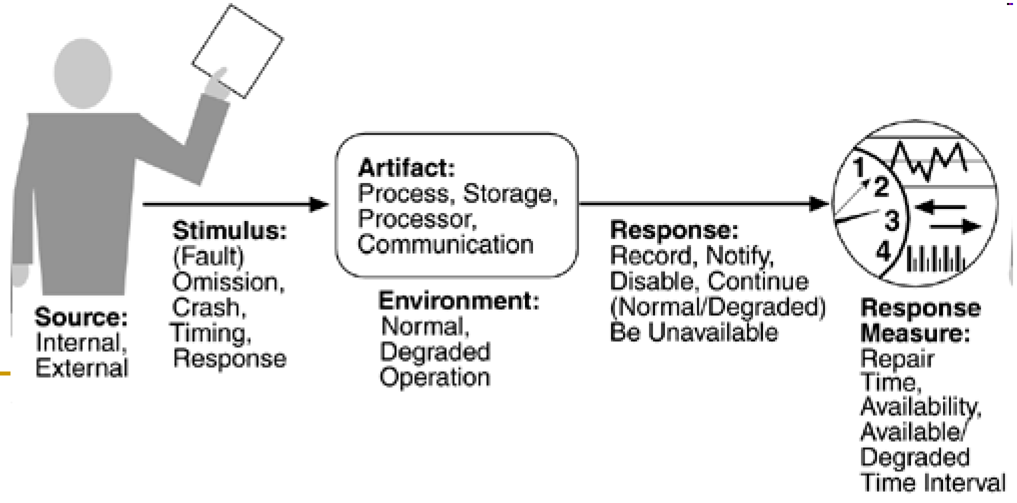
- Solution: use quality attribute ‘scenarios’ and unified la language

An Analysis Framework for Specifying Quality Attributes

QUALITY ATTRIBUTE SCENARIOS

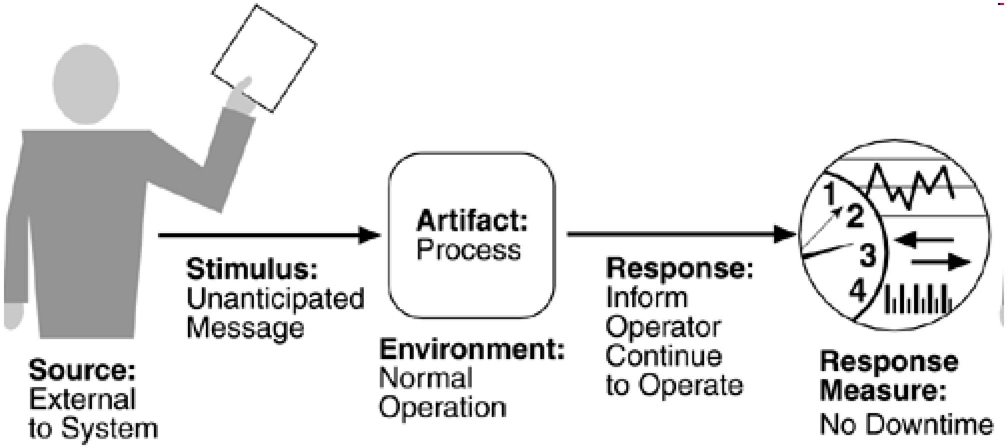
[For each quality-attribute-specific requirement.]

- *Source of stimulus*. This is some entity (a human, a computer system, or any other actuator) that generated the stimulus.
- *Stimulus*. A condition that needs to be considered when it arrives at a system.
- *Environment*. The stimulus occurs within certain conditions. The system may be in an overload condition or may be idle when the stimulus occurs.
- *Artifact*. Some artifact is stimulated. This may be the whole system or some pieces of it.
- *Response*. The activity undertaken after the arrival of the stimulus.
- *Response measure*. When the response occurs, it should be measurable in some fashion so that the requirement can be tested.



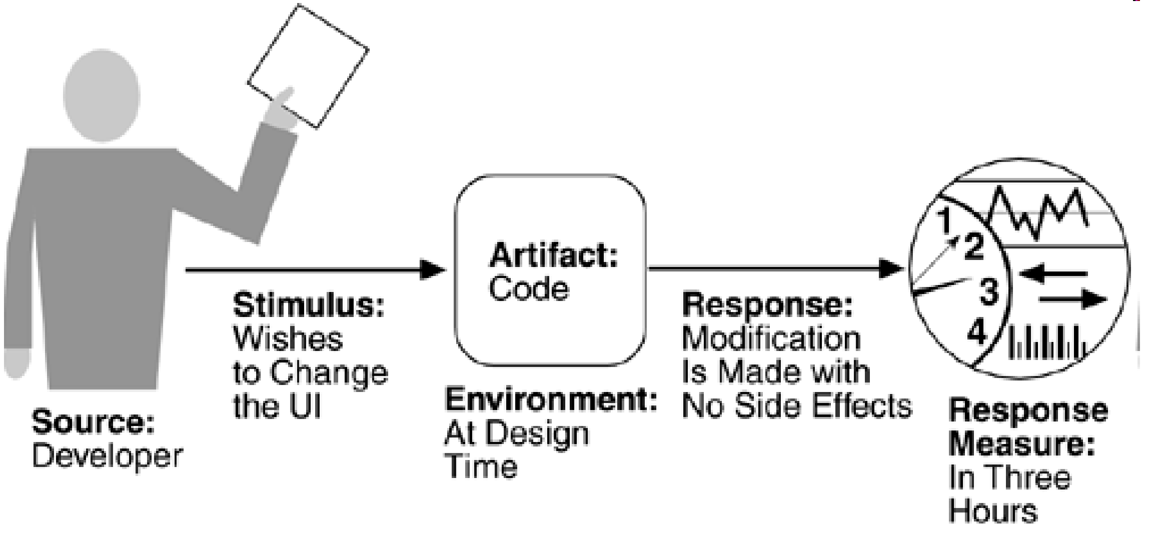
Availability Scenario Example

- System reaction to unanticipated message



Modifiability Scenario Example

- Specifying a modifiability QA requirement



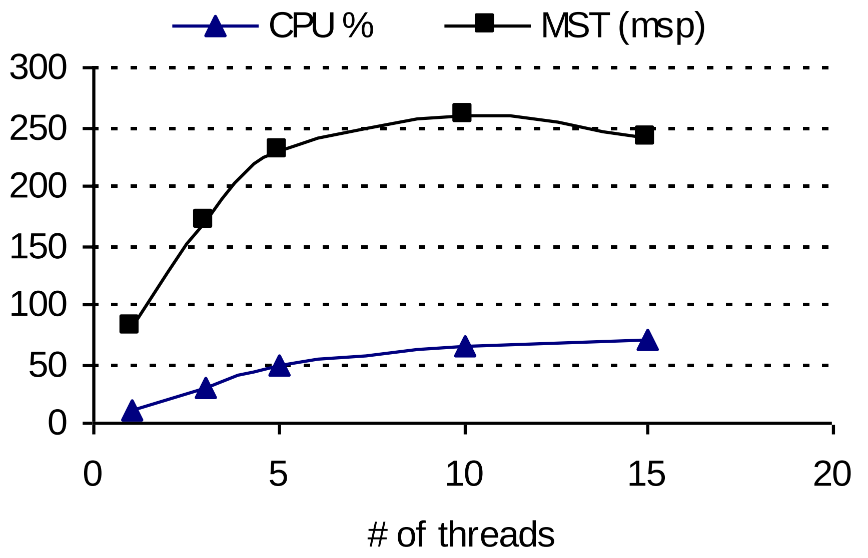
Performance

- Many examples of poor performance in enterprise applications
- Performance requires:
  - Multiple metrics: Throughput, response time, deadlines
  - Average (sustained) vs. peak.
  - Guarantees? Often specified as median and 99% tile.
- Enterprise applications often have strict performance requirements, e.g.
  - 1000 transactions per second
  - 3 second average latency for a request
  - Deadline that must be met

