Software Systems Architecture

FEUP-M.EIC-ASSO-2023-2024

Ademar Aguiar, Neil Harrison

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Visualizing/Documenting Architectures

"An architectural visualization defines how architectural models are depicted, and how stakeholders interact with those depictions."

A picture or visual representation of the architecture

Pretty important: what good is an architecture if we can't see it?

There is more than one way to see an architecture

Your Experience so far:

In the architecture documents you have seen:

What things do you like?

- Give examples
- Why do you like them?

What things do you NOT like?

- Give examples
- Why do you not like them?

Documentation Basic Concepts

Good Architecture documentation contains both:

- Diagrams
- Text

The diagrams generally show the modules

- And the connections among the modules

The text does many things:

- Explains diagrams
- Explains tradeoffs made
- Explains rationale
- Explains the use of the system
- And more

More basics

The goal of architecture documentation is to help people understand the architecture

Different people have different roles

- And need different views of the architecture

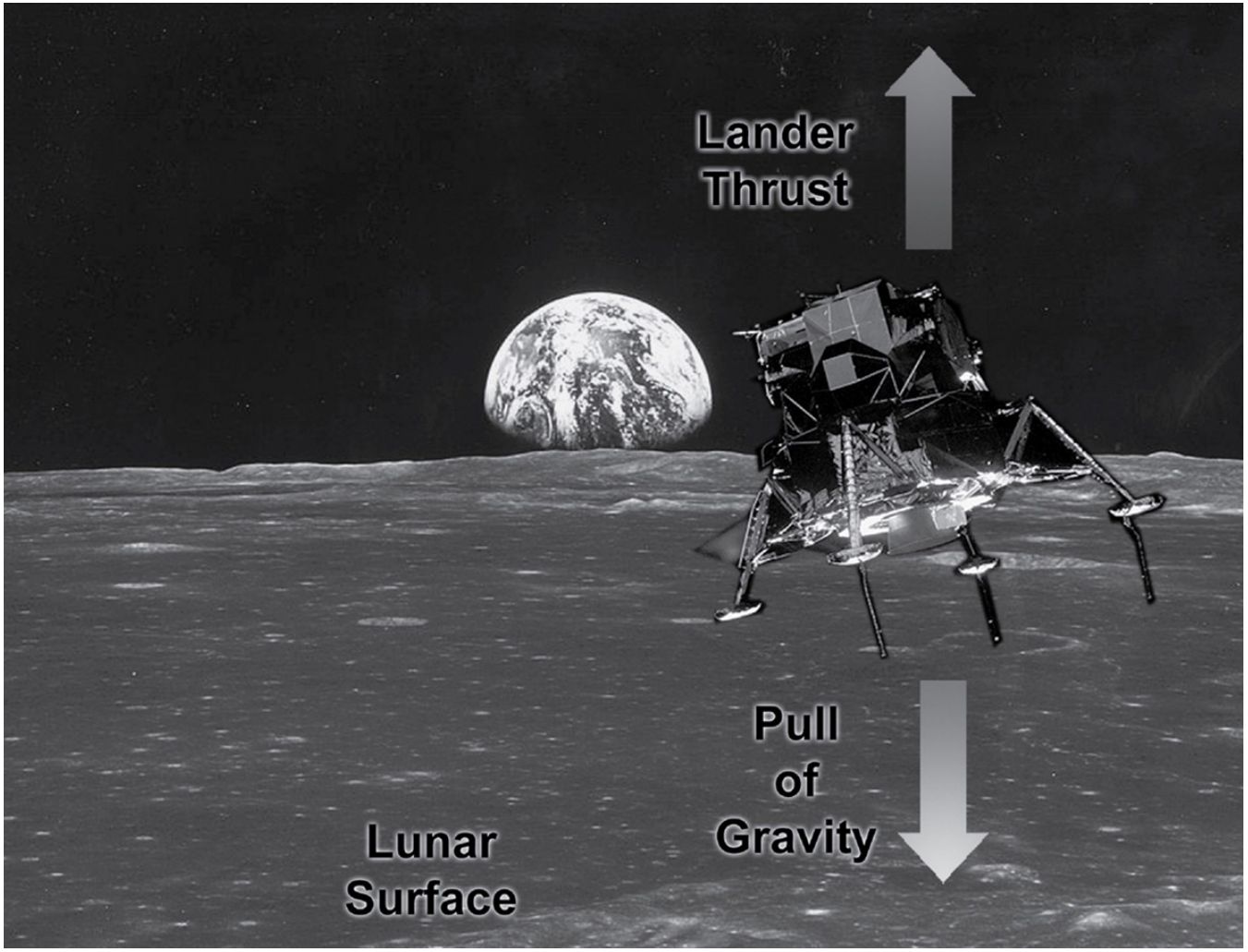
Assume that the reader does not know everything that you know about the system.

A gallery of visualizations

Which of the following do you like, or not like?

Why?

Lunar Lander1: Picture



Your reaction?

Can easily see general idea

Can see some important aspects of the system (thrust, gravity, surface of the moon)

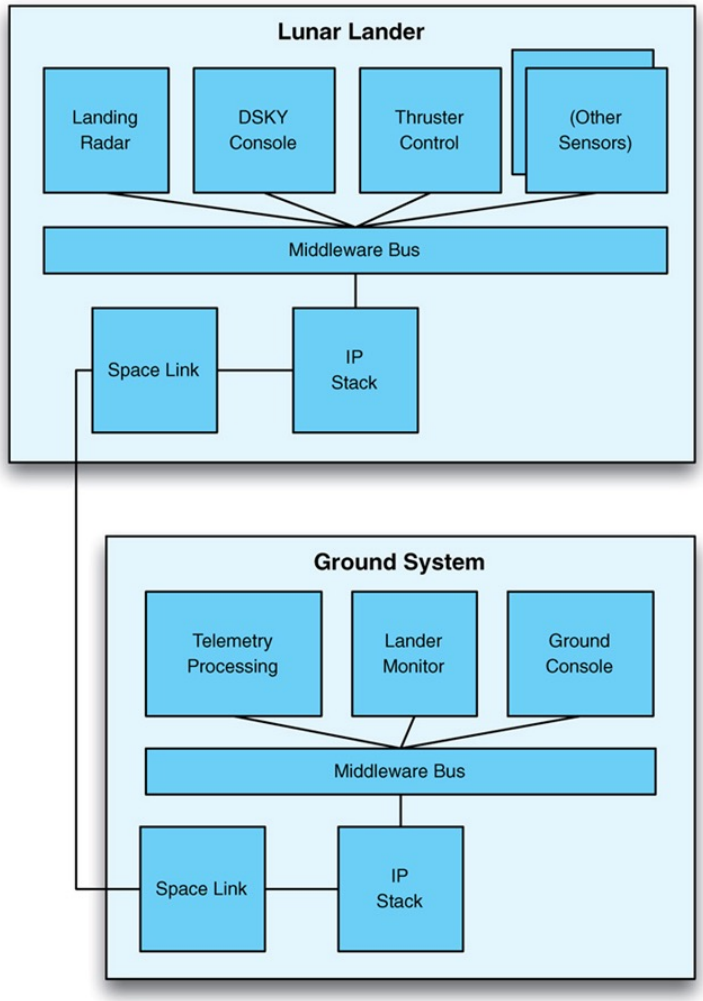
But inaccurate:

- why is the lunar lander tilted?
- May imply a three-dimensional aspect to the game

Doesn't give any info about the structure of the software

Is it useful as an implementation guide?

Lunar Lander 2: Boxes and Lines



Your reaction?

Gives some idea of the components

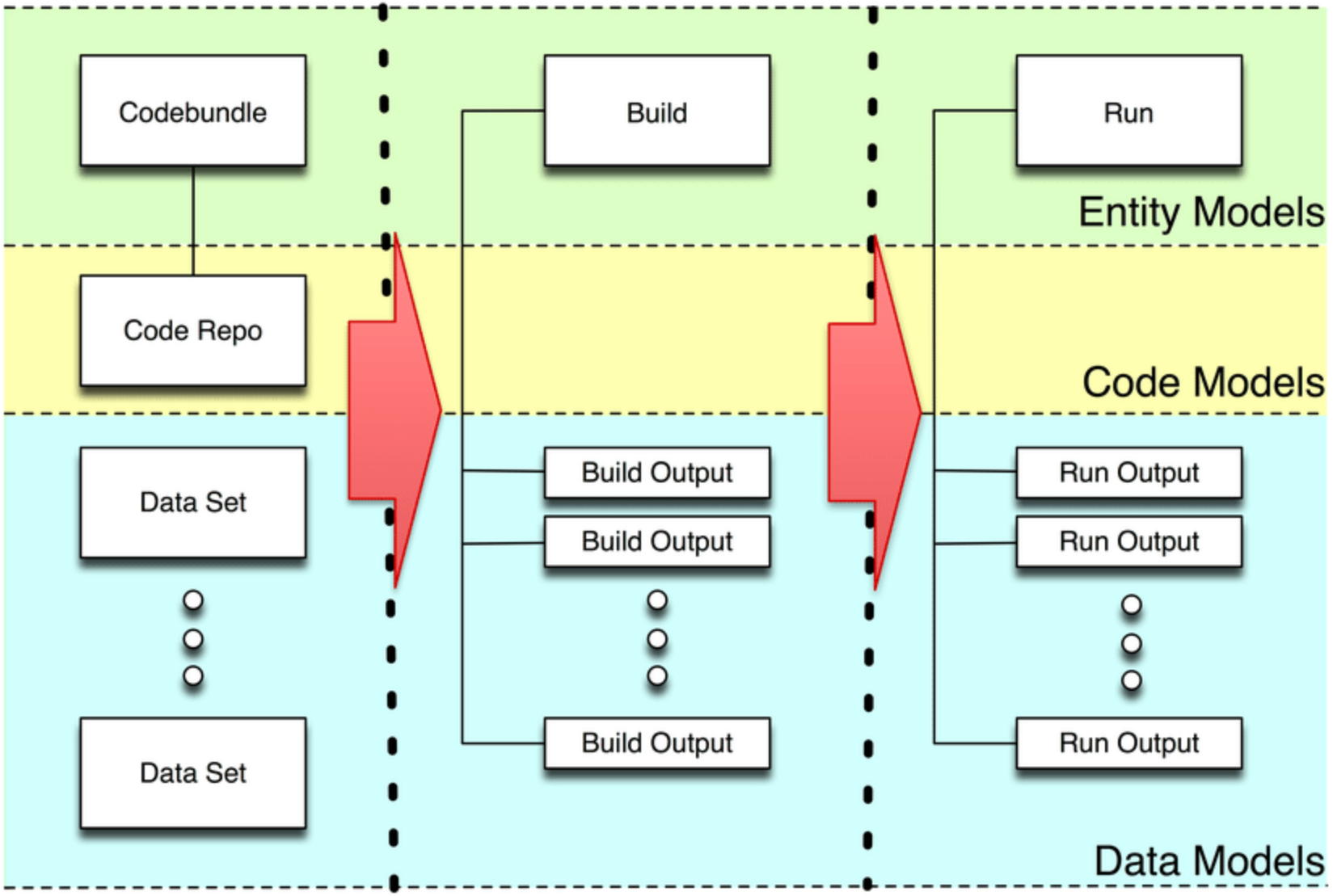
Gives some idea of how the components are connected to each other

