CSEE&T 2017
Software Architecture Educators' Workshop

Grace A. Lewis

glewis@sei.cmu.edu

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM17-0874

Background

Carnegie Mellon Software Engineering Institute has hosted educator workshops for the past 14 years

- Started in 2004 as the Architecture-Centric Engineering (ACE) Educators Workshop
- Changed its name to Software Engineering Educators Workshop in 2016 to broaden its scope beyond software architecture
- Free to any accredited college level educator in the field of software engineering
- “Entry fee” is a sharable artifact (established in 2011)

Three-day workshop

- First two days are training
- Third day is a facilitated group session to share experiences, ideas and shared artifacts for introducing software engineering topics into the college curriculum

Goals for Today

Provide a “very condensed version” of the educators workshop

- Architecture 101 in the morning
- ”Shared artifact” session in the afternoon

Brainstorm about the future of software architecture education

Plant the seed for creating a larger community of interest in software architecture (and software engineering) education

Agenda

Time	Topic
08:30 – 10:00	Architecture 101: Software Architecture Principles
10:00 – 10:30	Break
10:30 – 12:00	Architecture 101: Architecture Evaluation and Technical Debt
12:00 – 01:30	Lunch
01:30 – 03:00	Workshop — Part 1: Introduction and Group Work
03:00 – 03:30	Break
03:30 – 05:00	Workshop — Part 2: Presentations and Discussion

Software Architecture 101

CSEE&T 2017

Software Architecture Educators Workshop

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Agenda

What is Software Architecture?

Architecture Structures and Separation of Concerns

Architecture Evaluation

Technical Debt

Software Architecture 101

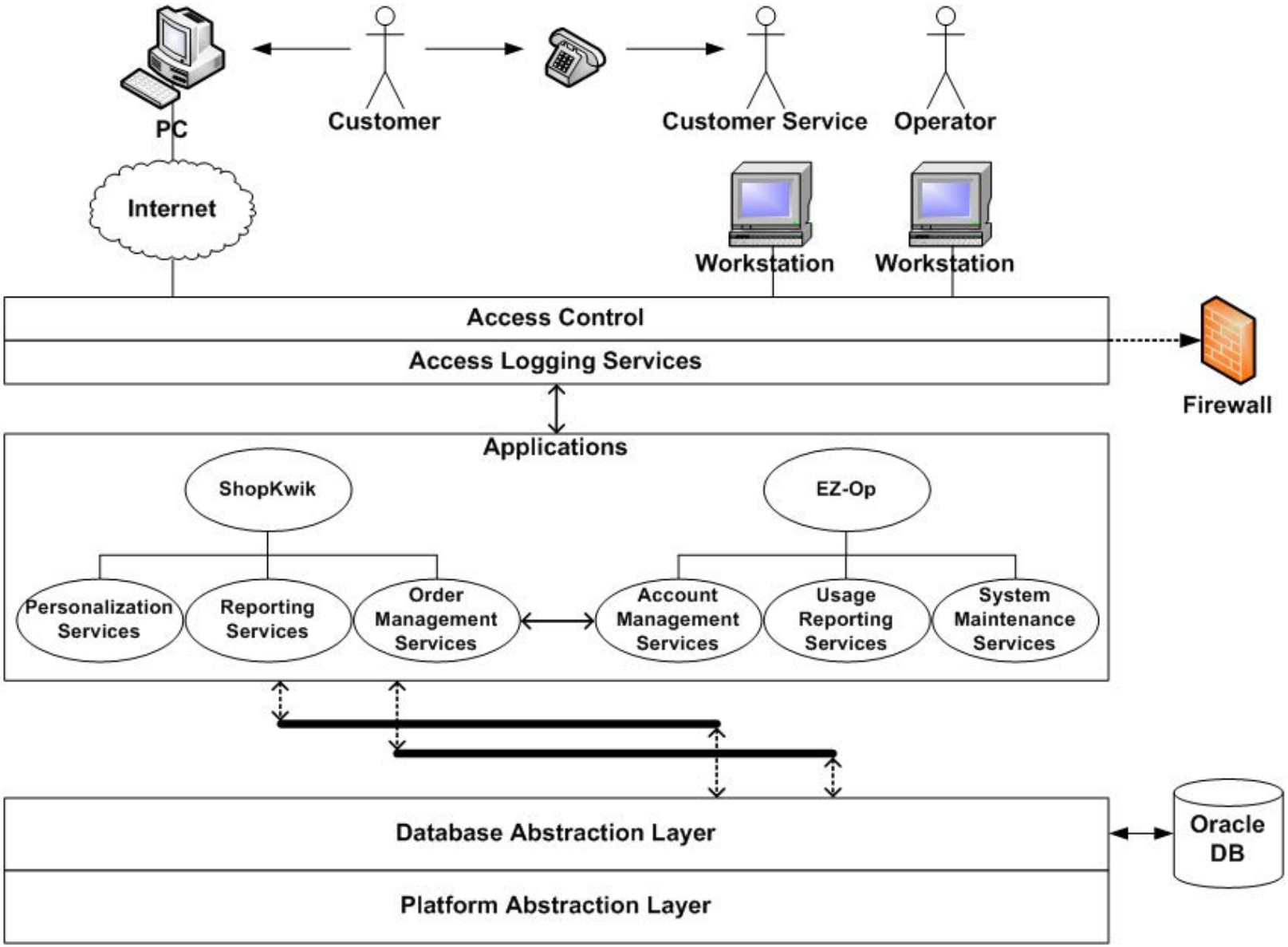
What is Software Architecture?

Typical Software Architecture 1

A software architecture is often depicted using an ad hoc box-and-line drawing of the system that is intended to solve the problems articulated by the specification.

- Boxes show elements or “parts” of the system.
- Lines show relationships among the parts.

Typical Software Architecture 2



What Can We Tell from This Picture?

The system consists of many elements?

The elements interact with each other over various networks?

Some of the elements represent layers and their relationships to one another?

The applications involved and the elements they comprise?

The system has multiple tiers?

The data, control, and communication mechanisms that are used?

What Does This Picture Omit? 1

What is the nature of the elements?

- What is the significance of their separation?
- Do they exist at runtime?
- Do they run at separate times?
- Are they processes, programs, or hardware?
- Are they objects, tasks, functions, or processes?
- Are they distributed programs or systems?

What are the responsibilities of the elements?

- What do they do?
- What functions do they provide in the context of the system?

What Does This Picture Omit? 2

What is the significance of the connections between the elements? Do the elements

- communicate with each other?
- control each other?
- send data to each other?
- use each other?
- invoke each other?
- synchronize with each other?

What is the significance of how the elements are positioned on the diagram?

- For example, does ShopKwik use, call, control, and/or contain Personalization Services, Reporting Services, and Order Management Services?

Architecture — Definitions

Extensive list of definitions at
http://www.sei.cmu.edu/architecture/definitions.html

"The structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time."

- Garlan, Perry. "Introduction to the Special Issue on Software Architecture," IEEE Transactions on Software Engineering, April 1995.

"The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both."

- Clements, et al. Documenting Software Architectures: Views and Beyond (2nd edition), Addison-Wesley, 2010.

Practitioner Definitions

"All architecture is design but not all design is architecture. Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change."

- Grady Booch

"Software architecture is the set of decisions which, if made incorrectly, may cause your project to be cancelled."

- Eoin Woods

"Decomposition of the problem in a way that allows your development organization to efficiently solve it, considering constraints like organizational structure, team locations, individual skills, and existing assets."

- John Klein

Complementary Perspectives

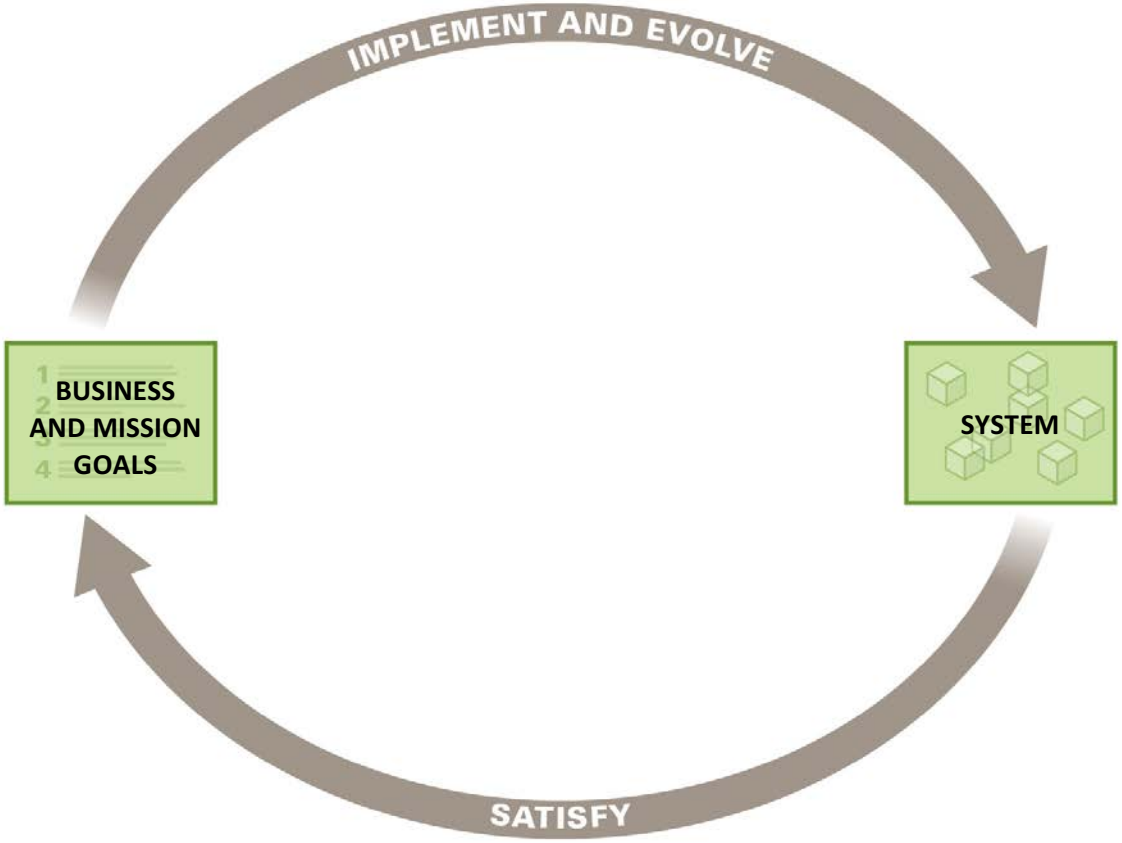
"Architecture is structure" (SEI and IEEE)

- But where do structures come from?
  - Architects make decisions!
  - Decisions create structures

