What is Software Architecture and Why It is important

March 2014

Ying Shen

SSE, Tongji University

Lecture objectives

This lecture will

- introduce and define the term “software architecture”
- explain the value that a software architecture brings to a development project
- describe how software architecture is composed of several different software structures
- give examples of several commonly used architectural structures and show their uses

Architecting a dog house

Can be built by one person

Requires

- Minimal modeling
- Simple process
- Simple tools

Architecting a house

Built most efficiently and timely by a team

Requires

- Modeling
- Well-defined process
- Power tools

Why we need software architecture

Why not just start coding and let things turn out somehow.

Software is increasingly complex.

Software is evolving, reused in new contexts.

The promise of off-the-shelf components with easy reuse is just partial.

Why we need software architecture

Sometimes we need to adapt, wrap or even modify these off-the-shelf components.

Too many components in software products, small and large.

Large number of connections/dependencies between these components.

How to make sure that SW development reaches the intended goals of a project.

Why we need software architecture

Software is complex:

- >10KLOC (KLOC = 1000 lines of code): internal projects, tools, hobby projects, minimum viable products.
- >100KLOC: small products, mobile apps.
- >1MLOC: operating systems, native frameworks, typical desktop software applications, server side applications, typical everyday product.

Why we need software architecture

How to understand software that has >1MLOC?

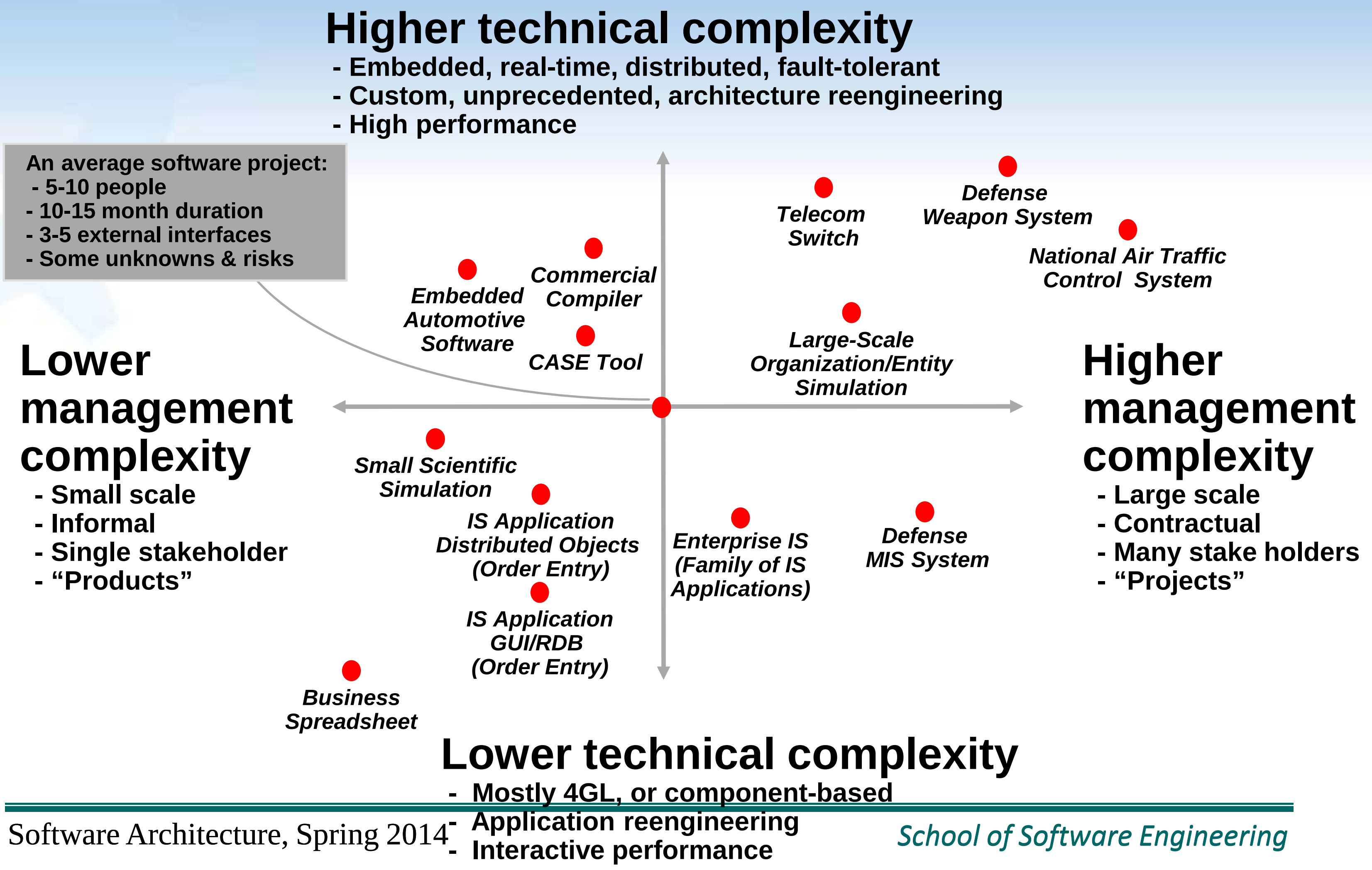
How to reach >1MLOC and still understand software?

How to reach >1MLOC without excessive complexity?

How a >1MLOC software evolves into the next version?

How a >1MLOC software or its components are reused (pivoted)?

Dimensions of software complexity


Higher technical complexity
- Embedded, real-time, distributed, fault-tolerant
- Custom, unprecedented, architecture reengineering
- High performance
An average software project:
- 5-10 people
- 10-15 month duration
- 3-5 external interfaces
- Some unknowns & risks
Lower management complexity
- Small scale
- Informal
- Single stakeholder
- "Products"
Higher management complexity
- Large scale
- Contractual
- Many stake holders
- "Projects"
Telecom Switch
Defense Weapon System
National Air Traffic Control System
Commercial Compiler
Embedded Automotive Software
CASE Tool
Large-Scale Organization/Entity Simulation
Small Scientific Simulation
IS Application Distributed Objects (Order Entry)
Enterprise IS (Family of IS Applications)
Defense MIS System
IS Application GUI/RDB (Order Entry)
Business Spreadsheet
Lower technical complexity
- Mostly 4GL, or component-based
- Application reengineering
- Interactive performance

Architectural description has a natural position in system design and implementation

Elements of a complete software system

User view of problem	User Model
Software view of problem	Requirement
Modules and connections	Architecture
Algorithms & data structures	Code
Data layouts, memory maps	Executable

Factors influencing architectures

Architectures are influenced by

- stakeholders of a system
- technical and organizational factors
- architect’s background

Customers

Customers are the people who pay for system development.

Customer concerns include

- cost of the system
- usability and lifetime of the system
- interoperability with other systems
- time to market
- platform portability

End users

End users are the people who use the system. They include

- “regular” users
- system administrators
- members of the development organization