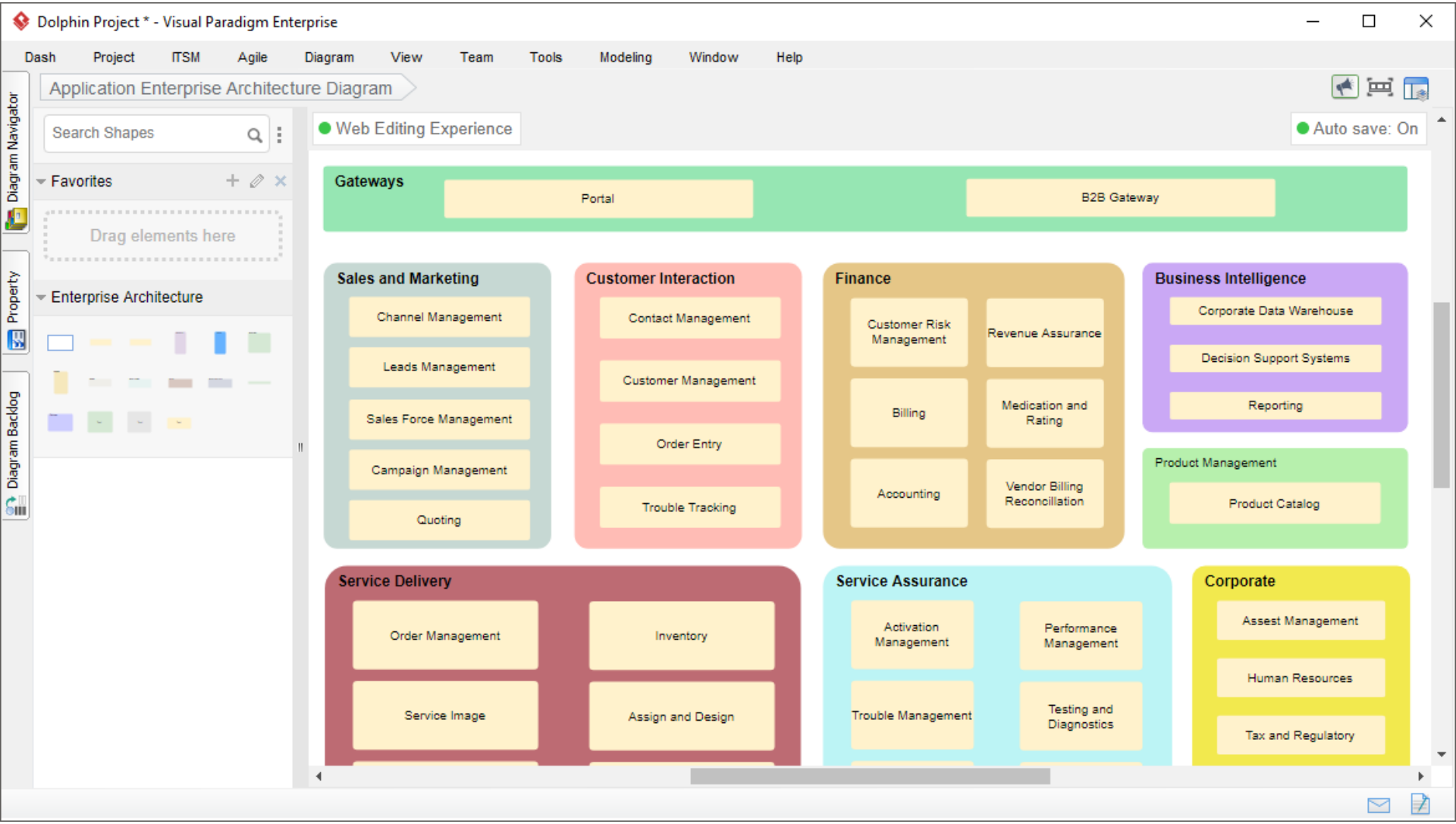
But lots of ambiguity!

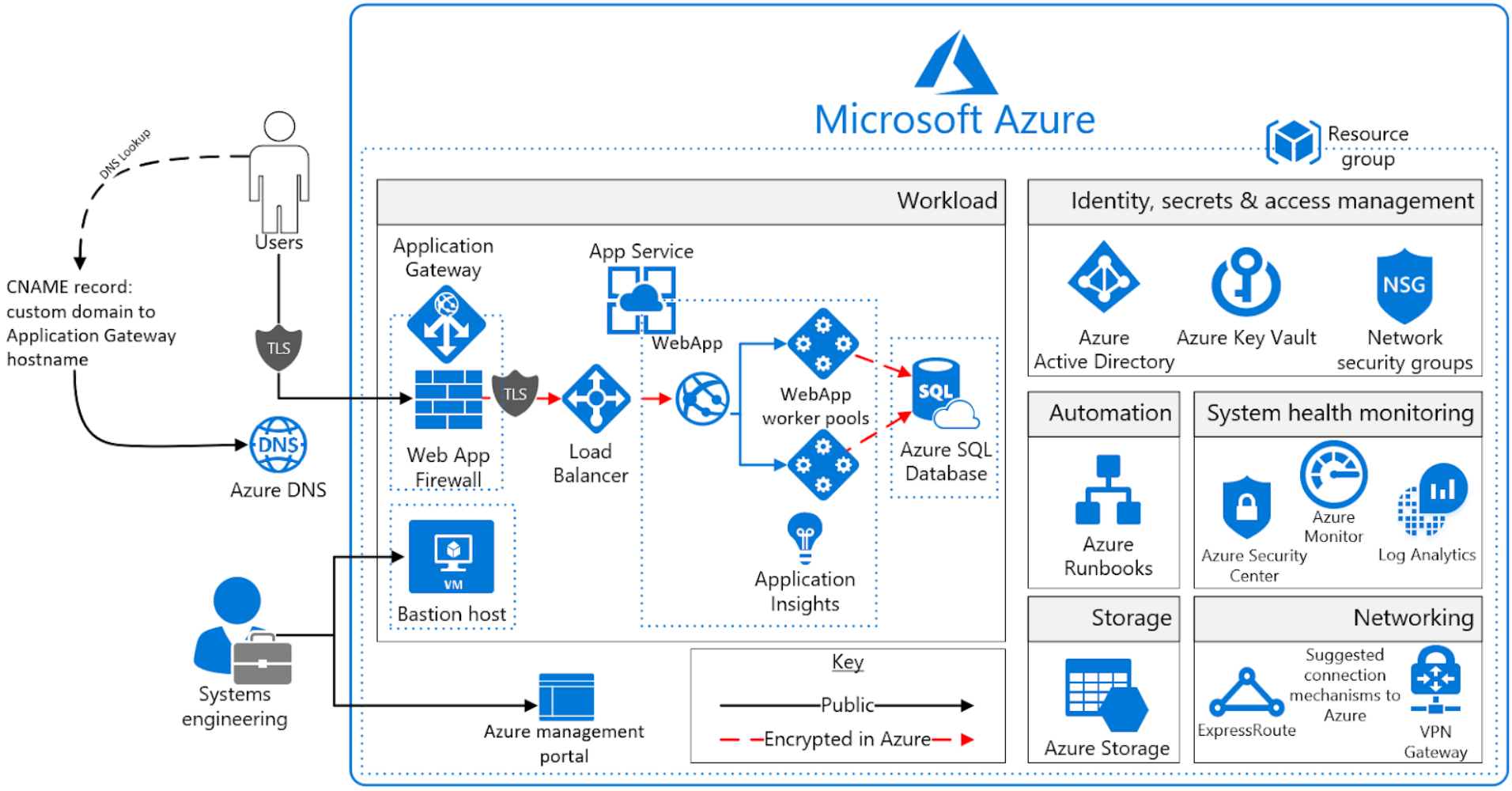
- Are all the dark blue boxes the same type of things? (um, no)
- What do the light blue boxes mean?
- What do the lines mean?

