Software (Systems) Architecture Foundations

Lecture #4

Evaluating Architecture

Alar Raabe

Recap of Last Lecture

> Designing an architecture without documenting it, is like winking at a girl in the dark – you know what you´re doing, but nobody else does
>
> E. Woods

- Creating an architecture is not enough – it **has to be communicated** properly to let others use it properly to do their jobs
- Architecture documentation is for
  - **communication** – primary communication vehicle between stakeholders
  - **education** – introducing new people to the system
  - **designing** – provides structure for design decisions
  - **analyzing** – provides information to analyze the system properties (quality attributes)
  - **constructing** – tells what to implement (must contain models to support automated construction)
- Write documentation
  - from the reader’s point of view, for clear purpose and record rationale
  - avoiding unnecessary repetition and ambiguity
  - using a standard organization

Recap of Last Lecture

Make your system capture its own current architecture automatically

- Document the **relevant views** (at least one per each major viewpoint), then add documentation that applies to more than one view (combine some views to reduce the number of views to create, keep consistent, and maintain)

- Choose the views depending on
  - who the **important stakeholders** are and what are their **concerns** towards the system
  - what **structures** are present in the architecture
  - budget, schedule and what skills are available

- Additionally to the views
  - document the **major design decisions** taken, how to use the architecture and the ways architecture is allowed to change
  - make a **single element catalog** for the whole architecture – because elements appear in more than one view
  - document a **mapping to requirements**, to show that no requirement was forgotten, nor contradicted
  - add a section to record open questions

Content

> Quality means doing it right when no one is looking
>
> Henry Ford

- Introduction
  - Architecture and Requirements
  - Software Quality Models
- Software Quality Attributes
  - Categories of Software Quality Attributes
  - Measuring Software Quality (CISQ)
- Evaluation of Software Architectures
  - Quality Attribute Scenarios
  - Architecture Trade-off Analysis Method (ATAM)
  - Software Metrics
- Cost and Value of Architecture
  - Value of Architecture
  - Valuation of Architecture Decisions (Option Value of Architecture Decisions)
- Conclusions

Which one is Better ?

Pictures © Wikipedia & Wikimedia Commons

Evaluation will be done … either before or after !

Pictures © Wikipedia & Wikimedia Commons

Stakeholders Concerns → Requirements

> You can have any combination of features the Air Ministry desires, so long as you do not also require that the resulting airplane fly
>
> W. Messerschmidt

- Concern – any interest in the system (purpose, functionality, structure, behavior, cost, supportability, safety, interoperability)

- Requirement is a statement that expresses a need and its associated constraints and conditions
  - a condition or capability that must be met or possessed by a system, system component, product, or service to satisfy an agreement, standard, specification, or other formally imposed documents

- Software requirements specification is the basis for an agreement between customers and developers (contractors or suppliers) – they express concerns of stakeholders

Architecture and Requirements

CMU SEI

- Functional requirements – state what the system must do, and how it must behave or react to run-time stimuli (describe the functions of the system)
  – satisfied by assigning an appropriate sequence of responsibilities throughout the design (a fundamental architectural design decision)

- Quality attribute requirements (a.k.a. Non-Functional requirements) – qualifications of the functional requirements or the overall system (e.g. how fast the function must be performed, how resilient it must be to erroneous input, the time to deploy the product or a limitation on operational costs)
  – satisfied by the various structures designed into the architecture, and the behaviors and interactions of the elements that populate those structures

- Constraints – a constraint is a design decision with zero degrees of freedom (i.e. a design decision that's already been made, an is no subject to negotiations and design trade-offs)
  – satisfied by accepting the design decision and reconciling it with other affected design decisions

Some Software Quality Models

Qualities related to:
1. Using
2. Operating
3. Building/Changing

- McCall's Quality Model a.k.a. General Electrics Model (J. A. McCall, 1977)
  - Product Revision – the ability of the product to undergo changes
  - Product Operations – the characteristics of the product operation
  - Product Transition – the adaptability of the product to new environments
- Boehm's Quality Model (B. W. Boehm, 1978)
  - As-is utility – how well, easily, reliably and efficiently can I use the software product as-is
  - Maintainability – how easy is it to understand, modify and retest the software product
  - Portability – how easy is to use the software product when the environment has been changed
- FURPS Quality Model (R. B. Grady 1992)
  - Functionality – feature sets, capabilities, and security
  - Usability – human factors, aesthetics, consistency of UI, help, user documentation and training materials
  - Reliability – frequency and severity of failure, recoverability, predictability, accuracy, and mean time between failures (MTBF)
  - Performance – speed, efficiency, availability, accuracy, throughput, response time, recovery time, and resource usage
  - Supportability – testability, extensibility, adaptability, maintainability, compatibility, configurability, serviceability, installability, and localizability

FURPS+ (Supplementary Requirements)

IBM RUP

- **Functionality** – what the customer wants (incl. security-related needs)?
- **Usability** – how effective is the product from the user's standpoint (aesthetics, documentation, etc.)?
- **Reliability** – what's the maximum acceptable system downtime, predictability, accuracy, …?
- **Performance** – how fast must it be, what's the response time, throughput, memory consumption?
- **Supportability** – is it testable, extensible, serviceable, installable, configurable, can it be monitored, ...?

- **Design** constraints – how the software must be built (e.g. computing platform,, technologies, ...)?
- **Implementation** requirements – need to adhere to standards (e.g. use of certain development methods, etc.)?
- **Interface** requirements – what other systems must this one interface with?
- **Physical** requirements – what hardware (or premises) must the system be deployable on?

Content

Quality means doing it right when no one is looking

Henry Ford

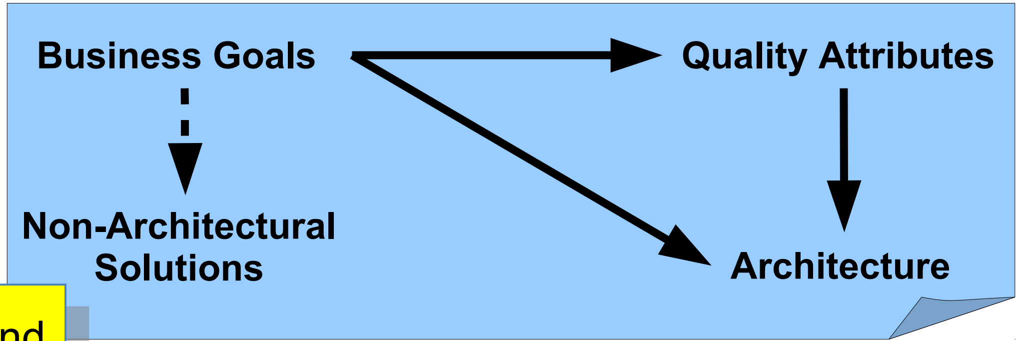
Software Quality and Quality Attributes

Quality is fitness for use

J. Juran

- (Software) Quality – degree to which the system satisfies the stated and implied needs of its various stakeholders, and thus provides value (ability of system to meet customer or user needs, expectations, or requirements)

- (Software) Quality Attribute – a measurable or testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders



As the systems in physical world become more and more software intensive the line between between Software and System Qualities is vanishing !

