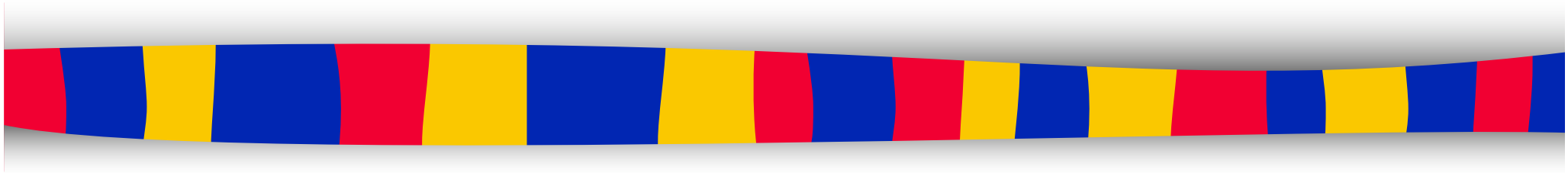
CISC 326

Game Architecture

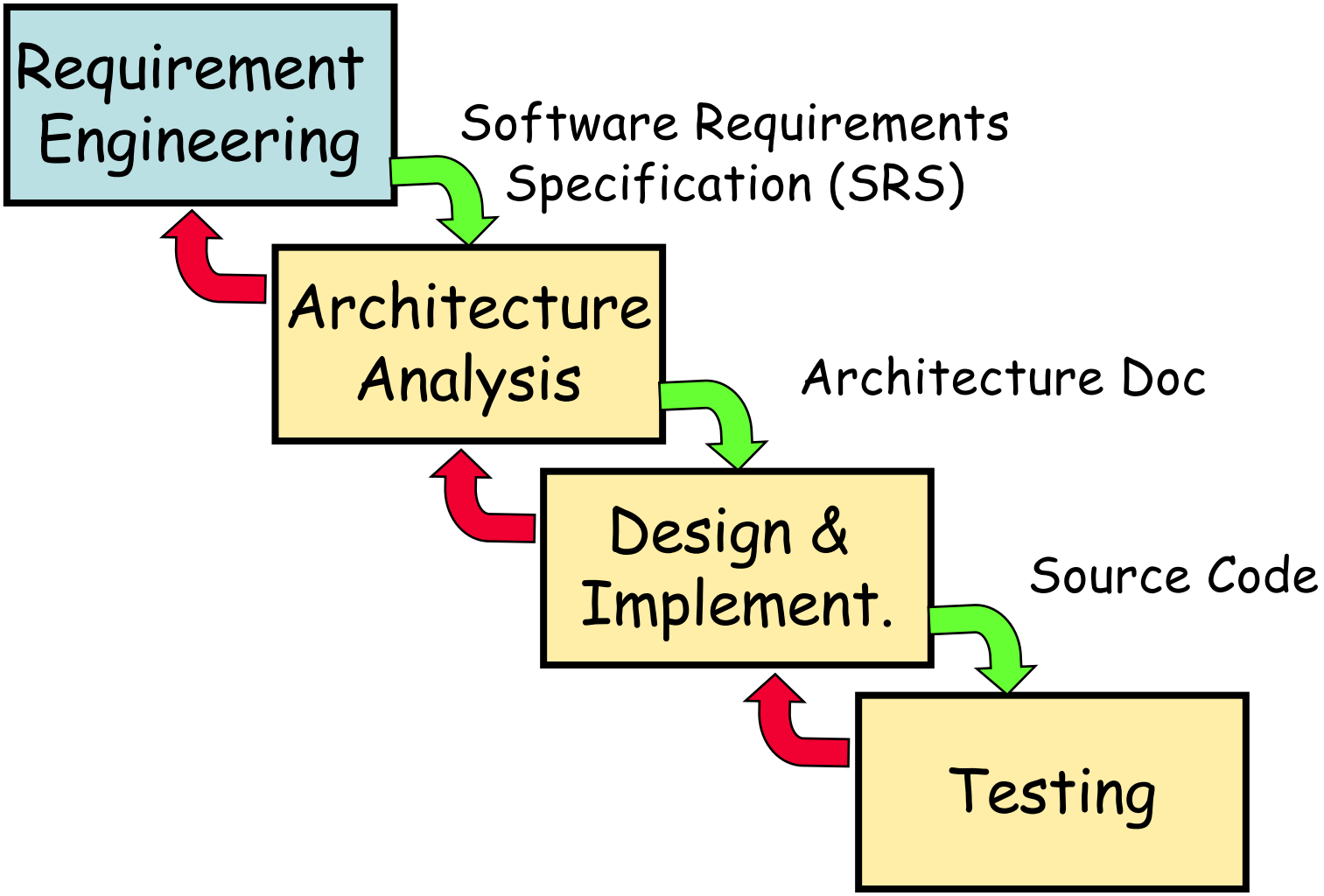


Module 03:

Non Functional Requirements (NFR) – Quality Attributes

Ahmed E. Hassan

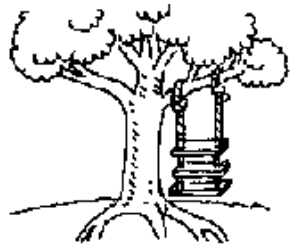
Waterfall Development Process



Where Do Requirements Come From?

- Requirements come from users and stakeholders who have demands/needs
- An analyst/requirement engineer:
  - Elicits these demands/needs (raw requirements)
  - Analyzes them for consistency, feasibility, and completeness
  - Formulates them as requirements and write down a specification
  - Validates that the gathered requirements reflect the needs/demands of stakeholders:
    - *Yes, this is what I am looking for.*
    - *This system will solve my problems.*

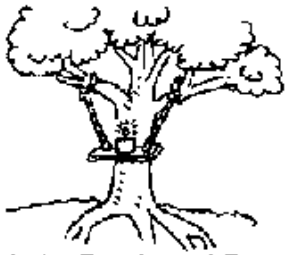
Many Stakeholders
Different Visions, Conflicting Goals