End users are concerned with

- ease of use
- availability of function

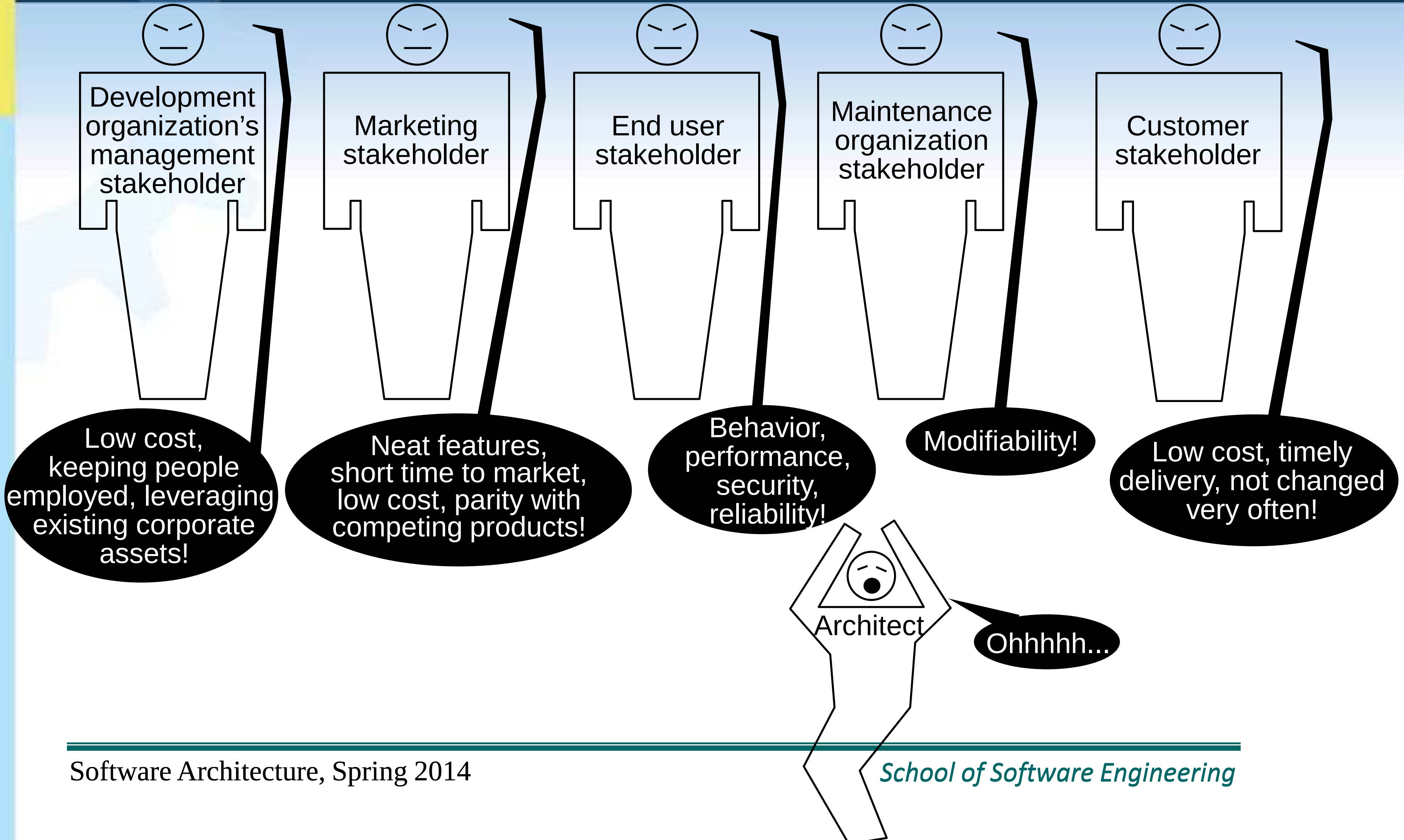
Other stakeholders

Development organization

Marketers

Maintenance organization

Stakeholders of a system


Development organization's management stakeholder
Marketing stakeholder
End user stakeholder
Maintenance organization stakeholder
Customer stakeholder
Low cost, keeping people employed, leveraging existing corporate assets!
Neat features, short time to market, low cost, parity with competing products!
Behavior, performance, security, reliability!
Modifiability!
Low cost, timely delivery, not changed very often!
Architect
Ohhhhh...

Development organization concerns

Immediate business issues

- amortizing the infrastructure
- keeping cost of installation low
- utilizing personnel

Long-term business issues

- investing in an infrastructure to reach strategic goals
- investing in personnel

Development organization concerns

Organizational structure issues

- furthering vested interests, e.g.,
  - maintaining an existing database organization
  - supporting specialized expertise
- maintaining the standard method of doing business

Technical environment

Current trends: today's information system will likely employ a

- database management system
- Web browser for delivery and distribution across platforms

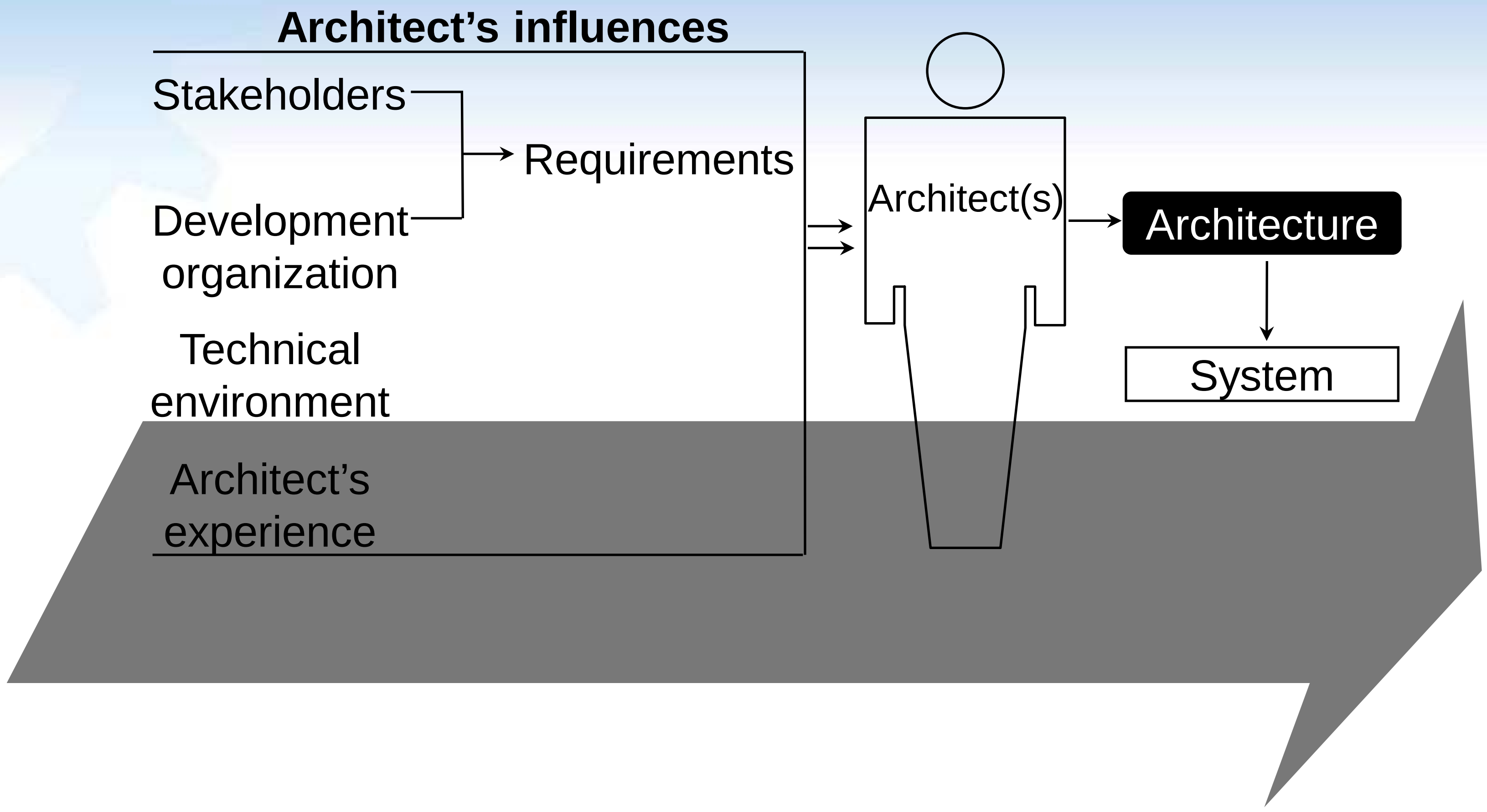
Available technology: decisions on using a centralized or decentralized system depend on processor cost and communication speed; both are changing quantities.

Architect’s background

Architects develop their mindset from their past experiences.

- Prior good experiences will lead to replication of prior designs.
- Prior bad experiences will be avoided in the new design.

Summary: influences on the architect



Factors influenced by architectures

Structure of the development organization

Enterprise goals of the development organization

Customer requirements

Architect’s experience

Technical environment

The architecture itself

Architecture influences the development organization structure

Short term: Work units are organized around architectural units for a particular system under construction.

Long term: When company constructs collection of similar systems, organizational units reflect common components (e.g., operating system unit or database unit).

Architecture influences the development organization enterprise goals

A successful system may establish a foothold in the market niche.

Being known for developing particular kinds of systems becomes a marketing device.

Architecture becomes a leveraging point for additional market opportunities and networking.

Architecture influences customer requirements

Knowledge of similar fielded systems leads customers to ask for particular features.

Customers will alter their requirements on the basis of the availability of existing systems.

Architecture influences the architect’s experience and technical environment

Creation of a system affects the architect’s background.