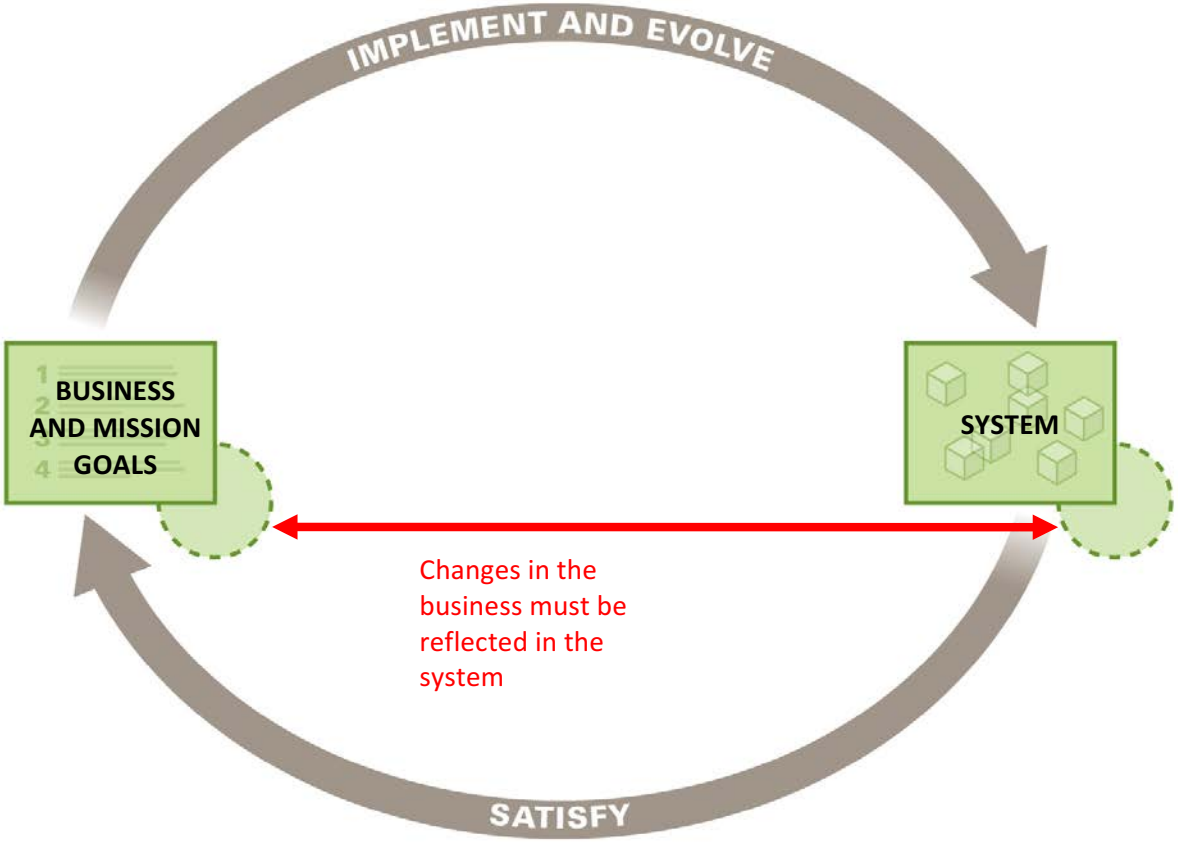
"Architecture is decisions" (Booch and Woods)

- A decision is a trade off
- Tyree and Ackerman (2005) proposed decisions as first-class architecture elements – emphasized capturing design rationale
- Helps to ground the architect's actions – why am I making this decision?
- Helps to clarify architect's limit of authority – who and what are impacted by this decision?

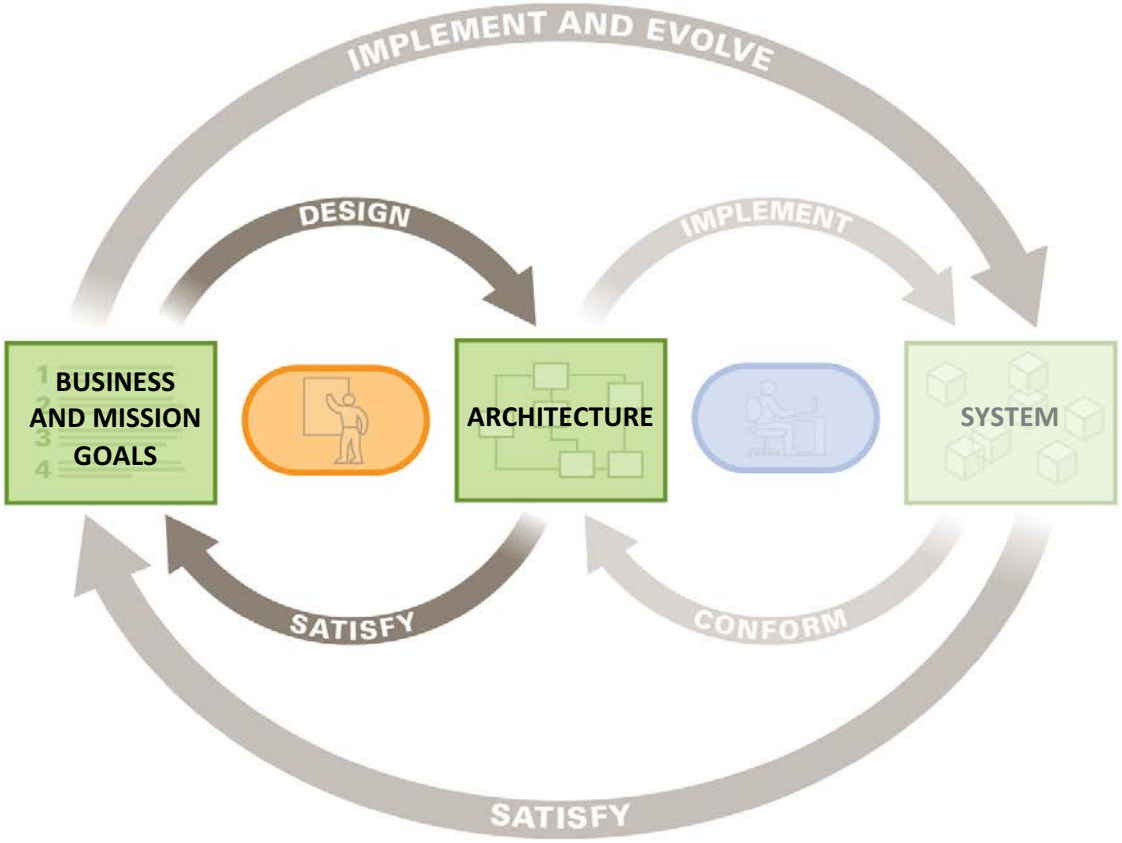
Architecture-Centric Engineering [1]



Architecture Centric Engineering 2



Architecture Centric Engineering 3



Siebel Systems
Sales Force Automation 1

Case Study

Business Goal (1993-1998)

- Laptop computers becoming affordable.
- Dial-up internet connectivity becoming ubiquitous – enable the new mobile sales workforce

Quality Attributes

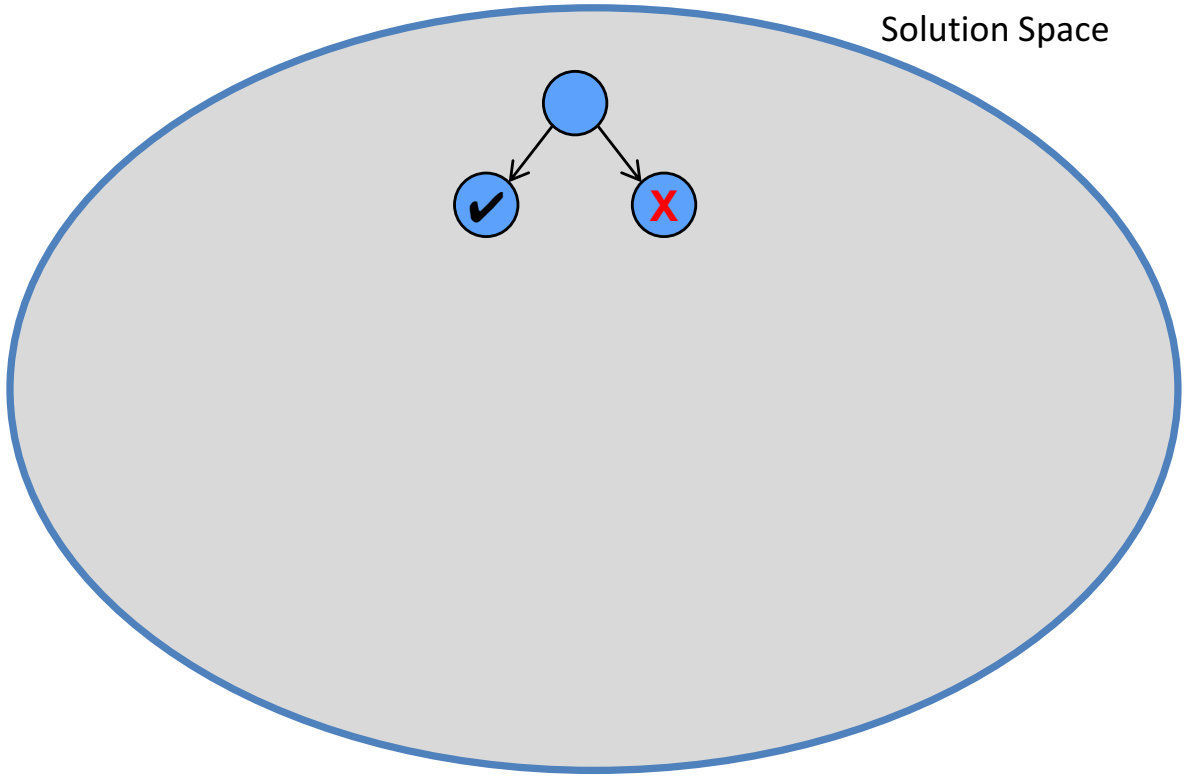
- Disconnected client operation
- Efficient bandwidth utilization (client connected via dial-up link)
- Customizability, migratability to new versions of the application

Architecture Solution

- Thick Client with local replica of partial repository
- Highly partitioned repository data model – each sales person replicates a personalized view with only the data needed
- Abstract, parameterized repository schema – easy to migrate
- Business logic customization through well defined interfaces at specific points in the business process — Had to use Siebel Tools to develop — Enabled automatic migration of customizations to new application versions.

Result – In 1999, Fortune Magazine named Siebel “the fastest growing company in the US”

“Architecture Manifests the Earliest Design Decisions”



Earliest Design Decisions 1

Architecture defines constraints on implementation

- The implementation must conform to prescribed design decisions such as those regarding
  - elements
  - interactions
  - behaviors
  - responsibilities
- The implementation must conform to resource allocation decisions such as those regarding
  - scheduling priorities and time budgets
  - shared data and repositories
  - queuing strategies

Architectures are both prescriptive and descriptive

Earliest Design Decisions 2

Architecture dictates the structure of the organization

- Architecture represents the highest level decomposition of a system and is used as a basis for
  - partitioning and assigning the work to be performed
  - formulating plans, schedules, and budgets
  - establishing communication channels among teams
  - establishing plans, procedures, and artifacts for configuration management, testing, integration, deployment, and maintenance

For managerial and business reasons, once established, an architecture becomes very difficult to change

Early Design Decisions 3

Architecture permits/precludes the achievement of a system's desired quality attributes. The strategies for achieving them are architectural

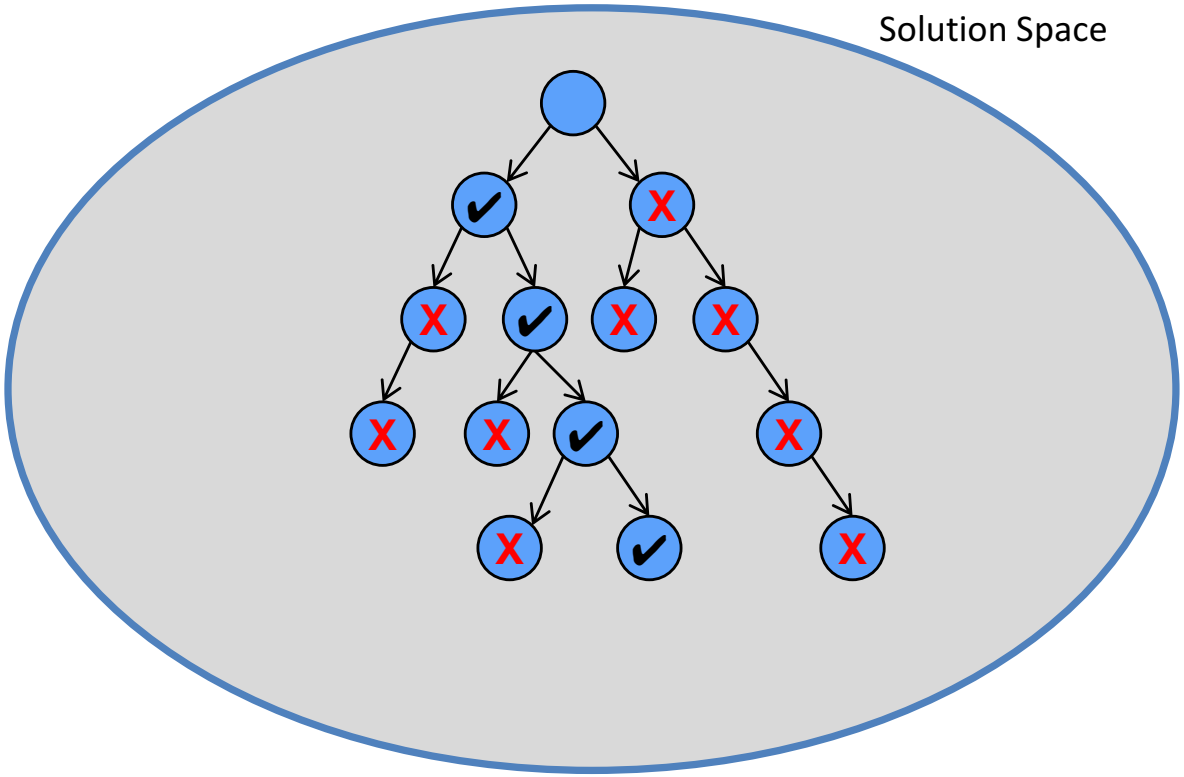
If you desire…	you need to pay attention to…
High performance	minimizing the frequency and volume of inter-element communication
Modifiability	limiting interactions between elements
Security	managing and protecting inter-element communication
Reusability	minimizing inter-element dependencies
Subsetability	controlling the dependencies between subsets and, in particular, avoiding circular dependencies
Availability	the properties and behaviors that elements must have and the mechanisms you will employ to address fault detection, fault prevention, and fault recovery
And so forth	…