1. As Management Requested It

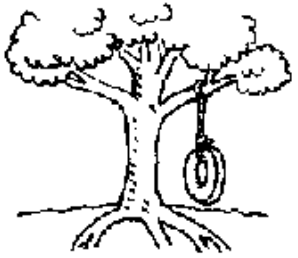
2. As Specified in the Project Request

3. As Designed By The Senior Analyst



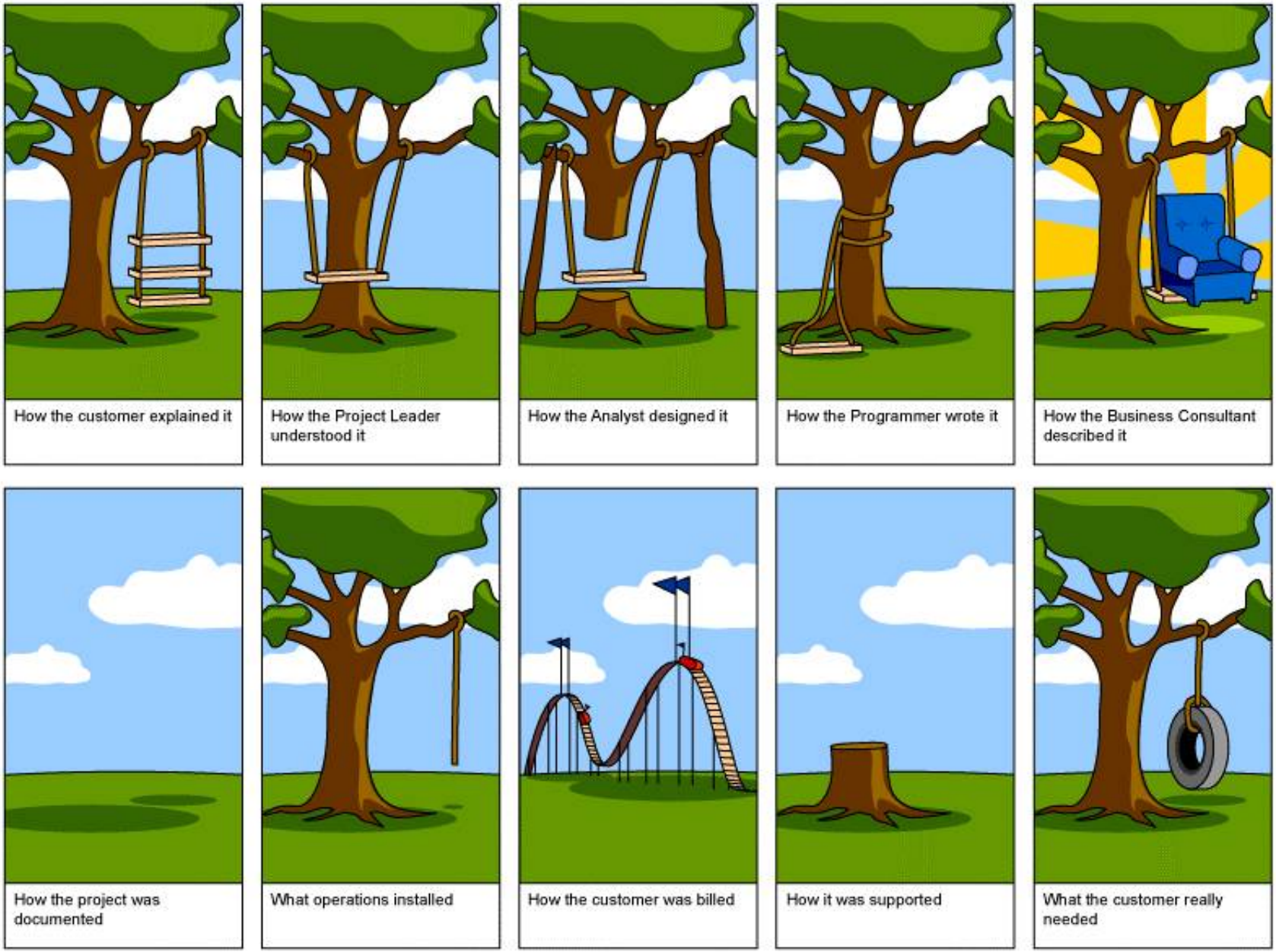
4. As Produced By The Programmers

5. As Installed



6. What The User Wanted

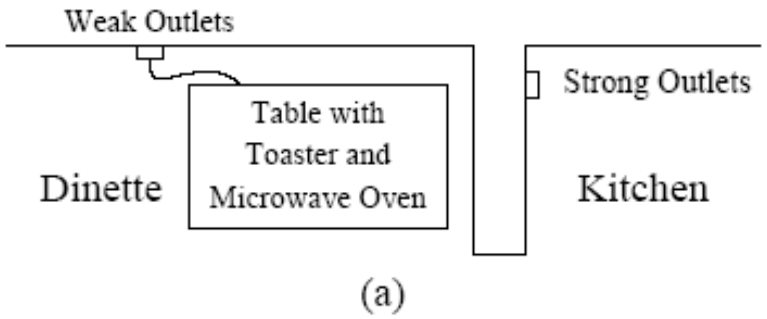
More Stakeholders



Questions that Arise During Requirement Gathering

- Is this a need or a requirement?
- Is this a nice-to-have vs. must-have?
- Is this the goal of the system or a contractual requirement?
- Do we have to program in Java? Why?

A Good Understanding of the Problem is Essential



(a)

[Berry 02]

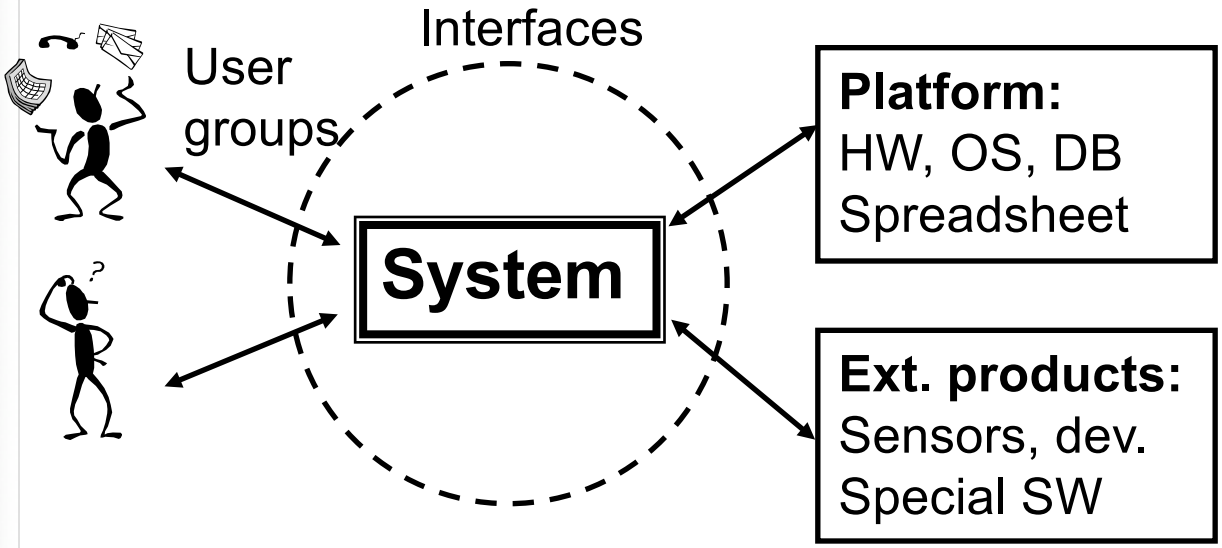
A Good Understanding of Problem is Essential