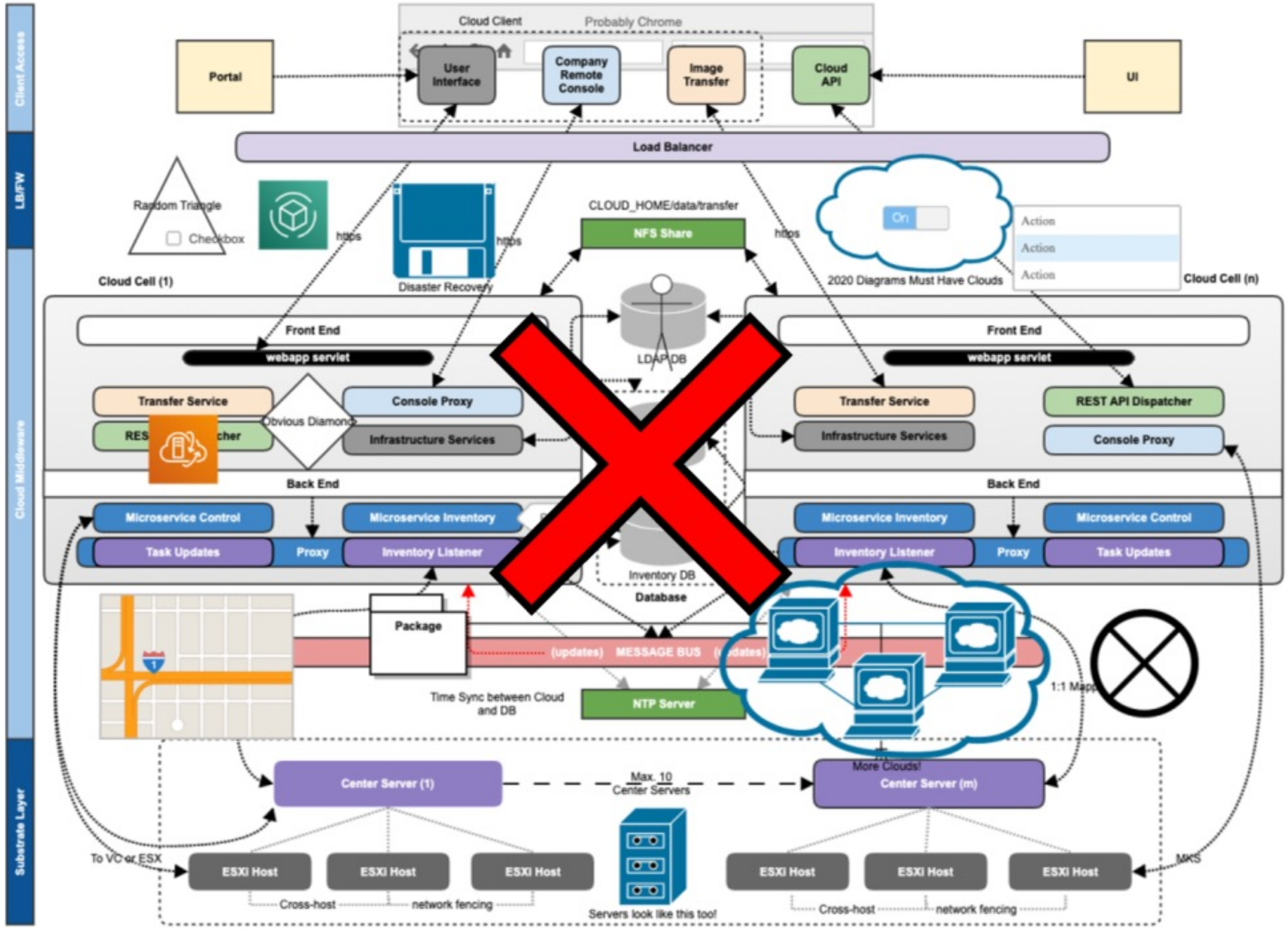
Can help you with implementation, but there are a lot of questions!

- Lots of room for misinterpretations


Codebundle
Build
Run
Entity Models
Code Repo
Code Models
Data Set
Build Output
Build Output
Run Output
Run Output
Data Set
Build Output
Run Output
Data Models


Dolphin Project * - Visual Paradigm Enterprise
Dash
Project
ITSM
Agile
Diagram
View
Team
Tools
Modeling
Window
Help
Application Enterprise Architecture Diagram
Diagram Navigator
Property
Diagram Backlog
Search Shapes
Favorites
Drag elements here
Enterprise Architecture
Web Editing Experience
Auto save: On
Gateways
Portal
B2B Gateway
Sales and Marketing
Channel Management
Leads Management
Sales Force Management
Campaign Management
Quoting
Customer Interaction
Contact Management
Customer Management
Order Entry
Trouble Tracking
Finance
Customer Risk Management
Revenue Assurance
Billing
Medication and Rating
Accounting
Vendor Billing Reconcillation
Business Intelligence
Corporate Data Warehouse
Decision Support Systems
Reporting
Product Management
Product Catalog
Service Delivery
Order Management
Inventory
Service Image
Assign and Design
Service Assurance
Activation Management
Performance Management
Trouble Management
Testing and Diagnostics
Corporate
Assest Management
Human Resources
Tax and Regulatory


Microsoft Azure
Resource group
Workload
Identity, secrets & access management
Application Gateway
App Service
WebApp
WebApp worker pools
Web App Firewall
Load Balancer
Azure SQL Database
Application Insights
Bastion host
Azure Active Directory
Azure Key Vault
Network security groups
NSG
Automation
Azure Runbooks
System health monitoring
Azure Security Center
Azure Monitor
Log Analytics
Storage
Azure Storage
Networking
ExpressRoute
Suggested connection mechanisms to Azure
VPN Gateway
Users
DNS Lookup
CNAME record: custom domain to Application Gateway hostname
TLS
TLS
DNS
Azure DNS
VM
Systems engineering
Azure management portal
Key
Public
Encrypted in Azure


Client Access
LB/FW
Cloud Middleware
Substrate Layer
Cloud Client
Probably Chrome
Portal
User Interface
Company Remote Console
Image Transfer
Cloud API
UI
Load Balancer
Random Triangle
Checkbox
https
Disaster Recovery
CLOUD_HOME/data/transfer
NFS Share
On
Action
Action
Action
2020 Diagrams Must Have Clouds
Cloud Cell (1)
Cloud Cell (n)
Front End
webapp servlet
Transfer Service
Console Proxy
Obvious Diamond
Infrastructure Services
REST API Dispatcher
Back End
Microservice Control
Microservice Inventory
Task Updates
Proxy
Inventory Listener
LDAP DB
Inventory DB
Database
Package
(updates) MESSAGE BUS
Time Sync between Cloud and DB
NTP Server
1:1 Mapp
More Clouds!
Center Server (1)
Max 10 Center Servers
Center Server (m)
To VC or ESX
ESXi Host
Cross-host
network fencing
Servers look like this too!
MKS

Textual: xADL (an architectural description language)

```
<instance:xArch xsi:type="instance:XArch">
 <types:archStructure xsi:type="types:ArchStructure"
                      types:id="ClientArch">
  <types:description xsi:type="instance:Description">
   Client Architecture
  </types:description>
  <types:component xsi:type="types:Component"
                   types:id="WebBrowser">
   <types:description xsi:type="instance:Description">
    Web Browser
   </types:description>
   <types:interface xsi:type="types:Interface"
                    types:id="WebBrowserInterface">
    <types:description xsi:type="instance:Description">
     Web Browser Interface
    </types:description>
    <types:direction xsi:type="instance:Direction">
     inout
    </types:direction>
   </types:interface>
  </types:component>
 </types:archStructure>
</instance:xArch>
```

```
xArch{
 archStructure{
  id = "ClientArch"
  description = "Client Architecture"
  component{
   id = "WebBrowser"
   description = "Web Browser"
   interface{
    id = "WebBrowserInterface"
    description = "Web Browser Interface"
    direction = "inout"
   }
  }
 }
}
```

Your reaction?

Precise

Can use tools to validate it, and to maybe generate some code

Not very easy to understand!

Can anyone tell me the structure of the architecture we just looked at?

NOTE #1: I have not seen something like this in practice.

Note #2: It is fundamentally WRONG!

- It is precise, but software architecture is **by definition imprecise**

What makes a good visualization?

(By extension, what are characteristics of a good visualization tool?)

Fidelity

How faithfully does a view represent the underlying model?

Everything displayed should be absolutely correct (matches the model)

But not everything in the model must appear in the model

- In fact, it's often a good idea to show only certain aspects of the model

Note that fidelity affects both depiction and interaction

- For example, some visualization tools may allow the user to make changes

Consistency

Are similar concepts displayed in a consistent manner?

Is a visualization mechanism (e.g., a box) used for different concepts?