Earliest Design Decisions 4

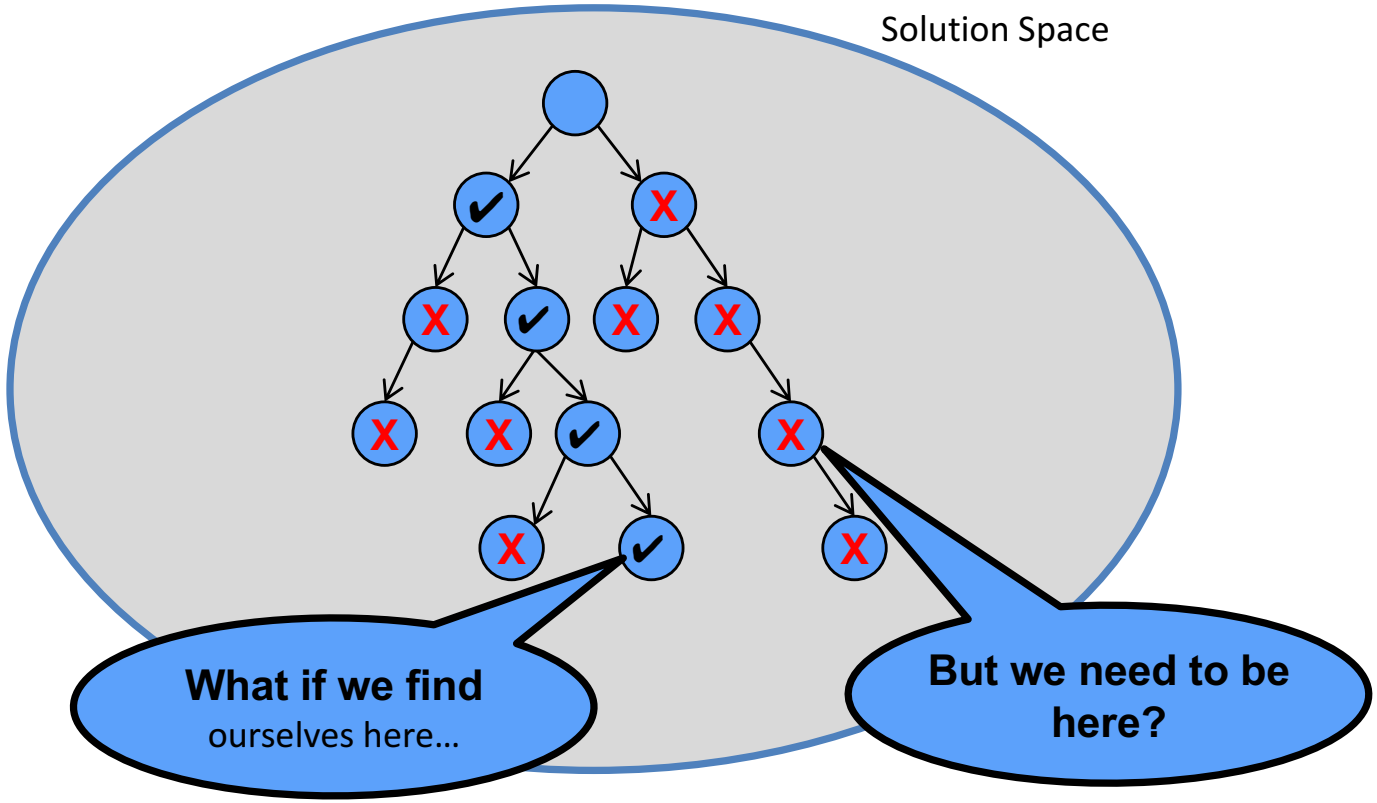
Architecture allows us to predict system quality attributes without waiting until the system is developed or deployed.

- Because architecture influences quality attributes in known ways, it follows that we can use architecture to predict how well quality attributes will be achieved
- We can analyze an architecture to evaluate how well it meets its quality attributes requirements

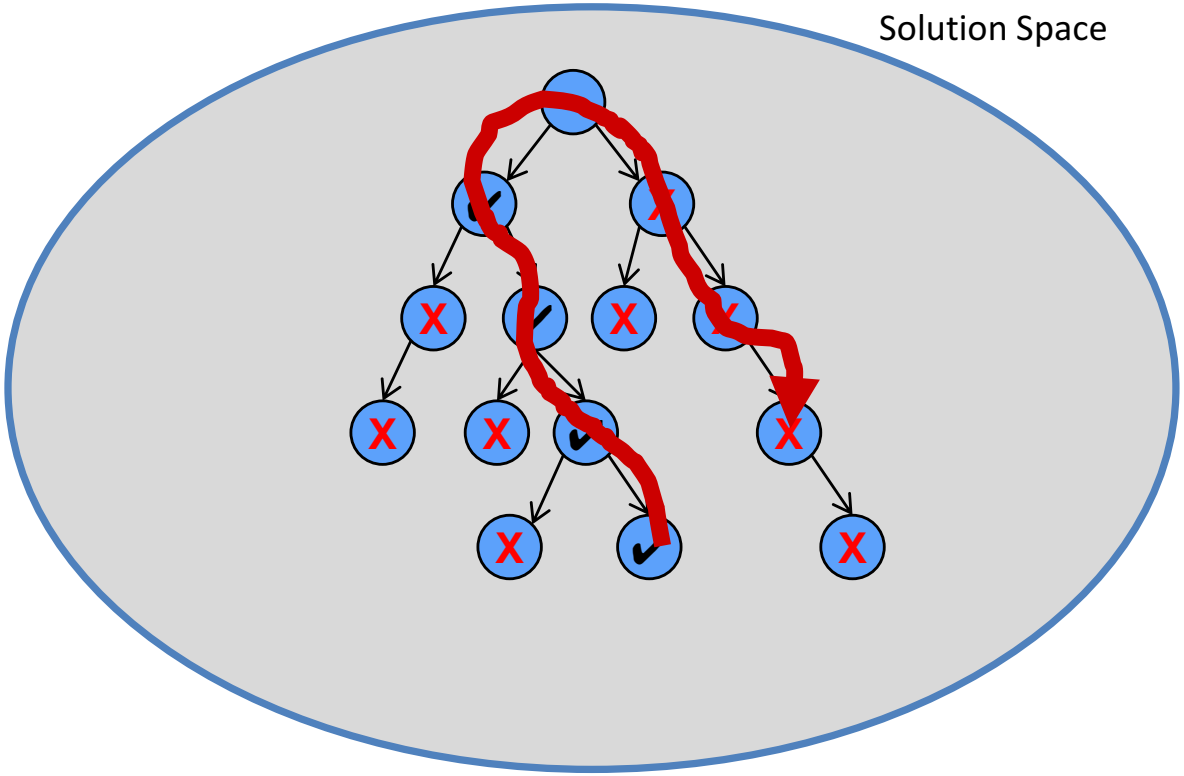
Decisions Limit Future Solution Options



Early Decisions Are the Most Important…



… Because They Are Expensive to Change



Siebel Systems
Sales Force Automation 2

Case Study

Changing business context (1998-2001)

- Dial-up → Broadband. First wired, then Wi-Fi, then cellular
- Thin clients – new IT best practice

Acquired Scopus Technology in 1998 for market share and thin client technology

- Lost focus on architecture as business grew through acquisition
- The architecture was unable to meet new quality attribute requirements:
  - "Always connected" clients caused repository performance to degrade
  - Thin client architecture did not separate customizations from common core, migration to new versions became a manual merging process

Result:

- By 2001, customers stopped upgrading, revenue growth stopped
- Salesforce.com founded in 1999 – early SaaS play
  - Grew to 393,000 subscribers and 50 million transactions/day by 2006
- Siebel acquired by Oracle in 2006

Summary

The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both

Architecture is important because it

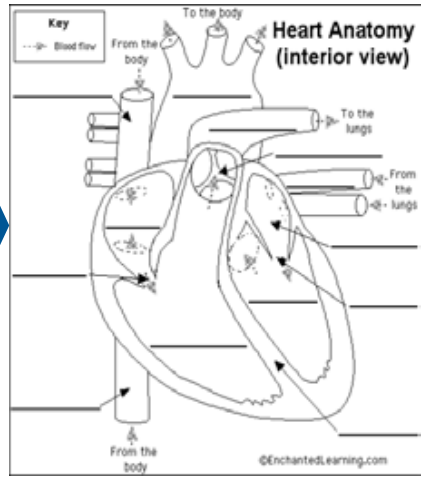
- is the result of the earliest design decisions
- provides a communication vehicle among stakeholders
- is a transferable, reusable abstraction of a system

Architecture 101

Architecture Structures and Separation of Concerns

Architectural Structures 1

A human body comprises multiple *structures*.



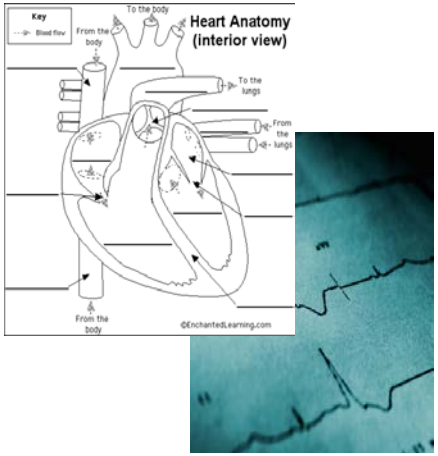
a *static view* of one human *structure*

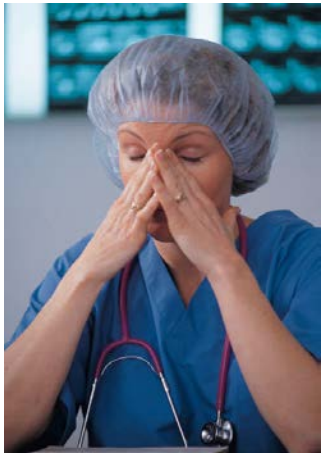
a *dynamic view* of that *structure*

One body has many structures. So it is with software…

Architectural Structures 2

These views are needed by a cardiologist…





…but they won't do for an orthopedist.

Different stakeholders are interested in different structures.

So it is with software…

Architectural Structures 3

Modern software systems are too complex to grasp all at once. At any moment, we restrict our attention to a small number of a software system's structures

To communicate meaningfully about an architecture, we must make it clear which structure or structures we are discussing

Architectural Structures 4

Architectural structures for software systems can be divided into three types

- **Module structures**: consisting of elements that are units of implementation called modules and the relationships among them.
- **Component-and-Connector structures**: consisting of runtime components (units of computation) and the connectors (communication paths) between them
- **Allocation structures**: consisting of software elements and their relationships to elements in external environments in which the software is created and executed

Examples of Module Structures

Decomposition structure: consisting of modules that are related via the "is a submodule of" relation

Uses structure: consisting of modules that are related via the "uses" relation (i.e., one module uses the services provided by another module)

Layered structure: consisting of modules that are partitioned into groups of related and coherent functionality. Each group represents one layer in the overall structure.