Performance - Throughput

- Measure of the amount of work an application must perform in unit time
  - Transactions per second
  - Messages per minute
- Is required throughput:
  - Average?
  - Peak?
- Many system have low average but high peak throughput requirements

Throughput Example



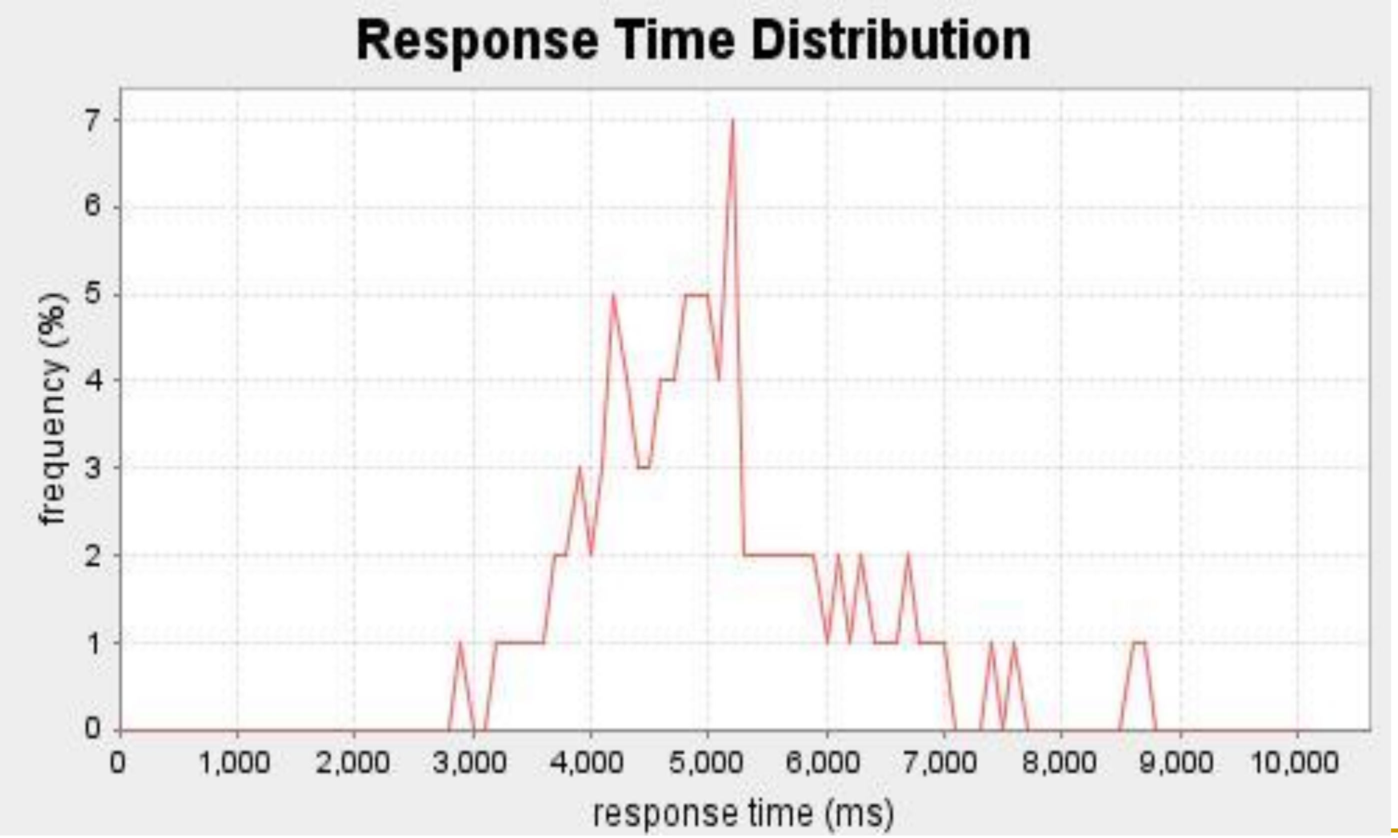
- Throughput of a message queuing system
  - Messages per second (msp)
  - Maximum sustainable throughput (MST)
- Note throughput changes as number of receiving threads increases

Performance - Response Time

- Measure of the latency an application exhibits in processing a request
- Usually measured in (milli)seconds
- Often an important metric for users
- Is required response time:
  - Guaranteed?
  - Average?
- E.g. 95% of responses in sub-4 seconds, and all within 10 seconds

Response Time

- Example shows response time distribution for a J2EE application



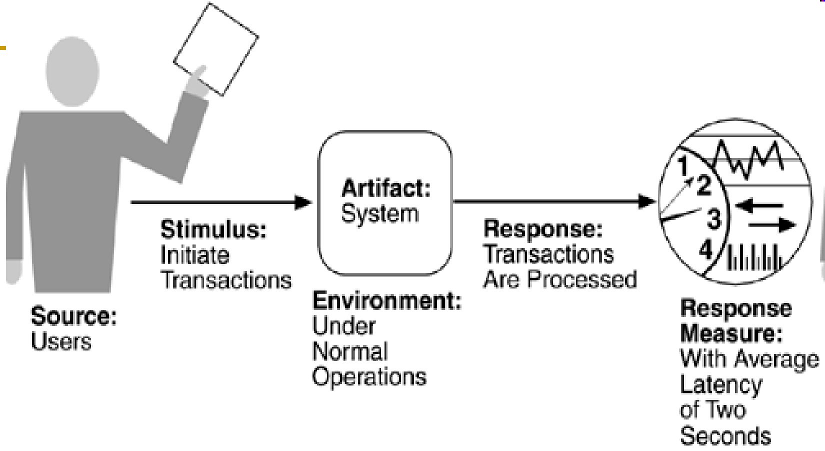
Performance - Deadlines

- ‘something must be completed before some specified time’
  - Payroll system must complete by 2am so that electronic transfers can be sent to bank
  - Weekly accounting run must complete by 6am Monday so that figures are available to management
- Deadlines often associated with batch jobs in IT systems.