In other words, is the visualization consistent with itself? (internal consistency)

Example, UML:

- UML always depicts an object as a rectangle with an underlined name
- But a dashed open-headed arrow means “dependency” in most UML diagrams, but it means “asynchronous invocation” in a UML sequence diagram

Note: there is a balance: extreme consistency can lead to a huge and confusing variety of symbols

Comprehensibility

VERY important

Should be able to understand it rather quickly

Notation, diagrams should be intuitive

Helps to keep scope of the visualization narrow

Don’t try to display too much information at once

Note the people may bring assumptions about what symbols may mean: don’t violate them

Dynamism

How well does the visualization support models that change over time

Information flows two ways:

- Change the model → change the visualization
- Change the visualization → change the model

Closely related to the tool(s) used

Note: pretty simple if the model has a single integrated visualization. Not so simple if there are multiple visualizations of the model.

View Coordination

Multiple views of a model must be consistent with each other

Note that a person might use multiple views simultaneously.

Aesthetics

Yes, it should look good

Lots of material available on how to make things look good

Yes, it is very subjective

Doesn’t seem important, but it might make a difference in acceptance by potential users

(See materials by Edward Tufte)

Edward Tufte (quick summary)

"The Visual Display of Quantitative Information"

- Lessons apply to architecture diagrams too

Parsimony:

- Data-ink ratio
- Avoid chartjunk (distractions: focus on the information)
- Does NOT mean minimalizing information

Envisioning information:

- 3-dimensional depictions
- Layering (multiple depictions that can overlay on each other)
- Time variation (different depictions over time)
- Small multiples (many small depictions of related data)

Extensibility

How easy is it to modify a visualization?

Tools can help (or not) (e.g. drawing tools)

Ability to add new capabilities

Grok Factor

(English geek-slang)

When you look at a visualization, how long does it take you to understand it?

It should be short

Constructing a Visualization

Borrow elements from Similar Visualizations

- The idea: if you use a common shape, the reader should immediately know what it stands for

Be consistent among visualizations

Give meaning to each visual aspect of elements

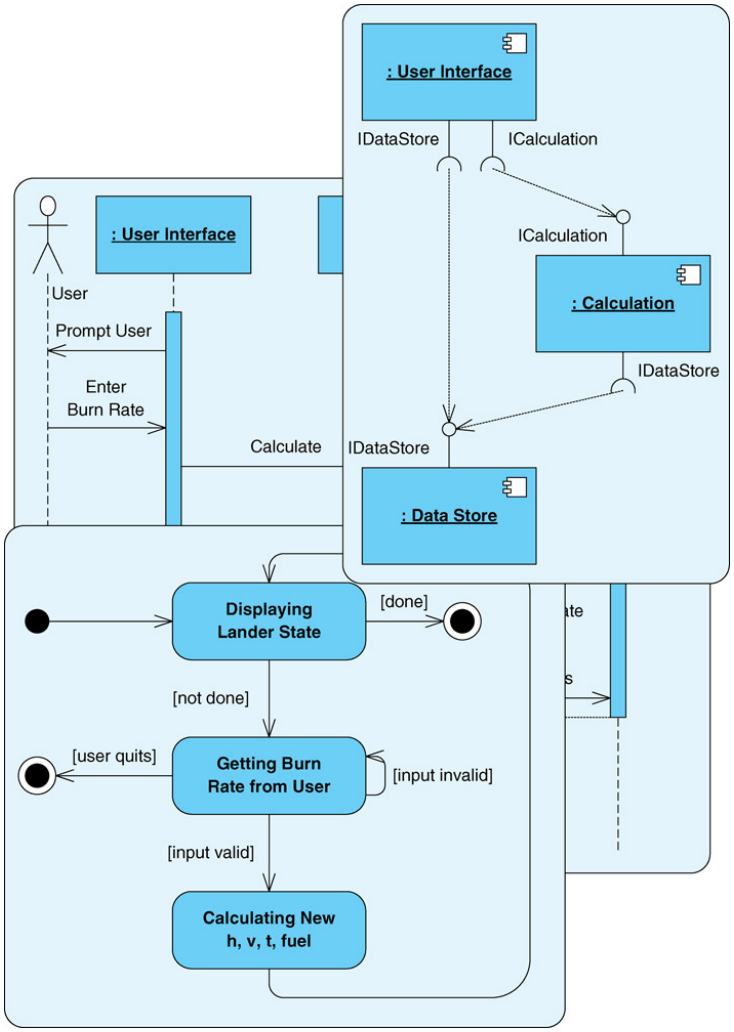
- For example, if you use colors

Document the meaning of visualizations

Balance traditional and innovative (user) interfaces

- E.g., traditional look and feel: PowerPoint, Visio

Example: different views in UML



Options

Use existing architecture visualization tools/notations

- UML is the most standard
- It's kind of heavy

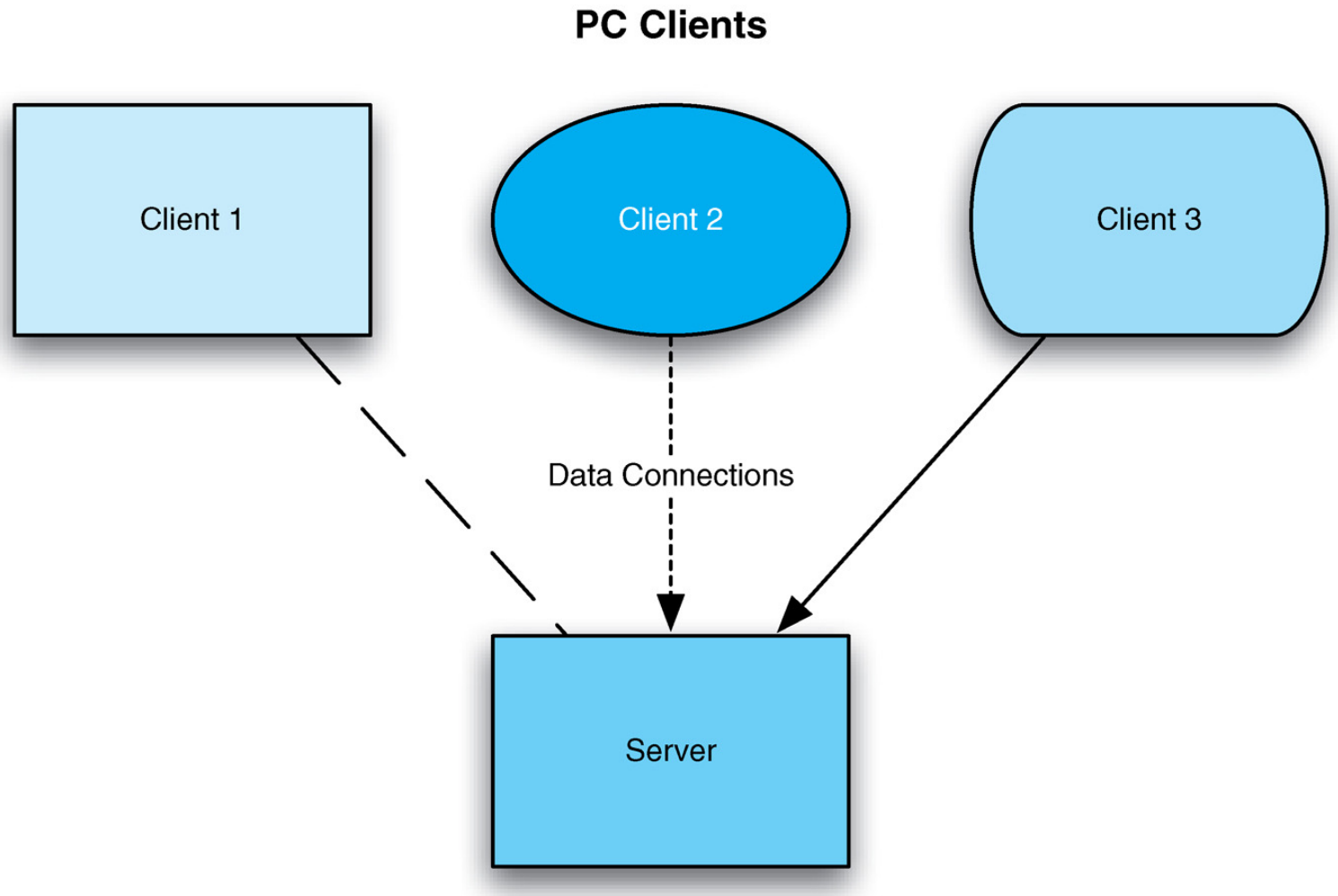
Make up your own diagrams

- Often easier
- But can easily lead to ambiguity and other problems!
- My Experience: Nearly everybody makes up their own styles