Class/Generalization structure: consisting of modules called classes that are related via the "inherits from" or "is an instance of" relations

Examples of Component-and-Connector Structures

Process structure: consisting of processes or threads that are connected by communication, synchronization, and/or exclusion operations

Concurrency structure: consisting of components and connectors where connectors represent “logical threads”

Shared-Data (repository) structure: consisting of components and connectors that create, store, and access persistent data

Client-Server structure: consisting of cooperating clients and servers and the connectors between them (i.e., the protocols and messages they share)

Examples of Allocation Structures

Deployment structure: consisting of software elements and their allocation to hardware and communication elements

Implementation structure: consisting of software elements and their mapping to file structures in the development, integration, and configuration control environments

Work Assignment structure: consisting of modules and how they are assigned to the development teams responsible for implementing and integrating them

What Makes a “Good” Architecture?

There is no such thing as an inherently good or bad architecture.

Architectures are either more or less fit for some purpose.

Architectures can be evaluated but only in the context of specific stated goals.

There are, however, good rules of thumb.

Process “Rules of Thumb”

The architecture should be the product of a single architect or a small group of architects with an identified technical leader.

- This approach leads to conceptual integrity and technical consistency.
- This recommendation holds for Agile and open source projects as well as “traditional” ones.
- There should be a strong connection between the architect(s) and the development team.

The architect (or architecture team) should base the architecture on a prioritized list of well-specified quality attribute requirements.

The architecture should be documented using views which address the concerns of the important stakeholders.

The architecture should be evaluated for its ability to deliver the system’s important quality attributes.

- This should occur early in the lifecycle and repeated as appropriate.

The architecture should lend itself to incremental implementation,

- Create a “skeletal” system in which the communication paths are exercised but which at first has minimal functionality.

Structural "Rules of Thumb" 1

The architecture should feature well-defined modules whose functional responsibilities are assigned on the principles of information hiding and separation of concerns.

- The information-hiding modules should encapsulate things likely to change.
- Each module should have a well-defined interface that encapsulates or "hides" the changeable aspects from other software.

Unless your requirements are unprecedented your quality attributes should be achieved using well-known architectural patterns and tactics specific to each attribute.

The architecture should never depend on a particular version of a commercial product or tool. If it must, it should be structured so that changing to a different version is straightforward and inexpensive.

Modules that produce data should be separate from modules that consume data.

Structural "Rules of Thumb" 2

Don't expect a one-to-one correspondence between modules and components.

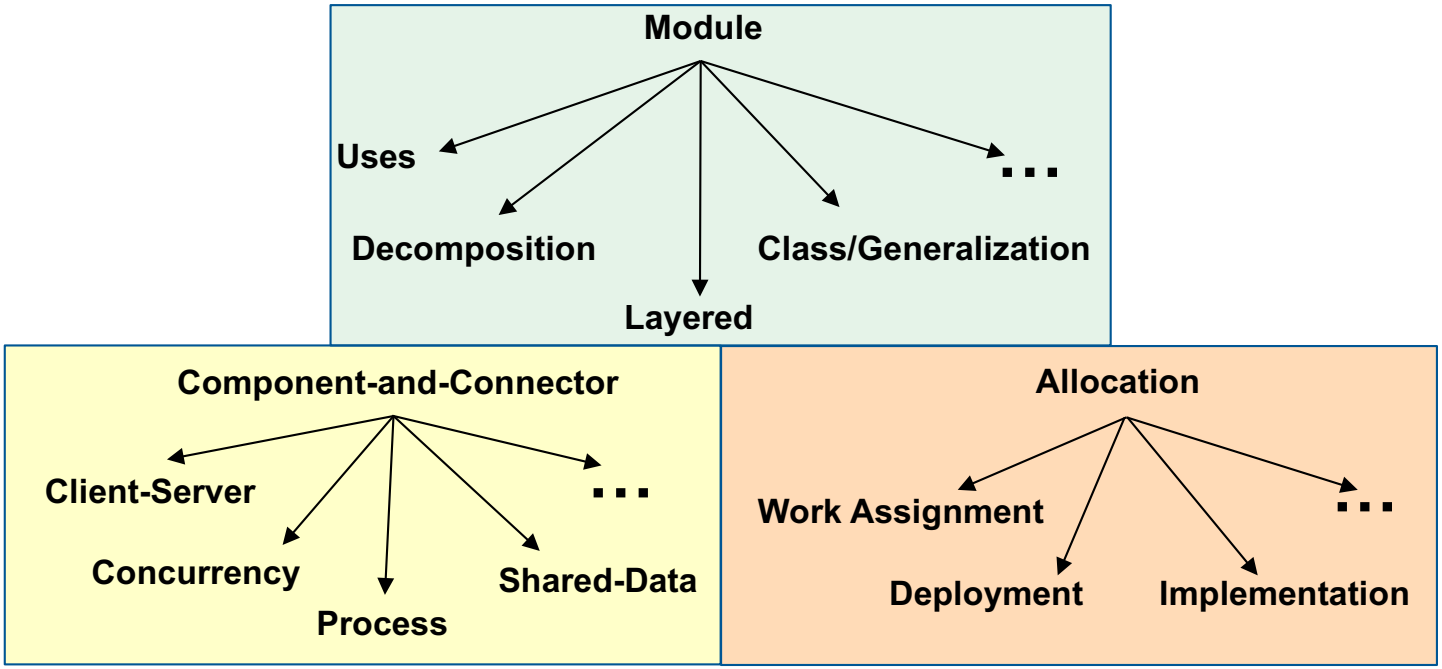
Every process should be written so that its assignment to a specific processor can be easily changed, perhaps even at runtime.

The architecture should feature a small number of ways for components to interact.

- The system should do the same things in the same way throughout.
- This will aid in understandability, reduce development time, increase reliability, and enhance modifiability.

The architecture should contain a specific (and small) set of resource contention areas, the resolution of which is clearly specified and maintained.

Summary



Architects must focus on whatever structures will provide them with the most leverage in achieving the desired quality attributes of a system

Software Architecture 101

Architecture Evaluation

Check Your Understanding: Critique A Design

Consider a company that builds hardware-based field systems for controlling building functions such as heating, ventilation, air conditioning, access, and safety

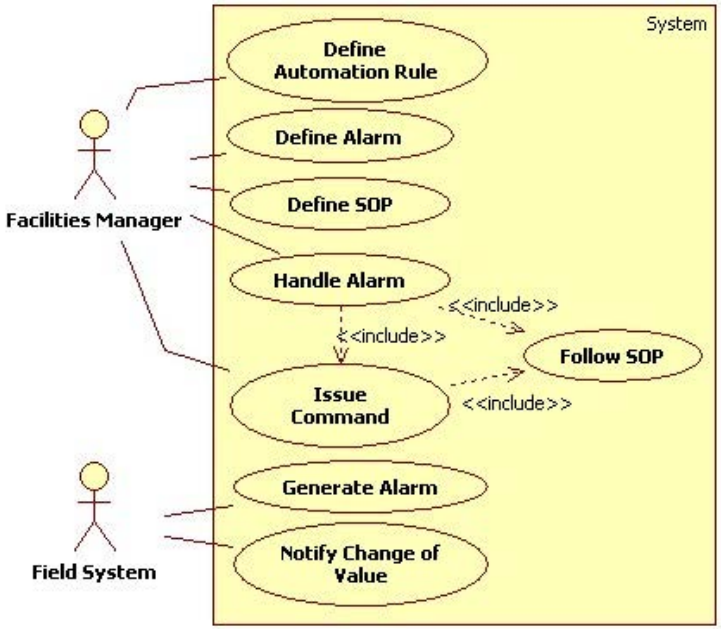
The company wants to develop a software-based system for facilities managers. The system would perform these functions:

- manage a network of hardware-based field systems
- issue commands to configure the field systems
- define rules based on property values that trigger commands
- for life-critical situations, trigger alarms notifying appropriate users

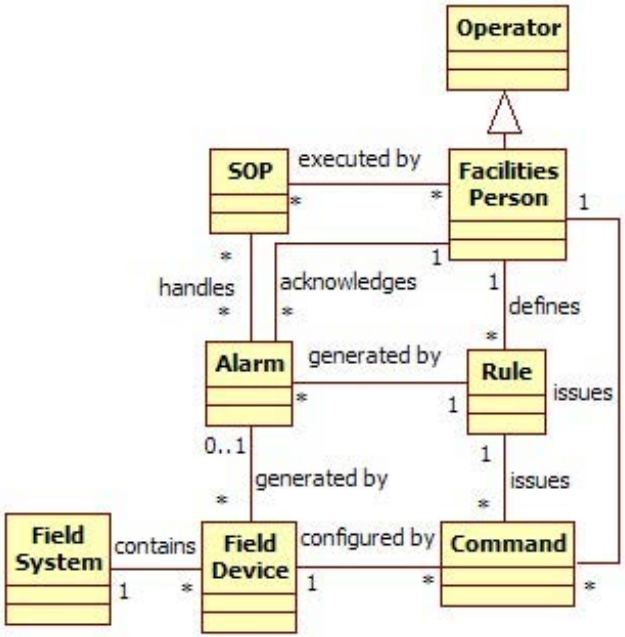
The company wants to offer this product in new markets, and expand its sales by allowing value-added resellers to sell the system under their own brands, supporting field systems from manufacturers of their choice

Check Your Understanding: Design 1

Object-oriented analysis and design artifacts showing (a) use cases (b) business domain model



(a)



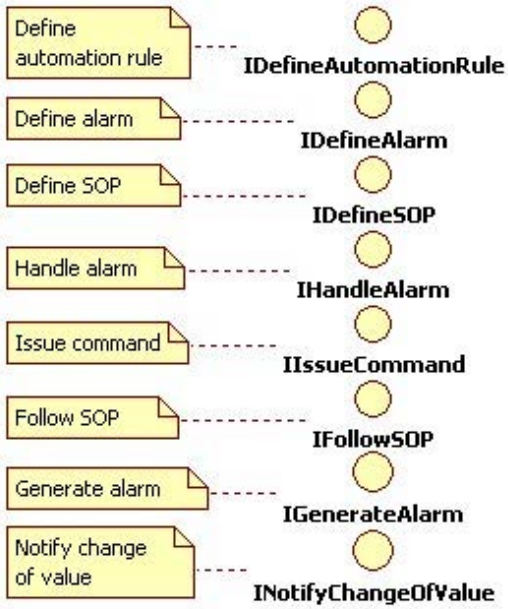
(b)

Key: UML

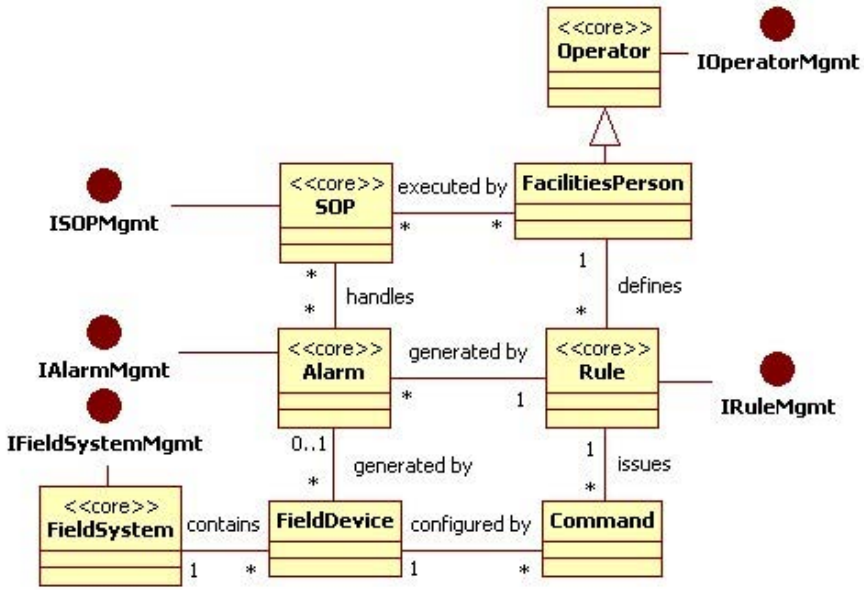
Source: Ipek Ozkaya, Len Bass, Raghvinder Sangwan and Robert Nord. "Making Practical Use of Quality Attribute Information." IEEE Software special issue on Software Quality Requirements, March/April 2008.