Occasionally, a system or an architecture will affect the technical environment.

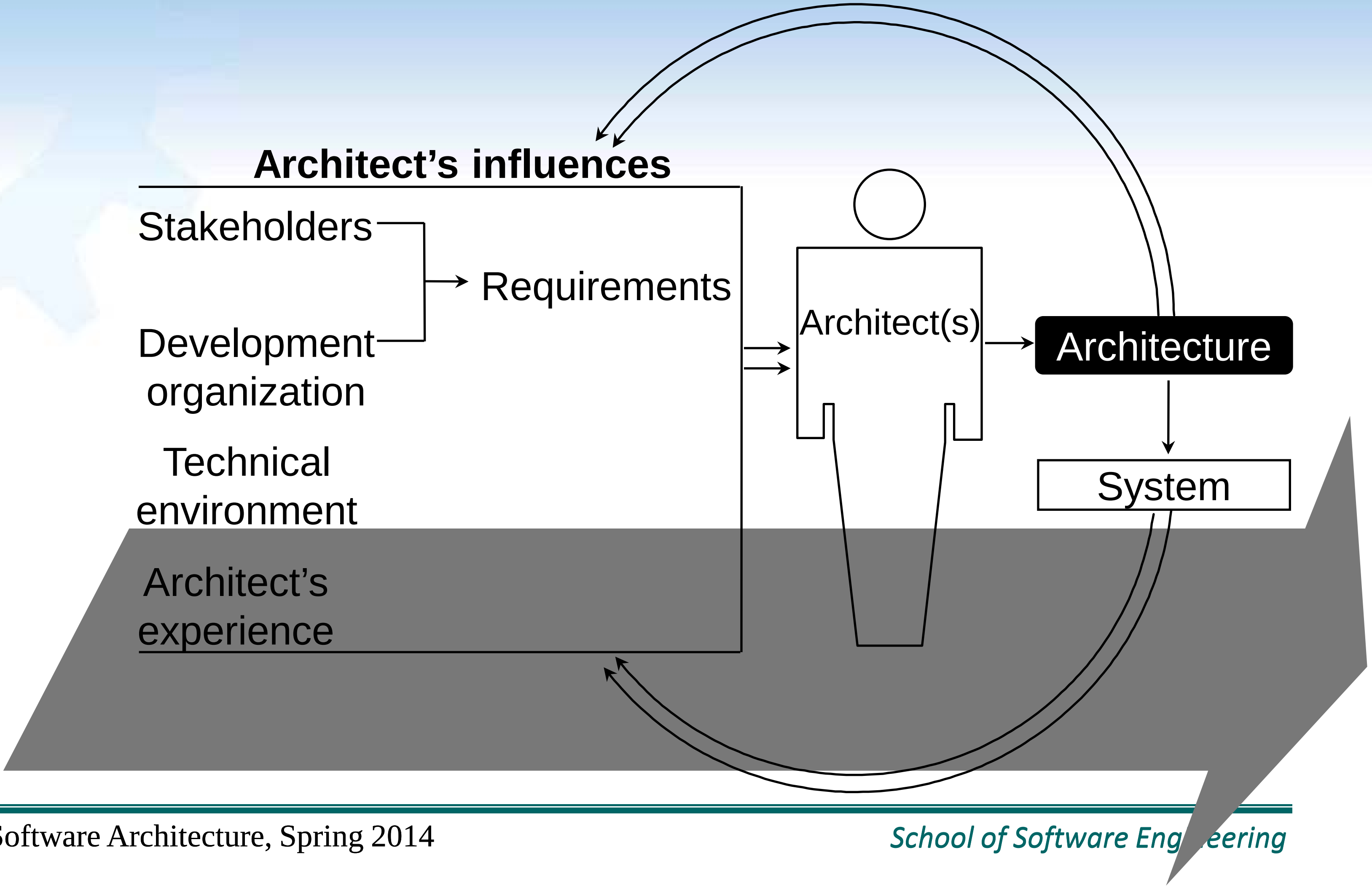
- the WWW for information systems
- the three-tier architecture for database systems

A cycle of influences

Architectures and organizations influence each other.

- Influences to and from architectures form a cycle.
- An organization can manage this cycle to its advantage.

Architecture Business Cycle (ABC)



What is Software Architecture?

Definition from *IEEE*:

Architecture is the fundamental ***organization*** *of a system embodied in its* ***components****, their* ***relationships*** *to each other, and to the environment, and the* ***principles*** *guiding its design and evolution.*

[*IEEE* 1471]

What is Software Architecture?

Definition from Kruchten: *Rational Unified Process*, 1999:

*An architecture is the set of significant **decisions** about the **organization** of a software system, the selection of structural **elements** and their **interfaces** by which the system is composed, together with their **behavior** as specified in the collaborations among those elements, the **composition** of these elements into progressively larger subsystems, and the **architectural style** that guides this organization -- these elements and their interfaces, their collaborations, and their composition.*

What is Software Architecture?

Definition from Bass et al.: *SA in practice*, 2012:

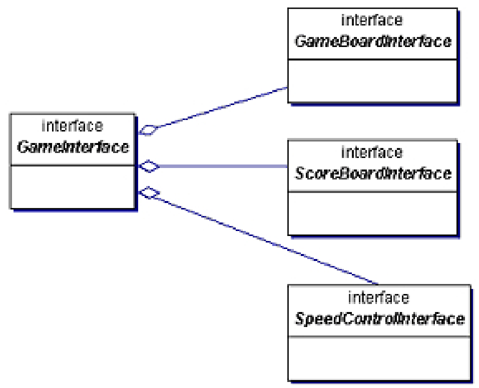
The software architecture of a system is the ***set of structures*** *needed to reason about the system, which comprise software* ***elements****,* ***relations*** *among them, and* ***properties*** *of both.*

The exact structures to consider and the ways to represent them vary according to engineering goals.

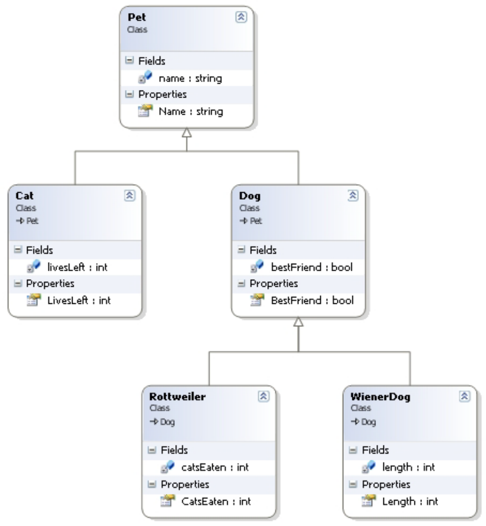
Implications of the definition

Architecture is a set of software structures

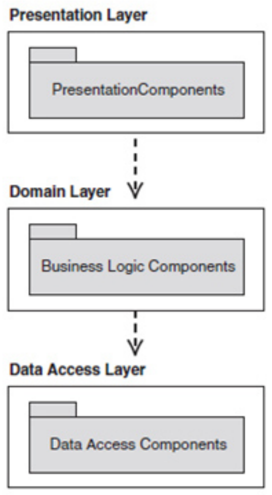
- Systems have many structures.
  - *Modules*: Static structures.



