Software Systems Architecture

FEUP-M.EIC-ASSO-2023-2024

Ademar Aguiar, Neil Harrison

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Homework Discussion

Present your “top five” questions:

1. WHY is this question important?
2. WHOM does this question impact?
3. WHEN can this question be answered?
4. HOW does the answer to this question impact the architecture of the system?

What is software architecture – My opinion

Oddly, this isn't really captured by most books

- (The original Shaw and Garlan book is probably closest)

1. The design of the entire system implementation (as envisioned)
   - Which may include hardware components, as well as networking/communication elements
   - Can be either an existing or proposed system
2. Includes partitions of the system
   - Some are physical (see above), some are purely software
   - If you have partitions, you must have connections between them
3. Abstract
   - Few, if any, details. This is intentional
4. Informal
   - Goes along with abstract
   - At odds with a lot of folks (mainly academics who haven't actually designed a system …)
5. It's all about conveying big-picture information
   - And providing guidance and a conceptual framework for detailed design and implementation
   - Which implies that architecture is intimately tied to its documentation (but see next slide)

Documentation, descriptions, etc.

Diagrams and views of the architecture are NOT the architecture

- Just like a UML diagram is NOT classes

The real architecture is in:

- The bits (if the system already exists)
- Our head (if it doesn't exist, and even if it does)
- (forward reference: programming as theory building)

But we often refer to the documentation as "the architecture"

- Wrong, but convenient…

Fundamentals of Software Architecture

Three basic understandings

Fundamental Understanding: 1

Every Application has an Architecture

Not every architecture is good!

Not every architecture is intentional

If an architecture is not intentional, where did it come from?

What are the properties that make an architecture good or bad?

Fundamental Understanding: 2

Every application has at least one architect

Why is it important that we understand this?

What are the implications of this understanding?

Fundamental Understanding: 3

Architecture is not a phase of software development

Where does this incorrect notion come from?

First: the idea that design of software is an activity that strictly precedes coding

Second: the idea that high-level design (called architecture) is an activity that strictly precedes low-level design

Architecture-Centric Software Development

Some people claim that software development should be architecture-centric.

I claim that software development IS by nature architecture-centric (if done rationally).

- In other words, software development is strongly influenced by architecture
- And in many cases, influences the architecture

If I'm right, what are the implications?

Implications:

You can’t design the architecture once and be done

But if it affects everything, upfront thought (and design) is necessary

- A good architecture makes development (and maintenance) easier
- A bad architecture makes development and maintenance harder
- Maybe that’s how we define a “good” or “bad” architecture! (see next slide)

If the architecture affects everything, it is important that everyone understand the architecture

- What does that say about the role of architecture?
- And the role of the architect?
- Naur: Programming as Theory Building

Implications #2

Good or bad architecture:

- In some cases, it's that a particular architecture is inappropriate for the particular system.
- A different architecture would make implementation of the system easier.
- We will explore this later with respect to architecture patterns and quality attributes.

Requirements

How are requirements important to the architecture?

How is the architecture important to requirements?

Design

Where is the line between the architecture and the design?

The process of design is a continuum:

- Sometimes you are at a system level
- Sometimes you are at a high level
  - (high level is not always at a system level)
- Sometimes you are at a detailed level

Early on, it's mostly system and high level

- Later, it's mostly detailed

Answer: the line is wherever we think it makes sense to draw it.

Coding, Testing, Deployment

How does architecture impact these?

How do these impact the architecture?

“Architect Also Implements”