Check Your Understanding: Design 2

Object-oriented analysis and design artifacts showing (c) system interfaces and (d) business interfaces



(c)

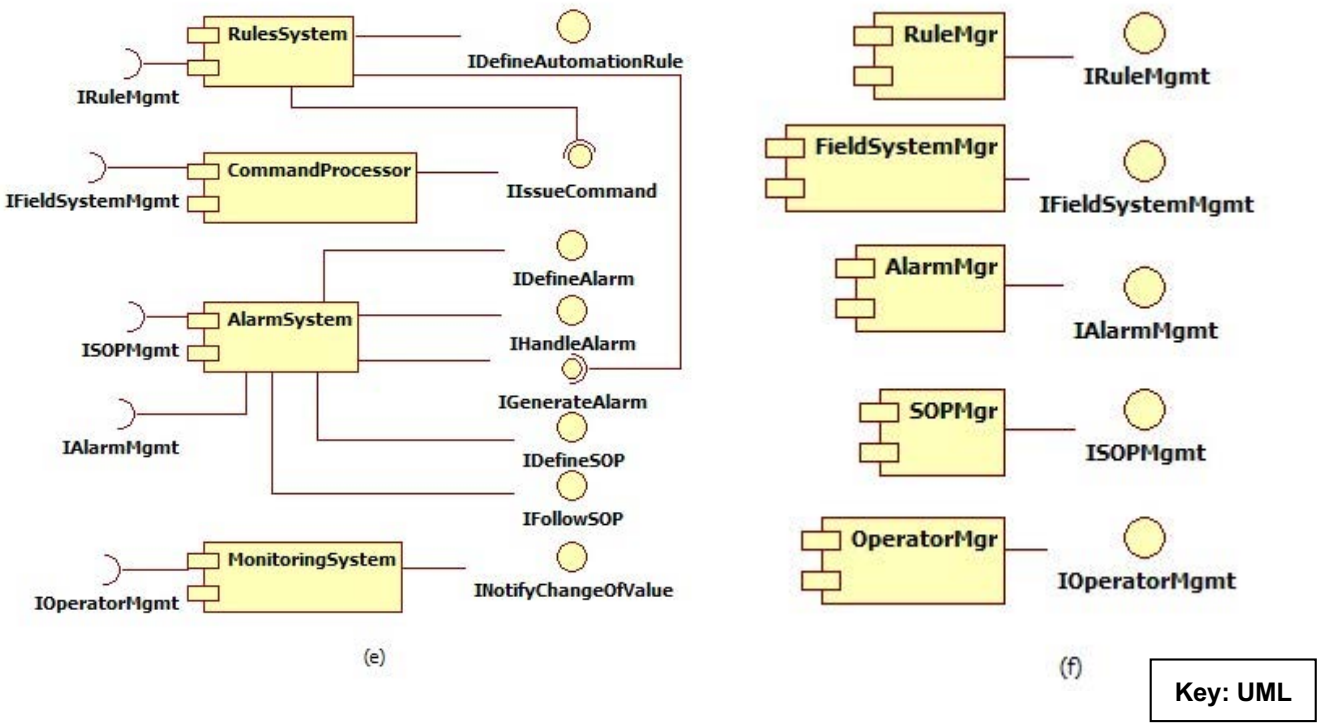


(d)

Key: UML

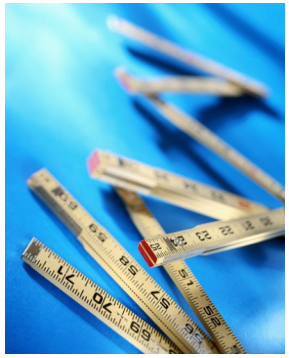
Check Your Understanding: Design 3

Object-oriented analysis and design artifacts showing (e) system components and (f) business components



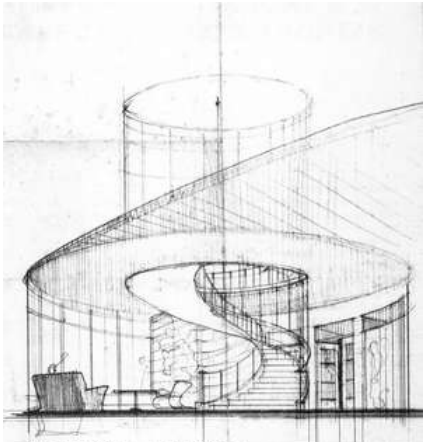
? What do you think of this design?

Things to Establish for Architecture Evaluation



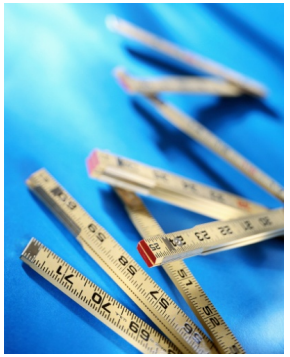
An objective measure

An understanding of the architecture



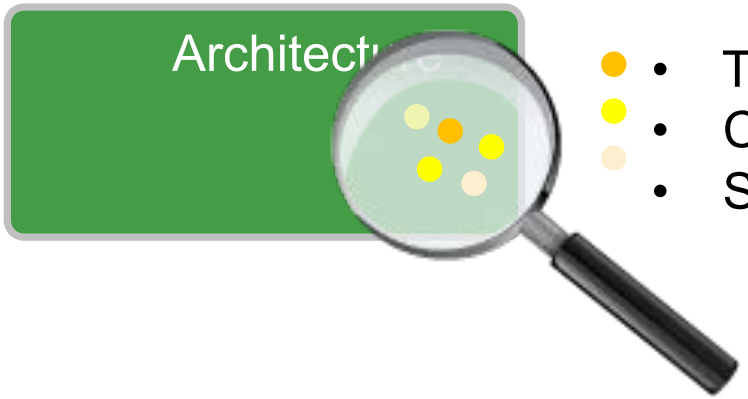
The analysis

An Objective Measure: Building the Yardstick



Without a yardstick to measure, every architecture is good or bad

Will the designed system solution have the properties to make the organization successful?

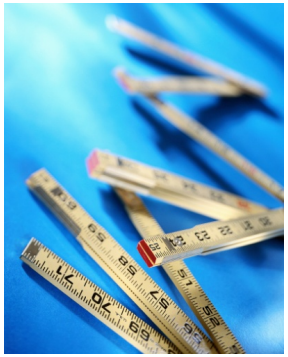


- Timing
- Cost of change
- Sensitive data

System properties of interest are the quality attribute properties of the functions that the system has.

Stakeholders

BUSINESS GOALS

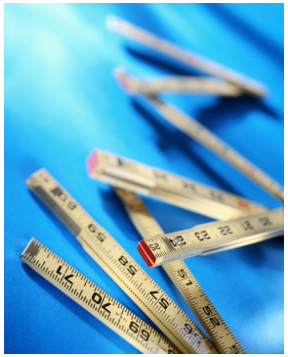


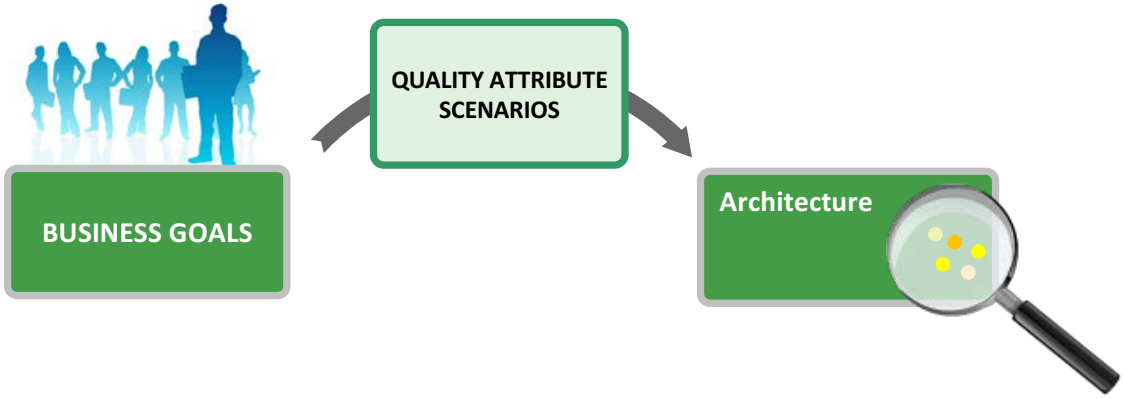
Only the stakeholders of an organization can specify what the expected important quality attributes are

- *We need to have a system that can easily be adapted to new market conditions*
- *Our systems cannot have any defects in the field*
- *We guarantee 24/7 availability*

These statements (business goals) describe how the organization plan to be successful with their business

Quality Attribute Scenarios



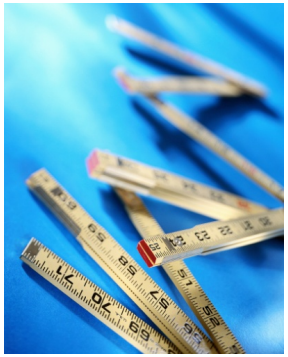


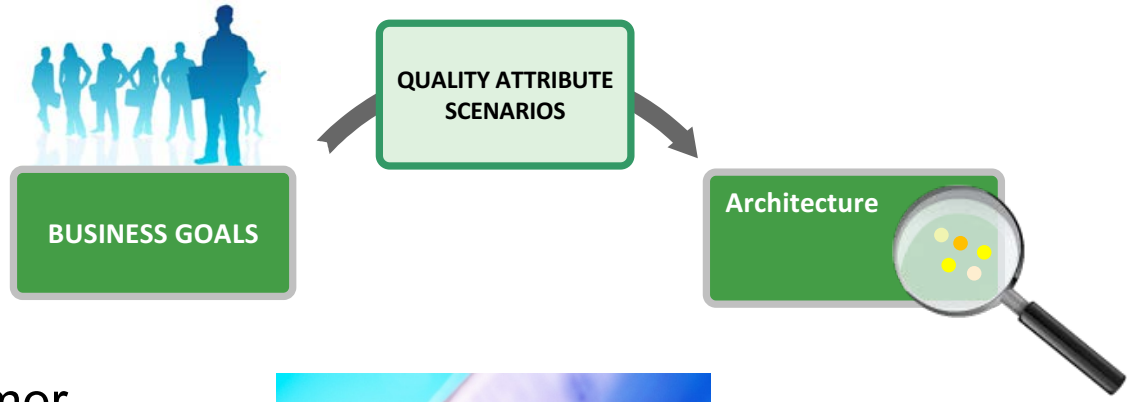
In most cases business goals are too vaguely defined to actually be able to measure their achievement

Quality attribute scenarios bridge business goals and architecture quality attribute properties

They describe what quality attribute properties the system is expected to possess to run a successful business

The Yardstick





A customer requires a new auction algorithm. A developer implements and integrates that function within one day of effort.

An error occurs in a fielded system. The system recovers from the error without manual intervention within 5 seconds.