- Elevators in skyscraper
- Toothpaste boxes
- Out of coverage simulator
- Ice cream store in Lake Como (Handicap service)
- High score tracking

Types of Requirements

- **Functional Requirements**
  - Specify the function of the system
  - F(input, system state) → (output, new state)
- **Non-Functional Requirements (Constraints)**
  - **Quality Requirements**
    - Specify how well the system performs its intended functions
    - Performance, Usability, Maintenance, Reliability, Portability
  - **Managerial Requirements**
    - When will it be delivered
    - Verification (how to check if everything is there)
    - What happens if things go wrong (legal responsibilities)
  - **Context / Environment Requirements**
    - Range of conditions in which the system should operate

Contents of Requirement Specification



Data requirements:

- System state: Database, comm. states
- Input/output formats

Functional requirements, each interface:

- Record, compute, transform, transmit
- Theory: F(input, state) -> (output, state)
- Function list, pseudo-code, activity diagram
- Screen prototype, support tasks xx to yy

Quality reqs:

- Performance
- Usability
- Maintainability
- . . .

Other deliverables:

- Documentation
- Install, convert, train . . .

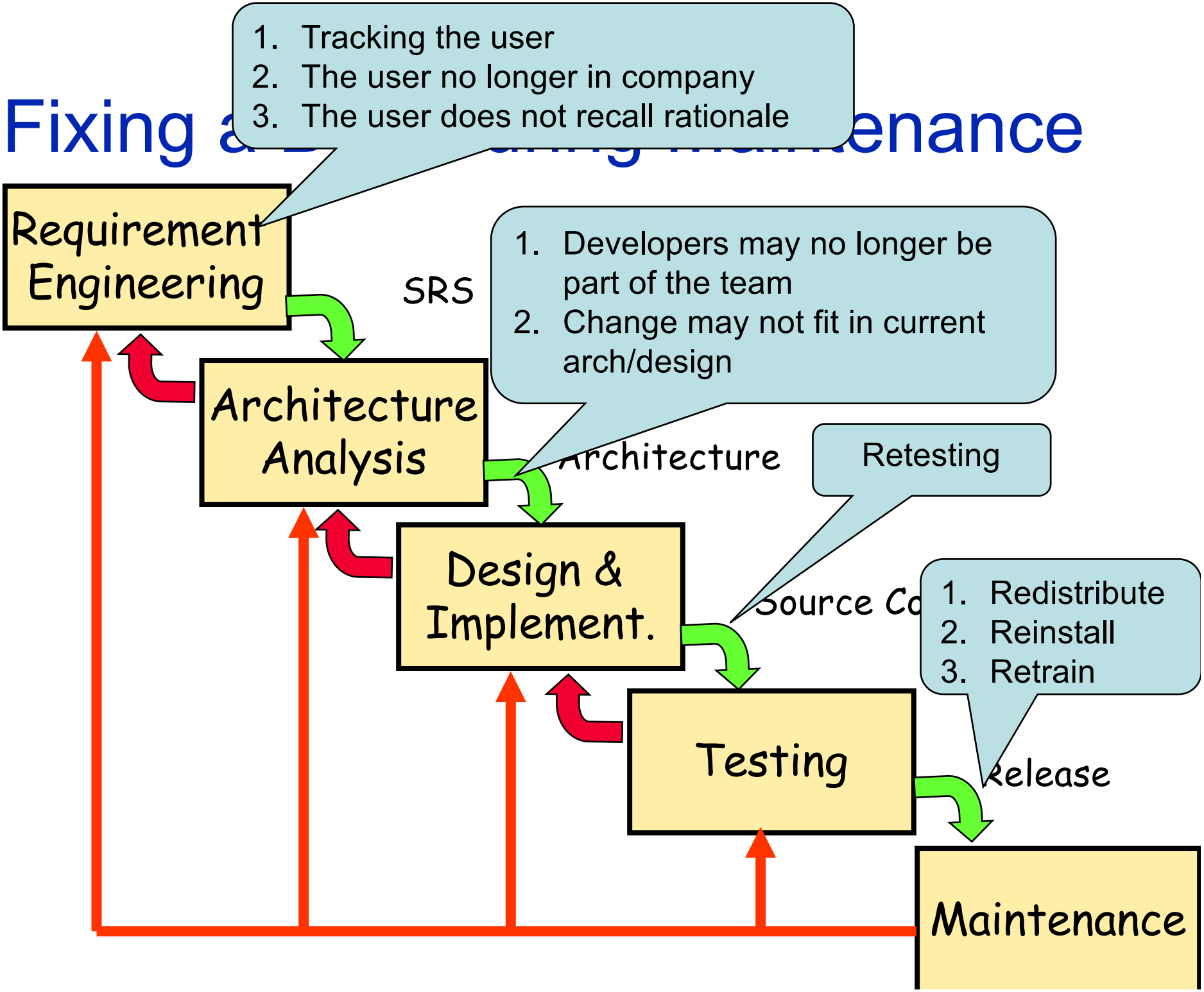
Managerial reqs:

- Delivery time
- Legal
- Development process . . .

Helping the reader:

- Business goals
- Definitions
- Diagrams . . .