Views of the Architecture Process

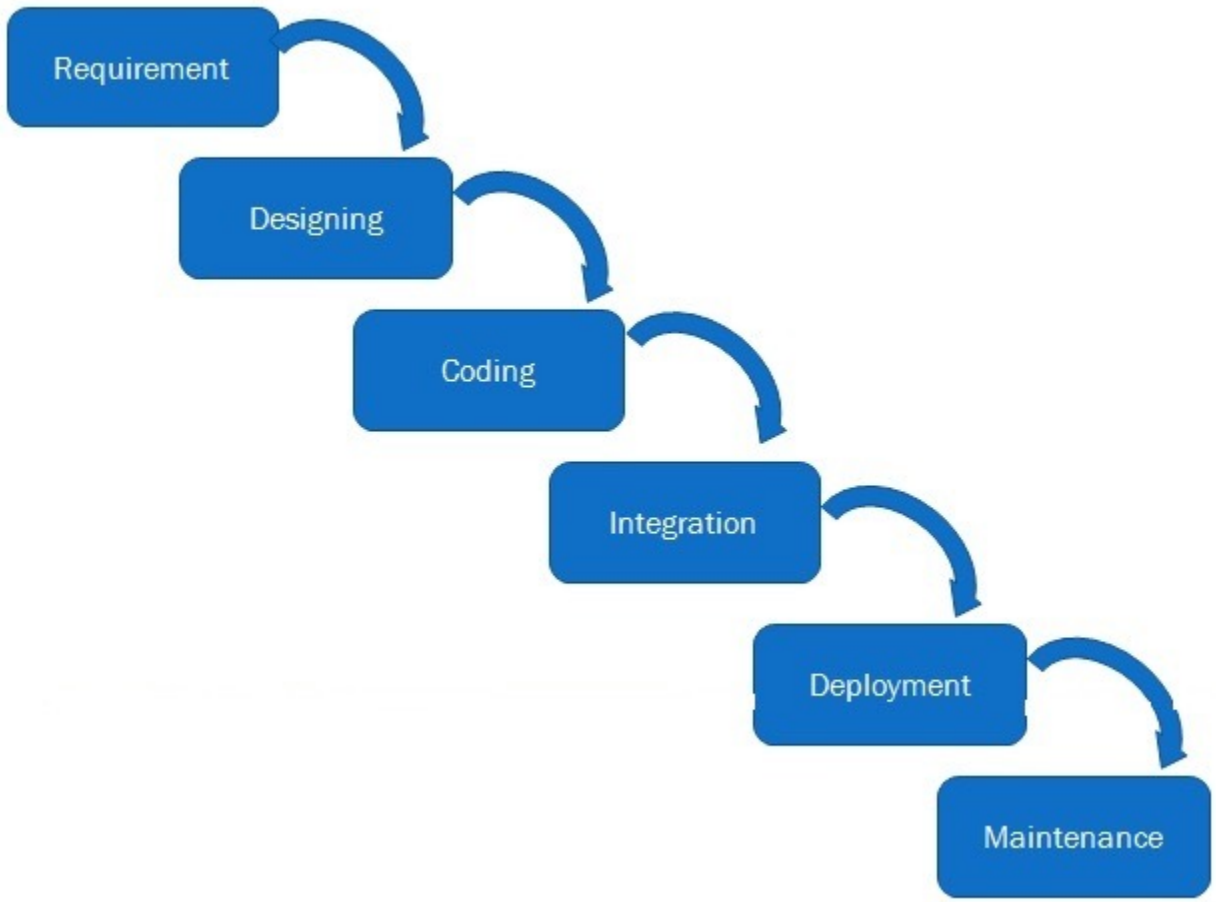
(a.k.a. the high level design process)

Two Extreme Views:

Traditional waterfall

XP: “Good design [architecture] emerges through refactoring” (Martin Fowler; also embraced by Ron Jefferies)

Architecture and Waterfall



Waterfall:

What's wrong with the waterfall view of design?

Designing and coding are iterative, highly integrated

If you think it of a model of time, it's wrong

What's RIGHT with the waterfall view of (architectural) design?

You need a general architecture beforehand in order to guide detailed design and coding.

It does show (part of) an information dependency model.

XP and TDD

Big Design Up Front (BDUF) is BAD

Good Architecture emerges through refactoring

Focus on unit tests and refactor to create a good design

What's wrong with this approach?

It's a recipe for accidental architecture; it doesn't acknowledge the need for intentional architectural design

- User stories don't live in a vacuum; they live in context of the entire system
- And architecture is all about that context

It violates the Second Law of Thermodynamics (i.e., it's a fantasy)

What's right with this approach?

Basically nothing

Goldilocks: How Much Architecture?

What's wrong with comprehensive architectural design up front?

What's wrong with letting the architecture emerge from the code?

What constitutes the "just right" level?

In practical terms, it's a question of when do we st coding

Here's one vie

When CAN yo start coding?