Module decomposition structure



Class diagram

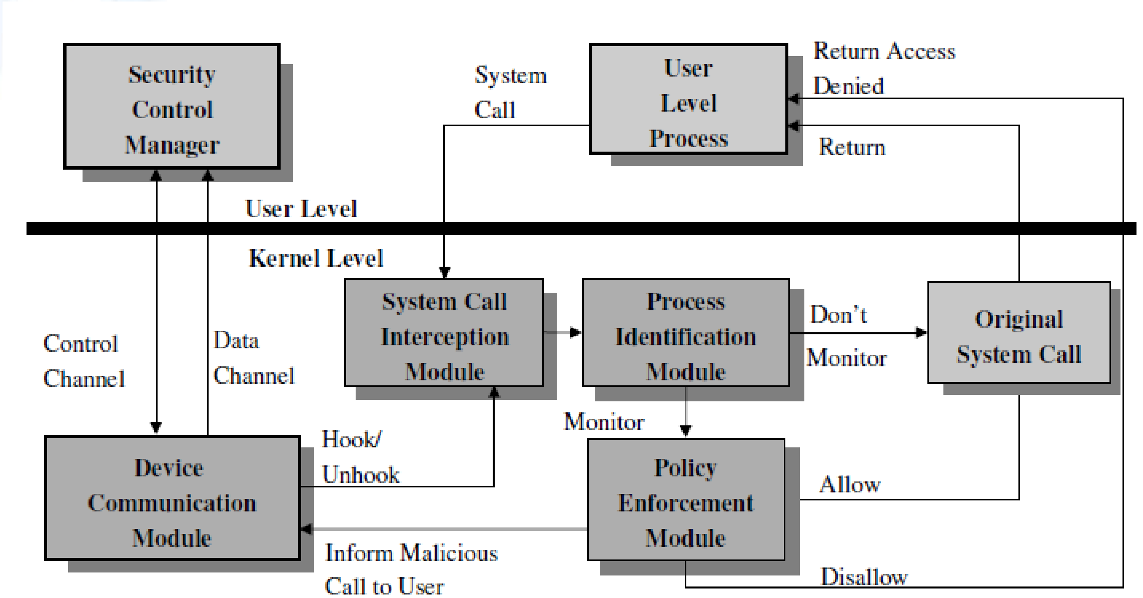


Layer structure

Implications of the definition

Architecture is a set of software structures

- Systems have many structures.
  - *Modules*: Static structures.
  - *Component-and-connector* (C&C) structure: runtime, dynamic structure. *Component* is a runtime entity.



Picture comes from (Lam et al. 2006)

Reference:
Lam et al., Secure Mobile Code Execution Service, in *Proc. LISA*, 2006

Implications of the definition

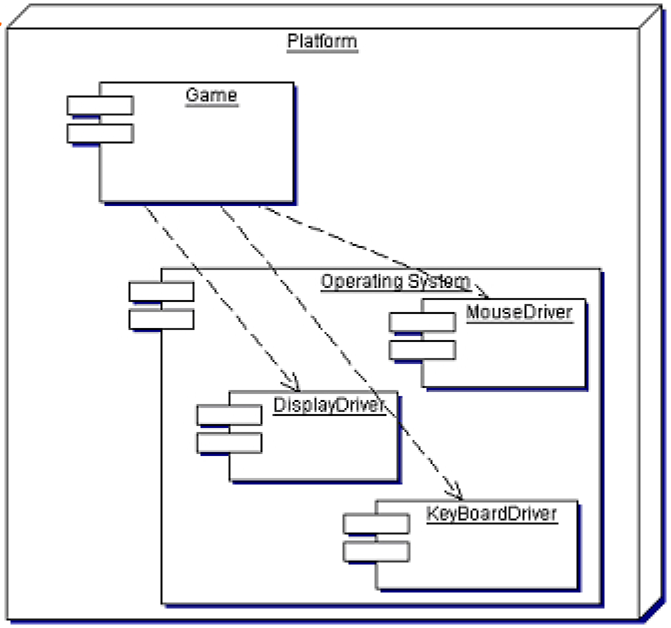
Architecture is a set of software structures

- Systems have many structures.
  - *Modules*: Static structures.
  - *Component-and-connector* (C&C) structure: runtime, dynamic structure. *Component* is a runtime entity.
  - Allocation structures

Implications of the definition

Architecture is a set of software structures

- Systems have many structures.
  - *Modules*: Static structures
  - *Component-and-connector* (C&C) structure: runtime, dynamic structure. *Component*
  - Allocation structures



Implications of the definition

Architecture is a set of software structures

- Systems have many structures.
    - *Modules*: Static structures
    - *Component-and-connector* (C&C) structure: runtime, dynamic structure. *Component* is a runtime entity.
    - Allocation structures
    - No single structure can be *the* architecture.
    - The set of candidate structures is not fixed or prescribed: whatever is useful for analysis, communication, or understanding.

Implications of the definition

Architecture is a set of software structures

- Systems have many structures.
  - *Modules*: Static structures
  - *Component-and-connector* (C&C) structure: runtime, dynamic structure. *Component* is a runtime entity.
  - Allocation structures
  - No single structure can be *the* architecture.
  - The set of candidate structures is not fixed or prescribed: whatever is useful for analysis, communication, or understanding.
- Not all structures of the software are architectural
  - It should support reasoning about the system and the system's properties

Implications of the definition

Architecture is an abstraction

- Architecture defines components and how they interact.
- Architecture suppresses purely local information about components; private details are not architectural.

Implications of the definition

Every software system *has* a software architecture.

- Every system is composed of elements and relationships among them.
- In the simplest case, a system is composed of a single element, related only to itself.

Just having an architecture is different from having an architecture that is known to everyone.

- architecture versus specification of the architecture
- architecture recovery and conformance
- rationale for the architecture

Implications of the definition

Architecture includes behavior

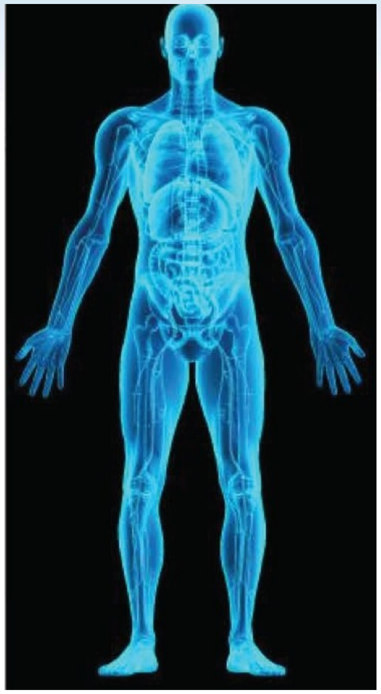
- This means that box-and-line drawings alone are *not* architectures, but a starting point.
- You might *imagine* the behavior of a box labeled “database” or “executive.”
- You need to add specifications and properties.

Architecture structures and views

Human body

skeletal

vascular



X-ray

Physiological Structures

Architecture structures and views

In a house, there are plans for

- rooms
- electrical wiring
- plumbing
- ventilation

Each of these constitutes a “view” of the house.

- used by different people
- used to achieve different qualities in the house
- serves as a description and prescription.

So it is with software architecture.

Architecture structures and views

A ***view*** is a representation of a coherent set of architectural elements. It consists of a representation of a set of elements and the relations among them.

A ***structure*** is the set of elements existing in software or hardware.

A view is a representation of a structure.

- Module view *vs*. module structure

Three kinds of structures

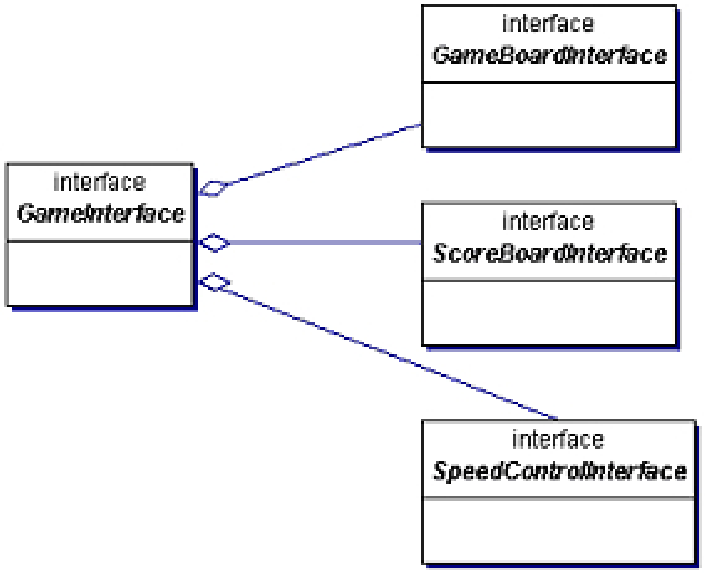
Module structures embody decisions as to how the system is to be structured as a set of code or data units that have to be constructed or procured.

- The elements are modules such as classes, layers.
- Focus on the functional responsibility of each module, not emphasize runtime issue.