Something to watch for …

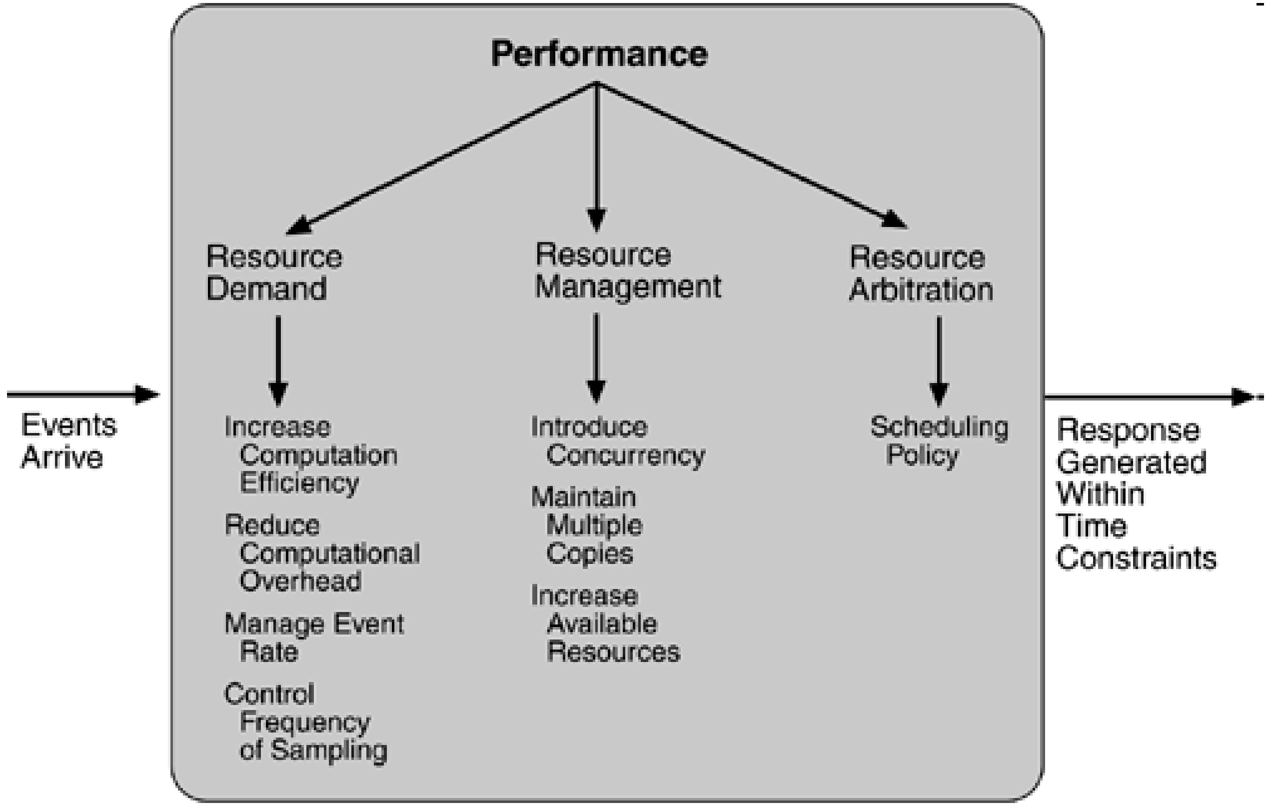
- What is a
  - Transaction?
  - Message?
  - Request?
- All are application / context specific measures.
  - System must achieve 100 mps throughput
    - BAD!!
  - System must achieve 100 mps peak throughput for *PaymentReceived* messages
    - GOOD!!!

Performance



Portion of Scenario	Possible Values
Source	One of a number of independent sources, possibly from within system
Stimulus	Periodic events arrive; sporadic events arrive; stochastic events arrive
Artifact	System
Environment	Normal mode; overload mode
Response	Processes stimuli; changes level of service
Response Measure	Latency, deadline, throughput, jitter, miss rate, data loss

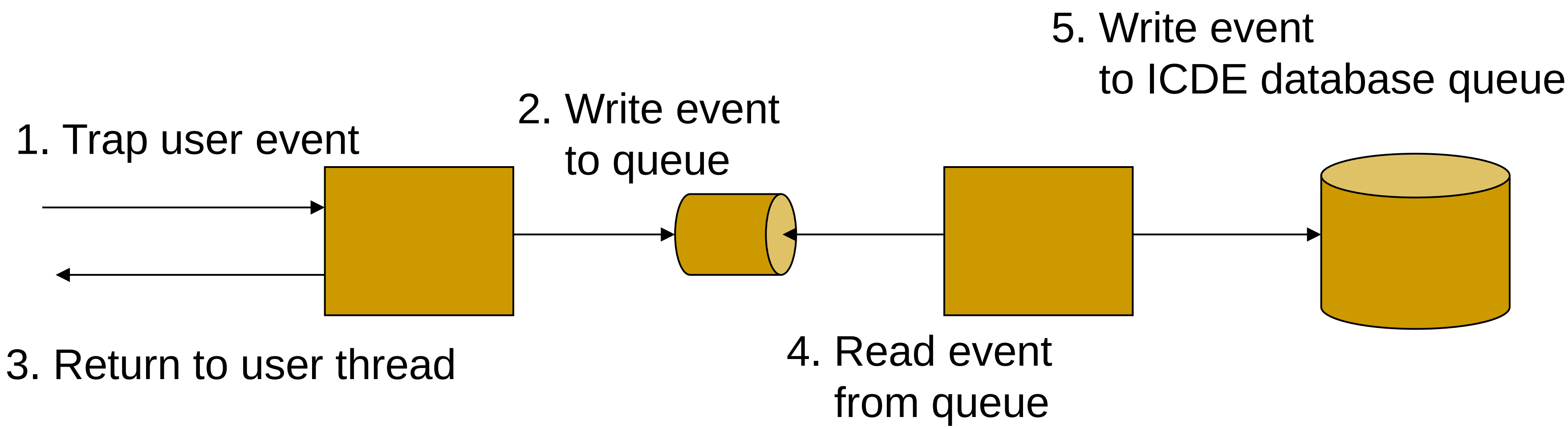
Performance Tactics



- Contributors to response time
  - Resource consumption
  - Blocked time
    - Contention for resources
    - Availability of resources
    - Dependency on other computations

ICDE Performance Issues

- Response time:
  - Overheads of trapping user events must be imperceptible to ICDE users
- Solution for ICDE client:
  - Decouple user event capture from storage using a queue