Goldilocks (again)

You can start coding (something) when you:

- Have a good (abstract) picture of that part of the system.
  - And …
- Have confidence that it (the picture) won't change a lot.

Or…

- You want to learn something

In other words:

- You can start some coding pretty early

BUT

- By definition, design precedes coding

Bottom line:

- The Waterfall folks aren't completely wrong, but aren't completely right.
- The XP folks aren't completely wrong, but aren't completely right.

Pitfalls...

Fallingwater (USA)

Designed by Frank Lloyd Wright, 1935

Unique, beautiful design

But plagued by problems:

- Structural stresses
- Building materials
- Leaks
- Mold

What went wrong?

Lessons from Fallingwater

The architecture must consider practical issues:

- Building: construction, laws of physics, etc.
- Software: how will the packages, libraries, etc., play together, performance, deployment, etc.

You have to consider the lifetime of the product:

- Building: will stresses cause problems later?
- Software: how easy is it to maintain? Will it evolve gracefully?

Architects do not live in ivory towers

But imagine how bad it would have been without a guiding architecture!

Quality Attributes

Fallingwater: Things like the ability to withstand stresses over time

We have lots in software

- We will talk a lot about them later

Quality attributes are system-wide properties

- Which leads to a problem…

Software Quality Attributes

When do you know for sure that:

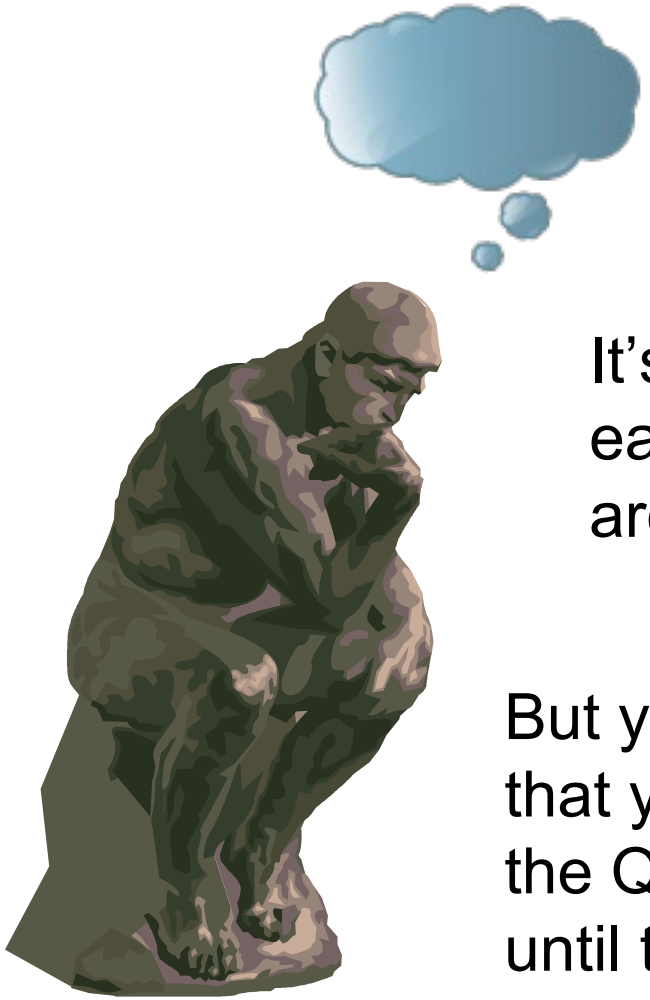
- Your software has sufficient performance?
- That it scales up easily?
- That it is secure against all types of security breaches?
- That it recovers gracefully from all types of faults?
- That is runs nonstop for months?

In general, at system test time. Why?

- Because you need to test the entire system
- In these aspects, the code is as strong as its weakest link

The Architect's Dilemma

Quality Attributes are strongly influenced by the Architecture



It's too late to easily change the architecture

And the architecture is designed EARLY in the project

But you don't know that you achieved the Quality Attributes until the system is complete!

Finding out Too Late …

The Vasa Warship 1628

- Sank on maiden voyage, less than 2km from shore
- Architecture did not support the top-heavy ship
- System test: 30 men ran from side to side on deck – Failed Stability Test!

Side Lesson from the Vasa

What does it say about changing requirements after the architecture is complete?