Quality Attributes addressed by Architecture

CMU SEI

- Main
  - Functionality
  - Availability (Reliability)
  - Interoperability
  - Modifiability
  - Performance (Efficiency)
  - Security
  - Testability
  - Usability
- Business
  - Time-to-Market
  - Cost vs. Benefits
  - Projected Life-Time
  - Targeted Market
  - Integration with Legacy
  - Roll-out (Roll-back) Schedule
- Other
  - Variability
  - Portability
  - Development Distributability
  - Scalability
  - Deployability
  - Mobility
  - Monitorability
  - Safety

  - Conceptual Integrity
  - Quality in Use
  - Marketability

Main Quality Attributes addressed by Architecture

CMU SEI

- Availability – a property of system that it is ready to carry out its task when needed (incl. reliability and recovery)
- Interoperability – quality of a system that enables it to work with other systems (incl. systems not yet known)
- Modifiability – ability of a system to grow and change over time (cost and risk of making changes)
- Performance (efficiency) – system's ability to meet timing requirements (the responsiveness of the system)
- Security – a measure of the system's ability to protect data and information from unauthorized access while still providing access to people and systems that are authorized (incl. confidentiality, integrity, and availability)
- Testability – the ease with which software can be made to demonstrate its faults through testing
- Usability – the ease for the user to accomplish a desired task and the kind of user support the system provides (user experience)

Quality Attributes addressed by Architecture

ISO/IEC 25010 (9126)

- Functional Suitability – incl. Functional Completeness, Functional Correctness, Functional Appropriateness
- Usability – incl. Appropriateness Recognizability, Learnability, Operability, User Error Protection, User Interface Aesthetics, Accessibility
- Compatibility – incl. Co-existence, Interoperability
- Reliability – incl. Maturity, Availability, Fault Tolerance, Recoverability
- Performance Efficiency – incl. Time Behavior, Resource Utilization, Capacity
- Security – incl. Confidentiality, Integrity, Non-Repudiation, Authenticity, Accountability

---

- Maintainability – incl. Modularity, Reusability, Analyzability, Modifiability, Testability
- Portability – incl. Adaptability, Installability, Replaceability

Categories of Software Quality Attributes

Characteristics of software that affect its ability to satisfy stated and implied needs

- CMU SEI
  - End User's Viewpoint
    - Functionality
    - Availability
    - Interoperability
    - Performance
    - Security
    - Usability
  - Developer's Viewpoint
    - Modifiability
    - Testability
  - Business's Viewpoint
    - Time-to-Market
    - Cost vs. Benefits
    - Projected Life-Time
    - Targeted Market
    - Integration with Legacy
    - Roll-out (Roll-back) Schedule

- ISO/IEC 25010
  - End User's Viewpoint
    - Functional Suitability
    - Reliability
    - Compatibility
    - Performance/Efficiency
    - Security
    - Usability
  - Developer's Viewpoint
    - Maintainability
    - Portability
  - Business's Viewpoint

MISSING !

Quality Attributes are often Conflicting → require Trade-Offs

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		-								+	-	

Measuring Software Size & Quality

Standard Quality Measures

- Consortium for IT Software Quality (CISQ)
  - organized (by CMU SEI & OMG) to develop standard measures for evaluating and bench-marking the reliability, security, performance efficiency, and maintainability of IT software

- Provides standards for
  - **Automated Function Points** – measures the functional size of software
  - **Automated Enhancement Points** – measures the size of both functional and non-functional code in one measure
  - **Automated Quality Characteristic Measures** (based on quality characteristics from ISO 25010)
    - **Security** – measures 22 violations in source code representing the most exploited security weaknesses in software (defined by CWE/Sans Institute and OWASP)
    - **Reliability** – measures 29 violations in source code impacting the availability, fault tolerance, and recoverability of software
    - **Performance Efficiency** – measures 15 violations in source code impacting response time and utilization of processor, memory, and other resources
    - **Maintainability** – measures 20 violations in source code impacting the comprehensibility, changeability, testability, and scalability of software
  - **Automated Technical Debt** – a measure of corrective maintenance effort due to violations (weaknesses) remaining in a software application

Content

Quality means doing it right when no one is looking

Henry Ford

- Introduction
  - Architecture and Requirements
  - Software Quality Models
- Software Quality Attributes
  - Categories of Software Quality Attributes
  - Measuring Software Quality (CISQ)
- Evaluation of Software Architectures
  - Quality Attribute Scenarios
  - Architecture Trade-off Analysis Method (ATAM)
  - Software Metrics
- Cost and Value of Architecture
  - Value of Architecture
  - Valuation of Architecture Decisions (Option Value of Architecture Decisions)
- Conclusions

Evaluation of Software Architectures

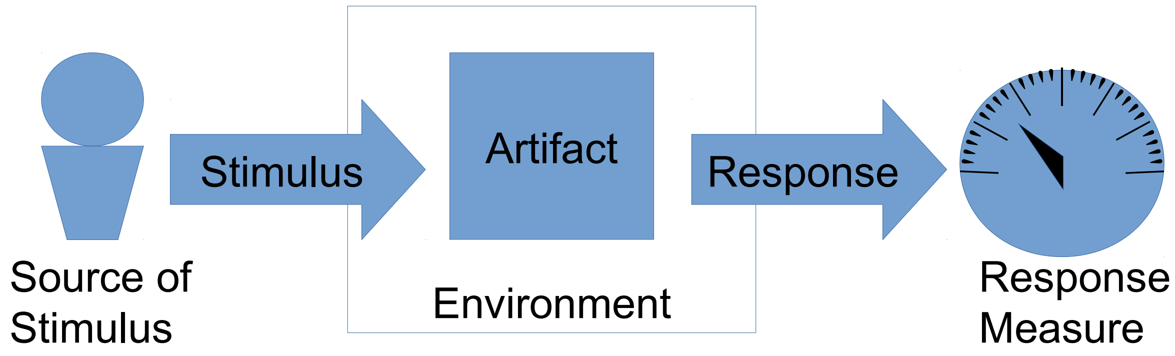
CMU SEI

- The earlier you find a problem in a software project, the better
  - architectural decisions are later hard or impossible to change
  - architectural decisions affect whether the system goals could be met
  - many structures related to the building of the system are organized around the architecture
- Architecture evaluation is a cheap way to avoid problems
- Architecture evaluation answers to
  - Is this architecture suitable for the system for which it was designed?
  - Which of two or more competing architectures is the most suitable one for the system at hand?
- Architecture is suitable if
  - The system that results from it will meet its quality goals (achieves required properties)
  - The system can be built using the resources at hand (staff, budget, time, …) – architecture is buildable

Quality Attribute Scenarios
(Provide Unambiguous & Testable Requirements)

CMU SEI

- **Stimulus** – a condition that requires a response when it arrives at a system
- **Source of stimulus** – some entity (a human, a computer system, or any other actuator) that generated the stimulus
- **Environment** – conditions under which the stimulus occurs (an overload, a normal operation, or some other relevant state)
- **Artifact** – artifact that is stimulated (a collection of systems, the whole system, or some piece or pieces of it)
- **Response** – the activity undertaken as the result of the arrival of the stimulus
- **Response Measure** – response should be measurable in some fashion so that the requirement can be tested



Types of Quality Attribute Scenarios

CMU SEI

- Use-case scenarios reflect the normal state or operation of the system
- Growth scenarios are anticipated changes to the system
  - These can be about the execution environment (e.g., double the message traffic) or about the development environment (e.g., change message format shown on the operator's console)
- Exploratory scenarios involve extreme changes to the system that may be unanticipated and that may occur in undesirable situations
  - Used to explore the boundaries of the architecture (e.g., message traffic grows 100 times, requiring the replacement of the operating system)

A Family of Quality Attribute Driven Methods based on Scenarios

CMU SEI

- Quality Attribute Workshop (QAW) – elicit and document quality attribute requirements accurately, resulting scenario descriptions

- Attribute-Driven Design (ADD) – shape design decisions around quality attribute considerations, resulting architecture description at least in three main views

- Architecture Trade-off Analysis Method (ATAM) – use scenarios to assess the consequences of architectural decision alternatives in light of quality attribute requirements (trade-offs among multiple quality attributes), resulting consequences of architectural decisions (identified trade-offs and risks)

