Software Architecture

Quality Attributes

Alar Raabe

Content

architecture is the primary carrier of quality attributes

- Summary from Previous Seminar
- Software Quality Attributes
  - Quality Attributes Addressed by Software Architecture
  - Categories of Software Quality Attributes
- Quality Attribute Driven Design
  - Attribute Trade-Off Analysis Method
  - SOA Quality Attribute Scenarios
- Group Work
- Discussion
  - Our common language
  - How we (should) evaluate software architectural decisions

What is (Software) Architecture

elements, relationships and principles that correspond to system concerns

- (Software) Architecture is a
  - **fundamental conception** of a (software) system in its
    - **environment** embodied in its
    - **elements**,
    - their **relationships** to each other and to the environment, and
    - **principles** guiding (software) system design and evolution
- (Software) Architectural Style
  - Characterizes a **family/class of system architectures** that are related by shared structural and semantic properties
  - Defines
    - a vocabulary of design **elements** (components & connectors)
    - design **rules**, or **constraints** (incl. topology)
    - **semantic interpretation**
    - **analyses** that can be performed on systems built in that style

Group Work Results: CDI-Hub

- stakeholders
- concerns
- viewpoints
- relevant architectural styles
- *properties of interest*

Quality Attributes of Software

quality is fitness for use

J. M. Juran

- (Software) Quality
  - the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs [ISO/IEC 9126]
- (Software) Quality Attribute
  - characteristic of software that affects its quality
- Categorization of Software Quality Attributes
  - End User's View – Developer's View – Business's View
  - Runtime Qualities – Non-Runtime Qualities
  - Quality Attributes Specific to the Architecture

Quality Attributes of Software

CMU SEI

- End User's View
  - *Functionality*
  - *Interoperability*
  - *Security*
  - *Performance (Efficiency)*
  - *Resource Efficiency*
  - *Availability and Reliability*
  - *Recoverability*
  - *Usability*
- Developer's View
  - *Modifiability*
  - *Portability (Extensibility)*
  - Reusability
  - Integrability
  - Testability
- Business's View
  - Time To Market
  - Cost vs. Benefits
  - Projected Life-time
  - Targeted Market
  - Integration with Legacy
  - Roll-out (Roll-back) Schedule

Quality Attributes addressed by Architecture (1)

CMU SEI

- Functionality
  - ability of the system to satisfy the purpose for which it was designed
  - drives the initial decomposition of the system
- Interoperability
  - quality of a system that enables it to work with other systems (incl. systems not yet known)
- Security
  - ability to enforce authorization, authentication, and deliberate denial of service attacks
- Performance (efficiency)
  - represents the responsiveness of the system (e.g., the time required to respond to events or the number of events processed in some period of time)

Quality Attributes addressed by Architecture (2)

CMU SEI

- Resource Efficiency
  - efficient utilization of resources
- Modifiability
  - ability to grow an architecture over time
- Availability and Reliability
  - availability is an attribute that measures the proportion of time the system is up and running
  - reliability is an attribute that measures the system's ability to continue operating over time
- Recoverability
  - ability of a system to pick up where it left off after a shutdown or crash

Quality Attributes addressed by Architecture (3)

CMU SEI

- Usability
  - usability with respect to the end user, system maintainers, operators, etc. (measured using scenarios)
  - composed of
    - Learnability and Memorability
    - Efficiency
    - Error avoidance and Error handling
    - Satisfaction
- Portability (Extensibility)
  - ability to reuse a component in a different application or operating environment (a special kind of modifiability)
  - related to
    - Adaptability
    - Installability
    - Conformance
    - Replaceability

Software Quality Attributes (1)

ISO/IEC 9126:2001

- Functionality
  - A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.
    - Suitability, Accuracy, Interoperability, Compliance, Security
- Reliability
  - A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time.
    - Maturity, Recoverability, Fault Tolerance
- Usability
  - A set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users.
    - Learnability, Understandability, Operability

Software Quality Attributes (2)

ISO/IEC 9126:2001

- Efficiency
  - A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions.
    - Time Behaviour, Resource Behaviour
- Maintainability
  - A set of attributes that bear on the effort needed to make specified modifications.
    - Stability, Analyzability, Changeability, Testability
- Portability
  - A set of attributes that bear on the ability of software to be transferred from one environment to another.
    - Installability, Replaceability, Adaptability, Conformance (similar to compliance, above, but here related specifically to portability, e.g. conformance to a particular database standard)

Comparing CMU SEI and ISO/IEC Software Quality Attributes

• Functionality • Interoperability • Security	• Functionality – Suitability, Accuracy, Interoperability, Compliance, Security
• Availability and Reliability • Recoverability	• Reliability – Maturity, Recoverability, Fault Tolerance
• Usability – Learnability, Memorability, Error avoidance & handling, Satisfaction	• Usability – Learnability, Understandability, Operability
• Performance (Efficiency) • Resource Efficiency	• Efficiency – Time Behaviour, Resource Behaviour
• Modifiability	• Maintainability – Stability, Analyzability, Changeability, Testability
• Portability (Extensibility) – Installability, Replaceability, Adaptability, Conformance	• Portability – Installability, Replaceability, Adaptability, Conformance