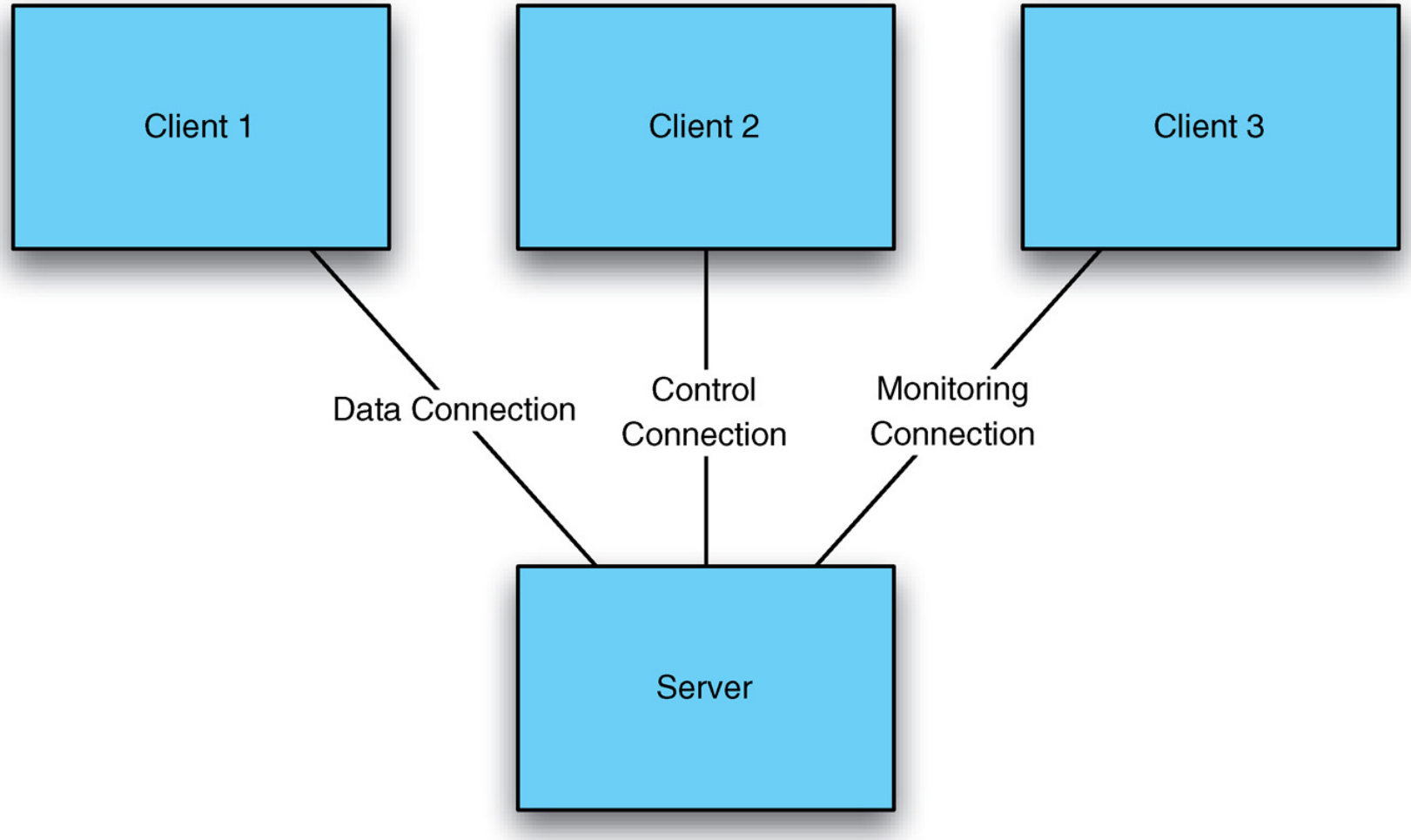
Common Issues (see next slides)

- Mostly in the "make your own" styles
- But can appear even with the use of standard notations

What is wrong with this picture?



What is wrong with this picture?



Notes on previous slide:

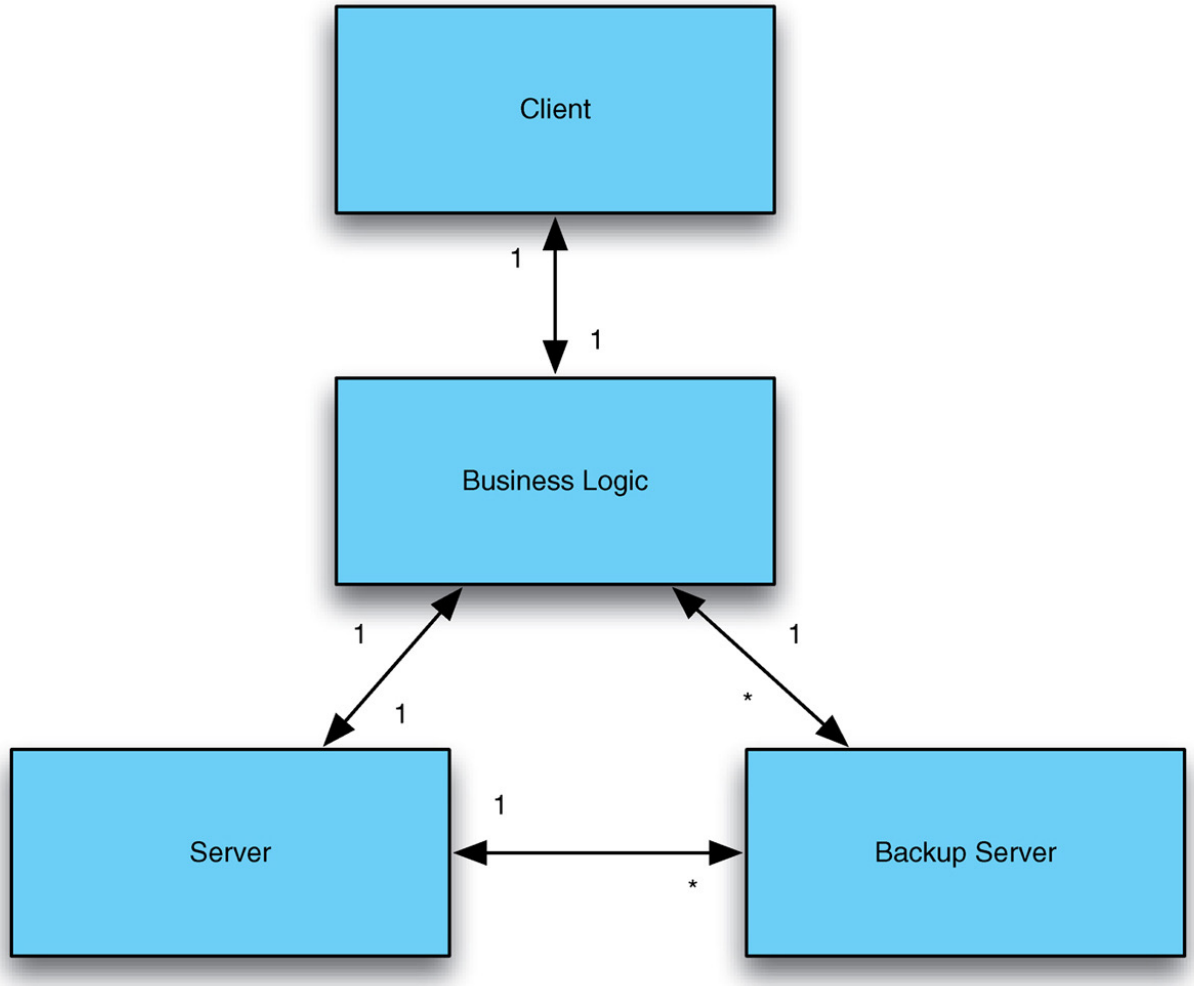
Clients and Server have the same size and color boxes

- That might be ok.
- But they need different labels, of course

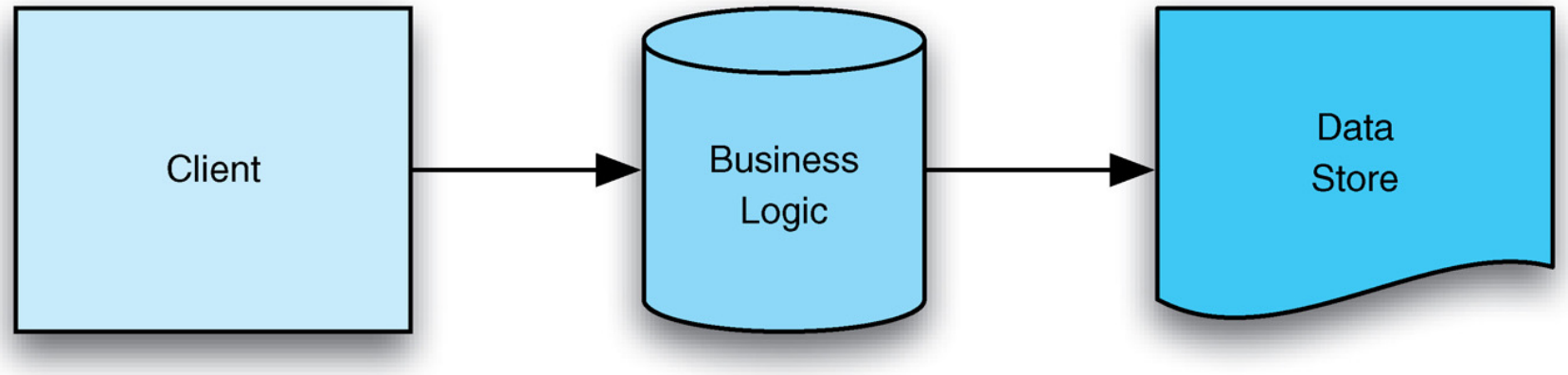
Different types of connectors use the same thing: lines