Three kinds of structures

Useful module structures:

- Decomposition structure

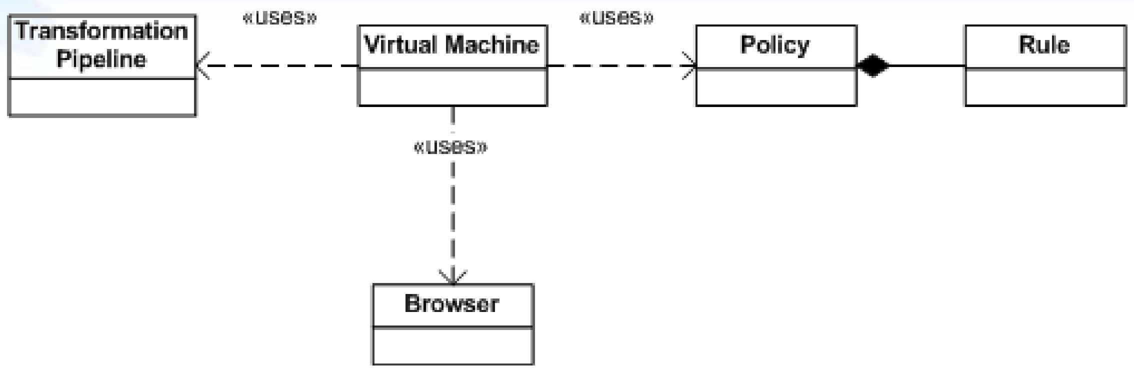


- Units: modules
- Relations: is a submodule of
- Used for: resource allocation and protect structuring and planning; information hiding, encapsulation; configuration control
- Affected attributes include: modifiability

Three kinds of structures

Useful module structures:

- Uses structure

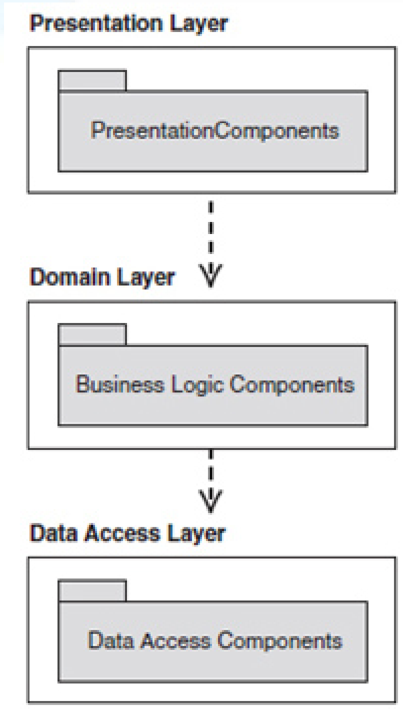


- ➢ Units: modules
- ➢ Relations: uses, i.e. requires the correct presence of
- ➢ Used for: engineering subsets, engineering extensions
- ➢ Affected attributes include: “subsetability”, extensibility

Three kinds of structures

Useful module structures:

- Layer structure

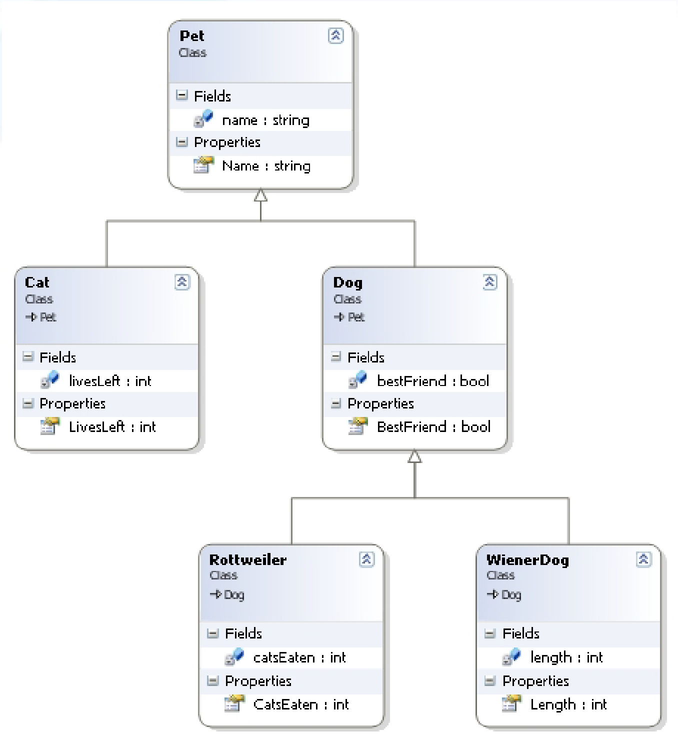


- Units: layers
- Relations: requires the correct presence of, uses the services of, provides abstraction to
- Used for: incremental development, implementing systems on top of "virtual machines"
- Affected attributes include: portability

Three kinds of structures

Useful module structures:

- Class (or generalization) structure

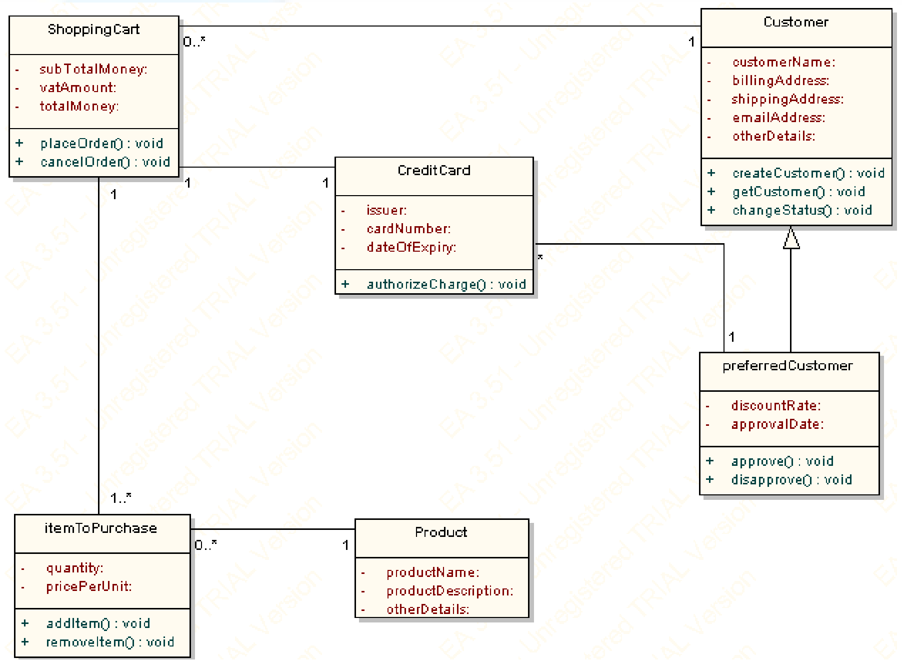


- Units: classes, objects
- Relations: inherits from or is an instance of
- Used for: in object-oriented design systems, factoring out commonality; planning extensions of functionality
- Affected attributes include: modifiability and extensibility

Three kinds of structures

Useful module structures:

- Data model



- ➢ Units: data entities
- ➢ Relations: {one, many}-to-{one, many}, generalizes, specializes
- ➢ Used for: engineering global data structures for consistency and performance
- ➢ Affected attributes include: modifiability, performance

Three kinds of structures

Module view help answer questions:

- ✓ *What is the primary functional responsibility assigned to each module?*
- ✓ *What other software elements is a module allowed to use?*
- ✓ *What other software does it actually use and depend on?*
- ✓ *What modules are related to other modules by generalization or specialization (i.e., inheritance) relationships?*

Looking at its module views is an excellent way to reason about a system's modifiability.