Quality Attribute Driven Design

- Example: Deriving REST (Fielding)
- CMU SEI Architecture-Centric Methods
- CMU SEI Architecture Tradeoff Analysis Method
  - Quality Attribute Scenarios
  - Quality Attribute Models
  - Risk Themes
- Example: SOA Quality Attribute Scenarios

Example: Deriving REST

Fielding

- Properties of Interest
  - Performance (network and user-perceived)
  - Scalability
  - Simplicity
  - Modifiability (incl. Extensibility and Reusability)
  - Visibility
  - Portability
- Constituents
  - Client Server Style → modifiability
  - Stateless Communication → visibility, reliability, scalability
  - Cache → network efficiency
  - Uniform Interface → simplicity, portability
  - Layered System Style → simplicity, scalability
  - Code-on-demand → modifiability (extensibility), simplicity

Architecture-Centric Methods

CMU SEI

- A Famility of Scenrio-Driven and Quality Attribute Driven Development Methods
  - Software Architecture Analysis Method (SAAM)
  - Architecture Tradeoff Analysis Method (ATAM)
    - to assess the consequences of architectural decision alternatives in light of quality attribute requirements
  - Quality Attribute Workshop (QAW)
  - Cost-Benefit Analysis Method (CBAM)
  - Active Reviews for Intermediate Designs (ARID)
  - Attribute-Driven Design (ADD)

ATAM Steps – Architect-Centric Phase

CMU SEI

1. Present the ATAM – quick overview of steps, techniques, outputs
2. Present the business drivers and context for architecture
3. Present the architecture
   - the architect presents an overview of the architecture
4. Identify architectural approaches
   - the evaluation team and the architect itemize the architectural approaches discovered in the previous step
5. Generate the quality attribute utility tree
   - a small group of technically oriented stakeholders identifies, prioritizes, and refines the most important quality attribute goals in a utility tree format
6. Analyze the architectural approaches
   - the evaluation team probes the architectural approaches in light of the quality attributes to identify risks, non-risks, and tradeoffs (to probe the architecture, they use quality attribute questions)

ATAM Steps – Stakeholder-Centric Phase

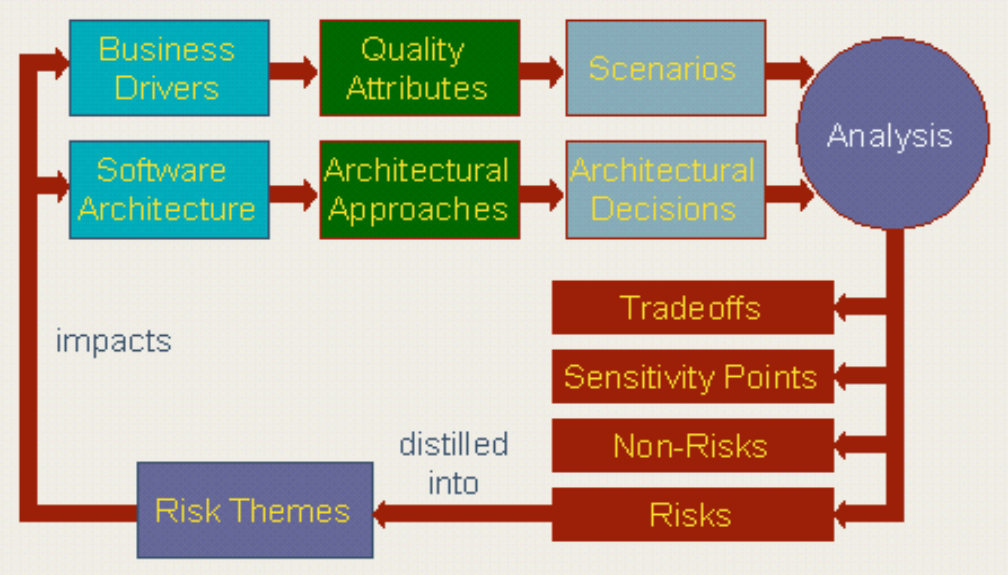
CMU SEI

7. Brainstorm and prioritize scenarios
   - a larger and more diverse group of stakeholders creates scenarios that represent their various interests; then the group votes to prioritize the scenarios based on their relative importance
8. Analyze architectural approaches
   - the evaluation team continues to identify risks, nonrisks, and tradeoffs while noting the impact of each scenario on the architectural approaches
9. Present results
   - the evaluation team recapitulates the ATAM steps, outputs, and recommendations