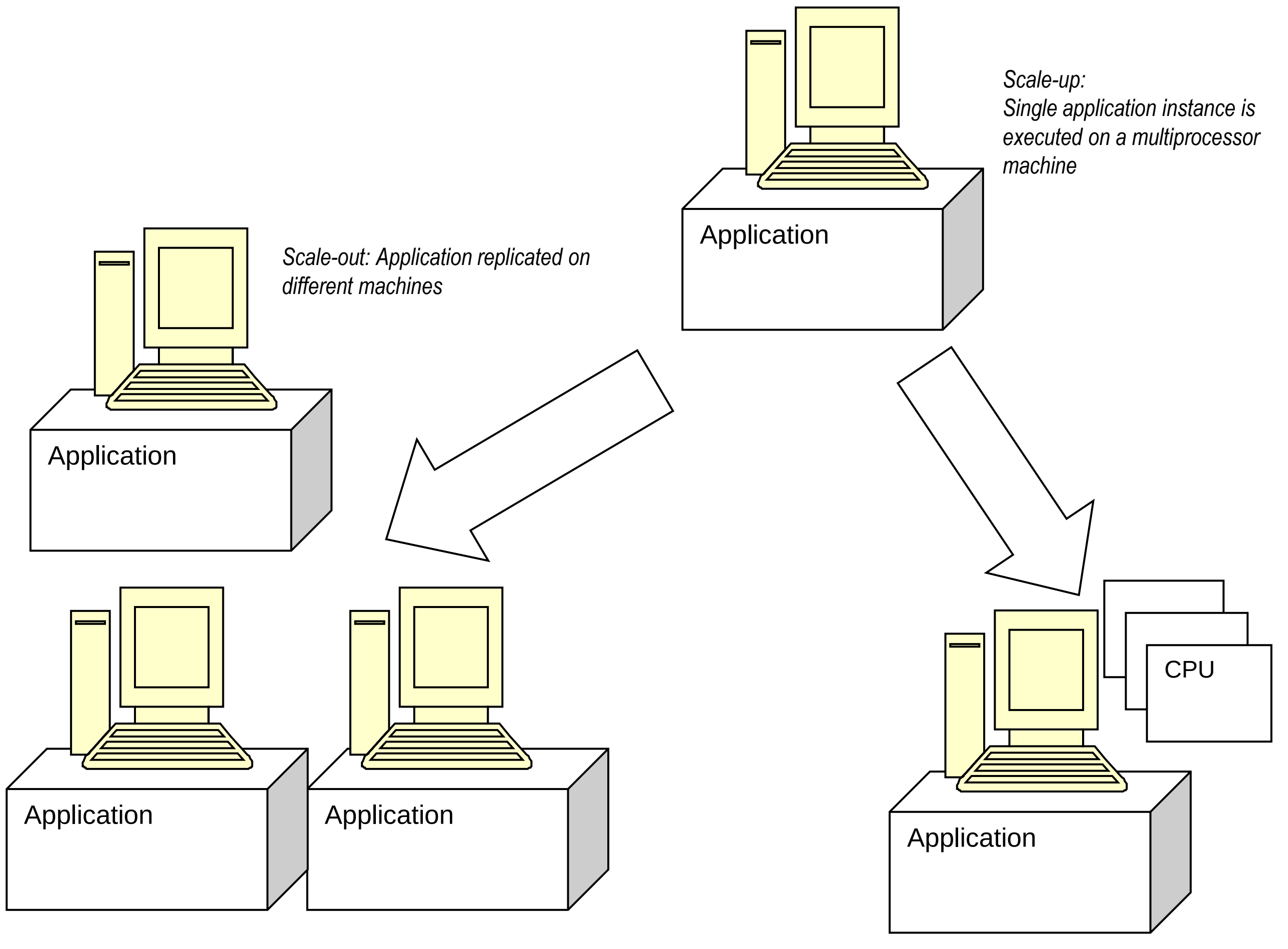
Scalability

- *"How well a solution to some problem will work when the size of the problem increases."*
- 4 common scalability issues in IT systems:
  - Request load
  - Connections
  - Data size
  - Deployments

Scalability – Request Load

- How does an 100 tps application behave when simultaneous request load grows? E.g.
  - From 100 to 1000 requests per second?
- Ideal solution, without additional hardware capacity:
  - as the load increases, throughput remains constant (i.e. 100 tps), and response time per request increases only linearly (i.e. 10 seconds).

Scalability – Add more hardware …



Scalability – the reality

- Adding more hardware should improve performance:
  - scalability must be achieved without modifications to application architecture
- Reality as always is different!
- Applications will exhibit a decrease in throughput and a subsequent exponential increase in response time.
  - increased load causes increased contention for resources such as CPU, network and memory
  - each request consumes some additional resource (buffer space, locks, and so on) in the application, and eventually these are exhausted

Scalability – J2EE example

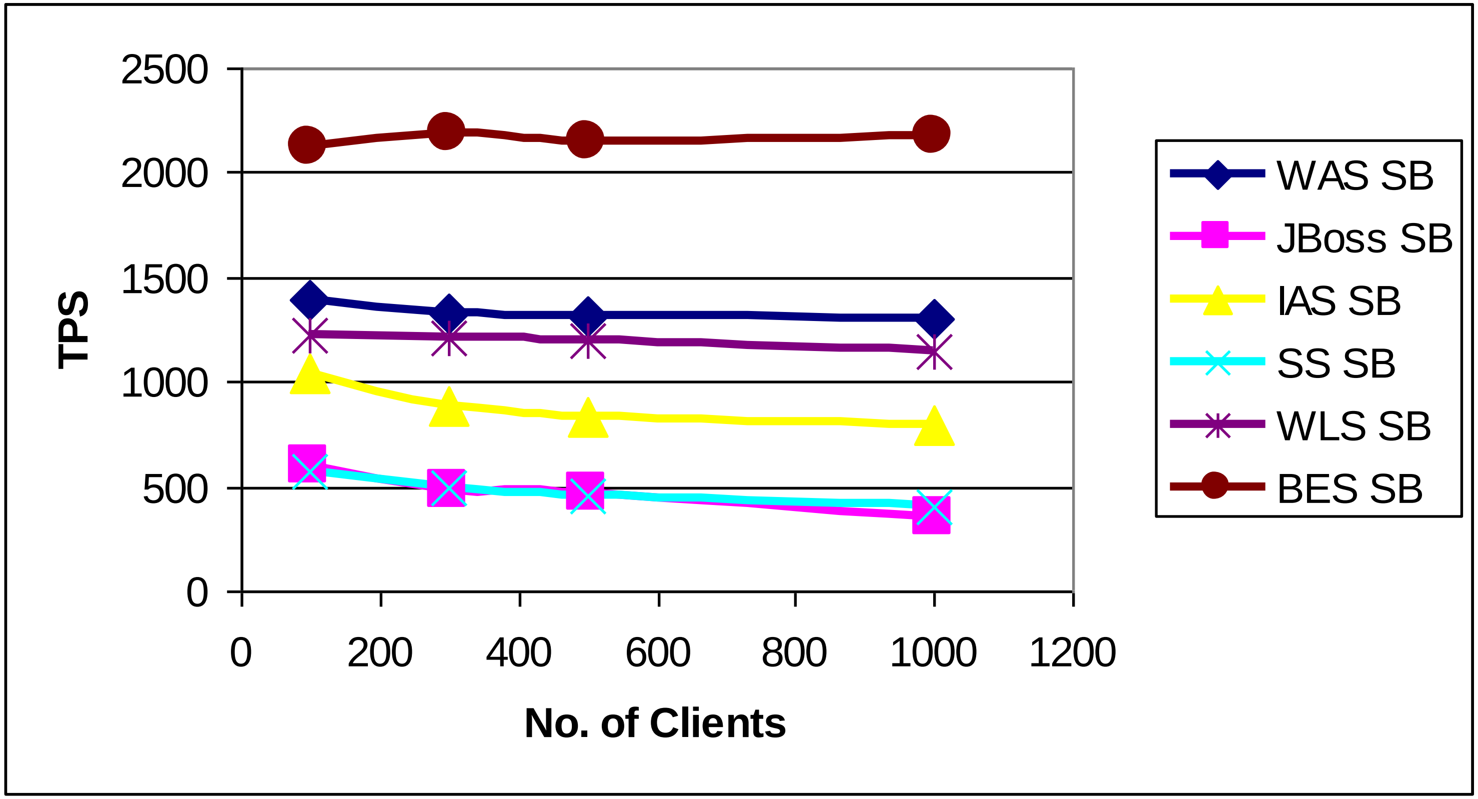


Figure shows how six different versions of the same application implemented using different JEE application servers perform as their load increases from 100 to 1,000 clients.

Scalability - connections

- What happens if number of simultaneous connections to an application increases
  - If each connection consumes a resource?
  - Exceed maximum number of connections?
- ISP example:
  - Each user connection spawned a new process
  - Virtual memory on each server exceeded at 2000 users
  - Needed to support 100Ks of users
  - Tech crash ….