Three kinds of structures

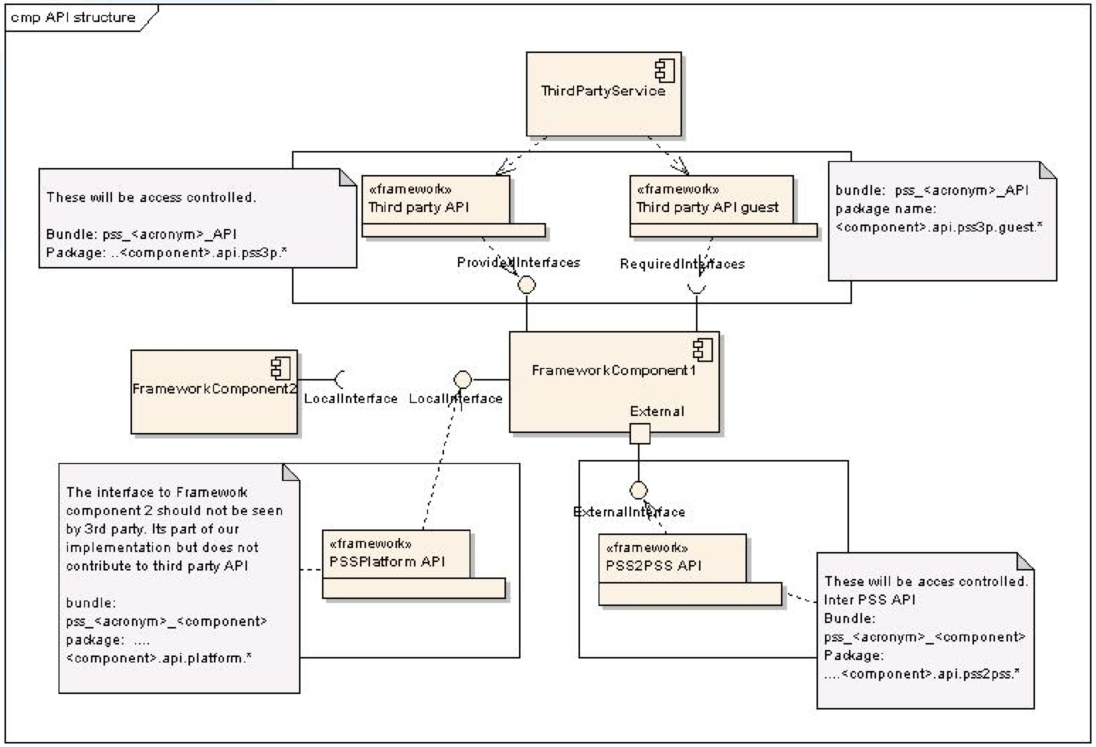
Component-and-connector structures embody decisions as to how the system is to be structured as a set of elements that have runtime behavior (components) and interactions (connectors).

- The elements are runtime components (such as services, peers, clients, servers, filters) and connectors (such as call-return, process synchronization operators, pipes).

Three kinds of structures

Useful C&C structure:

- Service structure



These will be access controlled.

Bundle: pss_<acronym>_API
Package: ..<component>.api.pss3p.*

bundle: pss_<acronym>_API
package name:
<component>.api.pss3p.guest.*

The interface to Framework component 2 should not be seen by 3rd party. Its part of our implementation but does not contribute to third party API

bundle:
pss_<acronym>_<component>
package: ....
<component>.api.platform.*

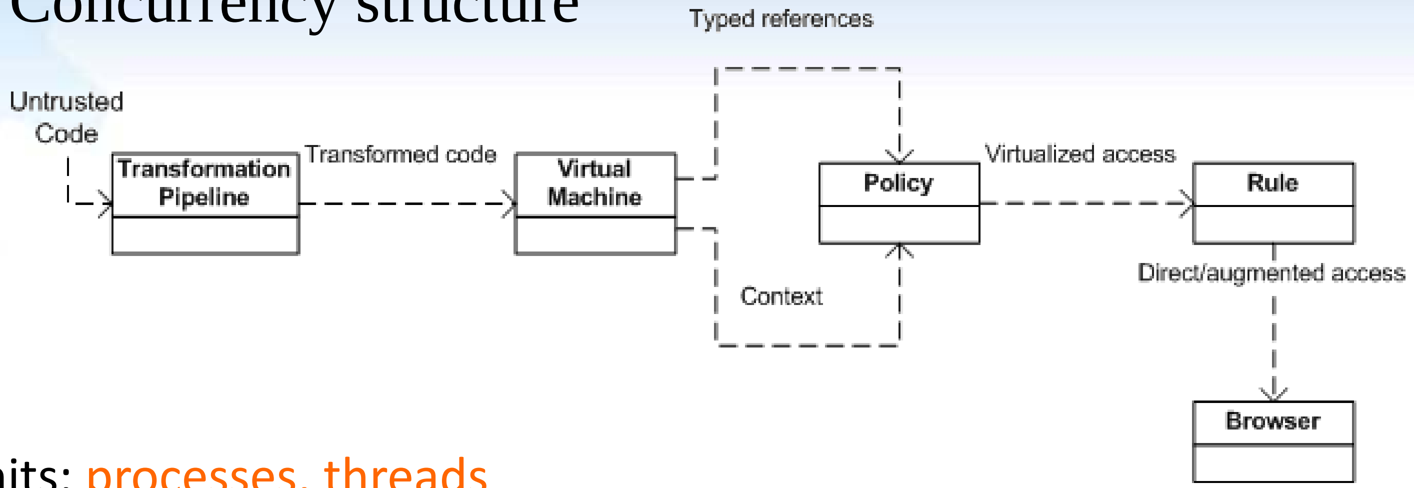
These will be acces controlled.
Inter PSS API
Bundle:
pss_<acronym>_<component>
Package:
....<component>.api.pss2pss.*

- Units: services, ESB, registry, others
- Relations: runs concurrently with, may run concurrently with, excludes, precedes, etc.
- Used for: scheduling analysis, performance analysis
- Affected attributes include: Interoperability, modifiability

Three kinds of structures

Useful C&C structure:

- Concurrency structure



- Units: processes, threads
- Relations: can run in parallel
- Used for: identifying locations where resource contention exists, or where threads may fork, join, be created, or be killed
- Affected attributes include: performance, availability

Three kinds of structures

C&C views help answer questions:

- ✓ *What are the major executing components and how do they interact at runtime?*
- ✓ *What are the major shared data stores?*
- ✓ *Which parts of the system are replicated?*
- ✓ *How does data progress through the system?*
- ✓ *What parts of the system can run in parallel?*
- ✓ *Can the system's structure change as it executes and, if so, how?*

C&C views are crucially important for reasoning about the system's runtime properties such as performance, security, availability.

Three kinds of structures

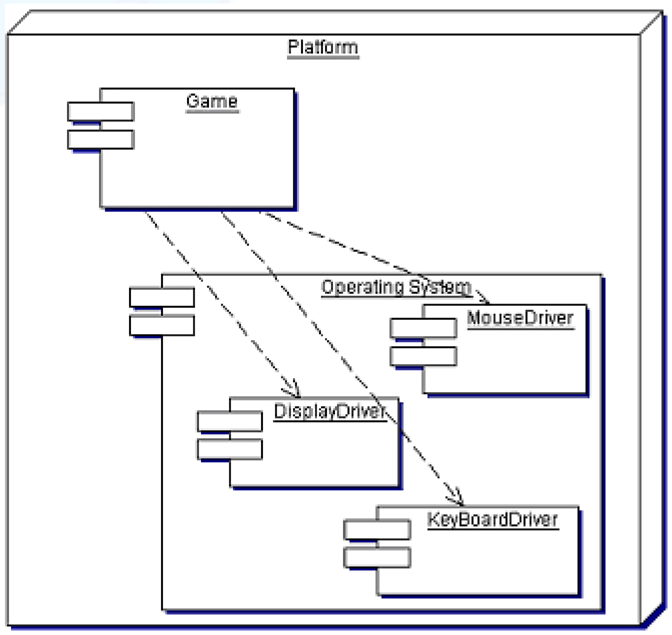
Allocation structures embody decisions as to how the system will relate to nonsoftware structures in its environment (such as CPUs, file systems, networks, development teams, etc.).

- They show the relationship between the software elements and elements in one or more external environments in which the software is created and executed.