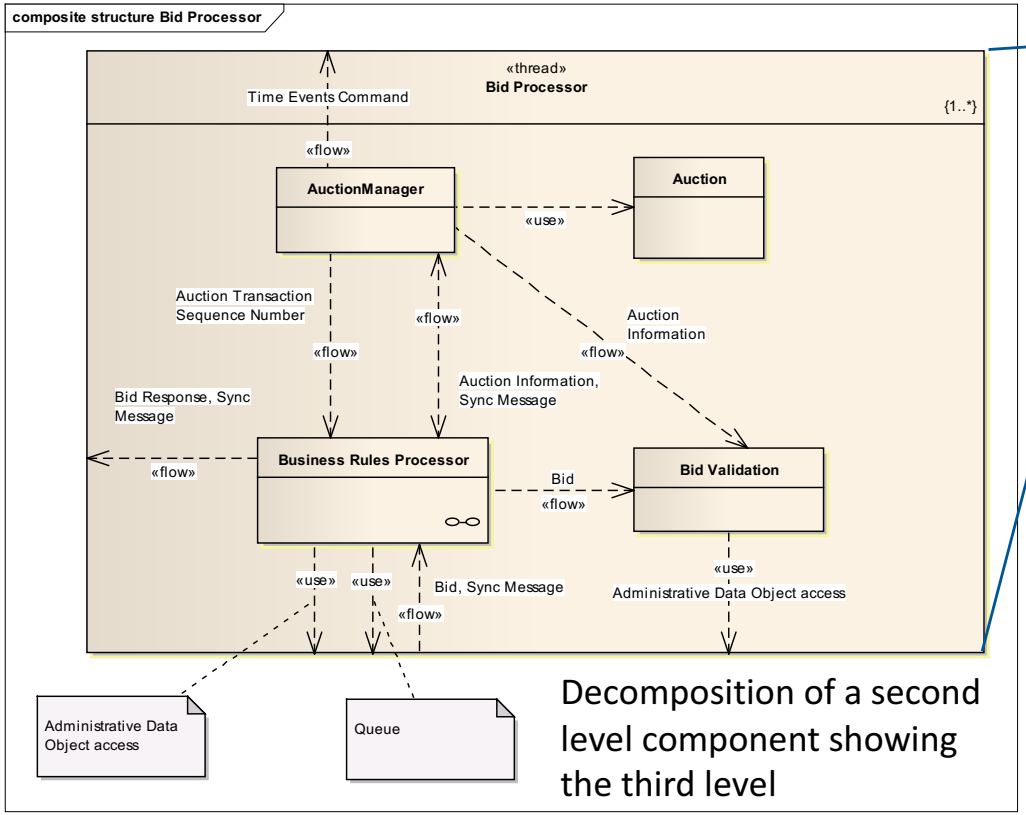
The Yardstick: Summary

Principle 1: Important quality attribute properties of the architecture need to be evaluated

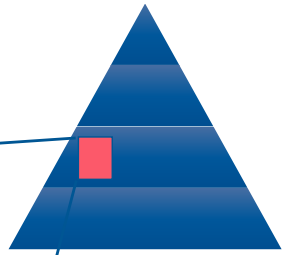
Principle 2: What is important is derived from the business goals

Principle 3: Quality attribute scenarios translate business goals into required quality attribute properties

An Understanding of the Architecture: Example Component Diagram



Decomposition of a second level component showing the third level



A typical piece of architecture documentation

What are the properties here?

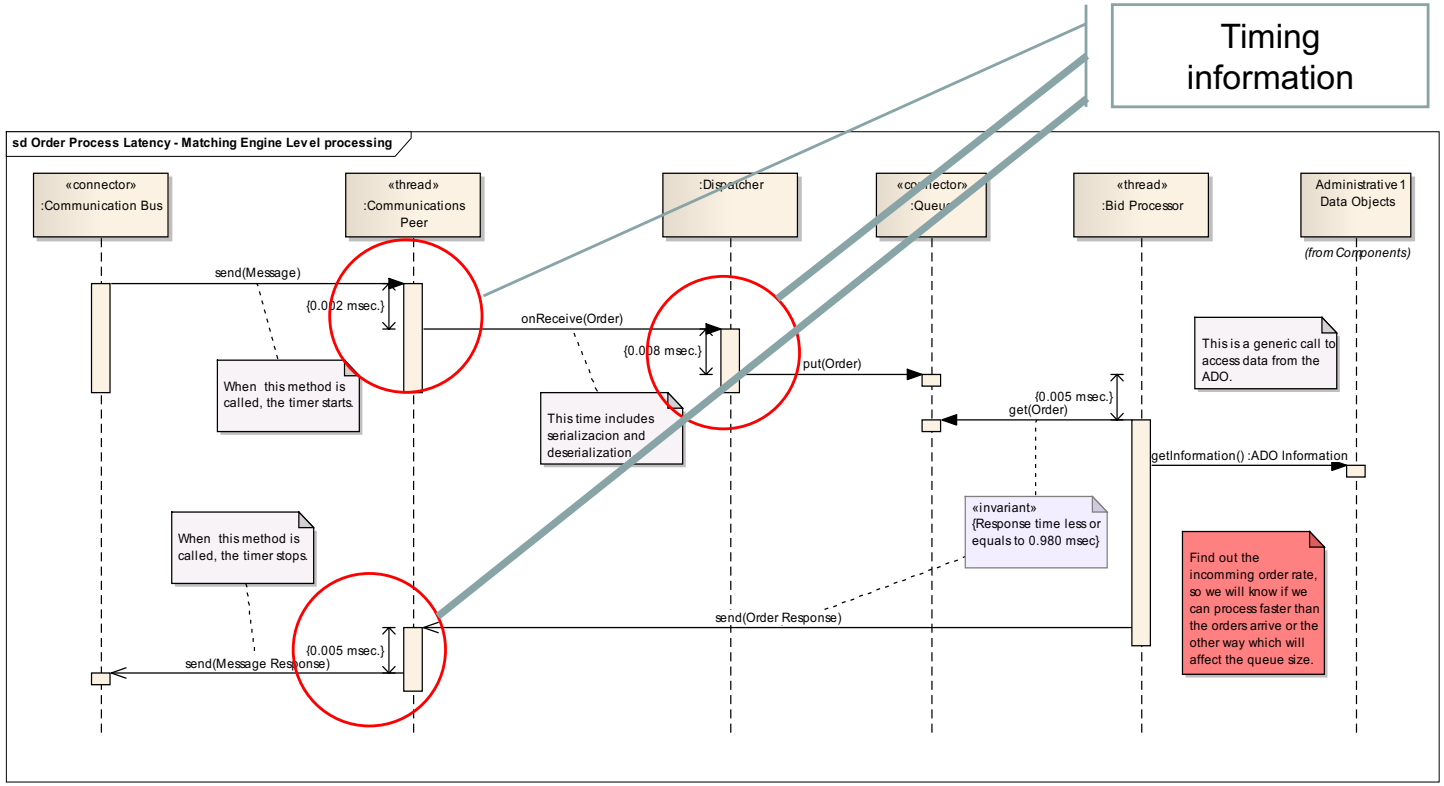
Where are they?

What can we say about

- Performance?
- Modifiability?
- Security?

Sequence Diagram with Timing

An even better piece of architecture documentation



Architecture Approaches 1

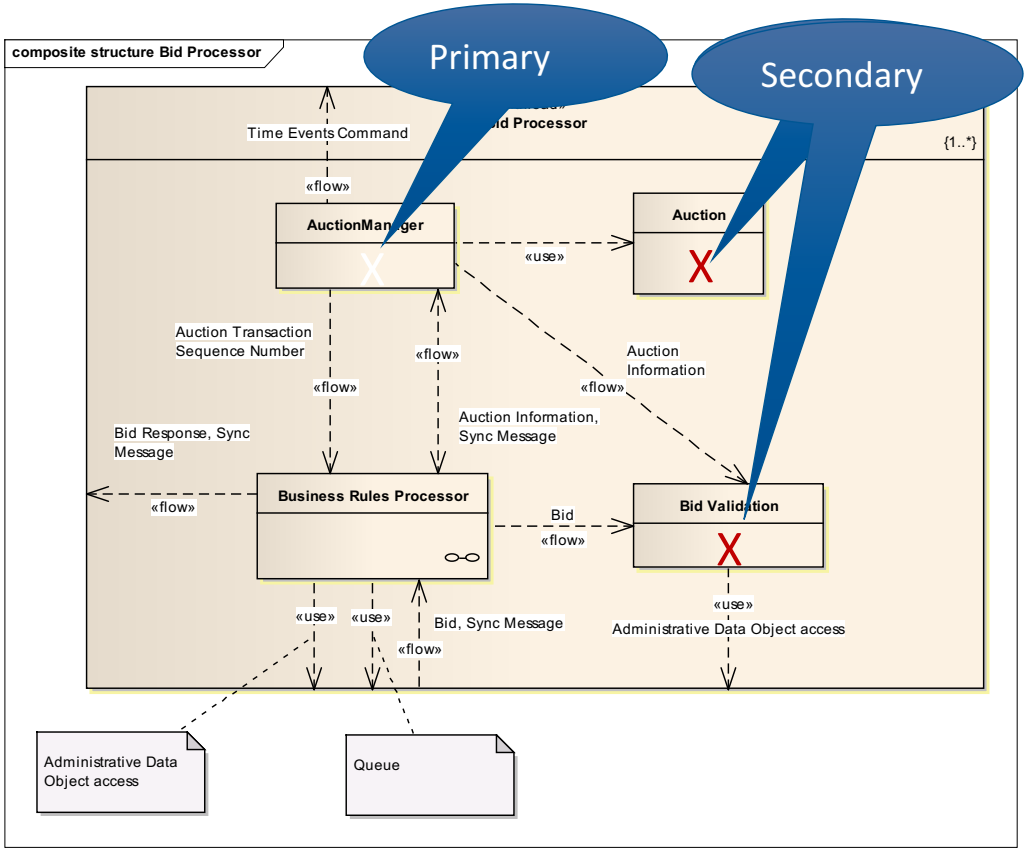
Narrow down the problem by focusing on a quality attribute scenario

A customer requires a new auction algorithm. A developer implements and integrates that function within one day of effort

First question to ask

- What are the components involved in this scenario?

Example Component Diagram



Architecture Approaches 2

Second question to ask:

- What are concepts put in place to make this change easy?
  - *Localize everything that needs to be changed in one component*
  - *Data structures have a generic interface that hides internal changes*
  - *Component also has a versioned interface to allow backward compatibility*

- These are architecture approaches to support this extensibility scenario

Architect vs. Documentation

An architecture documentation that can be used for evaluation needs to show where those concepts are and what their properties are

- Adding a new algorithm means specializing the generic class “AuctionManager”. A typical algorithm can be implemented in four hours
- New data structures can be added, but existing data structures cannot be changed without any impact on existing functions. Very seldom new structures are required. If so, they can be added within one hour
- Every component working with the AuctionManager requests an interface of a specific version. Creating a new version of an interface will take about one day

If this information is not in the architecture document then the answers have to come from the architect.

Side Effects

Every architecture approach used also has negative impact on other quality attributes

- Putting everything that needs to change in one place may introduce unnecessary dependencies to other components. Bad for security and other types of changes
- Data structures with generic interfaces may impose a performance penalty
- Versioned interfaces increase complexity, which is more difficult to test and the change for system crashes increases

The architect needs to be aware of these issues, needs to put mitigations into place, and document them

Understanding the Architecture: Summary

Principle 4: Identify relevant components by using scenarios

Principle 5: Identify architecture approaches with their quality attribute properties

Principle 6: Identify the side effects of those architecture approaches

The Analysis: Matchmaking

The architecture evaluation process has to answer the questions

- Do the quality attribute properties of the identified components support the given quality attribute scenario sufficiently?
- Is the negative impact of the chosen architecture approaches on other quality attribute scenarios acceptable?

If the answers are documented, then examining the documentation is sufficient for the evaluation

If not, the architect has to provide the answers

The evidence provided must be sufficient to convince quality attribute experts and stakeholders

- In many cases will require experimentation

Risks

No system design is without any risks

Successful organizations understand and handle the risks

An architectural risk is when

- The system properties do not completely satisfy the given quality attribute scenarios
- The approaches used for one scenario negatively impact other scenarios

If an important scenario cannot be supported, then one or more business goals are impacted

Analysis: Summary

Principle 7: The architect provides evidence that the system's quality attribute properties will indeed fulfill the scenarios

Principle 8: Mismatches between architecture properties and scenarios become risks to the business goals

The ATAM

The SEI developed the Architecture Tradeoff Analysis Method (ATAM) and has applied it to architectures for systems of wide-ranging sizes and domains