- (careful here!)

Conway's Law

The structure of the system and the structure of the committee inventing the system are isomorphic.

What does this mean?

Conway's Law

How does this apply to software projects?

Which comes first, the team or the architecture?

Design Exercise!

Divide into teams. Each team gets a paradigm:

Your paradigm:

Create a design for the following:

- The program inputs a positive integer, and outputs the equivalent Roman Numeral.

10-15 minutes: present a description (pictures, flowcharts, pseudo-code, or whatever you want)

1 = I

4 = IV

5 = V

9 = IX

10 = X

11 = XI

14 = XIV

15 = XV

50 = L

100 = C

Table driven (not strict): trivial

```
romn = [ 1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1]
roms = ['M', 'CM', 'D', 'CD', 'C', 'XC', 'L', 'XL', 'X', 'IX', 'V', 'IV', 'I' ]

def rom(n) :
   numeral = ''
   index = 0
   while (n > 0) :
      while (n >= romn[index]) :
         numeral = numeral + roms[index]
         n = n - romn[index]
      index = index + 1
   return numeral

for i in range (2000, 2026):
   print (rom(i))
```

What do we learn?

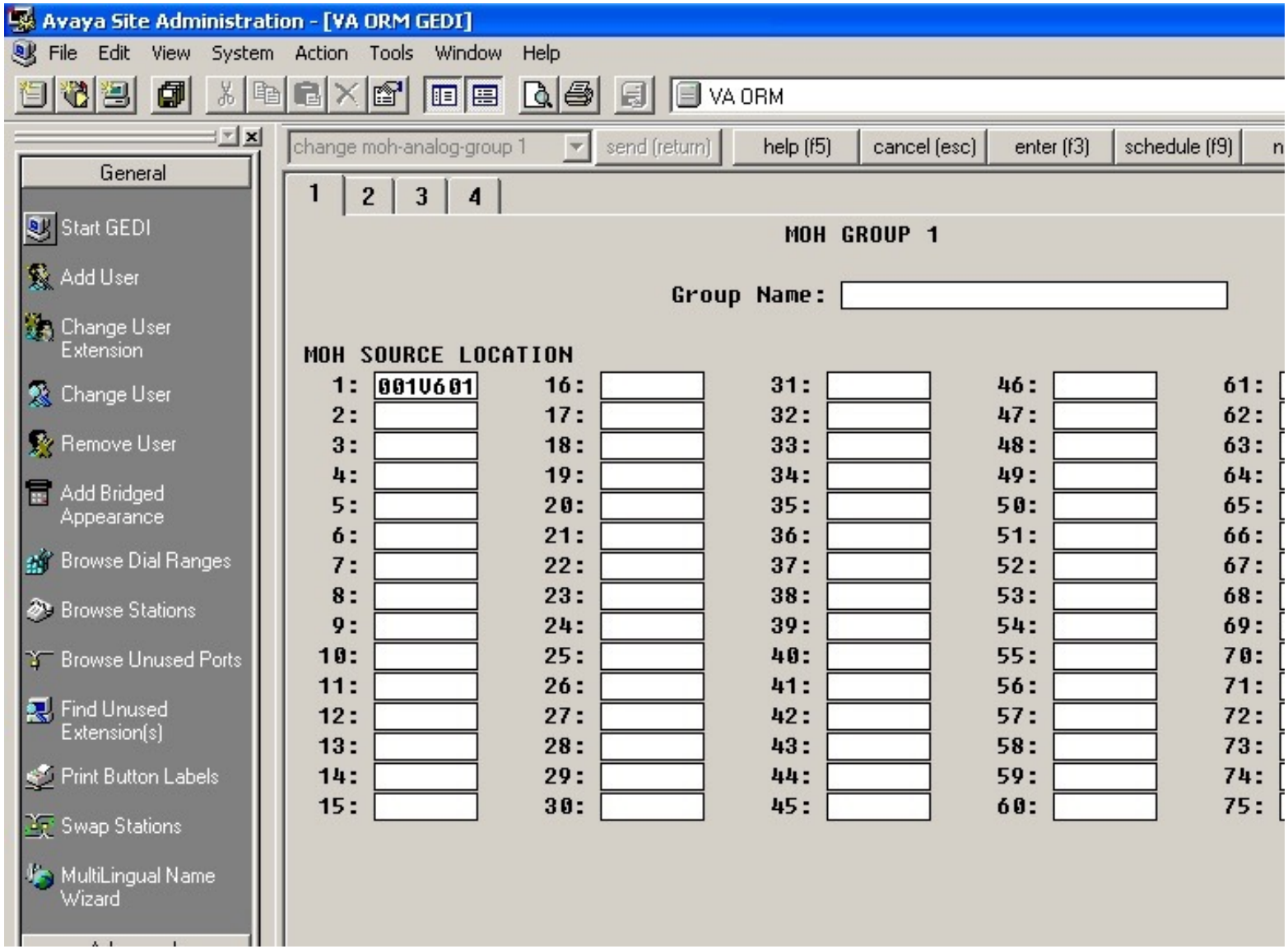
For a given problem, some approaches work better than others