Fixing a[illegible]enance

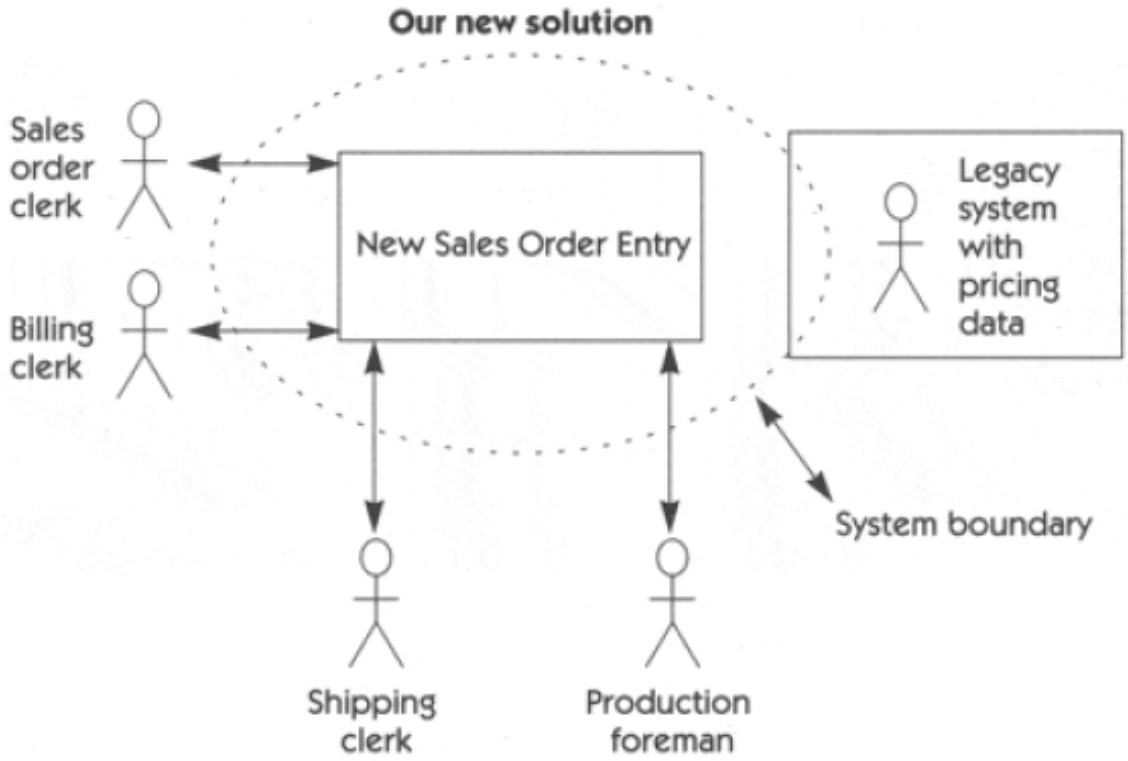


Software Specification

- Specification acts as a bridge between the real-world environment (demands of stakeholders) and the software system

System Perspective Diagram

- **System perspective** is a block diagram that describes the boundaries of the system, its users, and other interfaces



Example Constraints

Source	Constraint	Rationale
Operational	An exact copy of sales order data must remain on the legacy database for up to one year.	The risk of data loss is too great; we will need to run in parallel for up to one year.
Systems and OS	The applications footprint on the server must be less than 20 megabytes.	We have limited server memory available.
Equipment budget	The system must be developed on existing server and host; new client hardware for users may be provided.	Cost control and maintenance of existing systems.
Personnel budget	Fixed staffing resource; no outsourcing.	Fixed operating costs as per the current budget.
Technology mandate	New OO methodology to be used.	We believe that this technology will increase productivity and increase reliability of the software.

Fig 9.1 Quality criteria for a specification

Classic: A good requirement spec is:

Correct
Each requirement reflects a need.

Complete
All necessary requirements included.

Unambiguous
All parties agree on meaning.

Consistent
All parts match, e.g. E/R and event list.

Ranked for importance and stability
Priority and expected changes per requirement.

Modifiable
Easy to change, maintaining consistency.

Verifiable
Possible to see whether requirement is met.

Traceable
To goals/purposes, to design/code.

Necessary AND Feasible

Additional:

Traceable from goals to requirements.
Understandable by customer and developer.

From: Soren Lauesen:
Software Requirements

Non Functional Requirements (NFR)

- NFRs are often called “quality attributes”
- NFRs specify how well the system performs its functions:
  - How fast must it respond?
  - How easy must it be to use?
  - How secure does it have to be against attacks?
  - How easy should it be to maintain?

Non Functional vs. Functional Requirements

- **Functional requirements are like verbs**
  - The system should have a secure login
- **NFRs are like attributes for these verbs**
  - The system should provide a *highly* secure login
- Two products could have exactly the same functions, but their attributes can make them entirely different products

Non Functional vs. Functional Requirements

- Functional reqs must be met (ie. mandatory)
- NFRs could be:
  - Mandatory: eg. response time a valve to close
    - The system is unusable
  - Not mandatory: eg. response time for a UI
    - The system is usable but provides a non-optimal experience
- The importance of meeting NFRs increases as a market matures. Once all products meet the functional reqs, users start to consider NFRs

Expressing NFRs

- Functional are usually expressed in Use-Case form
- NFR cannot be expressed in Use-Case form since they usually do not exhibit externally visible functional behaviour
- NFRs are very important: Often represent 20% of the requirements and are the hardest to elicit and specify
- It is not enough to simply list that a system should satisfy a list of NFRs. The requirements should be clear, concise, and measurable
- Defining good NFRs requires not only the involvement of the customer but the developers too
  - Ease of maintenance (lower cost) vs. ease of adaptability
  - Realistic performance requirements

The effects of NFRs on high level design and code

- NFRs require special consideration during the software architecture/high level design phase
- They affect the various high level subsystems
- Their implementation does not map usually to a particular subsystem (except in the case of portability where an O/S abstraction layer may be introduced)
- It is very hard to modify an NFR once you pass the architecture phase:
  - Consider making an already implemented system more secure, more reliable, etc.

Examples of NFRs

- **Performance:** 80% of searches will return results in <2 secs
- **Accuracy:** Will predict cost within 90% of actual cost
- **Portability:** No technology should be used to prevent from moving to Linux
- **Reusability:** DB code should reusable and exported into a library
- **Maintainability:** Automated test must exist for all components. Over night tests must be run (all tests should take less than 24 hrs to ruin)
- **Interoperability:** All config data stored in XML. Data stored in a SQL DB. No DB triggers. Java
- **Capacity:** System must handle 20 Million Users while maintaining performance objectives!
- **Manageability:** System should support system admin in troubleshooting problems

Essential Software Architecture

Session 2:

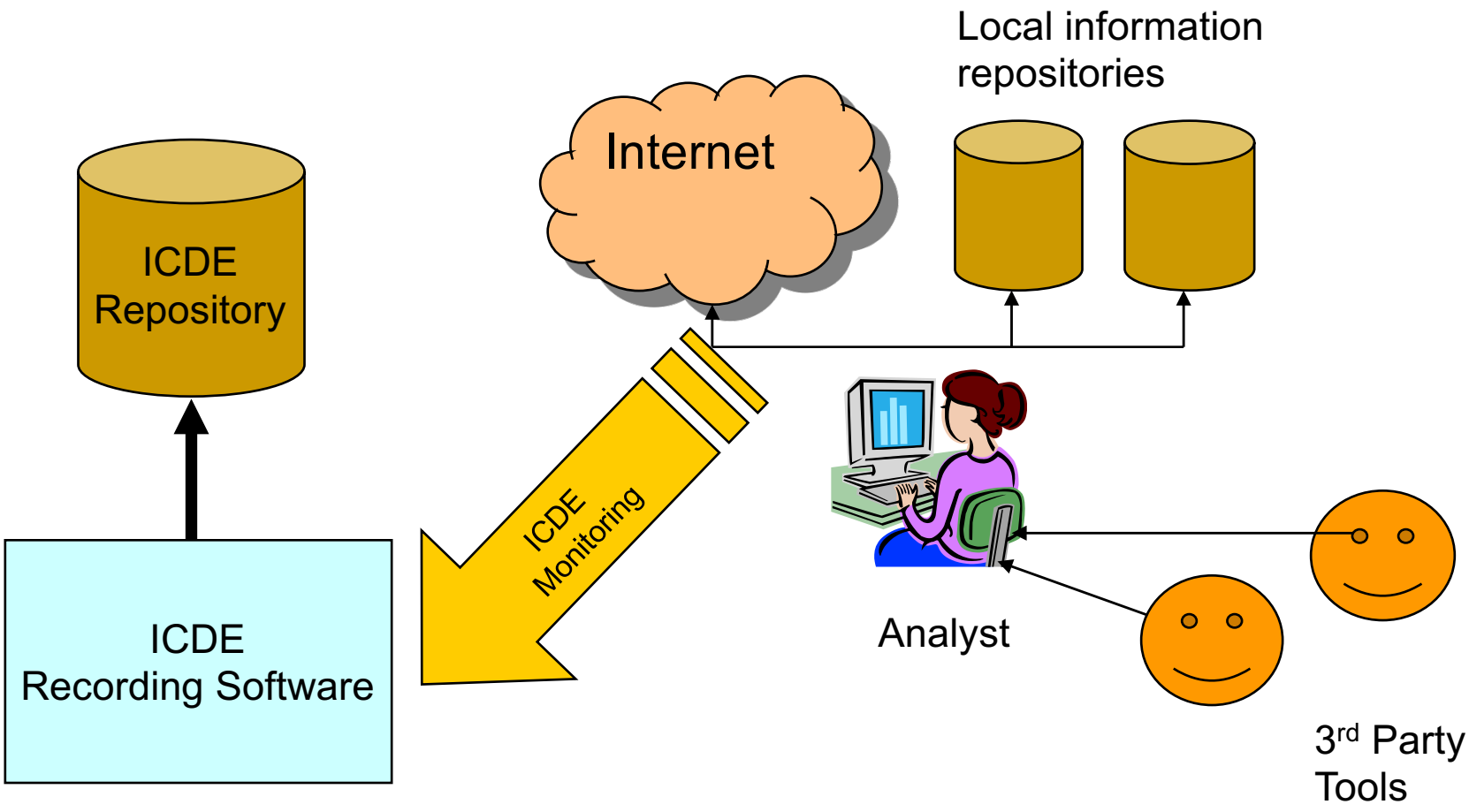
Introduction to the Case Study

[Slides by Ian Gorton]

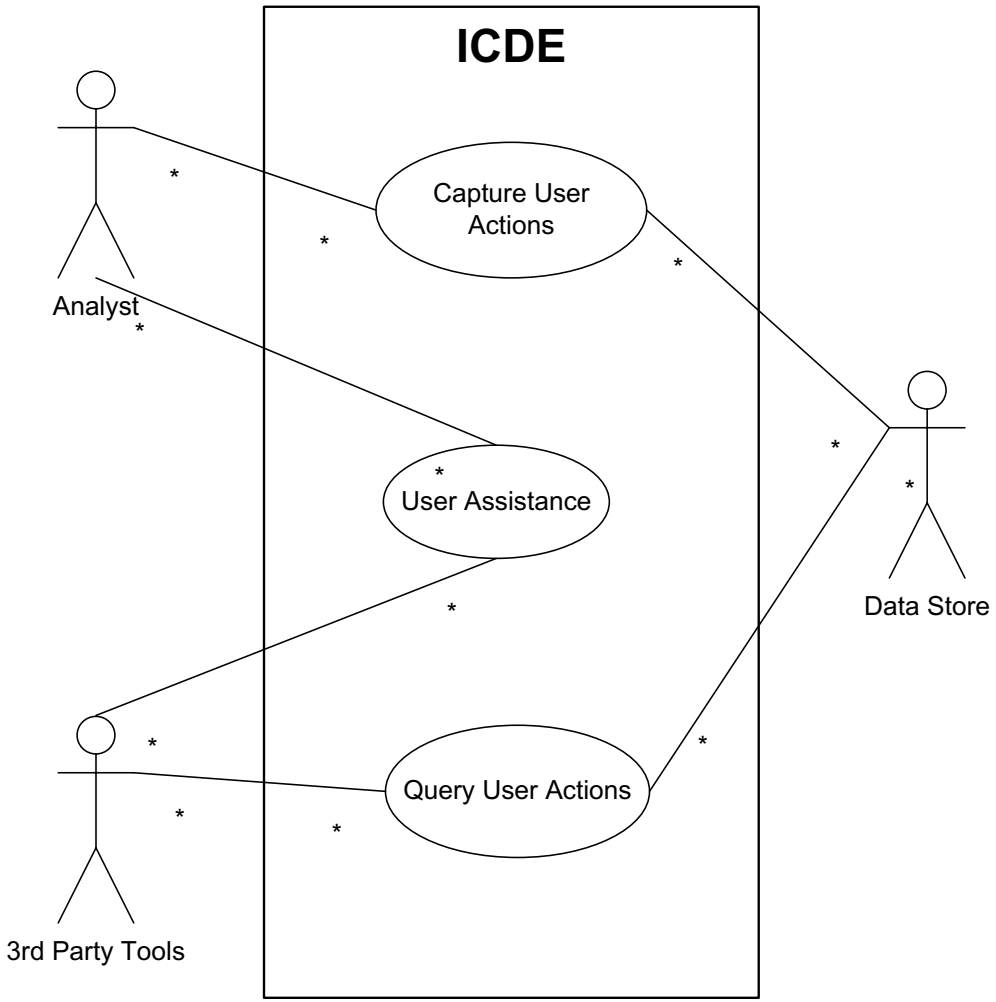
ICDE System

- Information Capture and Dissemination Environment (ICDE) is a software system for providing intelligent assistance to
  - financial analysts
  - scientific researchers
  - intelligence analysts
  - analysts in other domains

ICDE Schematic

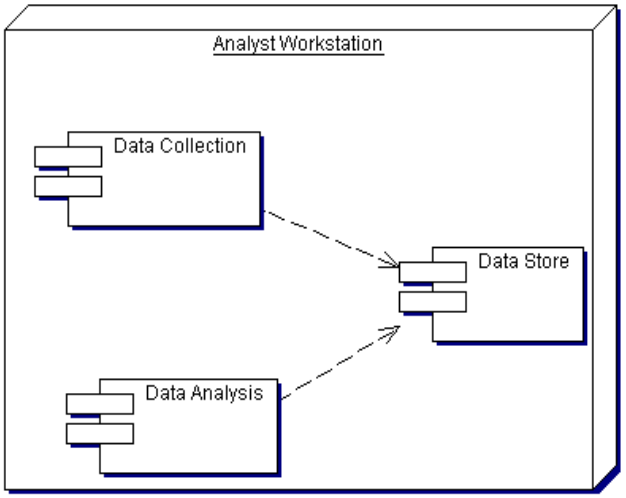


ICDE Use Cases



Case Study Context

- ICDE version 1.0 in production
- Basically a complex, *raw* information capture tool, GUI for looking at captured data
- 2 tier client-server, single machine deployment
  - Java, Perl, SQL,
  - Programmatic access to data through very complex SQL (38 tables, 46 views)