Architecture Tradeoff Analysis Method

CMU SEI

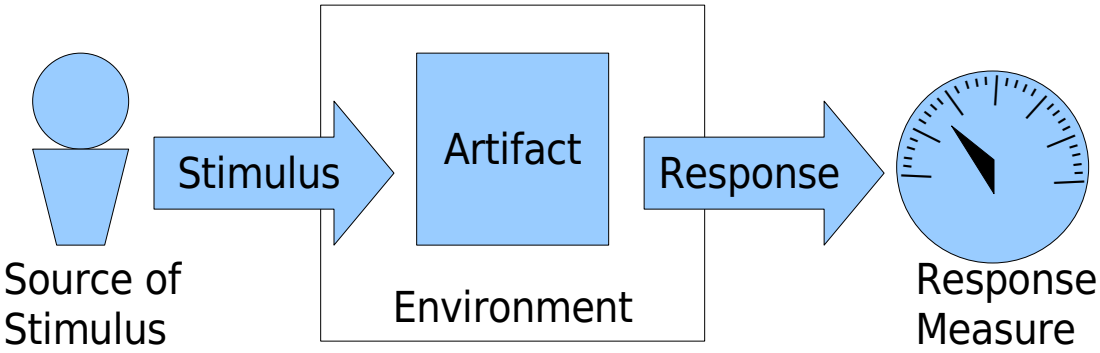
- Output
  - a set of identified architectural approaches
  - "utility tree" – driving architectural requirements
  - the set of scenarios mapped onto the architecture
  - a set of quality-attribute specific questions and responses
  - a set of identified risks
  - a set of identified non-risks
  - a set of risk themes that threaten to undermine the business goals for the system



Quality Attribute Scenarios

to represent stakeholders' concerns

- Source of stimulus
  - User or other System
- Stimulus
  -
- Environment
  - Conditions
- Artifact
  - System or some part
- Response
  -
- Response measure
  -

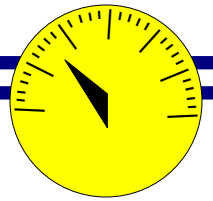


Quality Attribute Tradeoff Points

you can't eat your cake and have it too!

	Availability	Efficiency	Flexibility	Integrity	Interoperability	Maintainability	Portability	Reliability	Reusability	Robustness	Testability	Usability
Availability								+		+		
Efficiency			-		-	-	-	-		-	-	-
Flexibility		-		-		+	+	+		+		
Integrity		-			-				-		-	-
Interoperability		-	+	-			+					
Maintainability	+	-	+					+			+	
Portability		-	+		+	-			+		+	-
Reliability	+	-	+			+				+	+	+
Reusability		-	+	-				-			+	
Robustness	+	-						+				+
Testability	+	-	+			+		+				+
Usability		-								+	-	