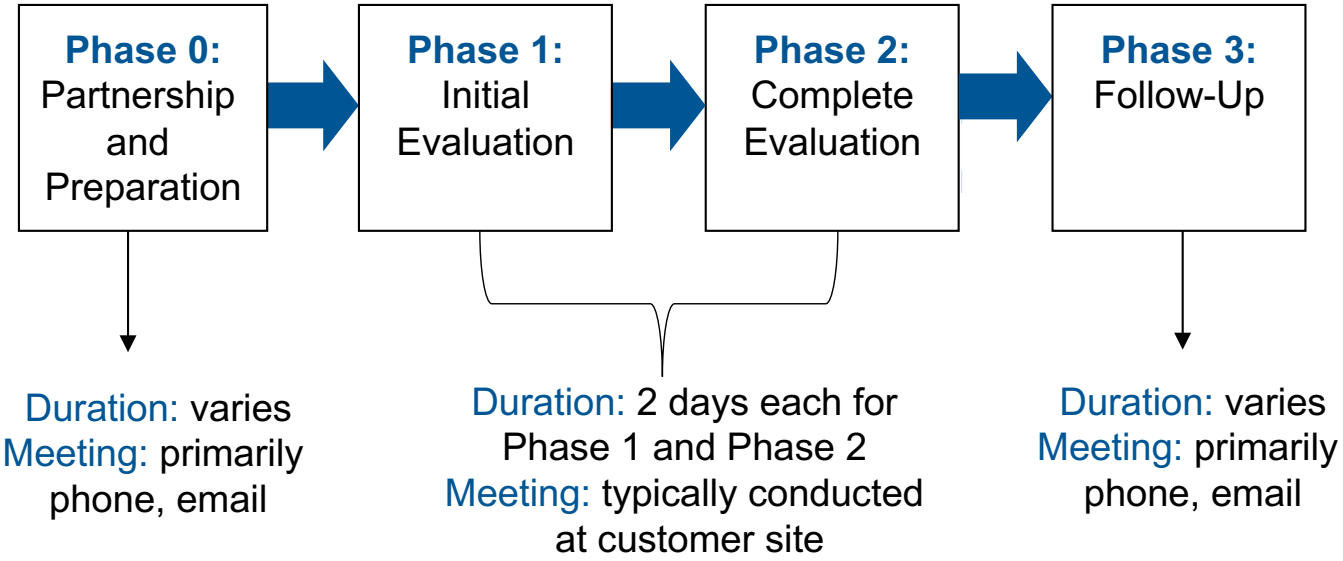
The purpose of the ATAM is to assess the consequences of architectural decisions in light of quality attribute requirements and business goals

The ATAM brings together three groups in an evaluation:

1. a trained evaluation team
2. an architecture's "decision makers" (architect, senior designers, project managers, customers)
3. representatives of the architecture's stakeholders

ATAM Phases

ATAM evaluations are conducted in four phases



ATAM Phase 1

Phase 1 involves a small group of predominantly technically oriented stakeholders

Phase 1 is

- architecture-centric
- focused on eliciting detailed architectural information and analyzing it
- a top-down analysis

ATAM Phase 1 Steps

1. Present the ATAM
2. Present business drivers
3. Present architecture
4. Identify architectural approaches
5. Generate quality attribute utility tree
6. Analyze architectural approaches
7. Brainstorm and prioritize scenarios
8. Analyze architectural approaches
9. Present results

Phase 1

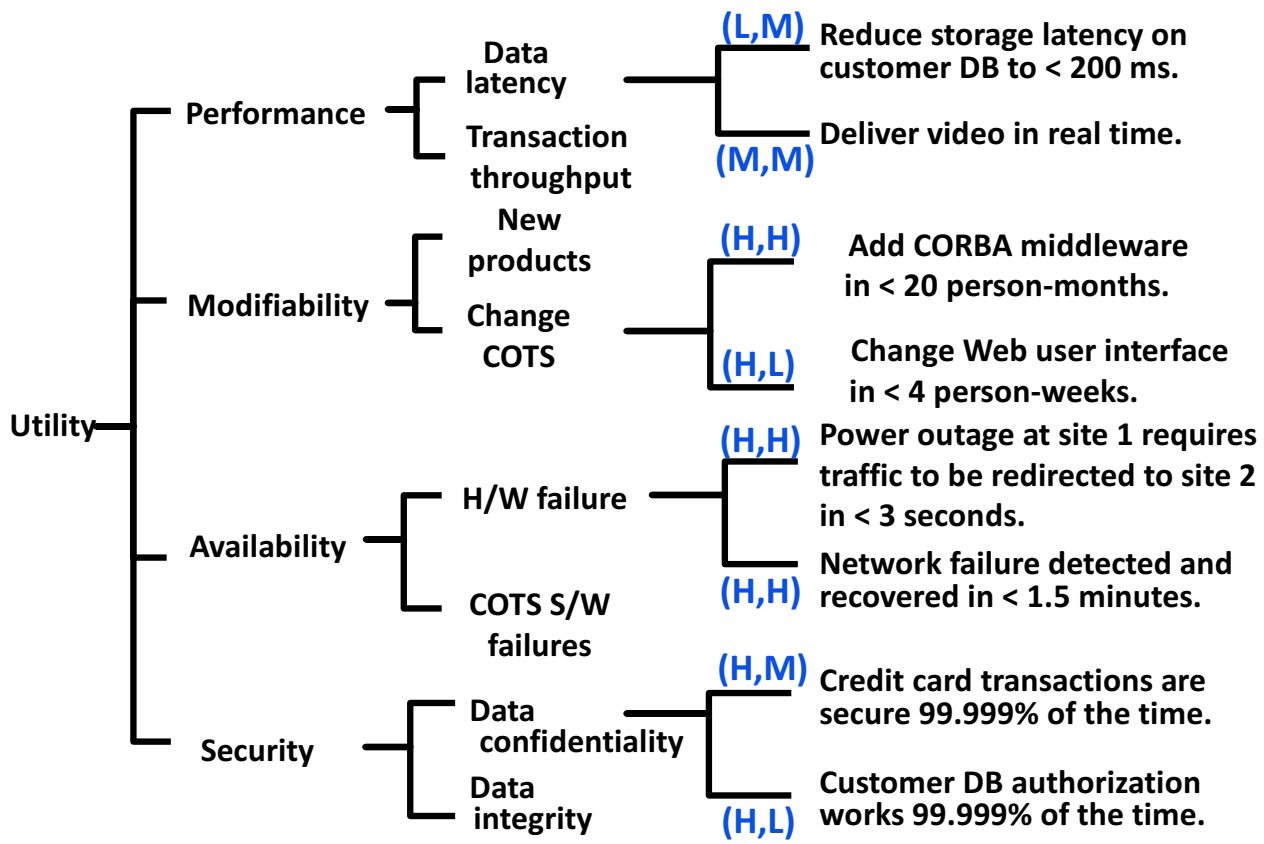
What Are Quality Attribute Utility Trees?

You can identify, prioritize, and refine the most important quality attribute goals by building a utility tree.

- A utility tree is a top-down vehicle for characterizing the “driving” attribute-specific requirements.
- The highest level nodes are typically quality attributes such as performance, modifiability, security, availability, and so forth.
- Scenarios are the leaves of the utility tree.

The utility tree is a characterization and a prioritization of specific quality attribute requirements.

Example of Quality Attribute Utility Tree



L = Low, M = Medium, H = High

How Scenarios Are Used 1

For design purposes, we use six-part scenarios

1. **Source**: an entity that generates a stimulus
2. **Stimulus:** a condition that affects the system
3. **Artifact(s):** the part of the system that was stimulated by the stimulus
4. **Environment:** the condition under which the stimulus occurred
5. **Response:** the activity that results because of the stimulus
6. **Response Measure:** the measure by which the system's response will be evaluated

How Scenarios Are Used 2

Scenarios are used to

- represent stakeholders' interests
- understand quality attribute requirements

Scenarios should cover a range of

- anticipated uses of the system (use case scenarios)
- anticipated changes to the system (growth scenarios)
- unanticipated stresses on the system (exploratory scenarios)

Scenarios are linked to business goals, for traceability

A good scenario clearly states the stimulus and the responses of interest

Examples of Scenarios

Use case scenario

- A remote user requests a database report via the Web during a peak period and receives it within 5 seconds

Growth scenario

- During maintenance, add an additional data server within 1 person-week

Exploratory scenario

- Half of the servers go down during normal operation without affecting the overall system availability

Scenarios should be as specific as possible

Stimuli, Environment, Responses

Use case scenario

- The remote user requests a database report via the Web during a peak period and receives it within 5 seconds

Growth scenario

- During maintenance, add an additional new data server within 1 person-week

Exploratory scenario

- Half of the servers go down during normal operation without affecting the overall system availability

Scenario Analysis Outputs

As each scenario is analyzed against the architecture, the evaluation team identifies risks.

A risk is a potentially problematic architectural decision.

- *"Rules for writing business logic modules in the second tier of your three-tier architecture are not articulated clearly. This could result in the replication of functionality, thereby compromising the modifiability of the third tier."*

ATAM Phase 2

Phase 2 involves a larger group of stakeholders.

Phase 2 is

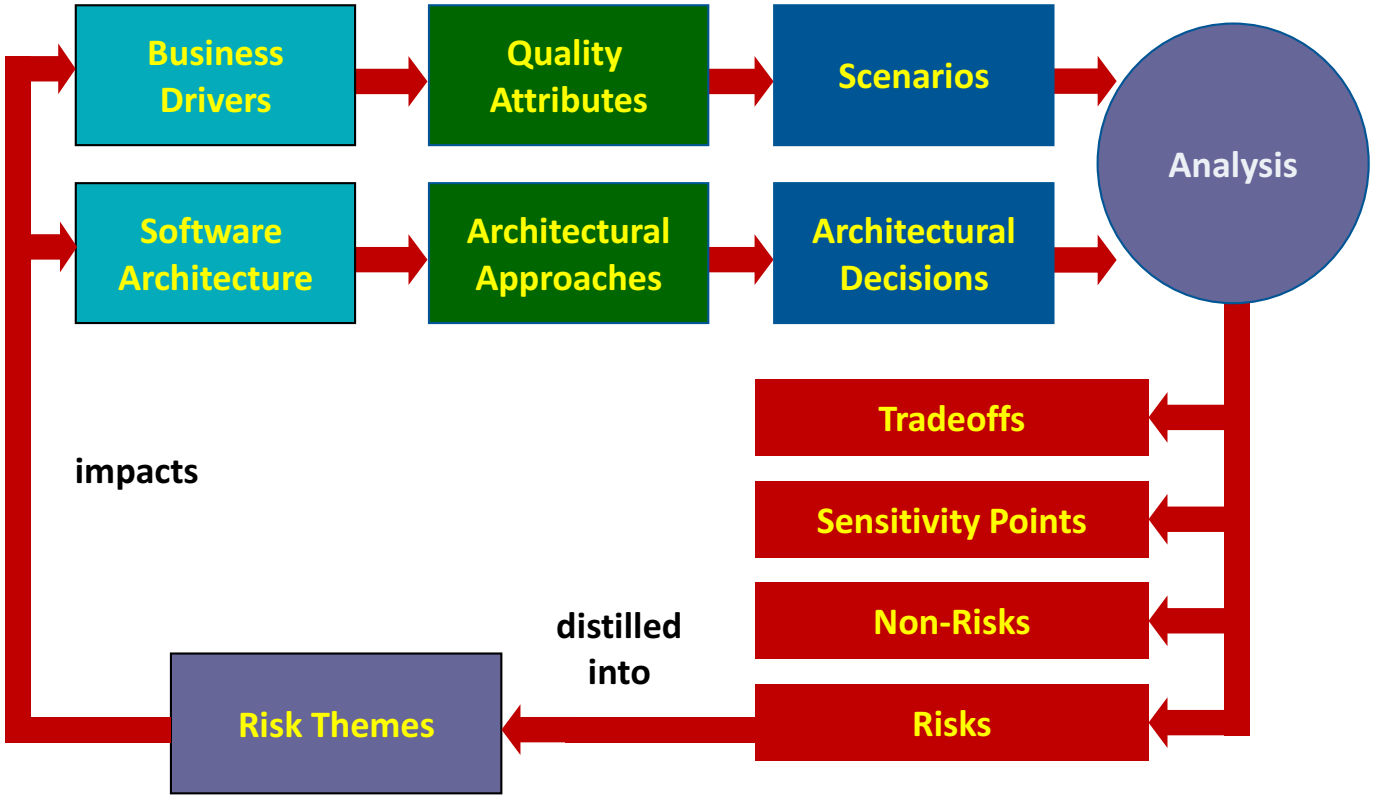
- stakeholder-centric
- focused on eliciting diverse stakeholders' points of view and on verifying the results of Phase 1
- bottom-up analysis

ATAM Phase 2 Steps

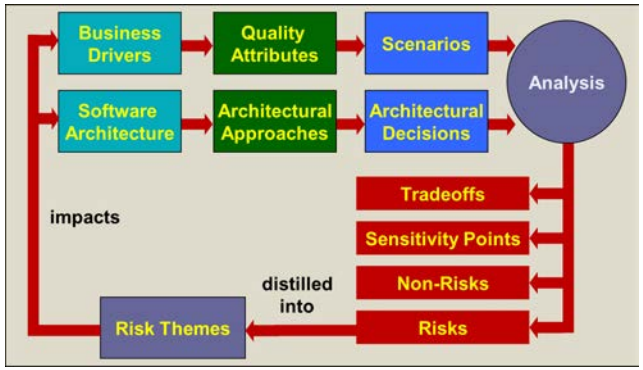
1. Present the ATAM.
2. Present business drivers.
3. Present architecture.
4. Identify architectural approaches.
5. Generate quality attribute utility tree.
6. Analyze architectural approaches.
7. Brainstorm and prioritize scenarios
8. Analyze architectural approaches
9. Present results

Phase 2

Conceptual Flow of the ATAM



Principles Apply to ATAM



Principle 1: Important quality attribute properties of the architecture need to be evaluated.