- The programming paradigms kind of forced some partitions (especially the OO paradigm)
- … which may or may not match the natural partitioning of the solution

A major part of architectural design is deciding on the primary (high level) partitions

- Many are logical (classes, functions, data entities, etc.)
- Some can be physical (e.g., client/server)

Table driven: switching system example



OO: the Paradigm du jour

Dominant approach these days

What are the advantages?

But what are the disadvantages?

Basic Concepts in Software Architecture

Terminology

- Architecture
- Reference Architecture
- Descriptive vs. Prescriptive Architecture
- Component
- Connector
- Architectural Style
- Architectural Pattern

Models

Processes

Stakeholders

Architecture Definitions (from books)

"The structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time." Perry & Wolf (probably the best definition)

"A software system's architecture is the set of principal design decisions made about the system." (from the textbook)

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

Architecture as a set of Design Decisions?

One popular notion

It's general – more than the structure of the system

Design decisions include those related to:

- System structure
- Functional behavior
- Interaction
- Non-functional properties
- Implementation

Just about everything is a decision

I don't particularly like this notion of architecture

- It doesn't give me a good system-wide view
- But it is still a useful concept

Reference Architecture

A single software architecture for a family of related software systems

"A reference architecture is the set of principal design decisions that are simultaneously applicable to multiple related systems, typically within an application domain, with explicitly defined points of variation."

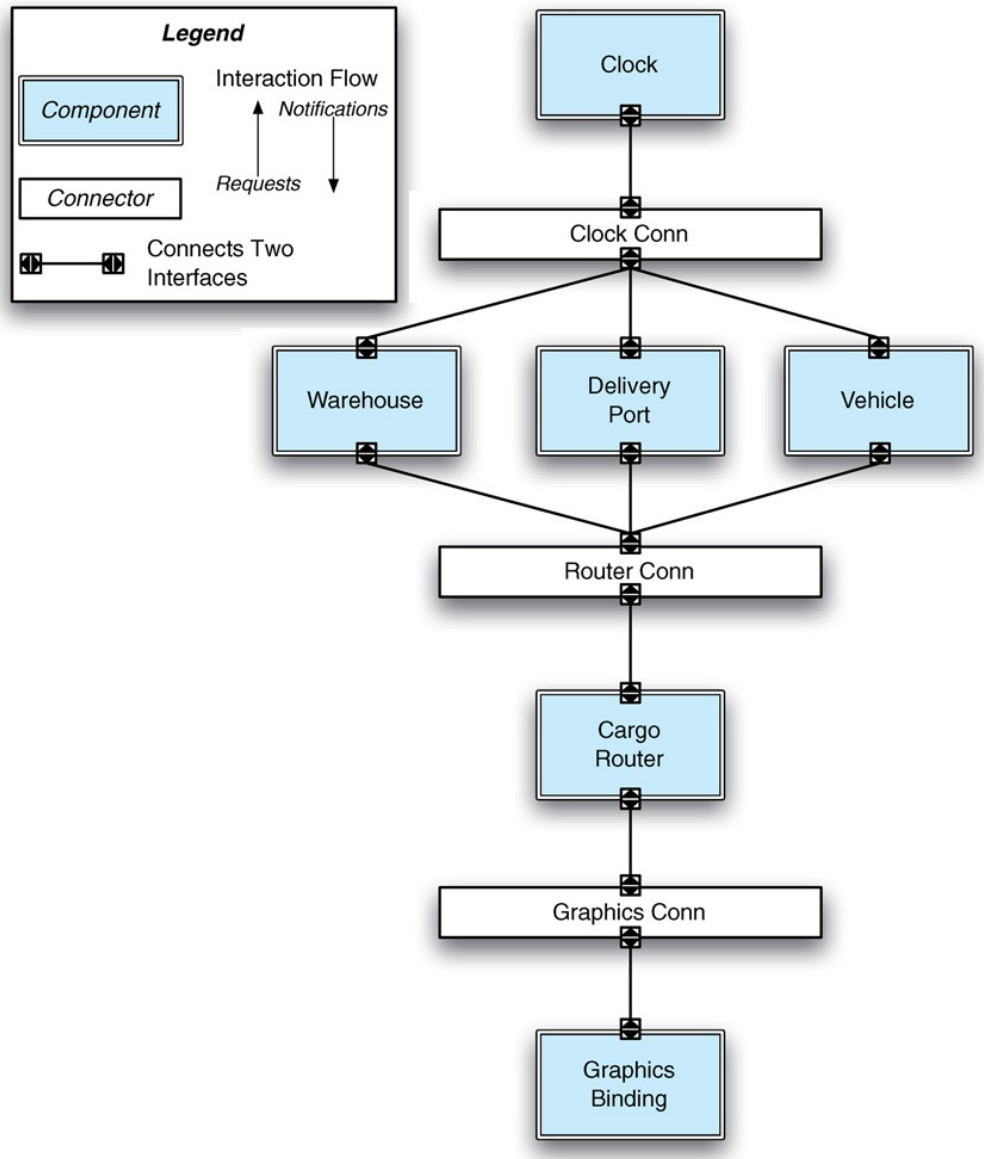
Descriptive vs. Prescriptive Architecture