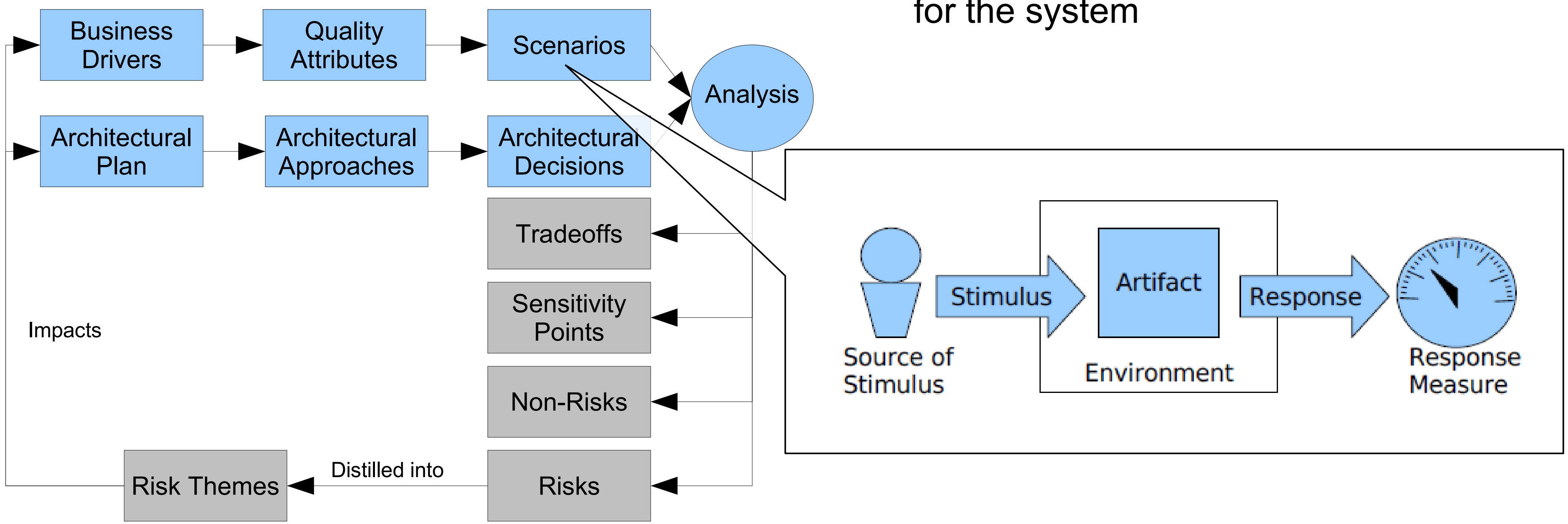
- Active Reviews for Intermediate Design (ARID) – blends Active Design Reviews with the ATAM, to asses partial designs, resulting issues/problems

- Cost-benefit Analysis Method (CBAM) – facilitates architecture-based economic analyses, resulting architectural strategies with associated cost and risks

Architecture Trade-off Analysis Method (ATAM)

CMU SEI

- Input
  - A set of identified architectural approaches
  - “utility tree” – driving architectural requirements
  - The set of scenarios mapped onto architecture
- Output
  - 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



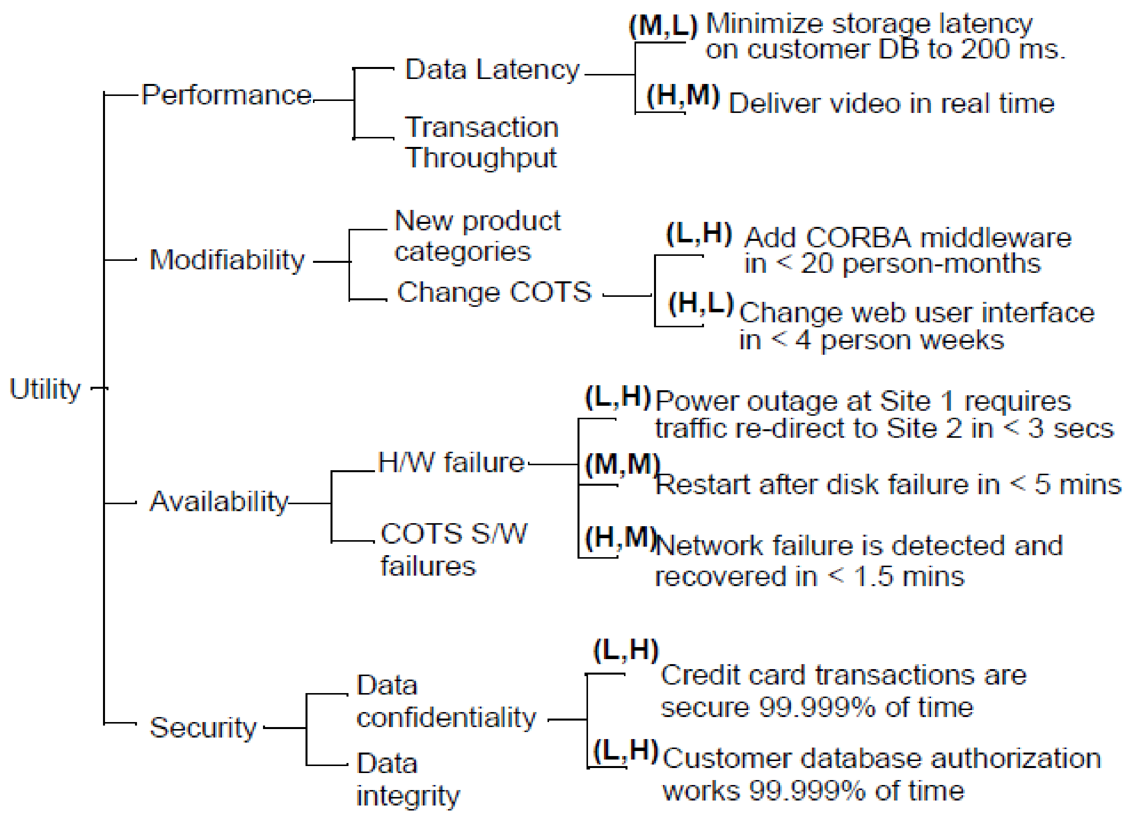
Utility Tree

CMU SEI

- Top-down elicitation to capture quality requirements by successively refining the top-most system quality goal (utility) into more and more specific quality goals (e.g. such as performance, modifiability, and availability)