Principle 2: What is important is derived from the business goals.

Principle 3: Quality attribute scenarios translate business goals into requires quality attribute properties.

Principle 4: Identify relevant components by using scenarios.

Principle 5: Identify architecture approaches with their quality attribute properties

Principle 6: Identify the side effects of those architecture approaches.

Principle 7: The architect provides evidence that the system's quality attribute properties will indeed fulfill the scenarios.

Principle 8: Mismatches between architecture properties and scenarios become risks to the business goals.

Software Architecture 101

Technical Debt

Essential Software Development Artifacts

	Visible	Invisible
Positive Value	New features and added functionality	Architectural, structural features
Negative Value	Defects	Technical Debt

Source: Kruchten, P. Nord, R.L., Ozkaya, I. 2012. Technical Debt: From Metaphor to Theory and Practice, IEEE Software, 29(6), Nov/Dec 2012.

What Is Technical Debt?

We define technical debt as a software design issue that

- exists in a system artifact, such as code, build scripts, automated test suites, data;
- is traced to several locations in the system, implying ripple effects if changes are made;
- has a quantifiable effect on system attributes of interest to developers, such as increasing number of defects, negative change in maintainability, and code-quality indicators.

The Debt Aware Organization – Find Where the Rubber Hits the Road

"We have a model-view controller framework. Over time we violated the simple rules of this framework and had to retrofit later many functionalities"

Modifiability violation, pattern conformance

"There were two modules highly coupled that should have been designed for from the beginning"

Modifiability violation, pattern conformance

"A simple API call turned into a nightmare <due to not following guidelines>"

Framework, pattern conformance

Test Your Understanding: Is this an actionable description of technical debt?

A training system, meeting the customers high-level requirements but not meeting the expected functionality of the end users of the system.

- Cause: poor requirements gathering
- Staff shortage, lack of software engineering experience, customer reps lack of understanding and outdated methods of getting from requirements to a design (text documents, no UML or similar model)
- Impact has been a huge re-design, many areas of functionality being altered in the way they work

No, because

- Many issues identified hint at reckless and inadvertent practices, organizational approaches are needed as well as system re-design
- Concrete system issue is not clear

Test Your Understanding: Is this an actionable description of technical debt?

One of two modules was upgraded

- Unfortunately, the second module required months of unplanned work, due to high coupling between the modules
- Developers disregarded the scoping rules, due to time / schedule / budget, which let to module coupling
- Impact: 12 KSLOC of unplanned work

Yes, because

- The areas affected in the system are identified
- It is possible to start to concretely understand the impact
- The analysis issues are identified

Technical Debt Item Template

Name	Shorthand Designation
Development Approach	Development artifact and rationale: Element of the system or the supporting work-products: design, code, documentation, tests, etc.
Symptoms	An observable qualitative or measurable consequence
Consequences	The effect on the value, quality or cost of the current or the future state of the system in the form of: • accumulation: additional costs due to reduced productivity, induced defects or loss of quality • remediation: the cost it would take now to develop a different or “better” solution
Analysis	The degree to which the development artifact meets stakeholder needs or expectations

Summary 1

Architecture is important because it is the result of the earliest design decisions.

An architecture is composed of many structures, each of which comprises software elements and their relationships.

- Each structure provides engineering leverage and insight on different quality attributes.
- Architects select those structures that help to achieve the desired quality attributes in the implementation.

Summary 2

Architecture evaluation

- why – to understand and analyze design tradeoffs
- when – throughout the lifecycle
- benefits – overall cost reduction, better understanding and prioritization of requirements and quality goals, early detection of problems, improved architectures

Evaluations are carried out via questioning and measuring techniques

The ATAM is the most widely used method for evaluating architectures with respect to multiple quality attributes

One aspect of an evaluation can be to uncover technical debt

Questions?

Workshop

CSEE&T 2017

Software Architecture Educators Workshop

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Workshop Evolution 1

From how to teach software architecture …

- 2004: Fitting essential concepts into a “small package”
- 2005: How to think architecturally — quality attributes and working in teams
- 2006: Exercises and tool support for exercises

… to teaching others …

- 2007: Forming and expanding the software architecture educator’s community
- 2008: Switch from “How can we do this?” to “Here’s how we do this in my programs”
- 2009: Half-day tutorial presented at CSEE&T on March 11, 2010
- 2010: Workshop at CSEE&T on May 22, 2011; group decision to ask for a shared artifact as the “entry fee” for the workshop.
- 2011: Workshop accepted for SIGCSE; low enrollment forced cancellation

Workshop Evolution 2

… to learning from others

- 2013: Unique opportunity to interact with like-minded teaching colleagues face to face as well as to connect to a growing community
- 2014: How to apply what I just learned
- 2015: How to include emerging topics in courses

… to moving beyond software architecture

- 2016: Expanding the software engineering educator's community

… to software engineering practices in 2017

Emerging Trend in Attendees

Software architecture educators are becoming very creative with respect to teaching methods and artifacts, triggered by

- Difficulty to get and keep the attention of students (especially undergraduates)
- Packed curricula forcing them to teach the same concepts in less time
- Job market starting to understand software engineering and software architecture
  - Using these as differentiators

Sample Teaching Techniques

Technique/Method	Description
Design Thinking (Case Method)	Centers around resolving real-life cases through individual preparation, group analysis, brainstorming, innovation and creative ideas.
Flipped (Inverted) Classroom	Delivers instructional content, often online, outside of the classroom. Moves activities, including those that may have traditionally been considered homework, into the classroom.
Gamification	Uses game design elements in non-game contexts to teach concepts.
Jigsaw	Breaks classes into groups and breaks assignments into pieces that the group assembles to complete the (jigsaw) puzzle.

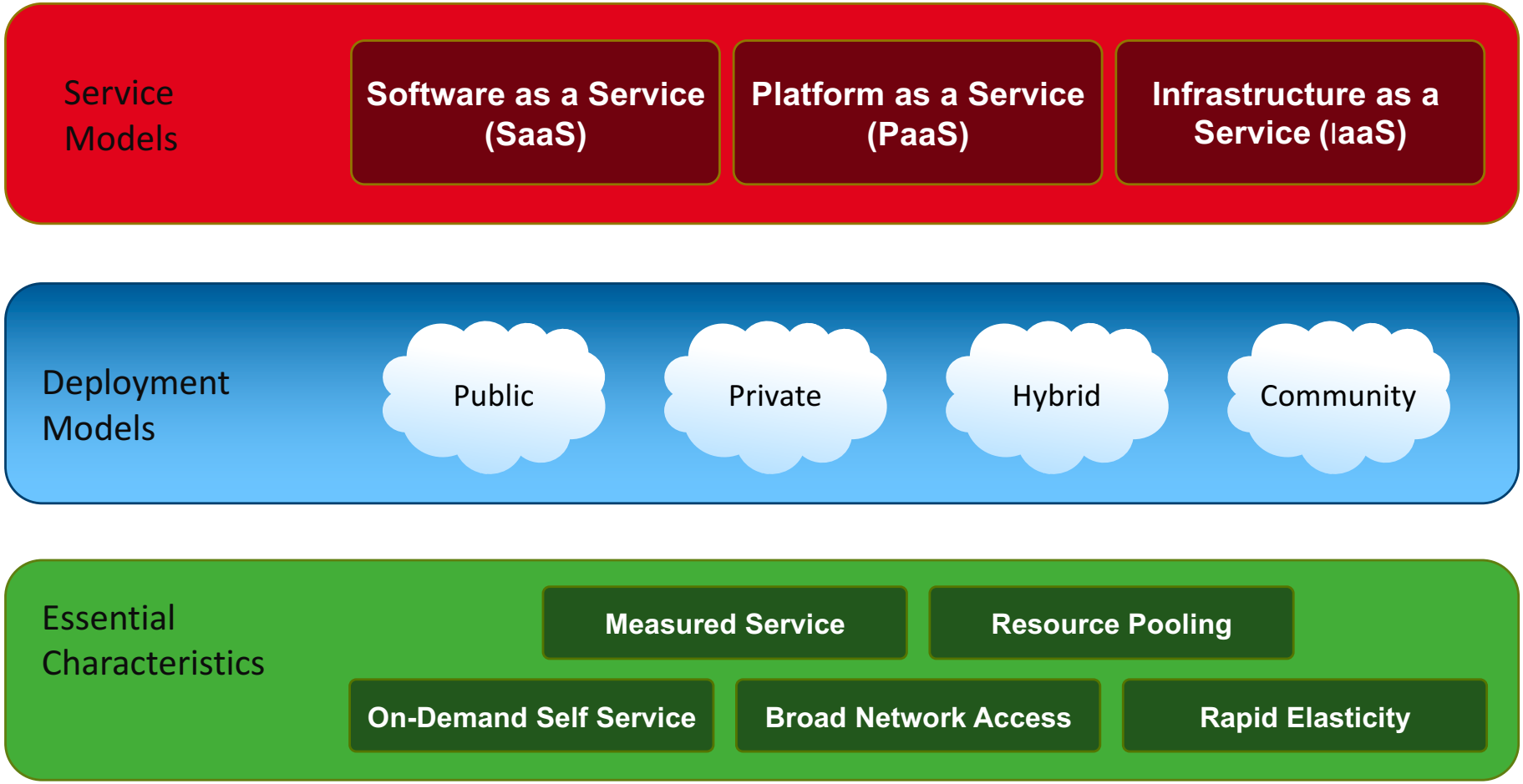
Resources:

- Design Thinking for Educators. http://designthinkingforeducators.com/
- Flipped Classroom. http://www.knewton.com/infographics/flipped-classroom/
- Gamification. https://www.knewton.com/infographics/gamification-education/
- The Jigsaw Classroom. http://www.jigsaw.org

Workshop Logistics

1. Facilitator provides a quick introduction of a software architecture lesson to teach
2. Attendees break up into groups based on previous list of teaching techniques
3. Each group discusses how to teach the lesson using the assigned teaching technique
4. Each group summarizes and presents the main elements of their proposal
5. After each presentation, the facilitator leads a discussion of the advantages and disadvantages of each technique

Lesson Topic: Architectural Implications of Cloud Computing 1



Source: National Institute of Standards and Technology (NIST), 2011

Lesson Topic: Architectural Implications of Cloud Computing 2

The cloud consumer typically uses resources in the cloud to

- SaaS: Use systems (or system components) on demand
- PaaS: Quickly deploy systems for internal and/or external use
- IaaS: Fully or partially deploy systems, fully or partially store transactional data, archive data, or complement existing resources

Some quality attributes of interest to cloud consumers are therefore (in no particular order)

- Security
- Interoperability
- Scalability and Elasticity
- Monitorability
- Availability

Lesson Topic: Architectural Implications of Cloud Computing 3

How would you put together a lesson, using the assigned teaching technique, for students to understand the architectural implications of cloud computing?

Rules of the game

- Make any assumptions regarding time available for lecture(s), pre-requisite knowledge, and how it fits into the rest of the syllabus
- Create a single sheet with the output from the team
- Select a presenter

Final Thoughts

Computer Science and Software Engineering Education is losing traction as a community

However … "*even though colleges have increased production of computer science bachelor's degrees, it's not enough to fill the jobs available in the field*"*

- There is a huge opportunity to reach and engage students to fill this gap

We need help

- Replicate the CS/SE/SA Educators Workshop at your organizations
- Create communities of interest around CS/SE/SA education

* Source: Falling Behind. https://www.insidehighered.com/news/2017/10/27/even-booms-student-enrollment-not-enough-degrees-keep-jobs-computer-science