Scalability – Data Size

- How does an application behave as the data it processes increases in size?
  - Chat application sees average message size double?
  - Database table size grows from 1 million to 20 million rows?
  - Image analysis algorithm processes images of 100MB instead of 1MB?
- Can application/algorithms scale to handle increased data requirements?

Scalability - Deployment

- How does effort to install/deploy an application increase as installation base grows?
  - Install new users?
  - Install new servers?
- Solutions typically revolve around automatic download/installation
  - E.g. downloading applications from the Internet

Scalability thoughts and ICDE

- Scalability often overlooked.
  - Major cause of application failure
  - Hard to predict
  - Hard to test/validate
  - Reliance on proven designs and technologies is essential
- For ICDE - application should be capable of handling a peak load of 150 concurrent requests from ICDE clients.
  - Relatively easy to simulate user load to validate this

Modifiability

- Modifiability measures how easy it **may** be to change an application to cater for new (non-) functional requirements.
  - **'may'** – nearly always impossible to be certain
  - Must estimate cost/effort
- Modifiability measures are only relevant in the context of a given architectural solution.
  - Components
  - Relationships
  - Responsibilities

Modifiability

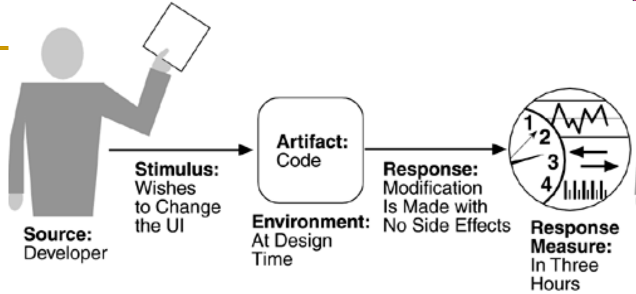
- Modifications to a software system during its lifetime are a fact of life.
  - Ideal: modifiable systems that are easier to change/evolve

- Modifiability should be assessed in context of how a system is likely to change
  - No need to facilitate changes that are highly unlikely to occur
  - Over-engineering!

- Impact of designing for modifiability is rarely easy to quantify

- One strategy: Minimizing dependencies
  - Changes isolated to single components likely to be less expensive than those that cause ripple effects across the architecture.

Modifiability



- *Source of stimulus*. Who makes the changes – e.g., developer, a system administrator, or an end user.
- *Stimulus*. What changes? Addition of a function, the modification of an existing function, deletion of a function; changing the qualities of the system
- *Artifact*. Specifies what is to be changed-the functionality of a system, its platform, its user interface, its environment, or another system with which it interoperates.
- *Environment*. When the change can be made-design time, compile time, build time, initiation time, or runtime.
- *Response*. Constraints on the change, how to test and deploy it.
- *Response measure*. Quantitative measure of cost.

Modifiability Scenarios

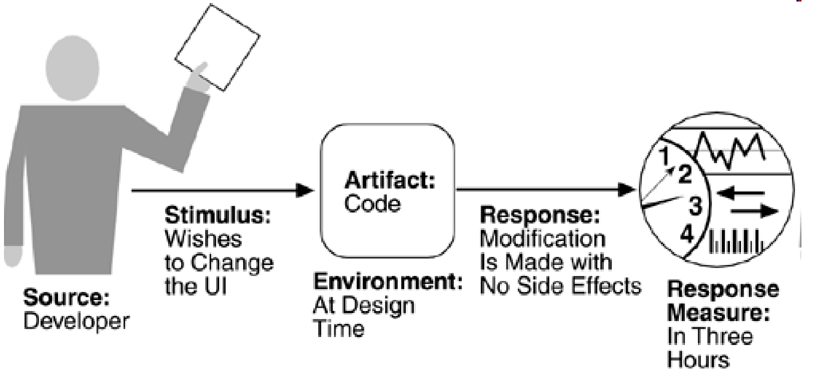
- Provide access to the application through firewalls in addition to existing “behind the firewall” access.
- Incorporate new features for self-service check-out kiosks.
- The COTS speech recognition software vendor goes out of business and we need to replace this component.
- The application needs to be ported from Linux to the Microsoft Windows platform.

Modifiability Analysis

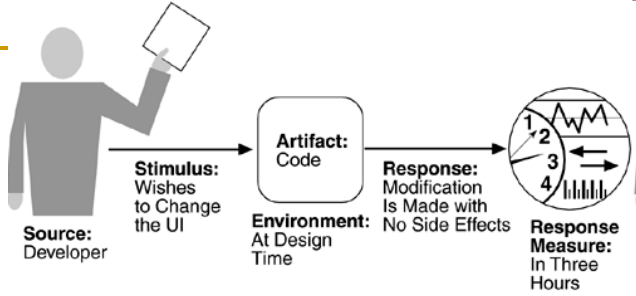
- Impact is rarely easy to quantify
- The best possible is a:
  - Convincing impact analysis of changes needed
  - A demonstration of how the solution can accommodate the modification without change.
- Minimizing dependencies increases modifiability
  - Changes isolated to single components likely to be less expensive than those that cause ripple effects across the architecture.

Modifiability Tactics

- Goals:
  - Reduce the number of modules affected by a change
    - ➔ localize modifications
  - Limited modifications of these modules
    - ➔ prevent ripple effects
  - Control deployment time and cost
    - ➔ defer binding time



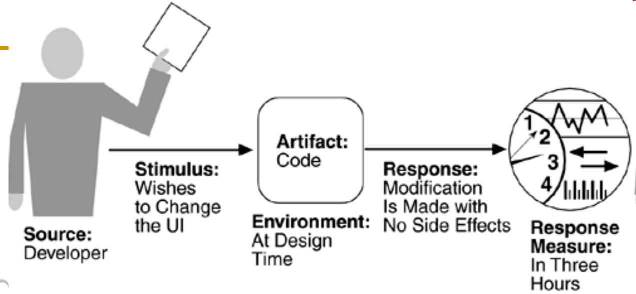
Modifiability Tactics



- Goals:
  - Reduce the number of modules affected by a change
    - ➔ localize modifications
  - Limited modifications of these modules
    - → prevent ripple effects
  - Control deployment time and cost
    - ➔ defer binding time

- *Techniques*
  - *Maintain semantic code coherence*
    - Coupling & cohesion metrics
  - *Abstract common services*
  - *Anticipate expected changes*
  - *Generalize the module*

Modifiability Tactics



- Goals:
  - Reduce the number of modules affected by a change
    - ➔ localize modifications
  - **Limited modifications of these modules**
    - **➔ prevent ripple effects**
  - Control deployment time and cost
    - ➔ defer binding time

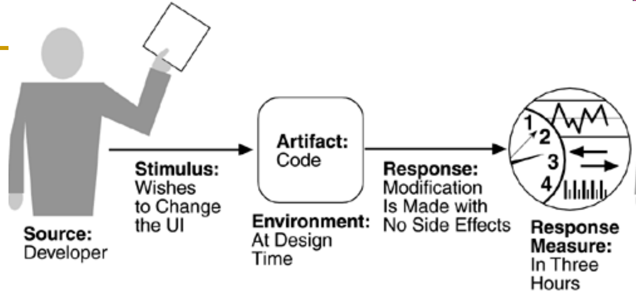
A taxonomy of dependencies between modules

- *Syntax of data and service invocation*
- *Semantics of data and service*
- *Sequence of data and control*
- *Identity of interfaces*
- *Location of called service*
- *QoS provided*

Tactics:

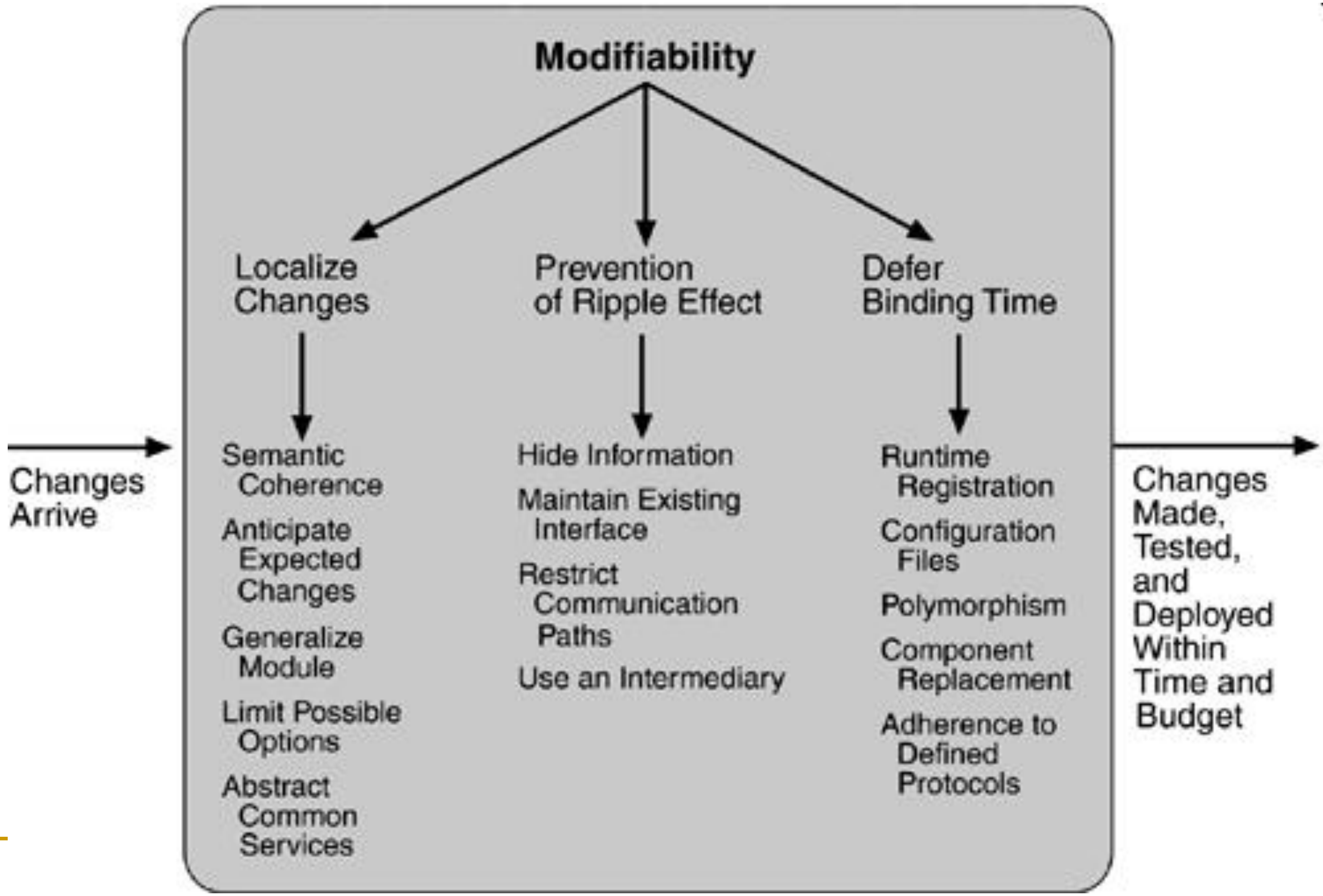
- *Hide information*
- *Maintain existing interfaces*
- *Restrict communication paths*
- *Use an intermediary (proxy)*

Modifiability Tactics



- Goals:
  - Reduce the number of modules affected by a change
    - ➔ localize modifications
  - Limited modifications of these modules
    - → prevent ripple effects
  - Control deployment time and cost
    - ➔ defer binding time
- Issues:
  - Reduce time to deploy
  - Allow non-programmers to make changes
- *Tactics*
  - Runtime registration
  - Configuration files
  - Dynamically loaded code

Modifiability Tactics Summary



Modifiability for ICDE

- The range of events trapped and stored by the ICDE client to be expanded.
- Third party tools to communicate new message types.
- Change database technology used
- Change server technology used

Security

- Difficult, specialized quality attribute:
  - Lots of technology available
  - Requires deep knowledge of approaches and solutions
- Security is a multi-faceted quality …

Security

- **Authentication:** Applications can verify the identity of their users and other applications with which they communicate.
- **Authorization:** Authenticated users and applications have defined access rights to the resources of the system.
- **Encryption:** The messages sent to/from the application are encrypted.
- **Integrity:** This ensures the contents of a message are not altered in transit.
- **Non-repudiation:** The sender of a message has proof of delivery and the receiver is assured of the sender’s identity. This means neither can subsequently refute their participation in the message exchange.

Security Approaches

- SSL(Security Socket Layer)
- PKI(Public Key Infrastructure)
- Web Services security
- JAAS
  - Java Authentication and Authorization Service
- Operating system security
- Database security
- Etc.

ICDE Security Requirements

- Authentication of ICDE users and third party ICDE tools to ICDE server
- Encryption of data to ICDE server from 3rd party tools/users executing remotely over an insecure network

Availability

- Key requirement for most IT applications
- Measured by the proportion of the required time it is useable. E.g.
  - E.g.
    - 100% available during business hours
    - No more than 2 hours scheduled downtime per week
    - 24x7x52 (100% availability)
  - MTTF / (MMTF + MTTR)
  - Note: scheduled downtime is excluded
- Related to an application’s reliability
  - Unreliable applications suffer poor availability

Availability

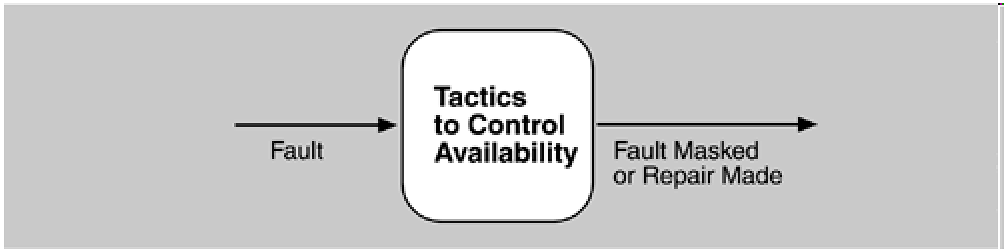
- Period of loss of availability determined by:
  - Time to detect failure
  - Time to correct failure
  - Time to restart application
- Strategies for high availability:
  - Eliminate single points of failure
  - Replication and failover
  - Automatic detection and restart
- Distinguish between faults and failures
- Recoverability (e.g. a database)
  - the capability to reestablish performance levels and recover affected data after an application or system failure