Descriptive:

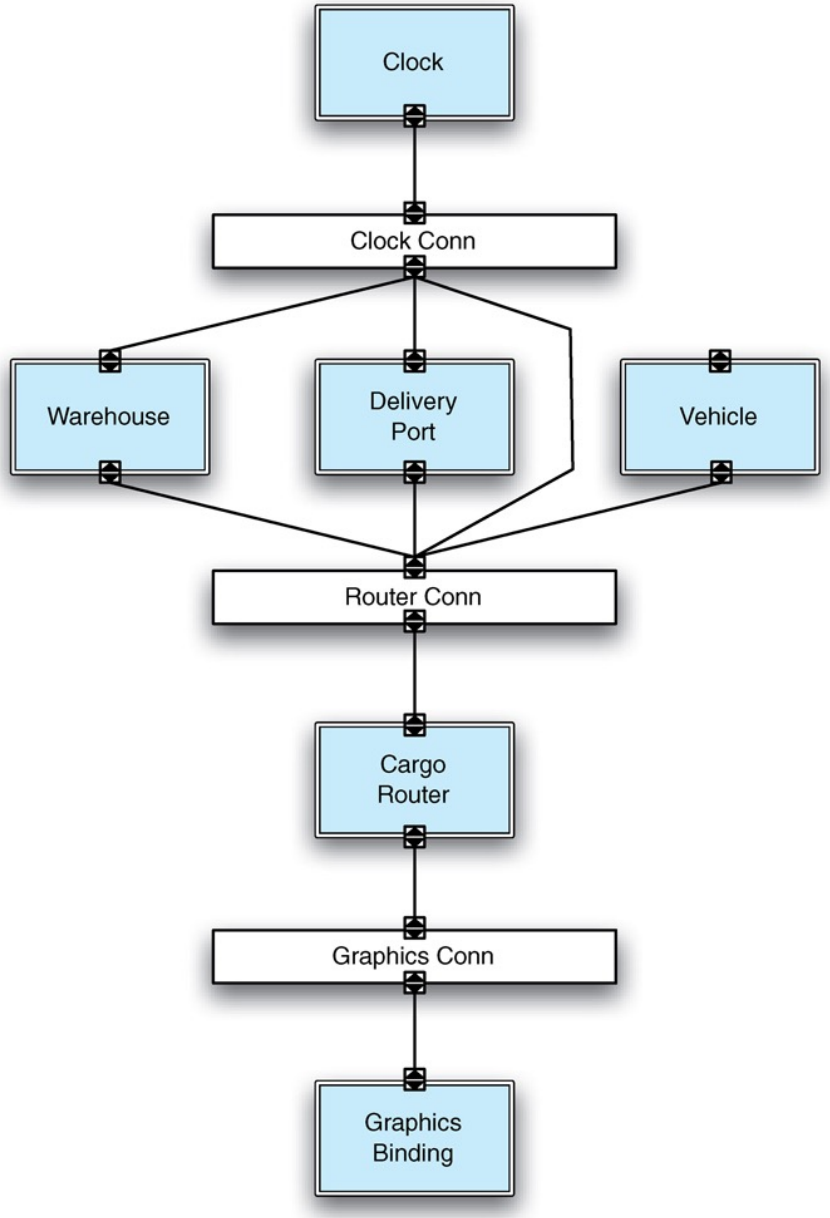
- Describes what is there
- Purposes:
  - Convey understanding of how a system is organized
  - Convey rationale for how the system is organized
  - Convey the theory of the program (includes the previous two)
  - Aid in maintenance and enhancement of the system
- Prescriptive
  - Describes the structure of the system to be created
  - Establish the theory of the program
  - Specifies some technical/development issues