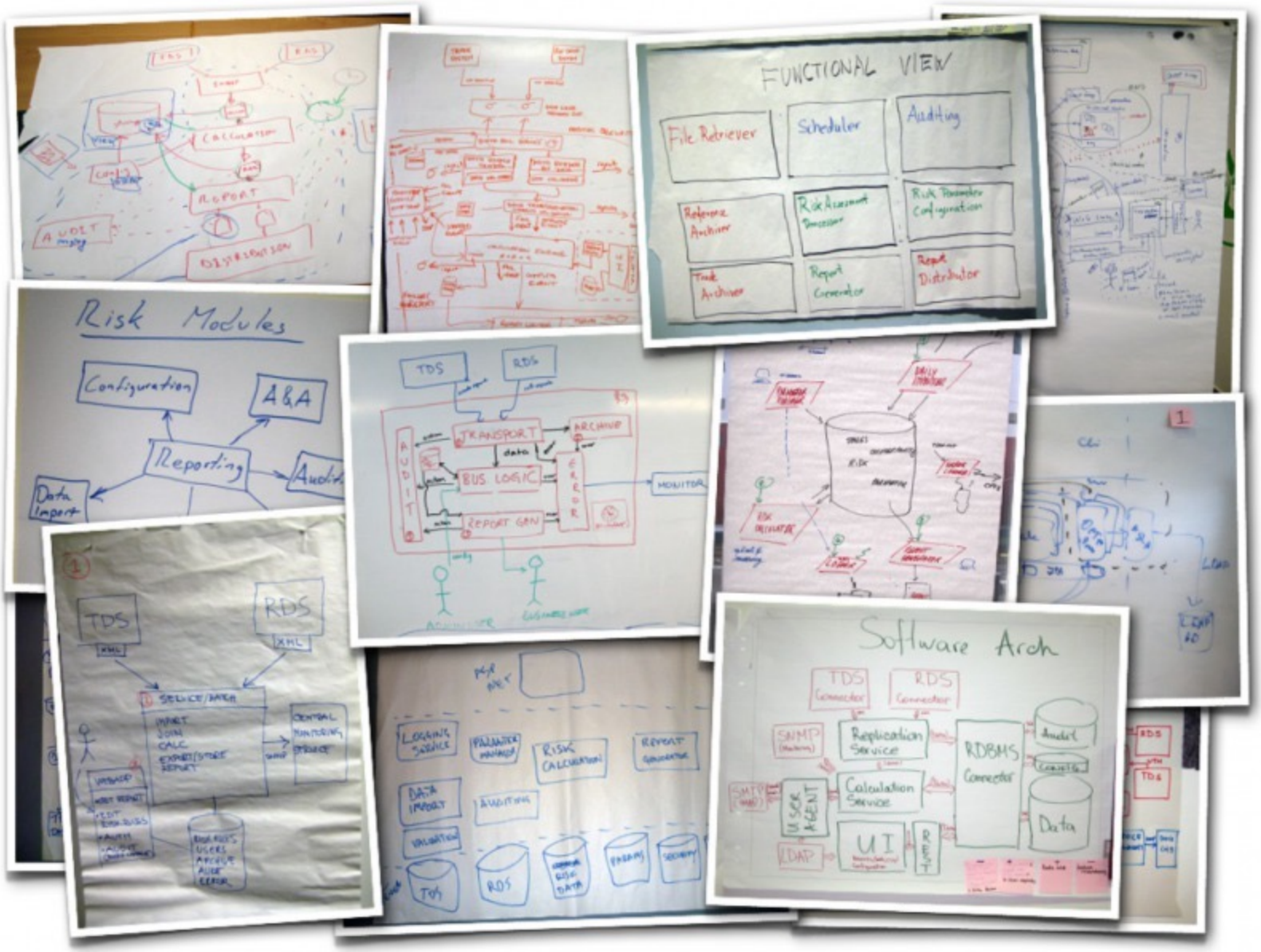
- That might also be ok.
- If you have different types of lines, you need a legend, and it gets messy.
- It may be clearest just to have labels and explanations.

Issue: decorations without meaning



What is wrong with this picture?




FUNCTIONAL VIEW
File Retriever
Scheduler
Auditing
Reference Archiver
Risk Assessment Processor
Risk Parameter Configuration
Trade Archiver
Report Generator
Report Distributor
Risk Modules
Configuration
A&A
Reporting
Data Import
TDS
RDS
TRANSPORT
ARCHIVE
BUS LOGIC
REPORT GEN
AUDIT
ERROR
MONITOR
Software Arch
TDS Connector
RDS Connector
SNMP
Replication Service
RDBMS Connector
Calculation Service
User Agent
UI
LDAP
Data
LOGGING SERVICE
RISK CALCULATION
REPORT GENERATOR
DATA IMPORT
AUDITING
VALIDATION

Comments about the previous slide

Color-coding is not explained or is inconsistent.

The purpose of diagram elements (i.e. different styles of boxes and lines) is not explained.

Key relationships between diagram elements are missing or ambiguous.

Generic terms such as “business logic”, “service” or “data access layer” are used.

Technology choices (or options) are omitted.

Levels of abstraction are mixed.

Diagrams lack context or a logical starting point.

And yet, there is a lot of information there…

- But do we get it right?

Source: https://www.voxxed.com/2014/10/simple-sketches-for-diagramming-your-software-architecture/

But …

The diagrams were probably VERY useful to the team who drew them up:

- They probably know what the colors mean
- They probably have all the necessary context
  - (in their heads!)
- It's probably perfectly clear to them
- → They probably drew them while they were designing the architecture!
- → In other words, it is probably a decent model.

The trick is to make the diagrams meaningful to other people

- You probably must redo your initial diagrams (and add the information that's in your head)

Theory vs. Practice Alert!

Next slides show the intention

“Official” approach

4+1 view model architecture
(for distributed systems)

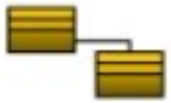
1. **Logical View (or Structural View)** - *an object model of the design*
2. **Process View (or Behavioral View)** - *concurrency and synchronization aspects*
3. **Development View (or Implementation View)** - *static organization (subset) of the software*
4. **Physical View (or Deployment View)** - *mapping of the software to the hardware*

+1. Use-cases View (or Scenarios) - *various usage scenarios*

1. Logical View = (The Object-oriented Decomposition)

- **Viewer:** End-user
- **Considers:** Functional Requirements- What the system should provide in terms of services to its users.

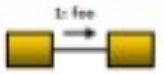
- **What this view shows?**
- *"This view shows the components (objects) of the system as well as their interaction/relationship".*
- Notation: Object and dynamic models
- **UML diagrams:** Class, Object, State Machine, Composite Structure diagrams



2. Process View = (The Process Decomposition)

- **Viewer:** Integrator(s)
- **Considers:** Non-Functional Requirements - (concurrency, performance, scalability, usability, resilience, re-use, comprehensibility, economic and technology constraints, trade-offs, and cross-cutting concerns - like security and transaction management)

- **What this view shows?**
- *"This view shows the processes (workflow rules) of the system and how those processes communicate with each other".*
- **UML diagrams:** Sequence, Communication, Activity, Timing, Interaction Overview diagrams



3. Development/Implementation View

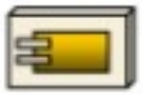
= (The Subsystem Decomposition)

- **Viewer:** Programmers and Software Managers
- **Considers:** Software module organization (Hierarchy of layers, software management, reuse, constraints of tools)
- **What this view shows?**
- *"This view shows the building blocks of the system".*
- **UML diagrams:** Component, Package diagrams
  - Package Details, Execution Environments, Class Libraries, Layers, Sub-system design

4. Physical/Deployment View

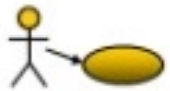
= (Software to Hardware Mapping)

- **Viewer:** System Engineers
- **Considers:** Non-functional Requirements for hardware (Topology, Communication)
- **What this view shows?** Non-functional
- *"This view shows the system execution environment".*
- **UML diagrams:** Deployment diagrams
- **Non-UML diagrams:** Network Topology (not in UML)



5. Use-case View/Scenarios = (putting it altogether)

- **Viewer:** All users of other views and Evaluators
- **Considers:** System consistency, validity
- **What this view shows?**
- *"This view shows the Validation and Illustration of system completeness. This view is redundant with other views.".*
- **UML diagrams:** Use-case diagram, User stories



Relationships between Views

- The *Logical View* and the *Process View* are at a conceptual level and are used from analysis to design.
- The *Development View* and the *Deployment View* are at the physical level and represent the actual application components built and deployed.
- The *Logical* View and the Development View are tied closer to functionality (functional aspect) . They depict how functionality is modeled and implemented.
- The *Process* View and Deployment View realizes the non-functional aspects using behavioral and physical modeling.
- *Use Case* View leads to structural elements being ***analyzed*** in the Logical View and ***implemented*** in the Development View. The scenarios in the Use Case View are ***realized*** in the Process View and ***deployed*** in the Physical View.

Why is it called the 4 + 1 instead of just 5?

- **The Use-case View:** The use case view has a special significance.

Views are effectively redundant (i.e. Views are interconnected).

- However, all other views would not be possible without use case view.
- It details the high levels requirements of the system.
- The other views detail how those requirements are realized.