Three kinds of structures

Useful allocation structures:

- Deployment structure

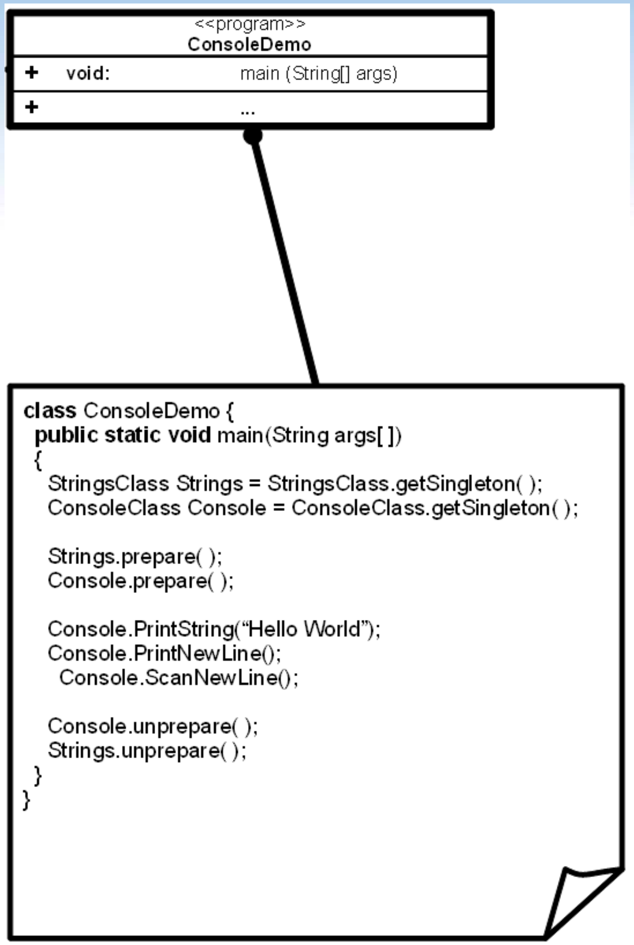


- Units: software elements (usually a process from a C&C view) and hardware entities (processors)
- Relations: allocated-to, migrates-to
- Used for: performance, availability, security analysis
- Affected attributes include: performance, data integrity, security, and availability

Three kinds of structures

Useful allocation structures:

- Implementation structure
  - Units: software elements (usually modules), file structure
  - Relations: stored in
  - Used for: configuration control, integration, test activities
  - Affected attributes include: development efficiency

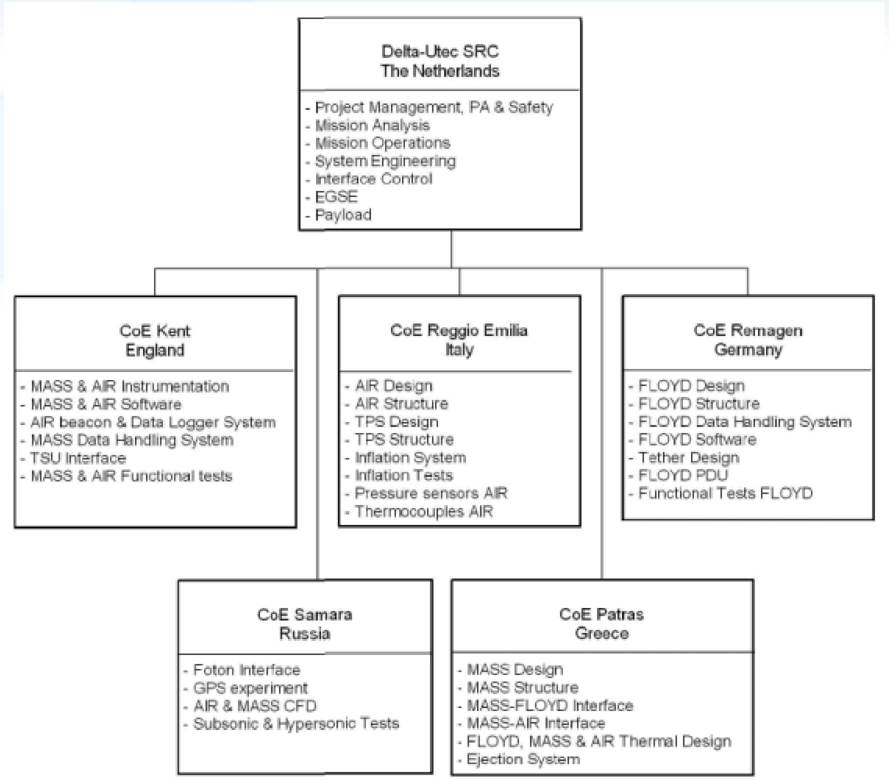


```
class ConsoleDemo {
 public static void main(String args[ ])
 {
  StringsClass Strings = StringsClass.getSingleton( );
  ConsoleClass Console = ConsoleClass.getSingleton( );

  Strings.prepare( );
  Console.prepare( );

  Console.PrintString("Hello World");
  Console.PrintNewLine();
   Console.ScanNewLine();

  Console.unprepare( );
  Strings.unprepare( );
 }
}
```

Three kinds of structures

Useful allocation structures:

- Work assignment structure



- Units: modules and organizational units
- Relations: assigned to
- Used for: Project management, best use of expertise and available resources, management of commonality
- Affected attributes include: development efficiency

Three kinds of structures

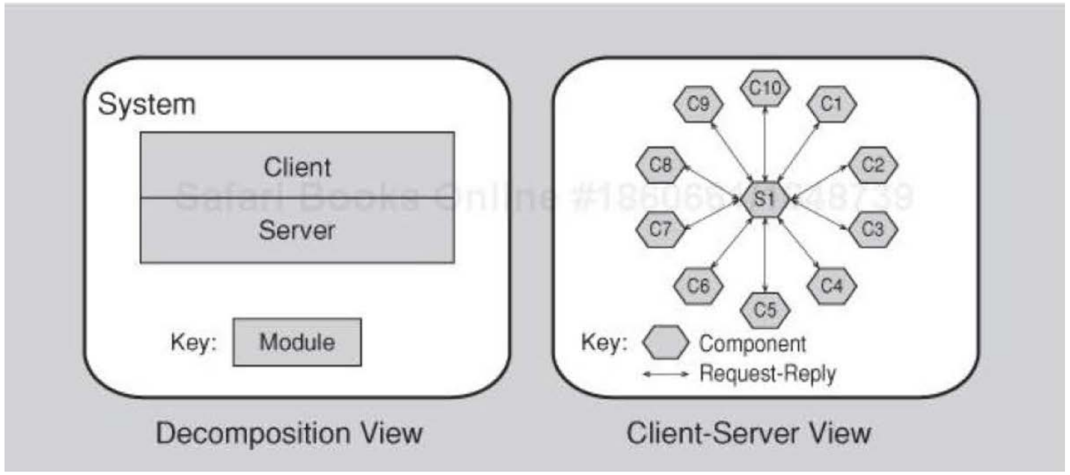
Allocation views help answer questions:

- ✓ *What processor does each software element execute on?*
- ✓ *In what directories or files is each element stored during development, testing, and system building?*
- ✓ *What is the assignment of each software element to development teams?*

Relationship between structures

Each of these structures provides a different perspective and design handle on a system.

They are not independent. Elements of one structure will be related to elements of other structures.



Two views of a client-server system

Which structures to choose?

You should think about how the various structures available provide insight and leverage into the system's most important quality attributes, and then choose the ones that will play the best role in delivering those attributes.

What are structures used for?

Documentation vehicle for

- current development
- future development
- managers
- customers

Engineering tool to help achieve qualities

Architectural structures summary

Structures are related to each other in complicated ways.

In some systems, different structures collapse into a single one. (For example, process structure may be the same as module structure for extremely small systems.)

Lesson: Choose the structures that are useful to the system being built and to the achievement of qualities that are important to you.

Views

An architecture is a very complicated construct -- too complicated to be seen all at once.

Views are a way to manage complexity.