Availability Scenarios

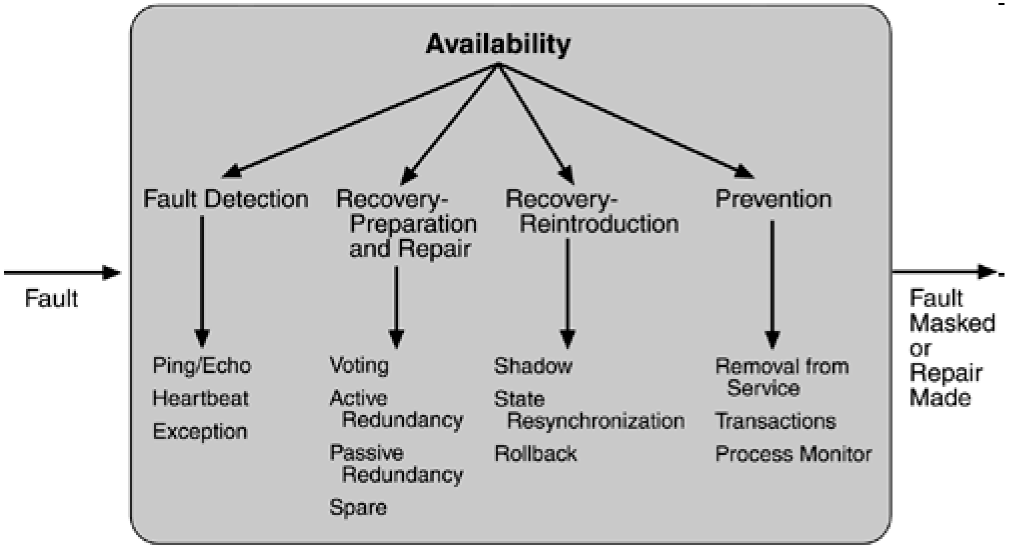
- *Source of stimulus.* Internal or external indications of faults or failure since the desired system response may be different.
- *Stimulus*. A fault of one of the following classes occurs.
  - omission. A component fails to respond to an input
  - crash. The component repeatedly suffers omission faults.
  - timing. A component responds but the response is early or late.
  - response (byzantine). A component responds with an incorrect value.
- *Artifact*. Specifies the resource that is required to be available,
- *Environment*. The state of the system when the fault or failure occurs may also affect the desired system response
- *Response*. Possible reactions to a system failure:
  - logging the failure, notifying selected users or other systems, switching to a degraded mode with either less capacity or less function, shutting down external systems, or becoming unavailable during repair.
- *Response measure*. Metric of success: e.g., availability percentage, or time to repair, etc.

Designing for Availability

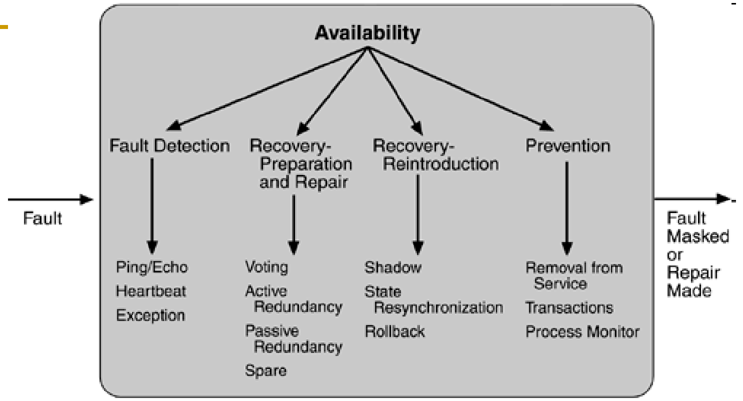
- Faults vs. Failures



- Tactics
  - Fault detection
  - Fault recovery
  - Fault prevention



Tactics for Availability



- Fault detection
  - Ping/echo;
  - Heartbeat;
  - Exceptions
- Fault recovery
  - Mostly redundancy based
    - [byzantine faults] Voting: multiple processes working in parallel.
    - [crash, timing] Active redundancy – hot restart
    - [crash] Passive redundancy (warm restart), spare.
  - Reintroduction: shadow operation, resynchronization, checkpoint/rollback
- Fault prevention
  - Removal from service; Transactions

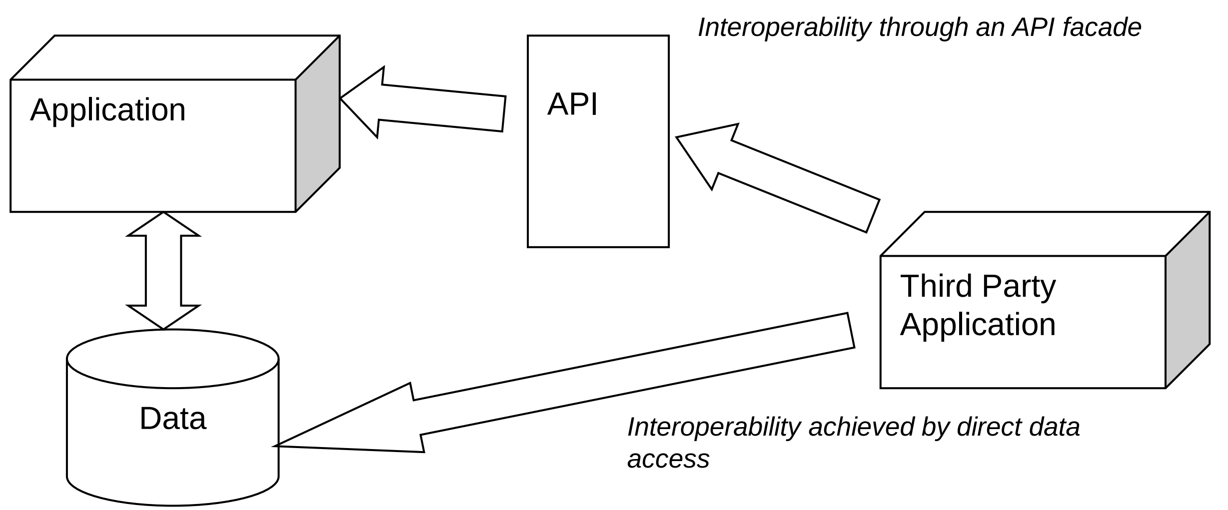
Availability for ICDE

- Achieve 100% availability during business hours
- Plenty of scope for downtime for system upgrade, backup and maintenance.
- Include mechanisms for component replication and failover

Integration

- Ease with which an application can be incorporated into a broader application context
  - Use component in ways that the designer did not originally anticipate
- Typically achieved by:
  - Programmatic APIs
  - Data integration

Integration Strategies



- Data – expose application data for access by other components
- API – offers services to read/write application data through an abstracted interface
- Each has strengths and weaknesses …

ICDE Integration Needs

- Revolve around the need to support third party analysis tools.
- Well-defined and understood mechanism for third party tools to access data in the ICDE data store.

Other Quality Attributes

- Portability
  - Can an application be easily executed on a different software/hardware platform to the one it has been developed for?
- Testability
  - How easy or difficult is an application to test?
- Supportability
  - How easy an application is to support once it is deployed?