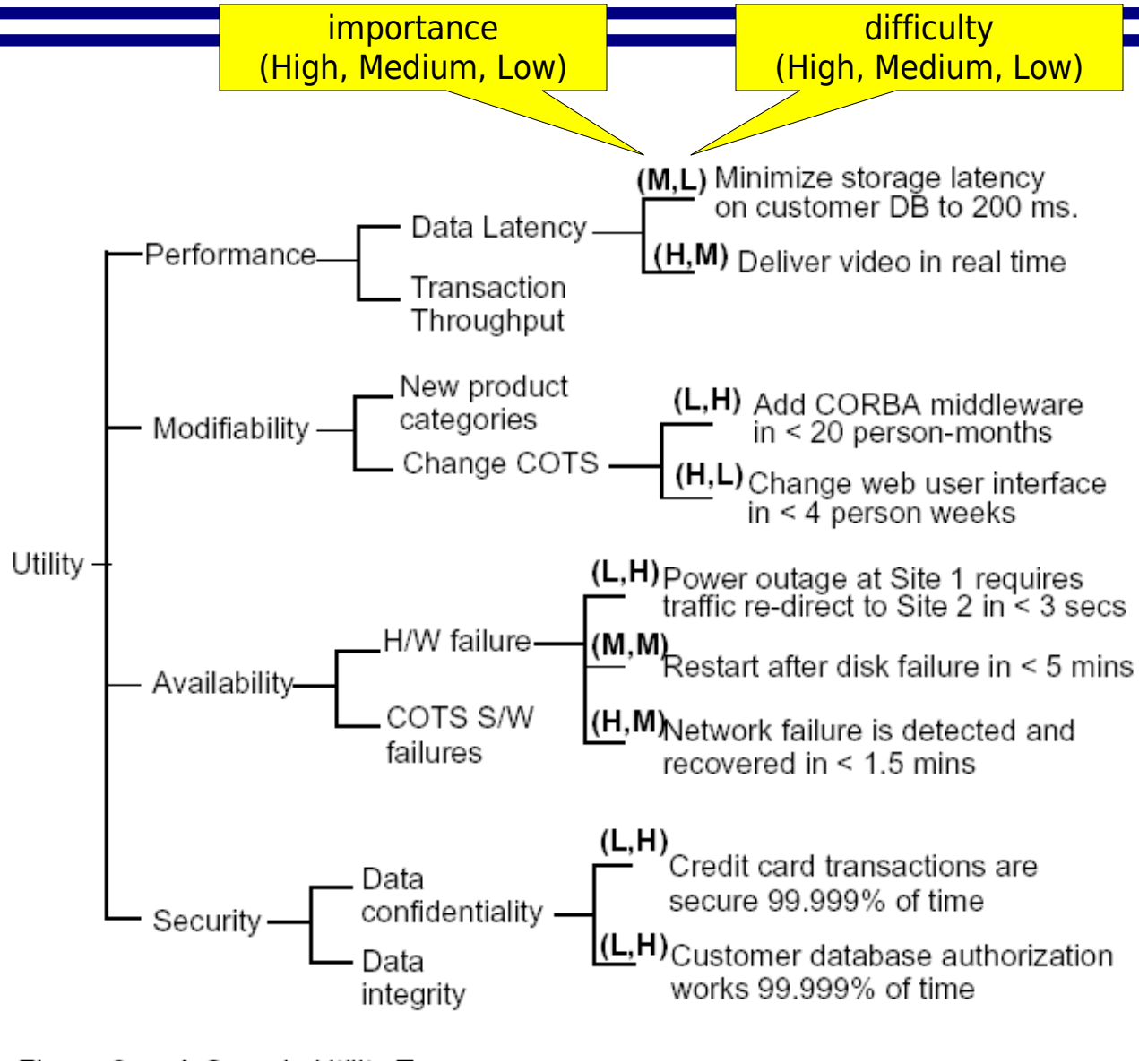
Metrics for Software Quality Attributes



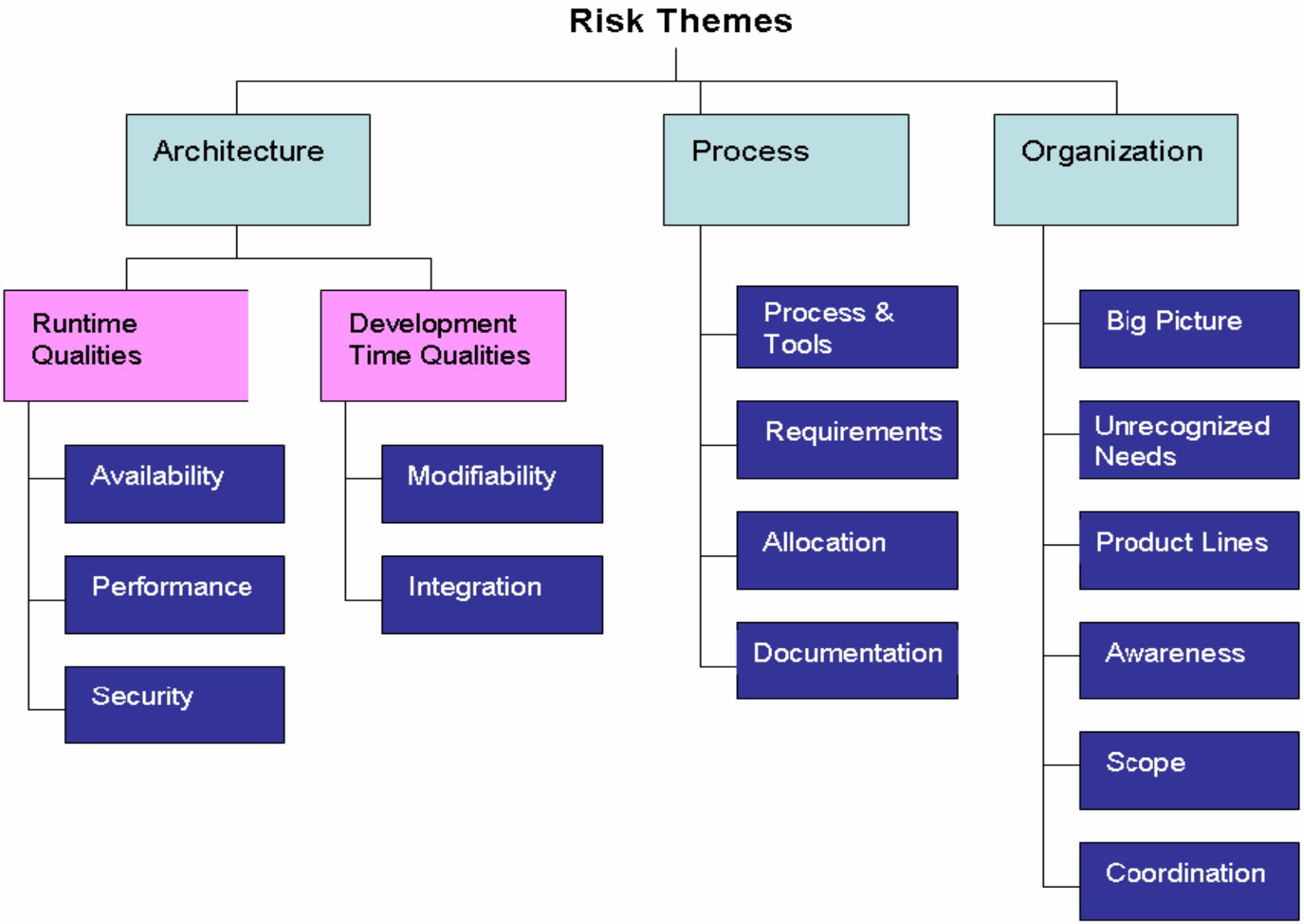
- Maintainability
  - Analyzability
    - cyclomatic number
    - number of statements
    - comments rate
    - calling proof
  - Changeability
    - number of jump
    - number of nested levels
    - average size of statement
    - number of variables