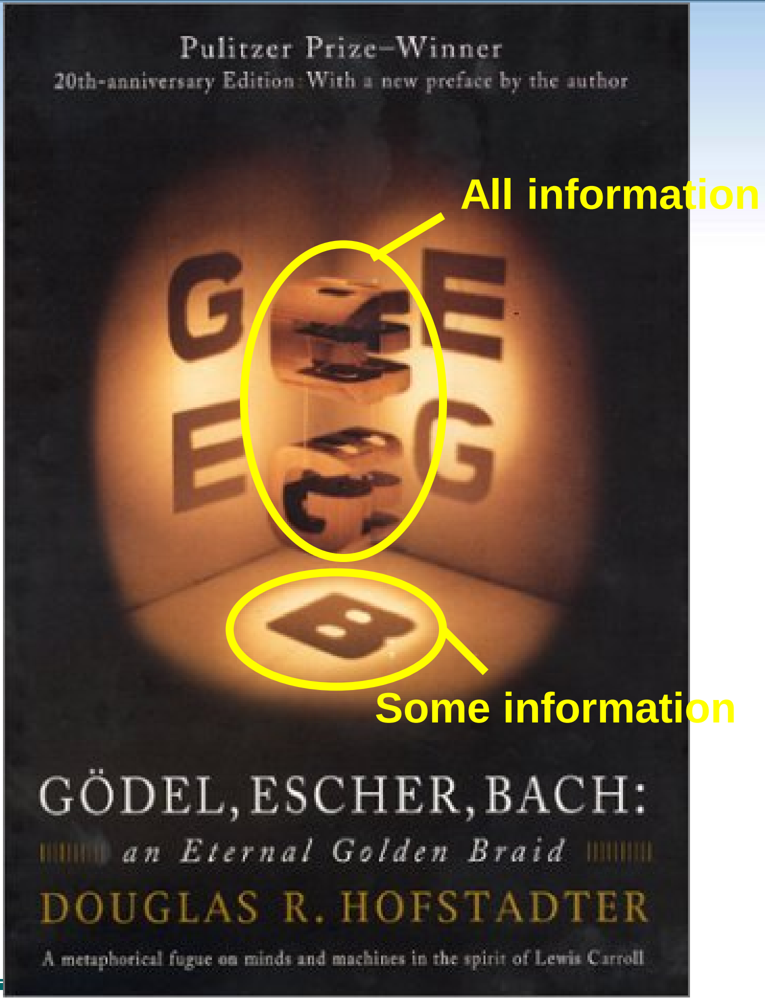
- 1974: Parnas observed that software is composed of many structures
- 1992: Perry and Wolf recognize that, similar to buildings (with plumbing and electrical and wall diagrams), different views of a system are required.
- 1995: Kruchten defined the “4+1 views” approach to software architecture.
- 2000: Hofmeister, Nord, and Soni defined the “Siemens Four Views” approach to software.

Views

A view is a representation of a set of architectural elements and the relations associated with them.

Not *all* architectural elements -- *some* of them.

A view binds element *types* and relation *types* of interest, and shows those.


Pulitzer Prize–Winner
20th-anniversary Edition With a new preface by the author
All information
Some information
GÖDEL, ESCHER, BACH:
an Eternal Golden Braid
DOUGLAS R. HOFSTADTER
A metaphorical fugue on minds and machines in the spirit of Lewis Carroll

Why is architecture important?

Architecture is important for the following reasons:

1. Inhibiting or enabling a system’s quality attributes
2. Reasoning about and managing change
3. Enhancing communication among stakeholders
4. Carrying earliest design decisions about a system
5. Defining constraints on an implementation
6. Influencing the organizational structure
7. Supplying a transferable, reusable model

….

Inhibiting or enabling a system's quality attributes

For example:

If you desire	***Examine***
performance	inter-component communication
modifiability	component responsibilities
security	inter-component communication, specialized components (e. g., kernels)
scalability	localization of resources
ability to subset	inter-component usage
reusability	inter-component coupling

The architecture influences qualities, but does not guarantee them.

Reasoning about and managing change

An architecture helps reason about and manage change.

- important since ≈ 80% of effort in systems occurs after deployment

Architecture divides all changes into three classes:

- local: modifying a single component
- non-local: modifying several components
- architectural: modifying the gross system topology, communication, and coordination mechanisms

A "good" architecture is one in which the most likely changes are also the easiest to make.

Enhancing communication among stakeholders

Architecture provides a common frame of reference in which competing interests may be exposed and negotiated.

- negotiating requirements with users and other stakeholders
- keeping the customer informed of progress and cost
- implementing management decisions and allocations

Carrying earliest design decisions about a system

Any design, in any discipline, can be viewed as a set of decisions.

- Example: painting a picture

An architecture design can also be viewed as a set of decisions.

- The early design decisions constrain the decisions that follow, and changing these decisions has enormous ramifications.

Carrying earliest design decisions about a system

Some early design decisions embodied by software architecture:

- Will the system run on one processor or be distributed across multiple processors?
- Will the software be layered? If so, how many layers will there be? What will each one do?
- Will components communicate synchronously or asynchronously? Will they interact by transferring control or data or both?
- Will the system depend on specific features of the operating system or hardware?
- Will the information that flows through the system be encrypted or not?
- What operating system will we use?

Defining constraints on an implementation

An architecture defines constraints on an implementation.

- Architectures are descriptive and prescriptive.
  - descriptive for communication
  - prescriptive for design and implementation
- Global resource allocation decisions constrain implementations of individual components.
- System tradeoffs regarding quality attributes are in the architectural realm.

Influencing the organizational structure

The architecture dictates organizational structure for development/maintenance efforts. Examples include

- division into teams
- units for budgeting, planning
- basis of work breakdown structure
- organization for documentation
- organization for CM libraries
- basis of integration
- basis of test plans, testing
- basis of maintenance

Once decided, architecture is extremely hard to change!

Architecture is basis for incremental development

An architecture helps with evolutionary prototyping and incremental delivery.

- Architecture serves as a skeletal framework into which components can be plugged.
- By segregating functionality into appropriate components, experimentation is easier.
- Risky elements of the system can be identified via the architecture and mitigated with targeted prototypes.

Architecture is a reusable model

An architecture is an abstraction: enables a one-to-many mapping (one architecture, many systems).

Systems can be built from large, externally developed components that are tied together via architecture.

Architecture is the basis for product (system) commonality.

Entire software product lines can share a single architecture.

Architecture is a reusable model

A component's functionality can be separated from its interconnection mechanisms.

Often, a component's packaging makes it difficult to reuse a component. For example, if you need

- an object, you can't use a task
- a task, you can't use an object
- to invoke with a pipe, you can't use a called program
- a program, you can't use a file

Architecture is a reusable model

Less is more: It pays to restrict the vocabulary of design alternatives.

Architectural styles serve as that restricted vocabulary of design alternatives.

Working with known styles

- reduces learning time
- enhances communication
- takes advantage of known style properties (e.g., performance, security, reliability)

Architecture is a reusable model

Architectures can enable template-based development.

Templates may be used to code component interaction frameworks. The developer fills in the unique part, and reuses the common part.

Templates enhance

- speed of development
- reliability
  - source of many errors eliminated
  - fixing one error improves many places

Architecture is a reusable model

In summary, an architecture forms a reusable model.

- enables product lines
- enables systems to be built from externally developed components
- separates functionality from interconnection mechanisms
- provides a vocabulary of design
- enables template-based component development

What makes a “Good” architect?

People skills: must be able to

- negotiate competing interests of multiple stakeholders
- promote inter-team collaboration

Technical skills: must

- understand the relationships between qualities and structures
- possess a current understanding of technology
- understand that most requirements for an architecture are not written down in any requirements document

What makes a “Good” architect?

Communication skills: must be able to

- clearly convey the architecture to teams (both verbally and in writing)
- listen to and understand multiple viewpoints

What makes a “Good” architecture?

Fitness for purpose

Achievable within a reasonable budget

Achievable within a reasonable time

What makes a “Good” architecture?

One measure of a good architecture is the how long it survives as a solution to a particular problem and in how many context can be used without changes.

What makes a "Good" architecture?

There is no clear, unified, objective way to define what is a good architecture.

There are good and bad practices.

There are patterns and anti-patterns.

If a solution is good, usually survives the test of time.

There are always good examples to get inspiration from.

Solutions that have been around for a long time.

Solutions that work in multiple contexts and environments independent of the product type.

Examples of good architectures

Good architectural styles exits now in most fields.

Pipes are a good model of combining multiple modules in processing data.

Pipes can be extended into flow-based programming.

Relational databases are a stable solution of a large group of database problems.

Examples of good architectures

The filing system is a good example of stable architecture:

- In early computing it was not clear how to access/organize persistent data.
- The hierarchical filing system solves this problem.
- A small, clearly defined and easy to use interface.
- Interface allows arbitrary implementation: various file systems focusing on speed, failure tolerance, etc.

For user interfaces the Cocoa (NextStep) UI framework provides a good inspiration.

POSIX is a good interface for OS designs.

Lecture summary

Architecture involves more than just technical requirements for a system. It also involves non-technical factors, such as

- the architect’s background
- the development environment
- the business goals of the sponsoring organization

Architecture influences the factors that affect it.

- Architects learn from experience.
- The development environment is expanded and altered.
- Businesses gain new marketing possibilities.

Discussion Questions

1. Software architecture is often compared to the architecture of buildings as a conceptual analogy. What are the strong points of that analogy? What is the correspondence in buildings to software architecture structures and views? What are the weaknesses of the analogy? When does it break down?

The End