ICDE version 2.0

- ICDE v2.0 scheduled for development in 12 month timeframe
  - Fixed schedule, budget
- Major changes to:
  - Enhance data capture tools (GUI)
  - Support 3rd party tool integration, testing, data access and large production scale deployments (100's of users)
- Very few concrete requirements for the 3rd party tool support or release to full production environment

ICDE v2.0 Business Goals

Business Goal	Supporting Technical Objective
Encourage third party tool developers	Simple and reliable programmatic access to data store for third party tools
	Heterogeneous (i.e. non-Windows) platform support for running third party tools
	Allow third party tools to communicate with ICDE users from a remote machine
Promote the ICDE concept to users	Scale the data collection and data store components to support up to 150 users at a single site
	Low-cost deployment for each ICDE user workstation

Architecturally Significant Requirements for ICDE v2.0

- ICDE project requirements:
  - Heterogeneous platform support for access to ICDE data
  - Instantaneous event notification (local/distributed)
  - Over the Internet, secure ICDE data access
  - Ease of programmatic data access
- ICDE Project team requirements:
  - Insulate 3rd party projects and ICDE tools from database evolution
  - Reliability for multi-tool ICDE deployments
  - Scalable infrastructure to support large, shared deployments
  - Minimize license costs for a deployment
- Unknowns
  - Minimize dependencies, making unanticipated changes potentially easier

Summary

- ICDE is a reasonably complex system
- Will be used to illustrate concepts during the remainder of this course

Essential Software Architecture

Session 3:

Quality Attributes

What are Quality Attributes

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

Quality Attribute Specification

- 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"

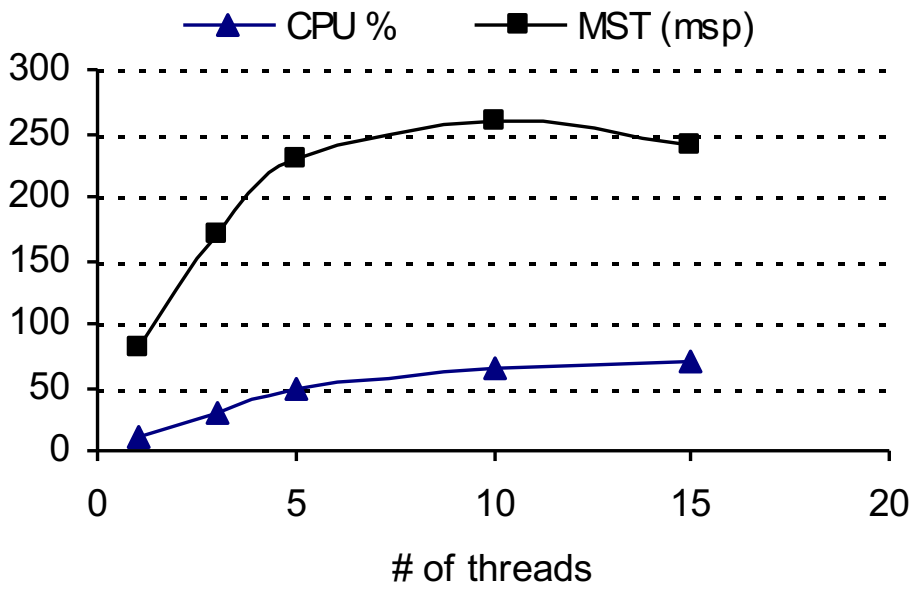
Performance

- Many examples of poor performance in enterprise applications
- Performance requires a:
  - Metric of amount of work performed in unit time
  - Deadline that must be met
- Enterprise applications often have strict performance requirements, e.g.
  - 1000 transactions per second
  - 3 second average latency for a request

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



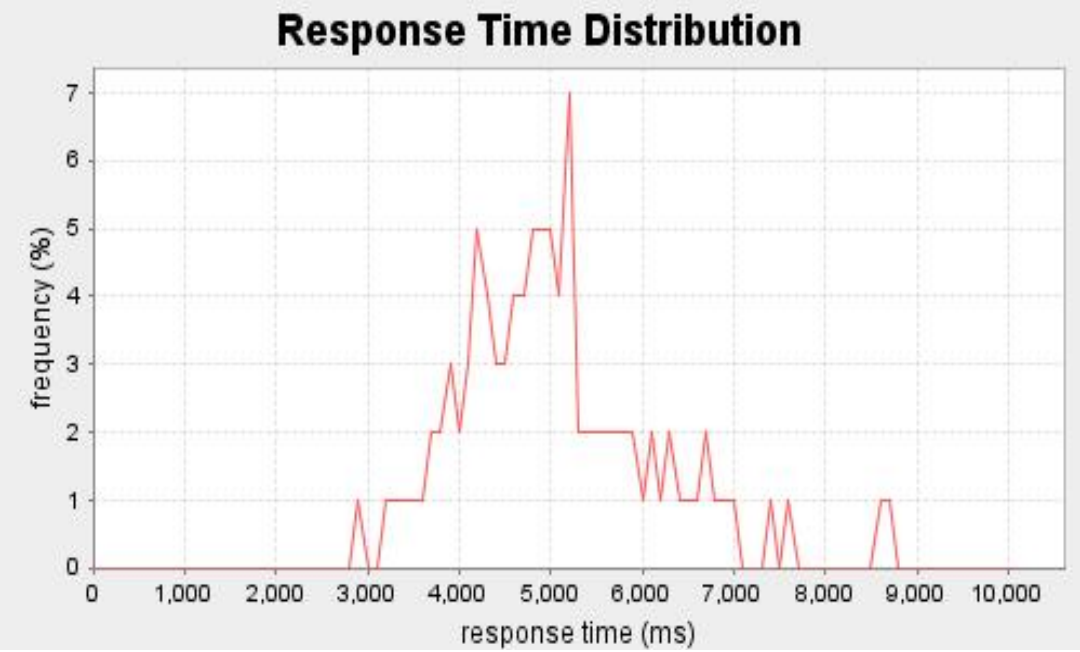
- Throughput of a message queuing system
  - Messages per second (msp)
  - Maximum sustainable throughput (MST)
- Note throughput changes as number of receiving threads increases