Other Quality Attributes

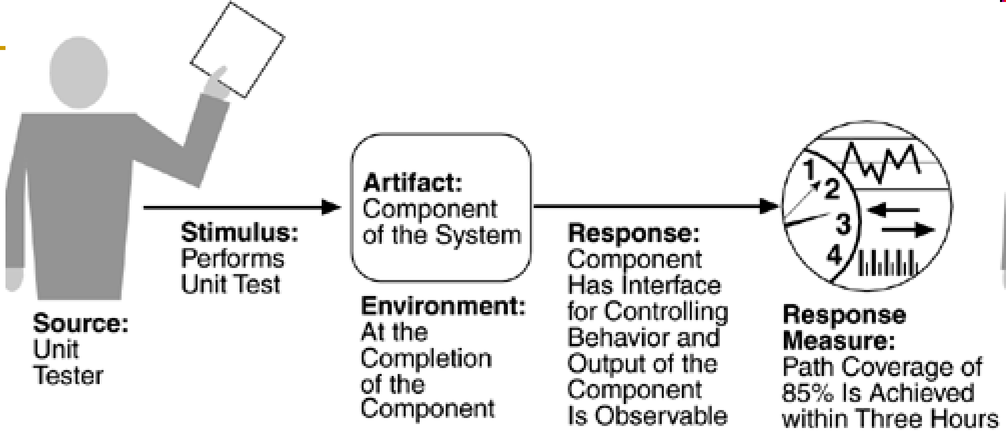
- Portability
  - Can an application be easily executed on a different software/hardware platform to the one it has been developed for?
- Testability
  - How easy or difficult is an application to test?
- Supportability
  - How easy an application is to support once it is deployed?

Testability

- Estimate: 40% of development cost goes to testing
- Testability: assuming that the software has at least one fault, the probability that this will be detected in the next testing round
- Need a system that is *controllable* and *observable*

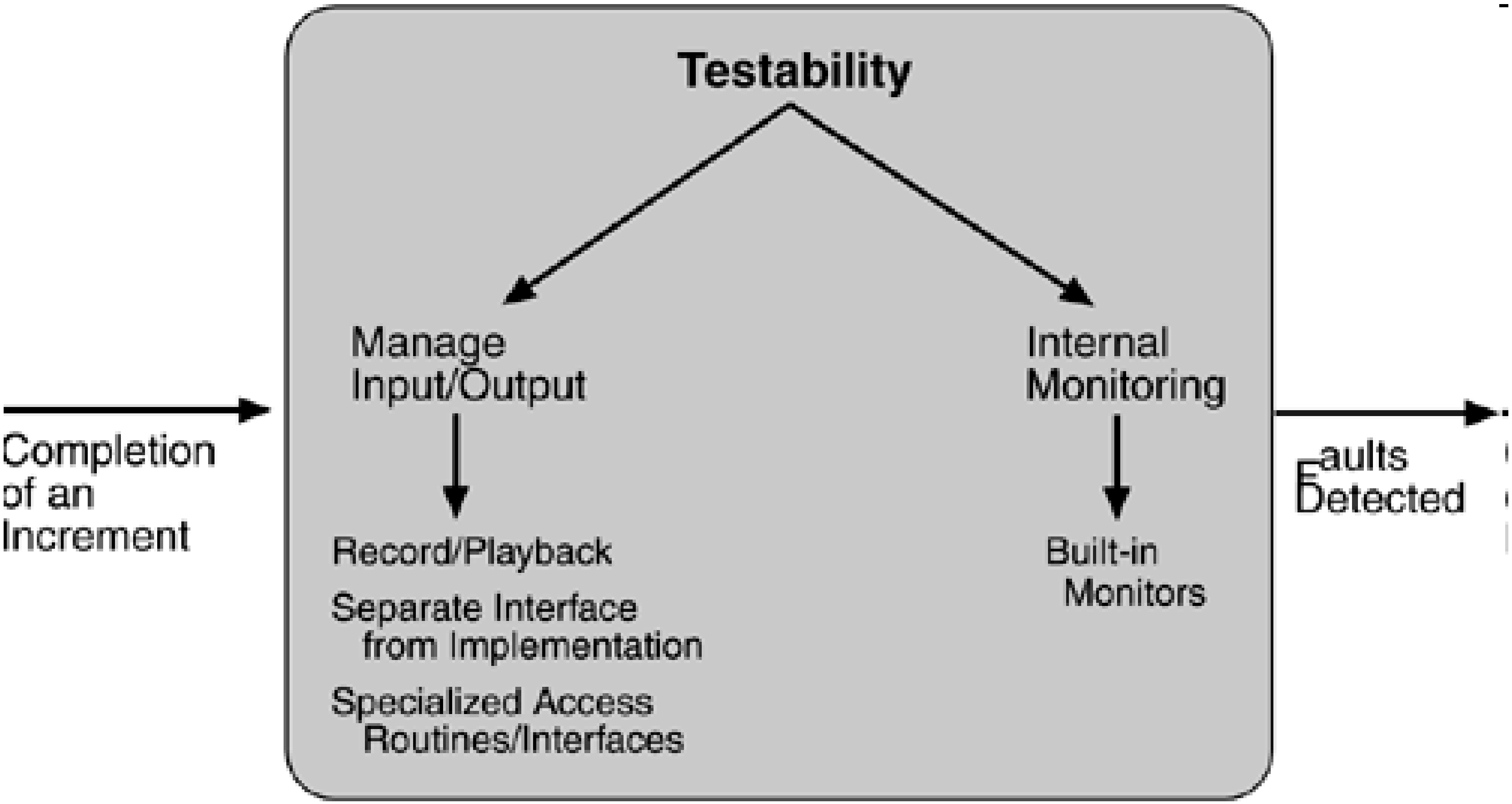
- Testing harness: control internal state of components,, pass inputs to the system, observe output

Testability



Portion of Scenario	Possible Values
Source	Unit test developer; or Increment integrator; or System verifier; or Client acceptance tester; or System user
Stimulus	Analysis, architecture, design, class, subsystem integration completed; system delivered
Artifact	Piece of design, piece of code, complete application
Environment	At design time, at development time, at compile time, at deployment time
Response	Provides access to state values; provides computed values; prepares test environment
Response Measure	Percent executable statements executed Probability of failure if fault exists Time to perform tests Length of longest dependency chain in a test Length of time to prepare test environment

Testability Tactics



Other Quality Attributes

- Portability
  - Can an application be easily executed on a different software/hardware platform to the one it has been developed for?
- Testability
  - How easy or difficult is an application to test?
- Supportability
  - How easy an application is to support once it is deployed?

Concern:

- Q: Are functionality and quality attributes orthogonal?
- Q: How to generate meaningful scenarios in practice?
- Q: Completeness?

Design Trade-offs

- QAs are rarely orthogonal
  - They interact, affect each other
  - highly secure system may be difficult to integrate
  - highly available application may trade-off lower performance for greater availability
  - high performance application may be tied to a given platform, and hence not be easily portable
- Architects must create solutions that makes sensible design compromises
  - not possible to fully satisfy all competing requirements
  - Must satisfy all stakeholder needs
  - This is the difficult bit!

Summary

- QAs are part of an application's non-functional requirements
- Many QAs
- Architect must decide which are important for a given application
  - Understand implications for application
  - Understand competing requirements and trade-offs

Selected Further Reading

- Ian Gorton, **Essential Software Architecture** (2nd Edition) , Springer; 2011.
- L. Chung, B. Nixon, E. Yu, J. Mylopoulos, (Editors). Non-Functional Requirements in Software Engineering Series: The Kluwer International Series in Software Engineering. Vol. 5, Kluwer Academic Publishers. 1999.
- J. Ramachandran. Designing Security Architecture Solutions. Wiley & Sons, 2002.
- I.Gorton, L. Zhu. *Tool Support for Just-in-Time Architecture Reconstruction and Evaluation: An Experience Report*. International Conference on Software Engineering (ICSE) 2005, St Loius, USA, ACM Press

Thank you ！