- Utility tree has 4 levels

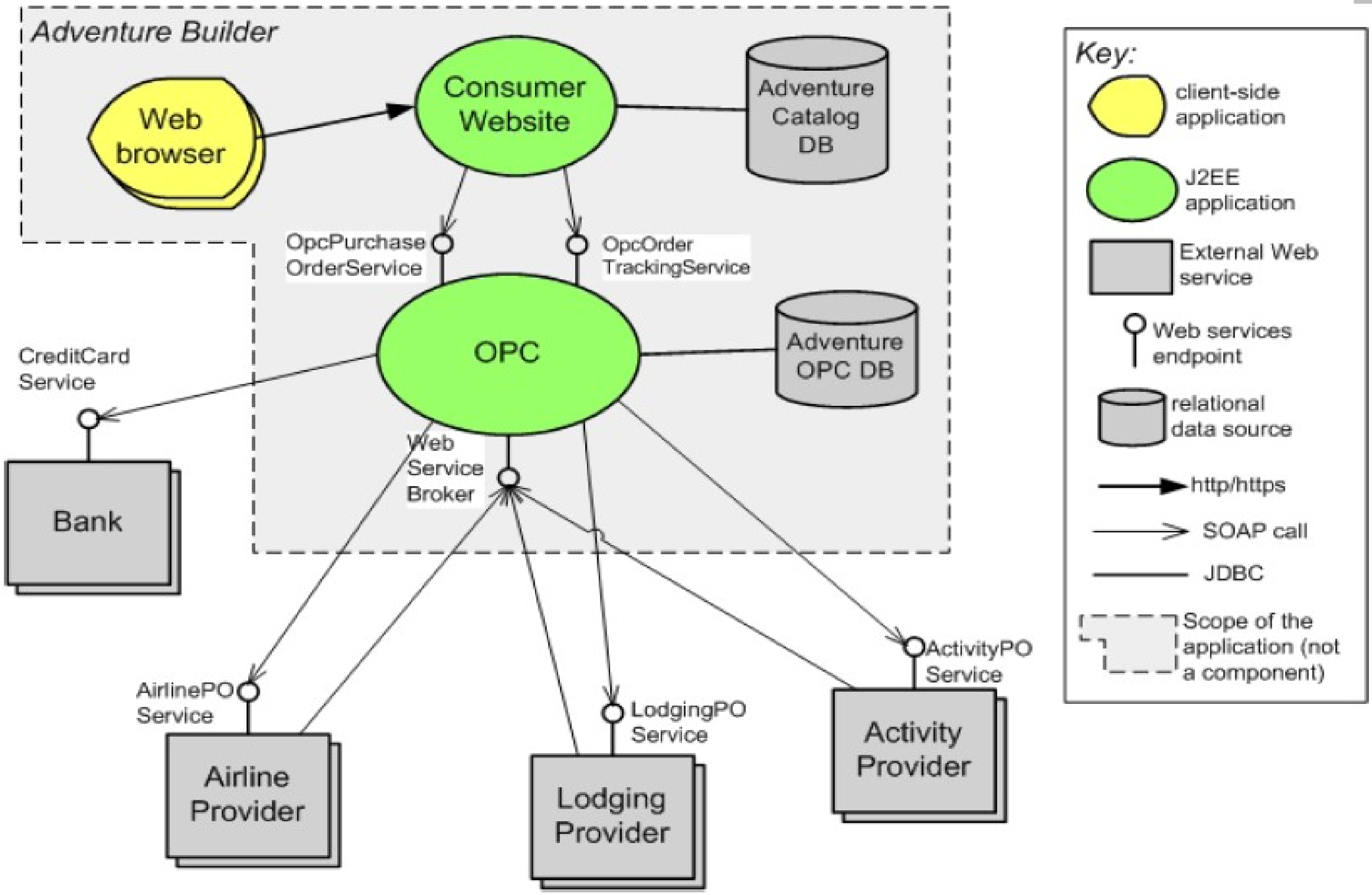
  Utility →
  Quality Attributes →
  Attribute Concern →
  Scenarios



Example

Travel Agency System Architecture

CMU SEI



Example

SOA Quality Attribute Scenario (Modifiability)

CMU SEI

Quality Attribute	Scenario
…	…
Scenario 2 Modifiability	• (**Source**) Business Analyst/Customer • (**Stimulus**) Add a new airline provider that uses its own Web services interface. • (**Artifact**) OPC (Order Processing Center) • (**Environment**) Developers have already studied the airline provider interface definition. • (**Response**) New airline provider is added that uses its own Web services. • (**Response Measure**) No more than 10 person-days of effort are required for the implementation (legal and financial agreements are not included).
…	…

Example

SOA Quality Attribute Scenario Analysis

CMU SEI

Analysis for Scenario 2	
Scenario Summary	A new airline provider that uses its own Web services interface is added to the system in no more than 10 person-days of effort for the implementation.
Business Goal(s)	Permit easy integration with new business partners.
Quality Attribute	Modifiability, interoperability
Architectural Approaches and Reasoning	• Asynchronous SOAP-based Web services • Interoperability is improved by the use of document-literal SOAP messages for the communication between OPC and external services. • Adventure Builder runs on Sun Java System Application Server Platform • Edition V8.1. This platform implements the WS-I Basic Profile V1.1, so interoperability issues across platforms are less likely to happen.
Risks	The design does not meet the requirement in this scenario, because it assumes that all external transportation providers implement the same Web services interface called ‘AirlinePOService’. The design does not support transportation providers that offer their own service interface.
Tradeoffs	The homogenous treatment of all transportation providers in OPC increases modifiability. However, intermediaries are needed to interact with external providers that offer heterogeneous service interfaces, as in this scenario. These intermediaries represent a performance overhead, because they may require routing messages and extensive XML processing.

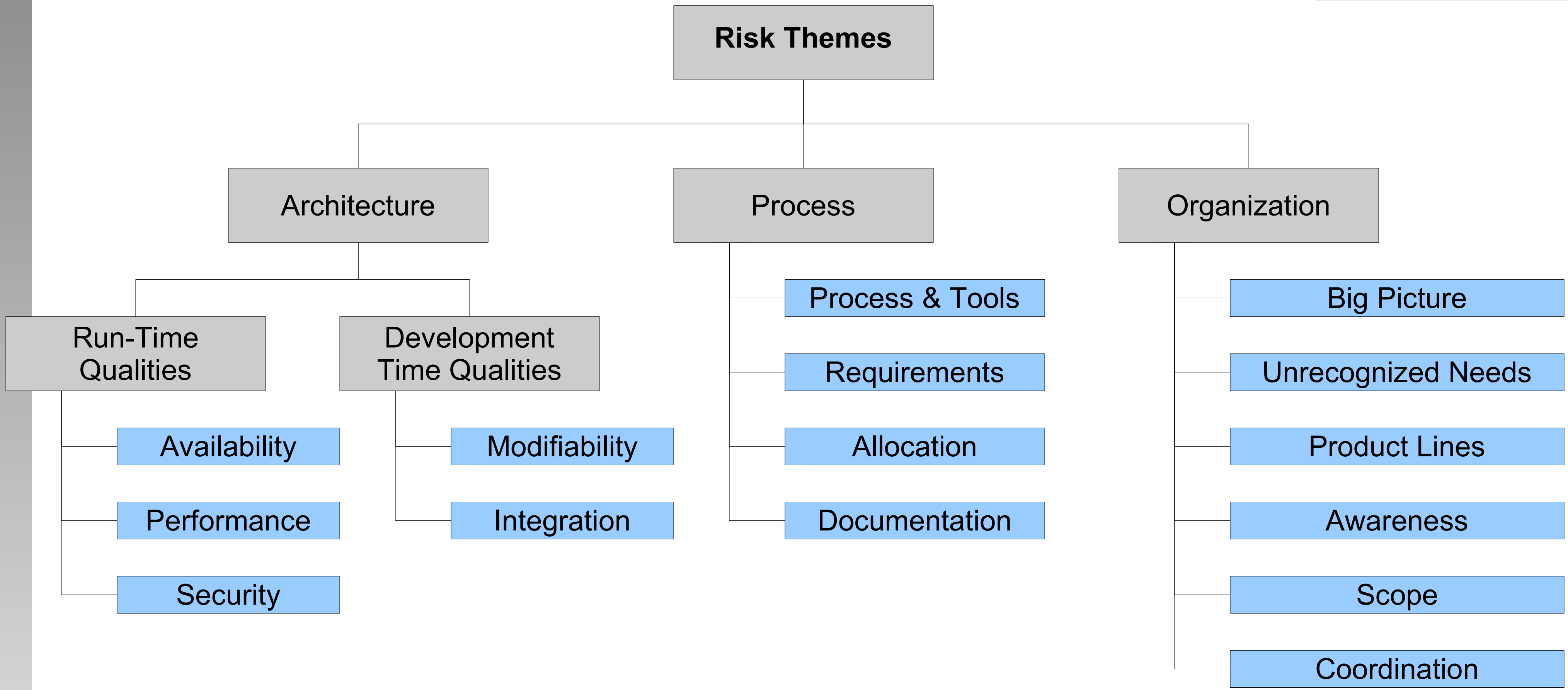
Some more SOA Quality Attribute Scenarios