Prescriptive: as imagined

A cargo routing application



Descriptive: what was implemented



Architectural Degradation

Architectural Drift:

- The introduction of principal design decisions into a systems descriptive architecture that are:
  - Not included in, encompassed by, or implied by its prescriptive architecture
  - Don't violate its design decisions

Architectural Erosion:

- Introduction of architectural design decisions into a systems descriptive architecture that:
  - Violate its architecture

My interpretation:

- Architectural drift moves the architecture
- Architectural erosion destroys it

Caveat: most literature doesn't distinguish, or calls everything "architectural drift".

Caveat #2: How do you distinguish between Drift and Erosion?

Component

(book): A software component is an architectural entity that

- Encapsulates a subset of the system's functionality and/or data
- Restricts access to that subset via an explicitly defined interface
- Has explicitly defined dependencies on its required execution context.

Another way of looking at it:

- A software component is a locus of computation and state in a system

Key aspect:

- You can "see" a component from the outside (and kind of ignore its innards)

Note: "component" is hard to pin down. The book's definition is as good as I've seen.

Connector

(book): A software connector is an architectural element tasked with effecting and regulating interactions among components

(Note: “effecting” means “to make happen”.)

Multiple types of connectors

Their importance is largely underappreciated

Architectural Styles, Patterns

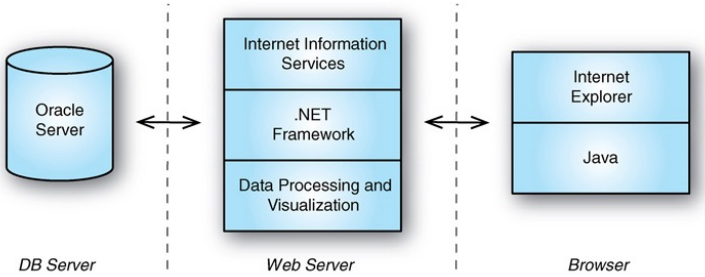
Some confusion about these terms: considerable overlap

Both are names for general architectural shapes

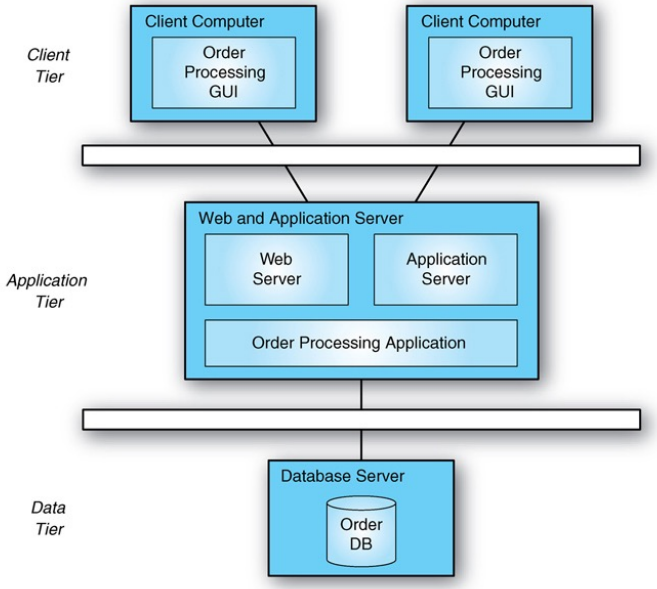
My opinion: architectural styles are broad architectural approaches, while patterns are often more context-specific.