- Maintainability (cont.)
  - Stability
    - number of parameters referenced
    - number of global variables
    - number of parameters changed
    - number of called relationships
  - Testability
    - number of non-cyclic path
    - number of nested levels
    - cyclomatic number
    - number of call-paths

Utility Tree – for Identifying & Prioritizing



Risk Themes


Risk Themes
Architecture
Process
Organization
Runtime Qualities
Development Time Qualities
Availability
Performance
Security
Modifiability
Integration
Process & Tools
Requirements
Allocation
Documentation
Big Picture
Unrecognized Needs
Product Lines
Awareness
Scope
Coordination

SOA Quality Attribute Scenarios (1)

Performance

- P1
  - A sporadic request for service ‘X’ is received by the server during normal operation
  - The system processes the request in less than ‘Y’ seconds
- P2
  - The service provider can process up to ‘X’ simultaneous requests during normal operation, keeping the response time on the server less than ‘Y’ seconds
- P3
  - The roundtrip time for a request from a service user in the local network to service ‘X’ during normal operation is less than ‘Y’ seconds

SOA Quality Attribute Scenarios (2)

Availability

- A1
  - An improperly formatted message is received by a system during normal operation
  - The system records the message and continues to operate normally without any downtime
- A2
  - An unusually high number of suspect service requests are detected (denial-of-service attack), and the system is overloaded
  - The system logs the suspect requests, notifies the system administrators, and continues to operate normally
- A3
  - Unscheduled server maintenance is required on server ‘X’
  - The system remains operational in degraded mode for the duration of the maintenance

SOA Quality Attribute Scenarios (3)

Availability

- A4
  - A service request is processed according to its specification for at least 99.99% of all requests
- A5
  - A new service is deployed without impacting the operations of the system
- A6
  - A third-party service provider is unavailable
  - Modules that use that service respond appropriately regarding the unavailability of the external service and the system continues to operate without failures

SOA Quality Attribute Scenarios (4)

Security

- S1
  - A third-party service with malicious code is used by the system
  - The third-party service is unable to access data or interfere with the operation of the system
  - The system notifies the system administrators
- S2
  - An attack is launched attempting to access confidential customer data
  - The attacker is not able to break the encryption used in all the hops of the communication and where the data is persisted
  - The system logs the event and notifies the system administrators

SOA Quality Attribute Scenarios (5)

Security

- S3
  - A request needs to be sent to a third-party service provider, but the provider's identity can not be validated
  - The system does not make the service request and logs all relevant information
  - The third party is notified along with the system administrator
- S4
  - An unauthorized service user attempts to invoke a protected service provided by the system
  - The system rejects the attempt and notifies the system administrator

SOA Quality Attribute Scenarios (6)

Security

- S5
  - An attacker is modifying incoming service requests in order to launch an attack on the system infrastructure
  - The system identifies and discards all tampered messages, logs the event, and notifies the system administrators
- S6
  - An attacker attempts to exploit the service registry in order to redirect service requests
  - The service registry denies access to information in the registry, logs the event, and notifies the system administrators

SOA Quality Attribute Scenarios (7)

Testability

- T1
  - An integration tester performs integration tests on a new version of a service that provides an interface for observing output
  - 90% path coverage is achieved within one personweek

SOA Quality Attribute Scenarios (8)

Interoperability

- I1
  - A new business partner that uses platform ‘X’ is able to implement a service user module that works with our available services in platform ‘Y’ in two person-days
- I2
  - A transaction of a legacy system running on platform ‘X’ is made available as a Web service to an enterprise application that is being developed for platform ‘Y’ using the Web services technology
  - The wrapping of the legacy operation as a service with proper security verification, transaction management, and exception handling is done in 10 person-days