The Real World

Common Practices:

- People rarely use all five views
  - Plus Use Cases/user stories
    - User stories more common
- Put logical and physical views in the same picture
- Dynamic views often neglected
- Connectors are usually poorly documented

Recommendations

- At least one static view, usually at least two
- Usually separate the physical and logical views
- Dynamic view is usually important
- Favor use cases over user stories
- Make connectors clear
- Notation (will cover that later)

“View”: One more thing

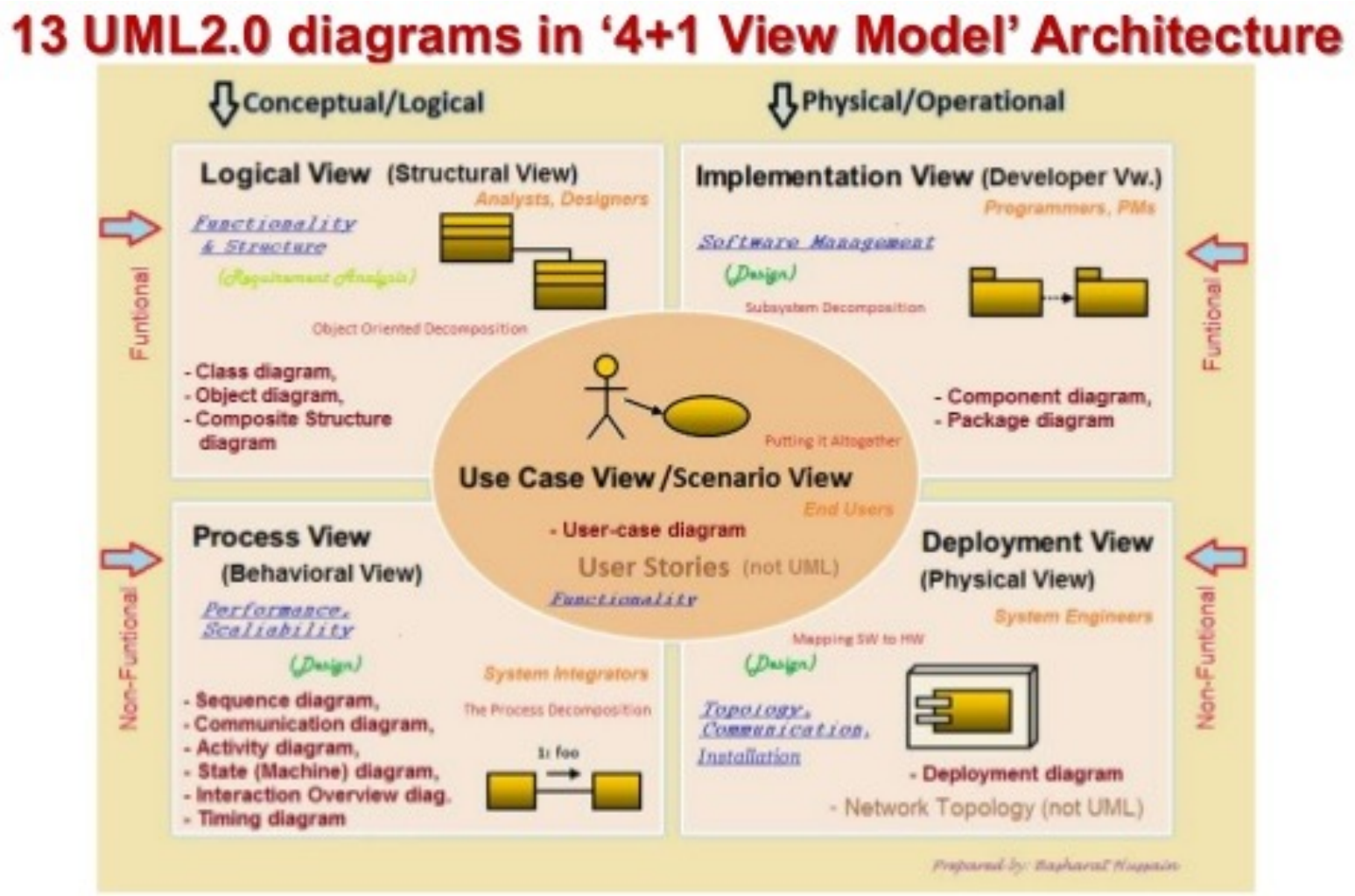
“View” doesn’t equate to “Diagram”

- A “View” is a “point of view”
- In other words: how someone looks at the architecture

Therefore:

- One view may have several diagrams
- A view may not have any diagrams at all
  - (Ok, that is rare)
  - Consider the “Deployment View”. Sometimes it is just a list of instructions in text.