Examples of 3-Tier Architecture pattern

Generic view:



Specific application: an order processing system



Architectural Model

(book): “An architectural model is an artifact that captures some or all of the design decisions that comprise a system’s architecture. Architectural modeling is the reification and documentation of those design decisions.”

(mine): An architectural model is a representation of the architecture. It’s usually expressed as a combination of text and pictures (although the text and pictures are NOT the model, but a description of it.) It helps us reason about the system (not just about the system’s architecture.)

(See Wikipedia)

References

Naur: Programming as Theory Building

Parnas: The Modular Structure of Complex Systems

- (Really what architecture is all about)

Architectural Recovery

(book) “Architectural recovery is the process of determining a software system’s architecture from its implementation artifacts.”

Artifacts include code and documents

By definition, it extracts a system’s descriptive architecture

- Note: It’s really hard to extract the theory of the program from the code alone

Architectural recovery tools may help (or not):

Find the Architecture

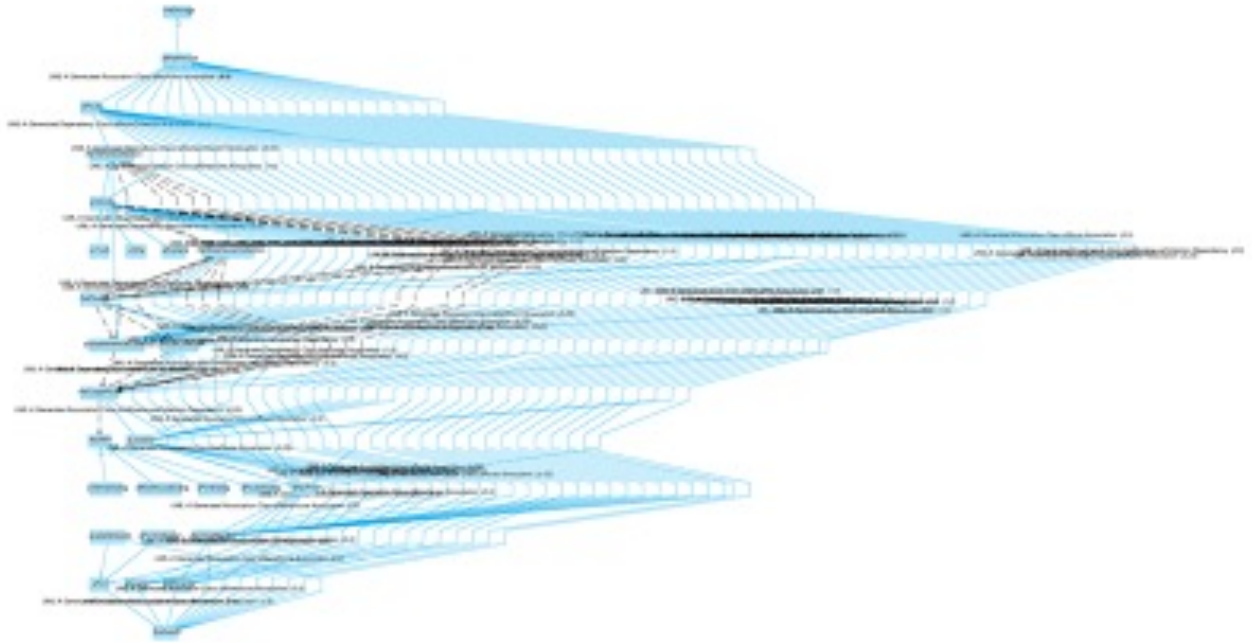


fig_03_06

Architectural Recovery

How would you go about it?

What information will you try to find?

Where to look?

(Assume a lack of architecture or design documentation)