SOA Quality Attribute Scenarios (9)

Modifiability

- M1
  - A service provider changes the service implementation, but the syntax and the semantics of the interface do not change
  - This change does not affect the service users
- M2
  - A service provider changes the interface syntax of a service that is publicly available
  - The old version of the service is maintained for 12 months, and existing service users are not affected within that period
- M3
  - A service user is looking for a service. A suitable service is found that contains no more than ‘X’ percentage of unneeded operations, so the probability of the service provider changing is reduced

SOA Quality Attribute Scenarios (10)

Reliability

- R1
  - A sudden failure occurs in the runtime environment of a service provider
  - After recovery, all transactions are completed or rolled back as appropriate, so the system maintains uncorrupted, persistent data
- R2
  - A service becomes unavailable during normal operation. The system detects and restores the service within two minutes

Group Work: CDI-Hub

- quality attributes
- quality attribute scenarios
- architectural styles

Value of Software Architecture

not documenting, but understanding!

- Value of Software Architecture to Stakeholders
- Measuring Value of Software Architecture
- Option Value of Software Architecture

Value of Software Architecture to Stakeholders

80% of time during maintenance is spent in design-rediscovery

Davidson, 2002

- Users and operators of the system
  - understand the external system behavior
  - understand how to operate system
- acquirers and owners of the system
  - understand economical issues connected to the system
- suppliers and developers of the system
  - plan development and construction
  - estimate system properties
- builders and maintainers of the system
  - understand the system internals

Value of Good Architecture (1)

- Developers & Suppliers
  - Speed & Freedom
    - Build and maintain loosely coupled components more independently, and replace old components with new technologies without breaking clients that depend upon them for services
  - Guidance
    - Clear definitions of responsibilities, answer "where should I put this functionality" before even asked, simplify building of components and layter understanding of these
  - Reuse of Effort, Skills & Know-How
    - Common design patterns, tools, hardware and software platforms allow to move from one system to another, and apply their current skills effectively
    - The goal of reusable components is achievable
  - Easy integration
    - Applications built to a consistent architecture are more easily integrated with each other

Value of Good Architecture (2)

- Owners & Maintainers
  - Standards-based systems are more easily integrated with external business partners and commercial "off the shelf" products
  - Careful design of the application and the infrastructure yield high availability and performance
  - Robust systems that can survive partial failure
  - Robust designs that can survive extension, adaptation, requirements changes, platform changes, etc.

Value of Modularity – Soft Value

- Maintainability
  - the ease with which a software system or component can be modified to change (flexibility) or add (extendability) capabilities, correct faults or defects, improve performance or other attributes, or adapt to a changed environment
- Portability
  - the ease with which a system or component can be transferred from one hardware or software environment to another
- Reliability
  - the ability of a system or component to perform its required functions under stated conditions for a specified period of time
- Testability

Measuring Value of Software Architecture

- Value of Software Architecture
  - cost of realization of risks comnpared to cost of architecture

$$value_{arch} = \sum_{i=1}^{n} (cost_{risk}(concern_i)) - cost_{arch}$$

- Value of Software Architecture Description
  - cost of perfroming activities without architecture description compared to cost of documenting architecture

$$value_{arch\,desc} = \sum_{i=1}^{n} (cost_{performing}(activity_i)) - cost_{arch\,desc}$$

Value of Modularity – Economic Value 1

- Real Option Theory [Baldwin & Clark]
  - Modularity
    - is a financial force
    - accommodates future uncertainty
    - creates choices that can be exercised in future
  - Design Structure Matrix – dependency matrix of design parameters
  - Modular Operators – correspond to options
    - split, substitute, exclude, augment (add), inversion, porting
  - Valuation of modularity as Real Options (American call)
    - Decision trees with probabilities (Markov Processes)
    - Dynamic programming algorithms
    - Monte Carlo simulations
  - Value

$$\mathrm{NPV}_{\mathrm{strategic}} = \mathrm{NPV}_{\mathrm{traditional}} + \mathrm{Value}_{\mathrm{real\ options}}$$

Value of Modularity – Economic Value 2

- Real Option Theory
  - Qualitative Design Principles [Sullivan]
    - If at any time, the expected value of future profits discounted to given time is at least by value of investment opportunity more than the direct costs, then commit to the design decision, otherwise do not
    - If the expected present value of the future profits that would flow from choice exceeds the direct cost of implementing it, then go ahead and implement the choice, otherwise implement other choice
    - If the expected present value of future profits that would flow from restructuring exceeds the direct cost of restructuring, then go ahead and restructure, otherwise do not
    - If the cost to effect a software decision is sufficiently low, then the benefit of investing to effect it immediately outweighs the benefit of waiting, so the decision should be made immediately
    - With other factors, including the static NPV, remaining the same, the incentive to wait for better information before effecting a design decision increases with risk (ie, with the spread, in possible benefits)
    - The incentive to wait before investing increases with the likelihood of unfavourable future events occurring
    - All else being equal, the value of the option to delay increases with variance in future costs

Discussion

What is/are for us ...