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

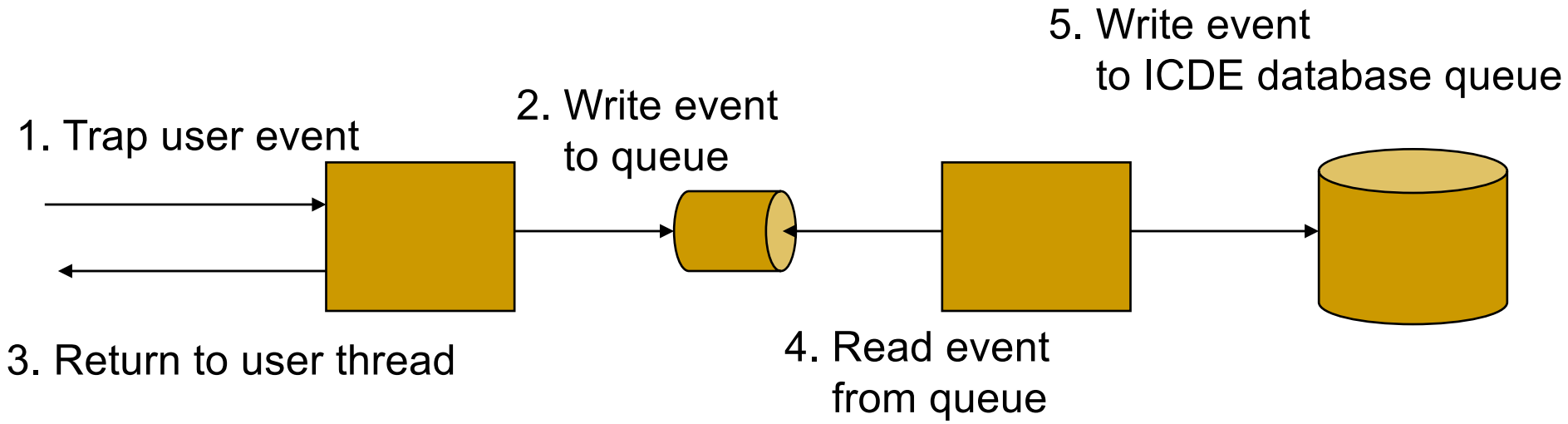
- ‘something must be completed before some specified time’
  - Payroll system must complete by 2am so that electronic transfers can be sent to bank
  - Weekly accounting run must complete by 6am Monday so that figures are available to management
- Deadlines often associated with batch jobs in IT systems.

Something to watch for …

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

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



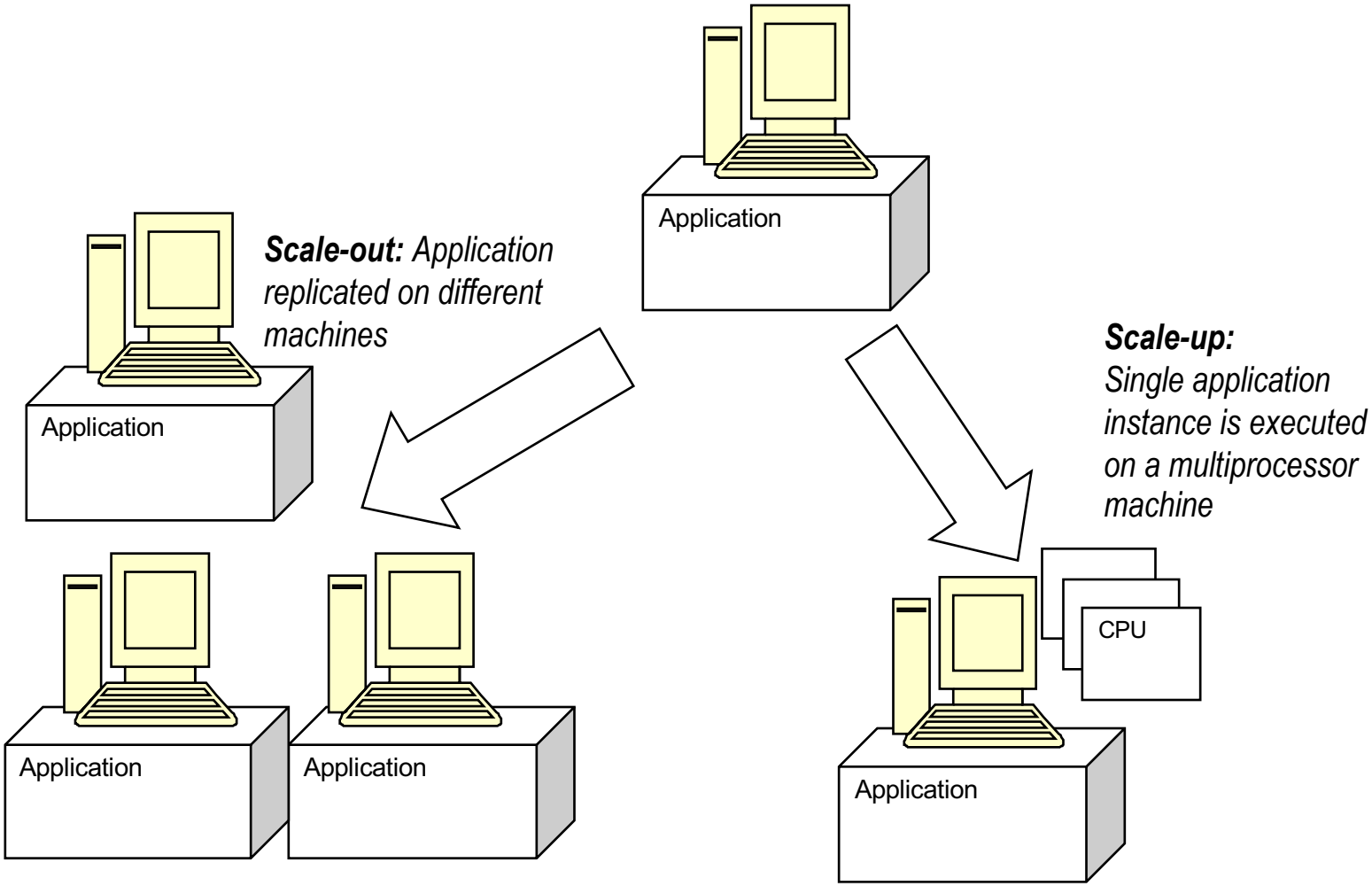
Scalability

- *"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).