CMU SEI

- Performance
  - A sporadic request for service ‘X’ is received by the server during normal operation → the system processes the request in less than ‘Y’ seconds
- Availability
  - 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
- Security
  - 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, and the system notifies the system administrators
- Testability
  - 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 person-week
- Interoperability
  - 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
- Modifiability
  - 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
- Reliability
  - 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

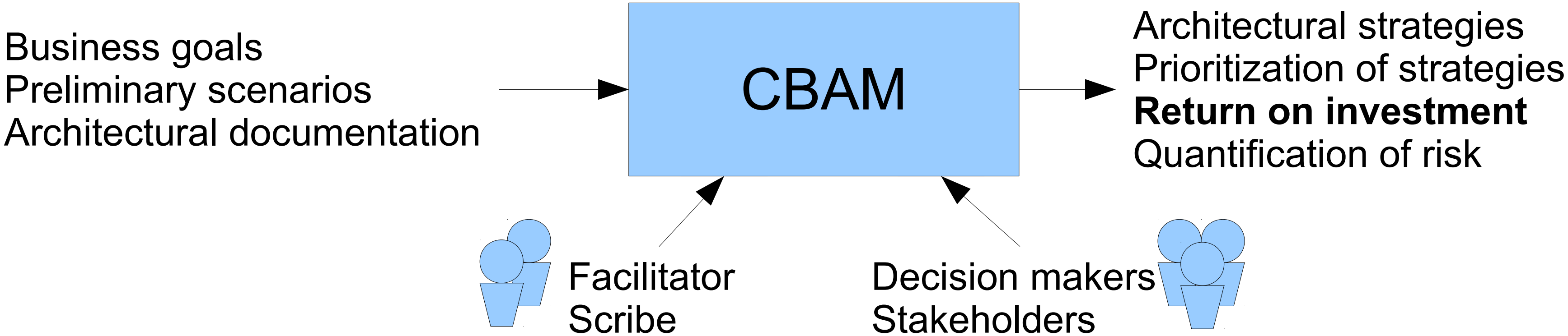
Risk Themes found in ATAM Evaluations

CMU SEI



Cost-Benefit Analysis Method CBAM (connecting architecture trade-offs to economics)

CMU SEI



1. Choose scenarios and architectural strategies
2. Assess quality attribute benefits
3. Quantify the benefits of architectural strategies
4. Quantify the costs and schedule implications of the architectural strategies
5. Calculate the desirability of each option
6. Make architectural design decisions

Some Simple Software Metrics

- Cyclomatic Complexity (T. J. McCabe, 1976)
  - measure of the complexity of a module's decision structure (number of linearly independent paths through a module)

    CC = Edges - Nodes + 2 * Parts

- Software Science (M. H. Halstead, 1977)
  - Measurable properties
    - $\mathbf{n_1}$ = number of unique or distinct operators
    - $\mathbf{n_2}$ = number of unique or distinct operands
    - $\mathbf{N_1}$ = number of total usage of all the operators
    - $\mathbf{N_2}$ = number of total usage of all the operands
  - Calculated properties
    - Vocabulary: $\mathbf{n = n_1 + n_2}$
    - Length: $\mathbf{N = N_1 + N_2}$ or estimated as ($\mathbf{N' = n_1 \log_2 n_1 + n_2 \log_2 n_2}$)
    - Volume: $\mathbf{V = N \log_2 n}$
    - Level: $\mathbf{L' = 2\, n_2 / n_1\, N_2}$
    - Intelligence Content (Complexity): $\mathbf{I = L'\, V = (2\, V\, n_2) / (n_1\, N_2)}$

Software Metrics – Measuring OO Programs (Chidamber & Kemerer)

- Coupling Between Object Classes (**CBO**) – number of other classes to which given class is coupled

  CBO = | { classes that **given class references U** classes that **reference given class** } |

- Lack of Cohesion in Methods (**LCOM**) – pairs of methods that share references to instance variables

  LCOM = P – Q *if* **P > Q** *else* **0**

  where
  - **P** is # of pairs of methods that do not share instance variables
  - **Q** is # of pairs of methods that share instance variables

- Others
  - Weighted methods per Class (**WMC**) – number of methods (orig. sum of the complexities of the methods of a class)
  - Response Set for a class (**RFC**) – cardinality of the set of methods that can be executed (directly or indirectly) in response to a message received by an object of that class – measures the degree of communication
  - Depth of Inheritance Tree of a class (**DIT**)
  - Number Of Children of a class (**NOC**)

Software Metrics – Measuring Entropy (E. B. Allen)

- Entropy is the average number of bits needed to describe the dependencies a program unit has on the rest of the system

- Entropy (average information (bits) per node)

$$H(S) = \Sigma\ 1/(n+1)\ (-log_2\ P_L)$$

  – where **S** is a CDG (Code Dependency Graph), $\mathbf{P_L}$ is probability of similar node, and **n** is the no. of nodes in **S**

- Total amount of information – estimated minimum description length

$$I(S) = (n+1)\ H(S) = \Sigma\ -log_2\ P_L$$

- Coupling (relationship between components or excess entropy)

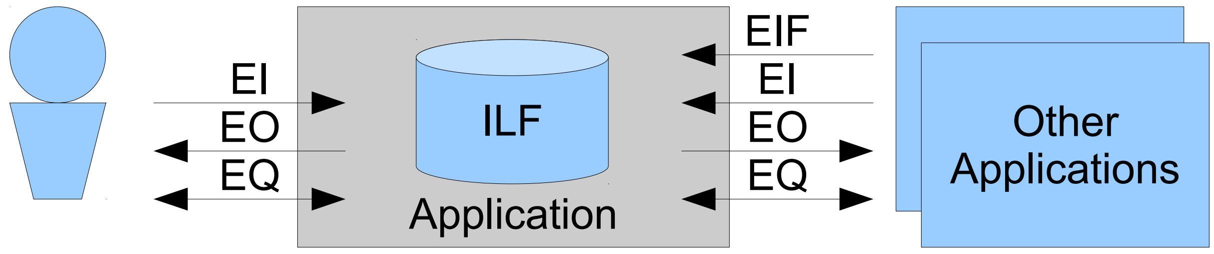
$$C(S) = \Sigma\ I(S_i) - I(S)$$

  – where **i** ranges on number of nodes

Software Metrics – Measuring Functional Volume

> Raising the number of function points to the 1.25 power predicts the number of defects
>
> C. Jones

- Counting (IFPUG, …, OMG CISQ)
  - External Inputs
  - External Outputs
  - External Inquiries
  - Internal Logical Files
  - External Interface Files



- Adjusting with
  - type
  - complexity
  - 14 system characteristics (degree of influence 0..5)
    - data communication, distributed functions, performance, heavily used configuration, transaction rate, on-line data entry, end user efficiency, online update, complex processing, reusability, installation ease, operational ease, multiple sites, facilitate change

	Simple	Average	Complex
External Input	3	4	6
External Output	4	5	7
Internal Logical File	7	10	15
External Interface File	5	7	10
External Inquiry	3	4	6

$$FP=\left[\sum_{i=1}^{n} Weight_i\right]\times\left[0.65+0.01\times\sum_{j=1}^{14} DegreeOfInfluenceOfGSC_j\right]$$

Example

Software Metrics – Measuring Functional Volume "Simple Store"

70.2 FPAs for Java … (C. Jones)
3.7 KLOC
164 test cases needed
256 potential defects
6.7 (Estonian ☺) man-months

- Description (Use Cases?)
  - Find/Add Customers
  - Check Customers' Credit
  - Enter Orders
  - Check Availability of Goods (if needed create back-orders)
  - Produce Invoices and Accept Payments
  - Update Stock & Customer Accounts
- Adjusting ...
  - 0.65 (if nothing special)
  - **FPA = 108 * 0.65 = 70.2**
  - 0.65 + 0.05 (very easy to change)
  - **FPA = 108 * 0.70 = 75.6**
  - 0.65 + 0.01 * 14 * 5 (very complex)
  - **FPA = 108 * 1.35 = 145.8**