- Concepts of
  - (Software) System
  - (Software) Architecture
  - (Software) Architecture Description
- Value of
  - (Software) Architecture
  - (Software) Architecture Description
- Most Relevant (Software) Architecture Styles
- Main Stakeholders
- Main System Concerns
- (Software) Architecture Framework
- (Software) Quality Attributes

Conclusion

- Value of (Software) Architecture
  - as fundamental conception of (software) system, architecture allows us to reason (answer questions) about the (software) system
  - as specific architectural styles address certain concerns (cause certain properties/qualities) of (software) systems, architecture allows us to address concerns (achieve required properties or qualities) of (sofware) systems
- Value of Architecture Description
  - as document, it provides guidance for constructing and evolving the (software) system, and allows us to record and communicate our knowledge and decisions about the (software) system architecture
  - as model, it allows us to reason (answer questions) about the (software) system architecture

Leftovers

- Conway's law (1968)
  - organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations

Thank You!

Terms (Glossary)

ISO/IEC 42010:2007

architecture	fundamental conception of a system in its environment embodied in elements, their relationships to each other and to the environment, and principles guiding system design and evolution
architecture decision	choice made from among possible options that addresses one or more architecture-related concerns
architecture description	collection of work products used to describe an architecture
architecture model	work product from which architecture views are composed
architecture rationale	explanation or justification for an architecture decision
architecture view	work product representing a system from the perspective of architecture-related concerns
architecture viewpoint	work product establishing the conventions for the construction, interpretation and use of architecture views
environment	context determining the setting and circumstances of developmental, technological, business, operational, organizational, political, regulatory, social and any other influences upon a system
model correspondence	relation on two or more architecture models
stakeholder	individual, team, organization, or class thereof, having concerns with respect to a system
purpose	*{one of system concerns}*
system	*{a conceptual entity defined by its boundaries}*
system concern	area of interest in a system pertaining to developmental, technological, business, operational, organizational, political, regulatory, social, or other influences important to one or moreof its stakeholders

Definitions 1

- System
  - a collection of interacting components organized to accomplish a specific function or set of functions within a specific environment
- Interface (Connection)
  - a shared boundary between two functional units, defined by various characteristics of the functions
  - component that connects two or more other components for the purpose of passing information from one to the other
- Module (Component)
  - a logically separable part of a system
- Encapsulation
  - isolating some parts of the system from the rest of the system
  - a module has an outside that is distinct from its inside (an external interface and an internal implementation)

Definitions 2

- Modularity
  - the degree to which a system is composed of discrete components such that a change to one component has minimal impact on other components
  - the extent to which a module is like a black box
- Coupling
  - the manner and degree of interdependence between modules
  - the strength of the relationships between modules
  - a measure of how closely connected two modules are
- Cohesion
  - the manner and degree to which the tasks performed by a single module are related to one another
  - a measure of the strength of association of the elements within a module