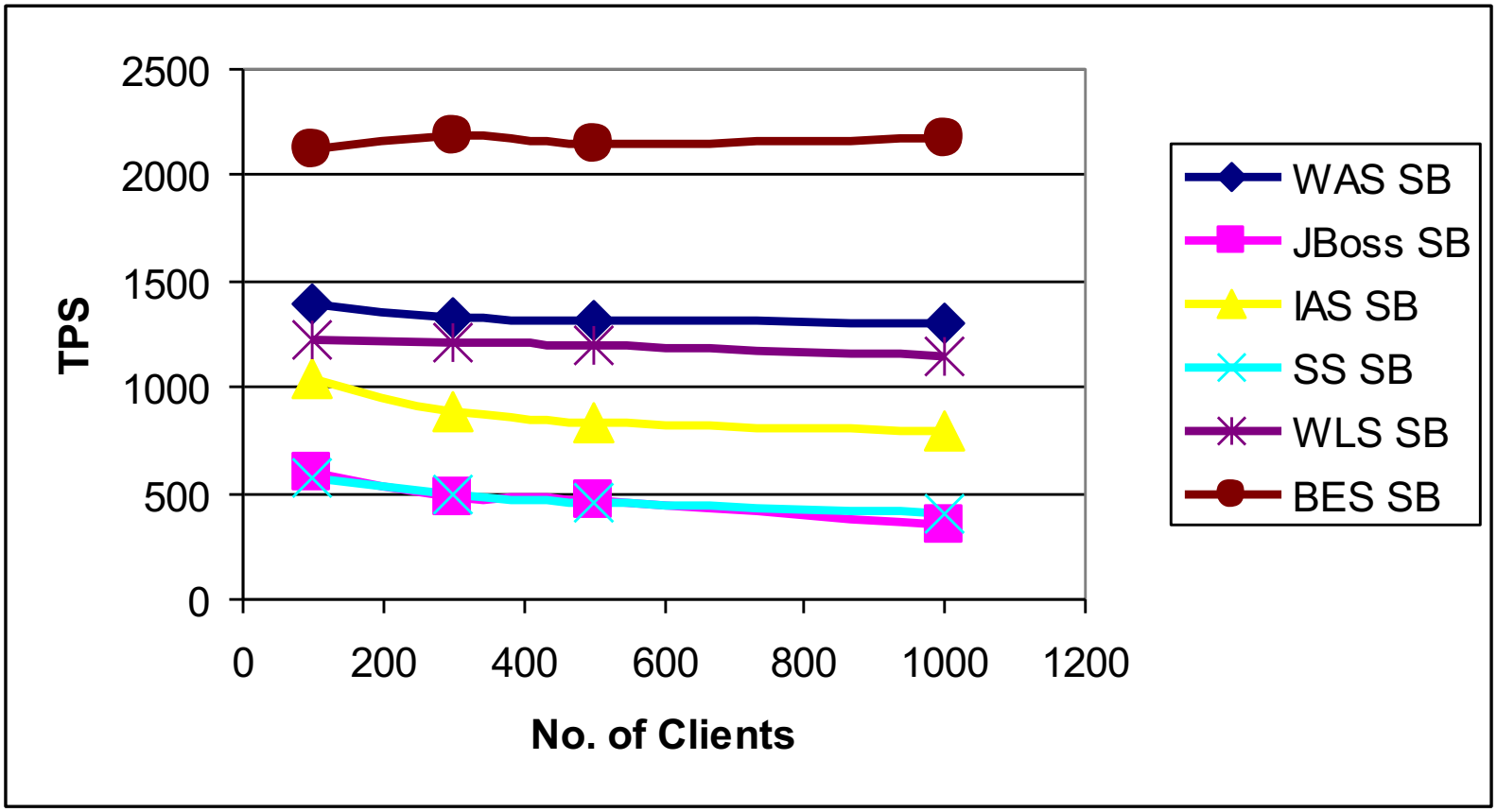
Scalability – Add more hardware …



Scalability - 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



I.Gorton, A Liu, *Performance Evaluation of Alternative Component Architectures for Enterprise JavaBean Applications, in IEEE Internet Computing, vol.7, no. 3, pages 18-23, 2003.*

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

- Modifications to a software system during its lifetime are a fact of life.
- Modifiable systems 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!

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 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 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
- PKI
- Web Services security
- JAAS
- Operating system security
- Database security
- Etc 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.
  - 100% available during business hours
  - No more than 2 hours scheduled downtime per week
  - 24x7x52 (100% availability)
- 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
- Recoverability (e.g. a database)
  - the capability to reestablish performance levels and recover affected data after an application or system failure

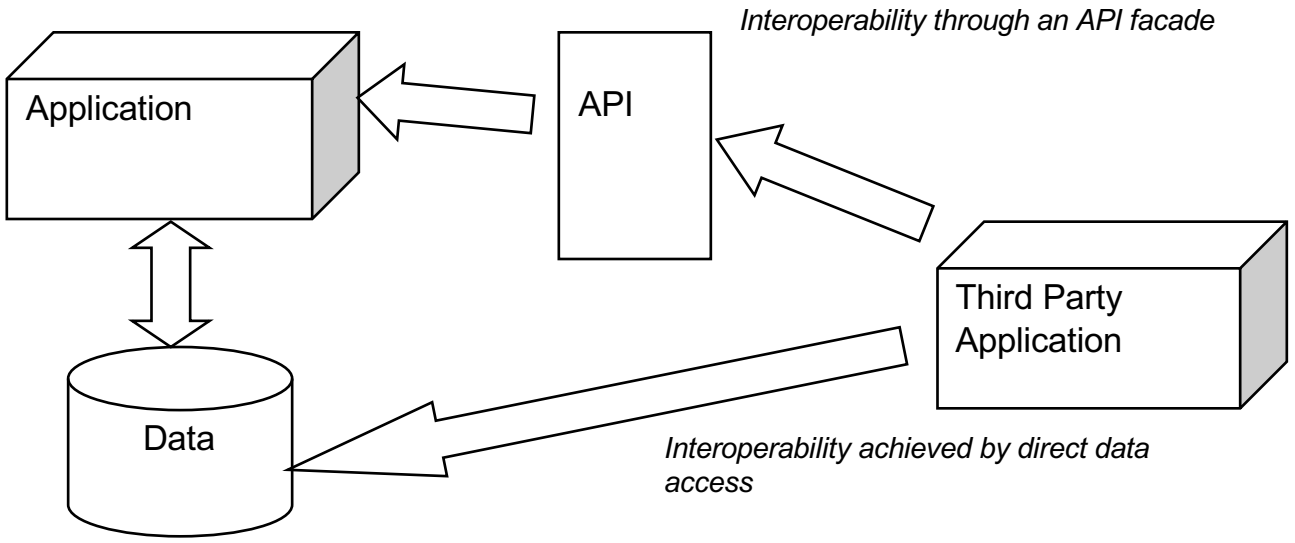
Availability for ICDE

- 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.

Misc. 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?

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

- 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