- Counting ...
  - 6 External Inputs
    - Customer No, Main Menu
    - Customer Details, Order Details, Stock Delivery Details, Payment Details
  - 6 External Outputs
    - Credit Rating
    - Invoice, Dispatch, Customer Details, Order Details, Stock Details
  - 3 External Inquiries
    - Customer Details, Order Details, Stock Details
  - 4 Internal Logical Files
    - Customers, Goods, Orders, Transactions
  - 1 External Interface Files
    - Customer Credit Details

	Simple		Average		Complex	TOTAL
External Input	**2**	3	**4**	4	6	**22**
External Output	**1**	4	**5**	5	7	**29**
Internal Logical File		7	**4**	10	15	**40**
External Interface File	**1**	5		7	10	**5**
External Inquiry		3	**3**	4	6	**12**
Unadjusted FPs						**108**

Content

> Quality means doing it right when no one is looking
>
> Henry Ford

Value of Software Architecture

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

Davidson, 2002

Value that Architecture provides

- Users and operators of the system
  - High availability and performance
  - Survival from partial failure
- Acquirers and owners of the system
  - Easy integration into environment
- Suppliers and developers of the system
  - Speed and freedom
  - Guidance
  - Reuse of effort, skills and know-how
  - Ease of integration
- Builders and maintainers of the system
  - Survival of extension, adaptation, requirements changes, platform changes, etc.

Value of Architecture (Description)

- 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

Measuring Value of Software Architecture

> Focus on quality and cost will decrease
> Focus on costs and quality will decrease
>
> W. E. Deming

- Value of Software Architecture
  - Cost of realization of risks compared to cost of architecture

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

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

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

Real Options for Valuation of Software Architecture

Real option

- is a right (opportunity), but not an obligation to make a decision in the future
- might be exercised multiple times (different from financial option)

- Applicable when
  - there is ***Uncertainty***
  - there is ***Business Change***
  - ***New Information*** should/could be exploited when it comes available
  - ***Action*** today should create
    - Possibility of ***future design choices***
    - Possibility of ***future value***

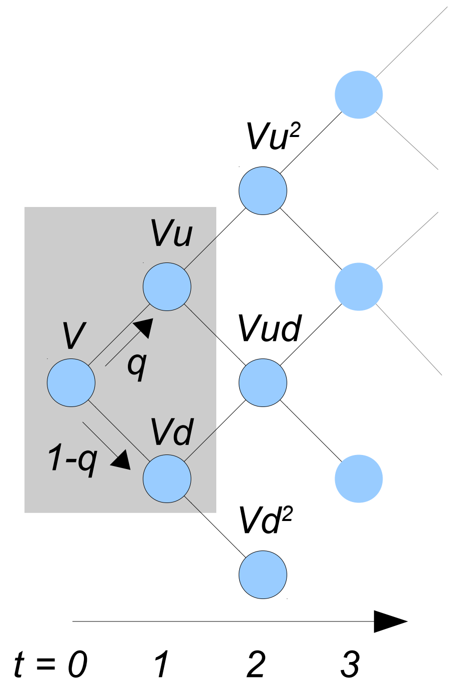
- Strategic Value with Real Options

$$NPV_{strategic} = NPV_{traditional} + Value_{real.options}$$

- Valuation of real options
  - Binomial lattices – decision trees with probabilities

$$V_{option} = (p\ V_{up} + (1\text{-}p)\ V_{down}) / (1+r)$$
$$p_{risk\ neutral} = (1+r\text{-}d) / u\text{-}d$$

  - Markov processes
  - Monte Carlo simulations



Example

Valuation of Software Architecture Decision as Option

- Suppose that
  - At first step of the project it is possible the make €1000 investment, which can with 50% probability be sufficient, but with 50% probability there will be need to invest €3000 more, to get business profit with NPV_{profit} €2200
- Then

  $NPV_{traditional}$ = €2200 - (€1000 + 50% * €3000) = - €300 → **don't invest**
- But
  - as the project can be canceled, when worst case materializes, then

  $NPV_{strategic}$ = 50% * €2200 - €1000 = €100 → **invest – good investment!**
  (investment of €1000 creates an option to get €2200 with 50% probability)

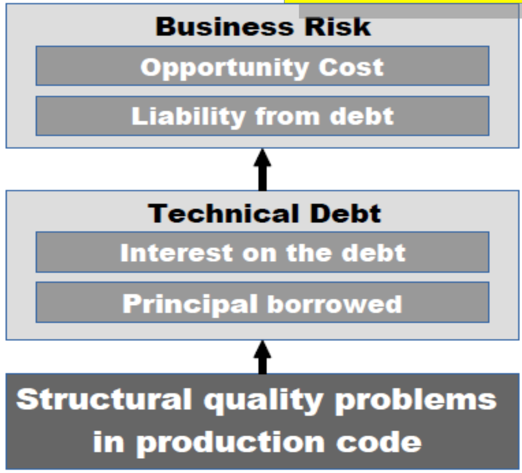
  $Value_{option} = NPV_{strategic} – NPV_{traditional}$ = ***€400***

Design Principles Guided by Economic Value

- Real Option Theory – Qualitative Design Principles (K. 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

Calculating Technical Dept

OMG CISQ



- *Technical Debt* – metaphor by W. Cunningham to describe effect of intentional decisions to release sub-optimal code to achieve some objectives (e.g. faster delivery)
  - S. McConnell extended *Technical Debt* to include both **intentional** and **unintentional violations** of good architectural and coding practice
- Standard way of measuring *Technical Debt* based on source code analysis (OMG CISQ) – looking for the specific violation patterns
  - *Opportunity cost* — benefits that could have been achieved had resources been put on new capability rather than retiring technical debt
  - *Liability* — business costs related to outages, breaches, corrupted data, etc.
  - *Interest* — continuing IT costs attributable to the violations causing technical debt, i.e, higher maintenance costs, greater resource usage, etc.
  - *Principal* — cost of fixing problems remaining in the code after release that must be remediated

1. Detect occurrences of patterns specified as weaknesses by OMG approved specifications
2. Assign an estimate of the amount of time to remediate each occurrence of a weakness
3. Collect qualification information about the occurrences of each weakness
4. Compute an adjustment factor as a function of qualification information about each of the occurrences to negatively or positively impact the effort estimate
5. Sum the total amount of time across all the occurrences for all 86 violations (variations in labor costs, skill levels, or currencies must be made based on local conditions)

Content

Quality means doing it right when no one is looking

Henry Ford

- Introduction
  - Architecture and Requirements
  - Software Quality Models
- Software Quality Attributes
  - Categories of Software Quality Attributes
  - Measuring Software Quality (CISQ)
- Evaluation of Software Architectures
  - Quality Attribute Scenarios
  - Architecture Trade-off Analysis Method (ATAM)
  - Software Metrics
- Cost and Value of Architecture
  - Value of Architecture
  - Valuation of Architecture Decisions (Option Value of Architecture Decisions)
- Conclusions

Conclusions

> It is not about bits, bytes and protocols, but profits, losses and margins
>
> Lou Gerstner

- (Software) Quality
  - degree to which the system satisfies the stated and implied needs of its various stakeholders, and thus provides value (ability of system to meet customer or user needs, expectations, or requirements)

- (Software) Quality Attribute
  - a measurable or testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders

- Economic Value of (Software) Architecture
  - Value of Architecture = cost of realization of risks compared to cost of architecture
  - Value of Architecture Description = cost of performing activities without architecture description compared to cost of documenting architecture

- (Software) Architecture creates choices/options, which have value – **designing and building an architecture is an investment activity**
  - Architecture Investment is a real option
    - provides an opportunity (right, but not an obligation) to make a decision in the future
    - might be exercised multiple times (different from financial option)

36. The soft overcomes the hard. The slow overcomes the fast. Let your workings remain a mystery. Just show people the results.