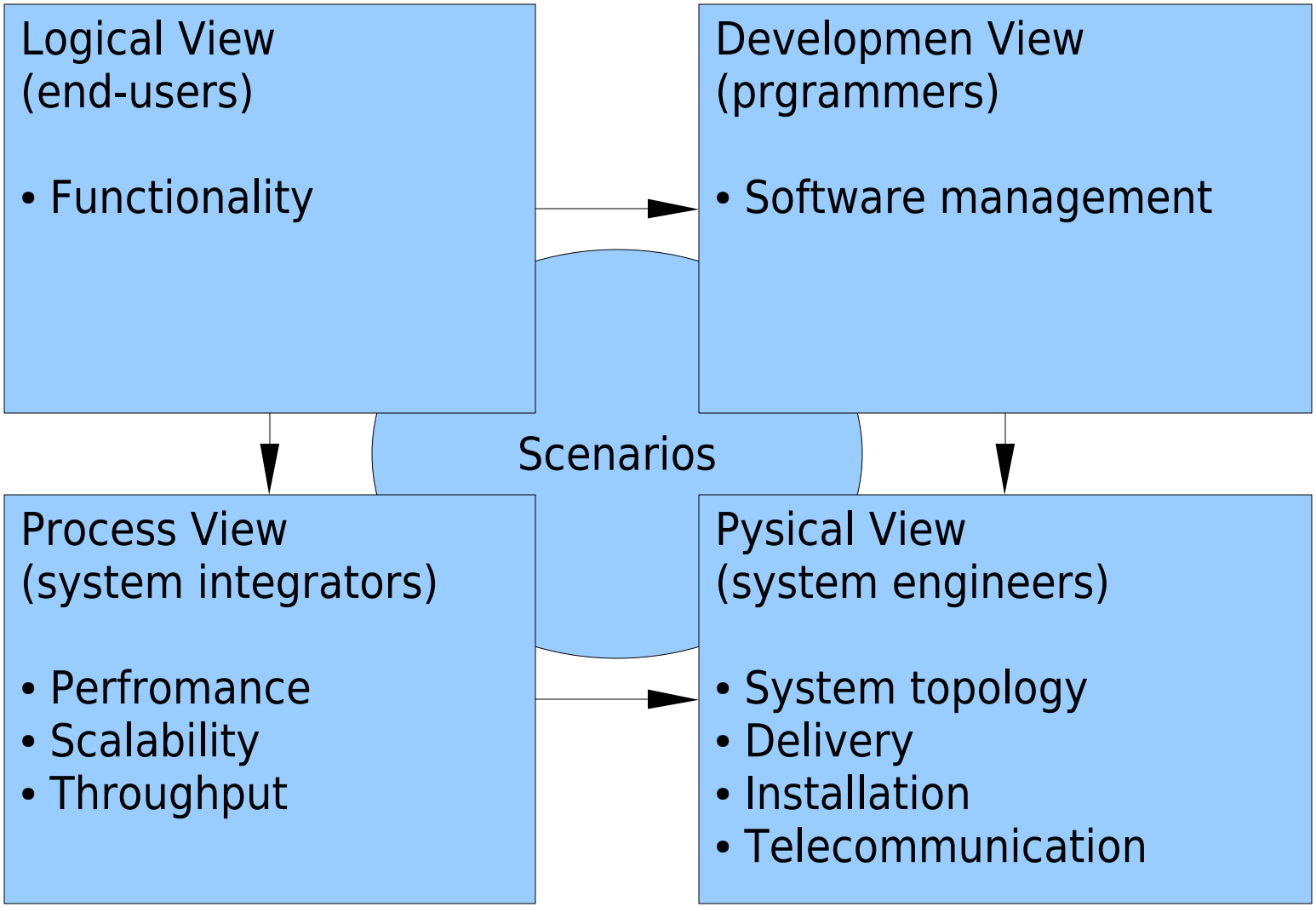
Definitions 3

- Model
  - an interpretation of a theory for which all the axioms of the theory are true
  - a semantically closed abstraction of a system or a complete description of a system from a particular perspective
  - anything that can be used to answer questions about system
    - to an observer B, an object M_A is a model of an object A to the extent that B can use M_A to answer questions that interest him about A

      Marvin Minsky
    - M is a model of A with respect to question set Q if and only if M may be used to answer questions about A in Q within tolerance T

      Doug Ross

4+1 Views (Kruchten)



What is a (Software) Architectural Style

- Characterizes a family of systems that are related by shared structural and semantic properties
- Defines
  - a vocabulary of design elements
  - design rules, or constraints (incl. topology)
  - semantic interpretation
  - analyses that can be performed on systems built in that style
- Classification of Architectural Styles
  - Constituent Parts
  - Control and Data Flows
  - Interaction of Control and Data Flows
  - Type of Reasoning

Architectural Design Decisions (1)

Kruchten

- Kinds of Architectural Design Decisions
  - Existence Decisions (*ontocrises*)
    - structural decisions
    - behavioral decisions
    - ban or non-existence decsisions (*anticrises*)
  - Property Decisions (*diacrises*)
    - constraints
    - design rules
    - guidelines
  - Executive Decisions (*pericrises*)
    - organizational decisions
    - process decisions
    - technology decisions
    - tool decisions

Architectural Design Decisions (2)

Kruchten

- Attributes of Architectural Design Decision
  - Epitome (the Decision itself)
  - Rationale (“why”)
  - Scope
  - State
    - 0: idea / obsolesced
    - 1: rejected
    - 2: tentative / challenged
    - 3: decided
    - 4: approved
  - Author, Time-Stamp, History
  - Categories (usability, security, ...)
  - Cost
  - Risk

Architectural Design Decisions (3)

Kruchten

- Relationship between Architectural Design Decisions
  - Constrains
  - Forbids (Excludes)
  - Enables
  - Subsumes
  - Conflicts with (mutually excluding)
  - Overrides
  - Comprises (is made of, decomposes into)
  - Is Bound to (strong)
  - Is an Alternative to
  - Is Related to (weak)
  - Dependencies
- Relationship with External Artifacts
  - Traces to
  - Does not Comply with

Questions

- How & who is using design artifacts?
- How to measure the cost and value of design knowledge?
- Who wants to pay for documents?
- Who wants to pay for exploring of various design alternatives?
- How tests are debugged?
- How to select architectural style?
- How to recover concepts?
- How to measure cost of having (or not having) architecture?
- How to evaluate the goodness of a method?