Lao Tsu (by Philippe Kruchten)

Thank You!

Questions

- List main software quality attributes
- How to evaluate the goodness of an architecture?
- What are parts of quality attribute scenario?
- Bring examples of quality attribute scenario for … ?
- Who wants to pay for documents?
- Who wants to pay for exploring of various design alternatives?
- ...

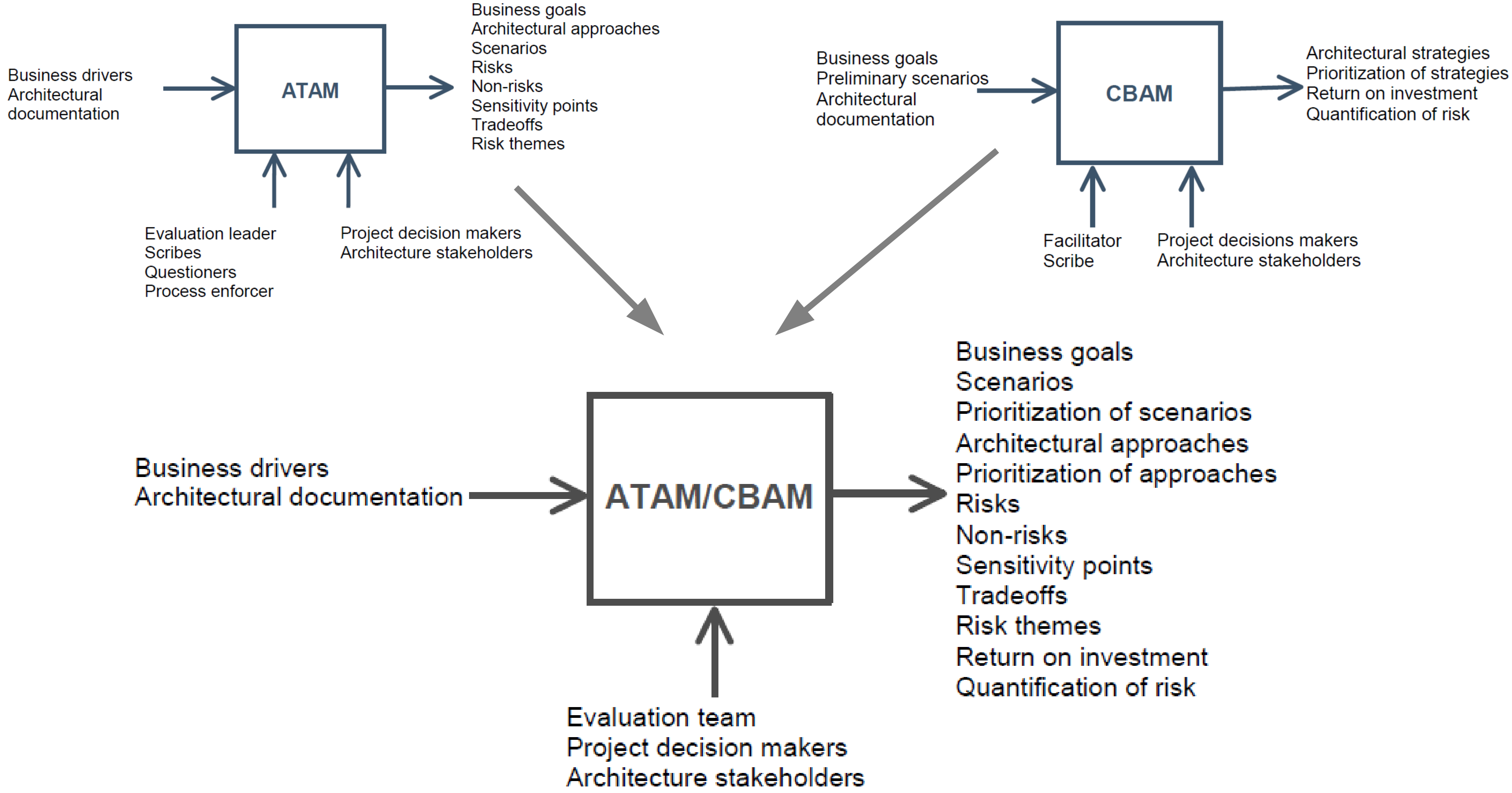
- How to measure complexity?
- How to measure functional volume?
- How to measure the cost and value of design knowledge?
- How to measure cost of having (or not having) good architecture?
- How to measure cost of having (or not having) good documentation?
- When it makes sense to delay the architecture decision?
- ...

Literature

- https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=5177
- https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=8443
- http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
- http://it-cisq.org/
- http://repository.cmu.edu/cgi/viewcontent.cgi?article=1327&context=sei
-
- https://pdfs.semanticscholar.org/2192/41cd2eb39264a32ed37f75252d17dfb11663.pdf
- http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.49.4666&rep=rep1&type=pdf
- http://www.ifpug.org/
- http://www.ifpug.org/wp-content/uploads/2017/04/IYSM.-Thirty-years-of-IFPUG.-Software-Economics-and-Function-Point-Metrics-Capers-Jones.pdf
- http://csse.usc.edu/tools/COCOMOII.php
- https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=8307
- https://www.researchgate.net/publication/220210234_Using_Binomial_Decision_Trees_to_Solve_Real-Option_Valuation_Problems
-
- … Google “software quality attribute” + “value of architecture” …

Joining Attribute-Based Methods

CMU SEI



Evaluation of Software Architectures

CMU SEI

1. Present Business Drivers
   – the participants are expected to understand the system and its business goals and their priorities

2. Present Architecture
   – all participants are expected to be familiar with the system (a brief overview of the architecture, using at least Module and Component & Connector views, and 1 - 2 scenarios are traced through these views)

3. Identify Architectural Approaches
   – the architecture approaches for specific quality attribute concerns are identified by the architect

4. Generate Quality Attribute Utility Tree
   – utility tree is created/updated, with new scenarios, new response goals, or new scenario priorities and risk assessments

5. Analyze Architectural Approaches
   – mapping the highly ranked scenarios onto the architecture

6. Present Results
   – review the existing and newly discovered risks, non-risks, sensitivities, and trade-offs and discusses whether any new risk themes have arisen

Generic Scenario for Availability

CMU SEI

	Possible Values
Source	Internal/external: people, hardware, software, physical infrastructure, physical environment
Stimulus	Fault: omission, crash, incorrect timing, incorrect response
Artifact	Processors, communication channels, persistent storage, processes
Environment	Normal operation, startup, shutdown, repair mode, degraded operation, overloaded operation
Response	Prevent the fault from becoming a failure Detect the fault: ■ Log the fault ■ Notify appropriate entities (people or systems) Recover from the fault: ■ Disable source of events causing the fault ■ Be temporarily unavailable while repair is being effected ■ Fix or mask the fault/failure or contain the damage it causes ■ Operate in a degraded mode while repair is being effected
Response Measure	Time or time interval when the system must be available Availability percentage (e.g., 99.999%) Time to detect the fault Time to repair the fault Time or time interval in which system can be in degraded mode Proportion (e.g., 99%) or rate (e.g., up to 100 per second) of a certain class of faults that the system prevents, or handles without failing

Generic Scenario for Interoperability

CMU SEI

	Possible Values
Source	A system initiates a request to interoperate with another system
Stimulus	A request to exchange information among system(s)
Artifact	The systems that wish to interoperate
Environment	System(s) wishing to interoperate are discovered at runtime or known prior to runtime
Response	One or more of the following: ■ The request is (appropriately) rejected and appropriate entities (people or systems) are notified ■ The request is (appropriately) accepted and information is exchanged successfully ■ The request is logged by one or more of the involved systems
Response Measure	One or more of the following: ■ Percentage of information exchanges correctly processed ■ Percentage of information exchanges correctly rejected

Generic Scenario for Modifiability

CMU SEI

	Possible Values
Source	End user, developer, system administrator
Stimulus	A directive to add/delete/modify functionality, or change a quality attribute, capacity, or technology
Artifact	Code, data, interfaces, components, resources, configurations, . . .
Environment	Runtime, compile time, build time, initiation time, design time
Response	One or more of the following: ■ Make modification ■ Test modification ■ Deploy modification
Response Measure	Cost in terms of the following: ■ Number, size, complexity of affected artifacts ■ Effort ■ Calendar time ■ Money (direct outlay or opportunity cost) ■ Extent to which this modification affects other functions or quality attributes ■ New defects introduced

Generic Scenario for Performance

CMU SEI

	Possible Values
Source	Internal or external to the system
Stimulus	Arrival of a periodic, sporadic, or stochastic event
Artifact	System or one or more components in the system
Environment	Operational mode: normal, emergency, peak load, overload
Response	Process events, change level of service
Response Measure	Latency, deadline, throughput, jitter, miss rate

Generic Scenario for Security

	Possible Values
Source	Human or another system which may have been previously identified (either correctly or incorrectly) or may be currently unknown (attacker may be from outside or from inside the organization)
Stimulus	Unauthorized attempt is made to display data, change or delete data, access system services, change the system's behavior, or reduce availability
Artifact	System services, data within the system, a component or resources of the system, data produced or consumed by the system
Environment	The system is either online or offline; either connected to or disconnected from a network; either behind a firewall or open to a network; fully operational, partially operational, or not operational
Response	Transactions are carried out in a fashion such that ■ Data or services are protected from unauthorized access ■ Data or services are not being manipulated without authorization ■ Parties to a transaction are identified with assurance ■ The parties to the transaction cannot repudiate their involvements ■ The data, resources, and system services will be available for legitimate use The system tracks activities within it by ■ Recording access or modification, and access data, resources, or services ■ Notifying appropriate entities when an apparent attack is occurring
Response Measure	One or more of the following: ■ How much of a system is compromised when a particular component or data value is compromised ■ How much time passed before an attack was detected ■ How many attacks were resisted ■ How long does it take to recover from a successful attack ■ How much data is vulnerable to a particular attack

CMU SEI

Generic Scenario for Testability

CMU SEI

	Possible Values
Source	Unit testers, integration testers, system testers, acceptance testers, end users, either running tests manually or using automated testing tools
Stimulus	A set of tests is executed due to the completion of a coding increment such as a class layer or service, the completed integration of a subsystem, the complete implementation of the whole system, or the delivery of the system to the customer
Artifact	The portion of the system being tested
Environment	Design time, development time, compile time, integration time, deployment time, run time
Response	One or more of the following: execute test suite and capture results, capture activity that resulted in the fault, control and monitor the state of the system
Response Measure	One or more of the following: effort to find a fault or class of faults, effort to achieve a given percentage of state space coverage, probability of fault being revealed by the next test, time to perform tests, effort to detect faults, length of longest dependency chain in test, length of time to prepare test environment, reduction in risk exposure (size(loss) × prob(loss))

Generic Scenario for Usability

CMU SEI

	Possible Values
Source	End user, possibly in a specialized role
Stimulus	End user tries to use a system efficiently, learn to use the system, minimize the impact of errors, adapt the system, or configure the system
Artifact	System or the specific portion of the system with which the user is interacting
Environment	Runtime or configuration time
Response	The system should either provide the user with the features needed or anticipate the user’s needs
Response Measure	One or more of the following: task time, number of errors, number of tasks accomplished, user satisfaction, gain of user knowledge, ratio of successful operations to total operations, or amount of time or data lost when an error occurs

Operations for Realizing Quality Attributes

CMU SEI

- *Separation* is an operation that places a distinct piece of functionality into a distinct component that has a well-defined interface to the rest of the world
- *Abstraction* is the operation of creating a virtual machine, a component whose function is to hide its underlying implementation
- *Compression* is the operation of removing layers or interfaces that separate system functions (the opposite of separation)
- *Composition* is the operation of combining two or more system components into a larger component (*Uniform Composition* is a restriction of this operation, limiting the composition mechanisms to a small set)
- *Replication* is the operation of duplicating a component within an architecture (used to enhance reliability (fault tolerance) and performance)
- *Resource sharing* is an operation that encapsulates either data or services and shares them among multiple independent consumers (typically there is a resource manager that provides the sole access to the resource)

Operations for Realizing Quality Attributes

CMU SEI

Operation	Software Quality Attribute							
	Scalability	Modifiability	Integrability	Portability	Performance	Reliability	Ease of Creation	Reusability
Separation	+	+	+	+	+/-		+/-	+
Abstraction	+	+	+	+	-		+	+
Compression	-	-	-	-	+		+/-	-
Uniform Composition	+		+				+	
Replication	-	-		-	+/-	+	-	-
Resource Sharing		+	+	+	+/-	-	+	+/-

Architectural Design Decisions for Quality Attributes

CMU SEI

- Allocation of responsibilities
- Coordination model
- Data model
- Management of resources
- Mapping among architectural elements
- Binding time decisions
- Choice of technology

Architecture and Quality Attributes

- For **high performance**
  - exploit potential parallelism by decomposing the work into cooperating or synchronizing processes
  - manage the inter-process and network communication volume and data access frequencies
  - be able to estimate expected latencies and throughputs
  - identify potential performance bottlenecks
- For **high accuracy**, pay attention to how the data elements are defined and used and how their values flow throughout the system
- For **security**
  - legislate usage relationships and communication restrictions among the parts
  - identify parts of the system where an unauthorized intrusion will do the most damage
  - possibly introduce special elements that have earned a high degree of trust
- For **modifiability** and **portability**, carefully separate concerns among the parts of the system, so that when a change affects one element, that change does not ripple across the system
- For deploying the system **incrementally** (releasing successively larger subsets), keep the dependency relationships among the pieces untangled, to avoid the “nothing works until everything works” syndrome

Software Architecture
Kinds of Structures & Quality Attributes

CMU SEI

Kind	Structure	Elements	Relations	Decisions	
Module Structures	**Decomposition**	Module	sub-module-of	Decomposition, structuring, information hiding, encapsulation	**Modifiability**
	Uses	Module	uses (requires)	Usable/useful sub-sets, extensions	**Extensibility, Subsetability**
	Layers	Layer	uses (requires), provides abstraction	Portability, ease of change and abstraction “virtual machines”	**Portability**
	Class	Class, Object	is-a (specializes), instance-of, ...	Reuse, commonality and planned incremental extension	**Modifiability, Extensibility**
	Data Model	Data Entity	{one,many}-to-{one,many}	Global data structures consistency	**Modifiability, Performance**
Component & Connector Structures	**Service**	Service, Bus, Registry, ...	runs-concurrently, excludes, precedes, ...	Independent development of components	**Interopertability, Modifiability**
	Concurrency	Process, Thread	can-run-parallel	Parallelism, access to resources	**Performance, Availability**
Allocation Structures	**Deployment**	Component, Hardware Devices, ...	allocated-to, migrates-to	Performance, security, availability	**Performance, Security, Availability**
	Implementation	Module, File Structure, ...	stored-in	Development, integration and testing	**Development Efficiency**
	Work Assignment	Module, Organization Unit, ...	assigned-to	Project management and communication	**Development Efficiency**