Common: 4+1 Views

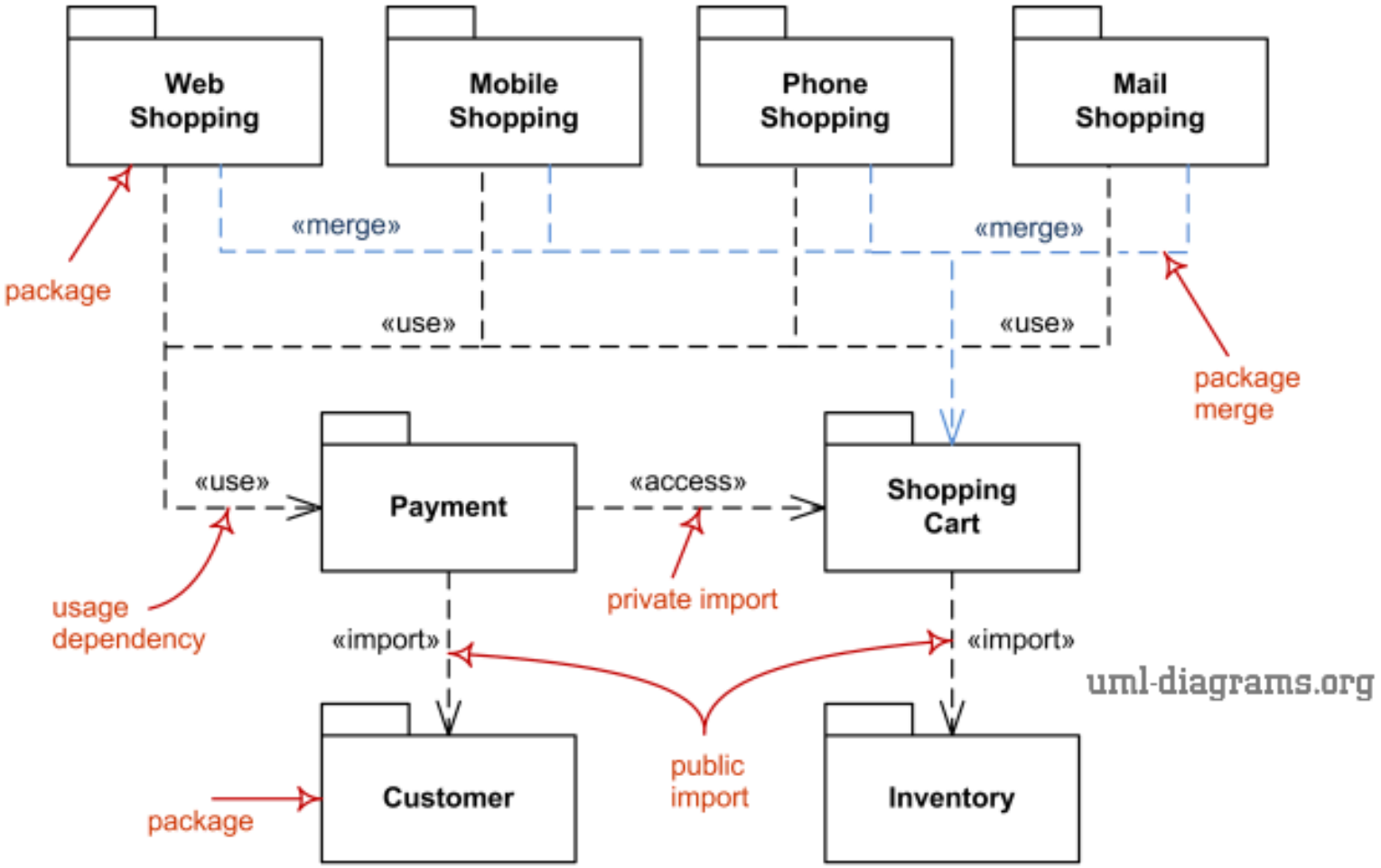


Notation

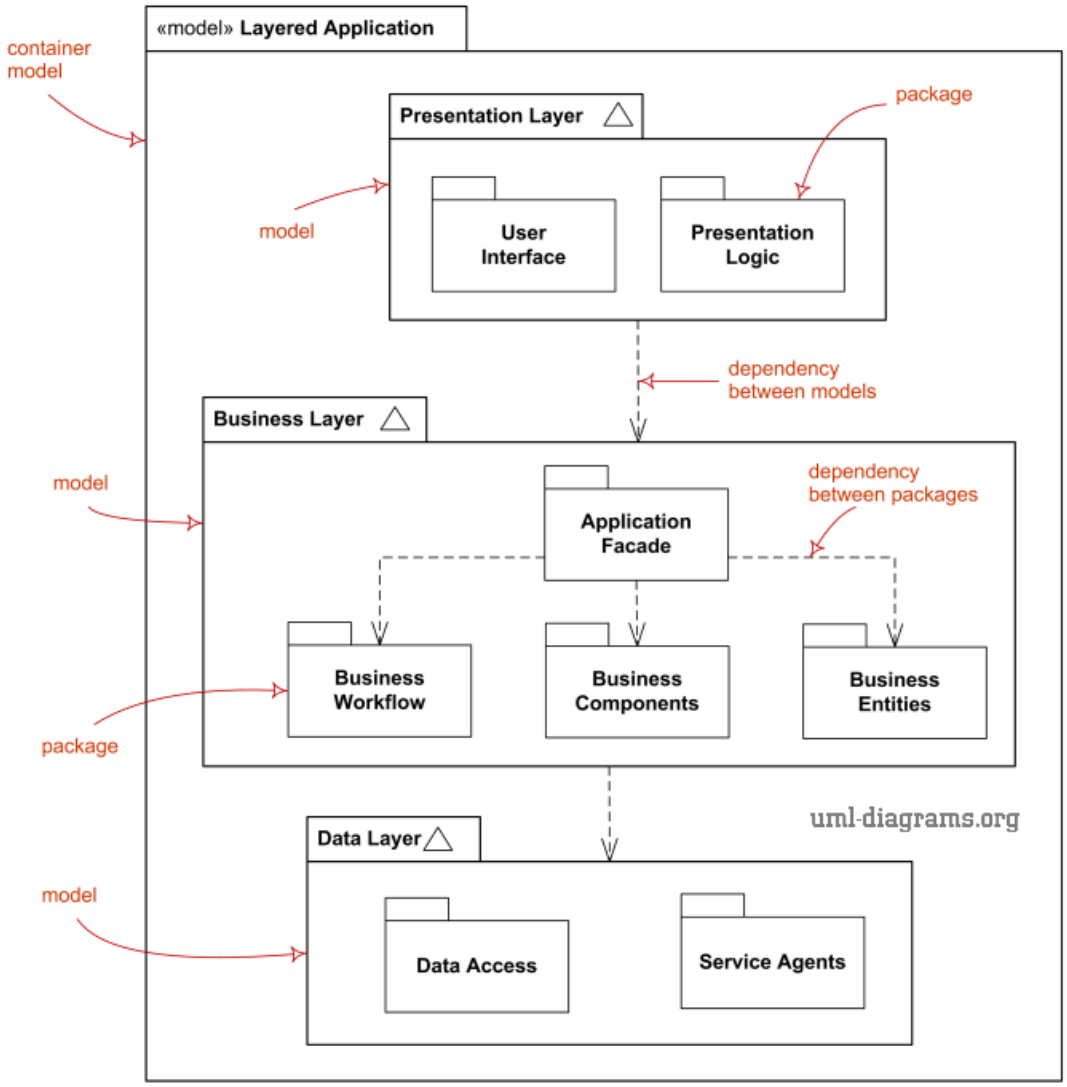
The next slides show UML notation

Note that not all UML is used commonly

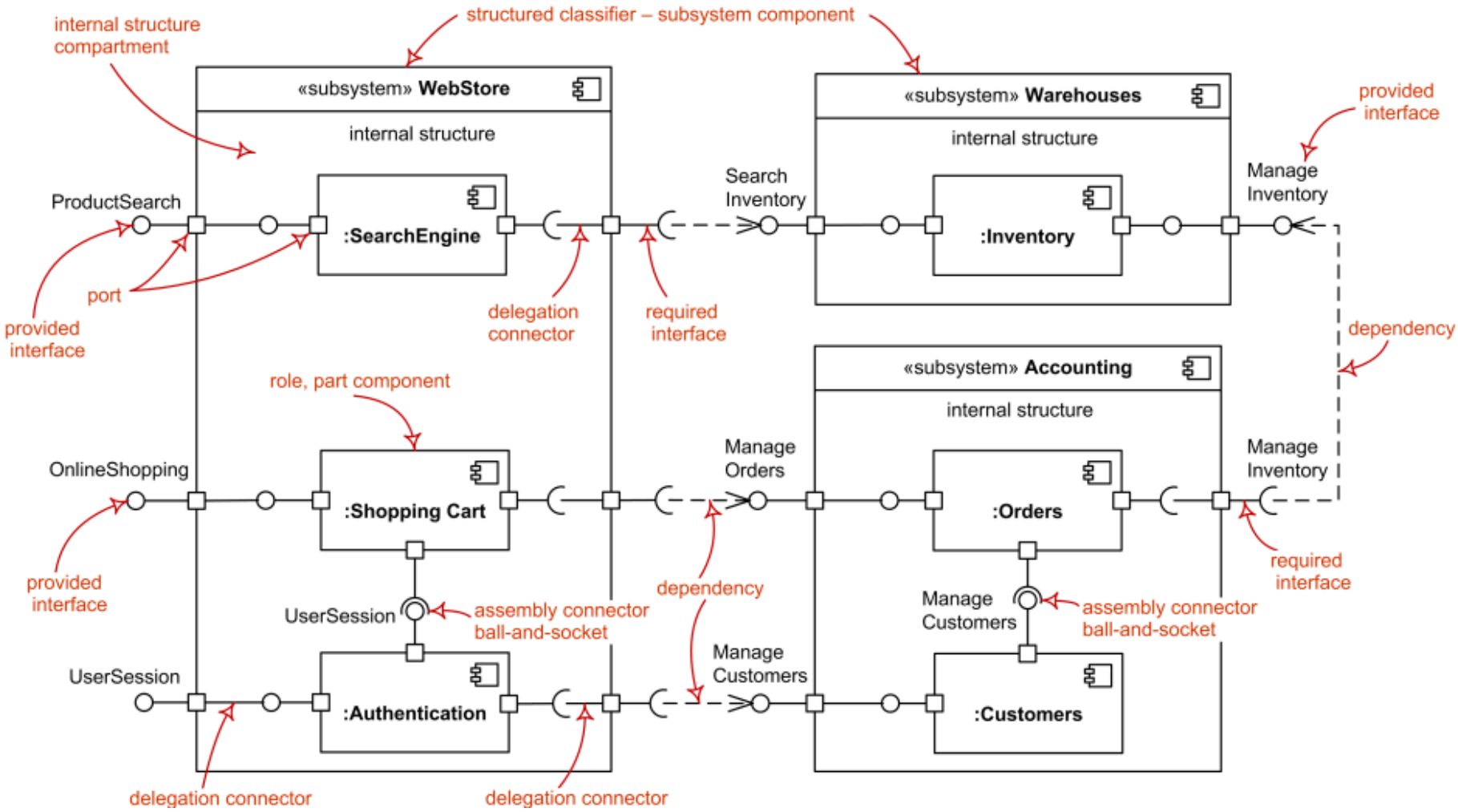
UML Package Diagram



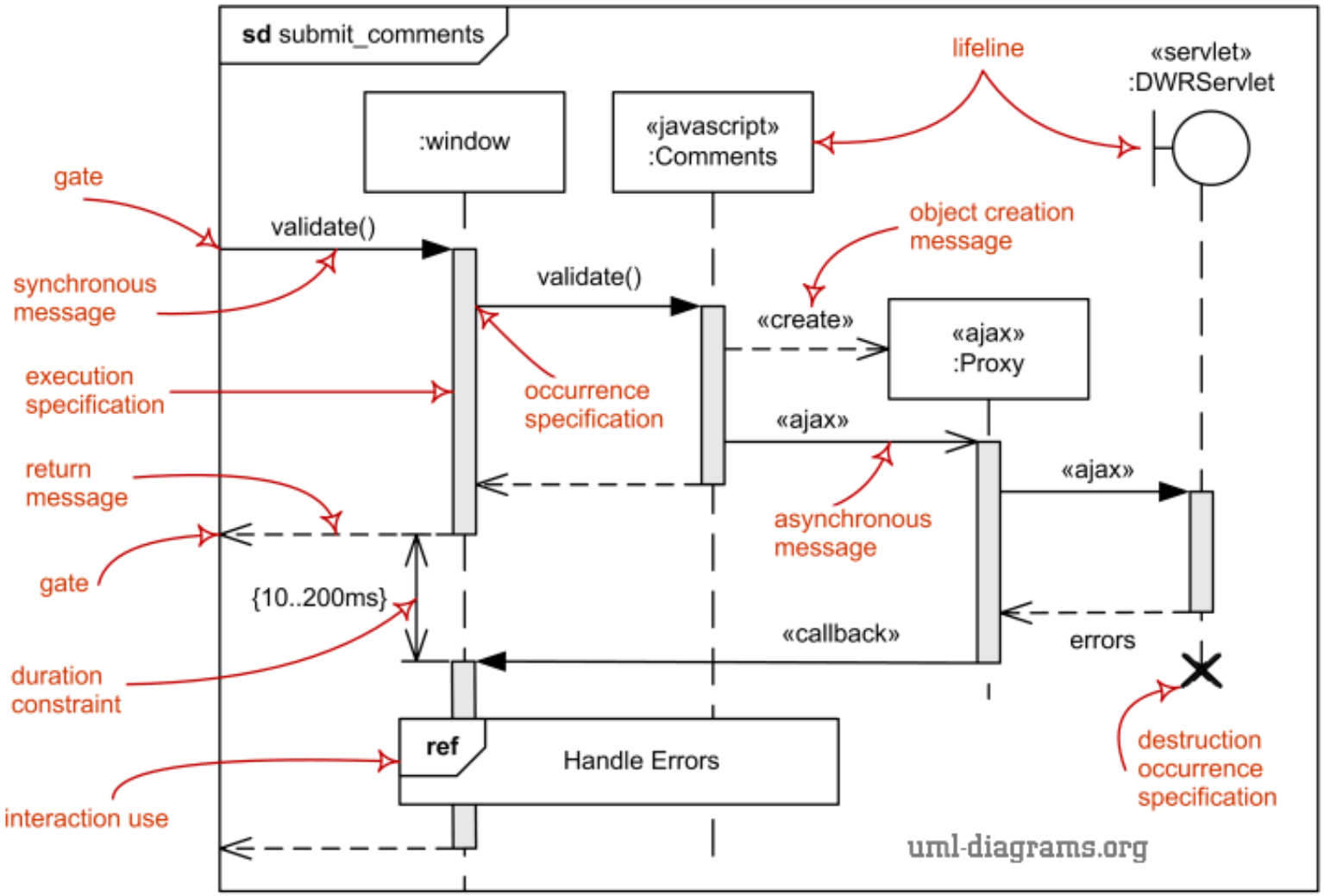
UML Model diagram



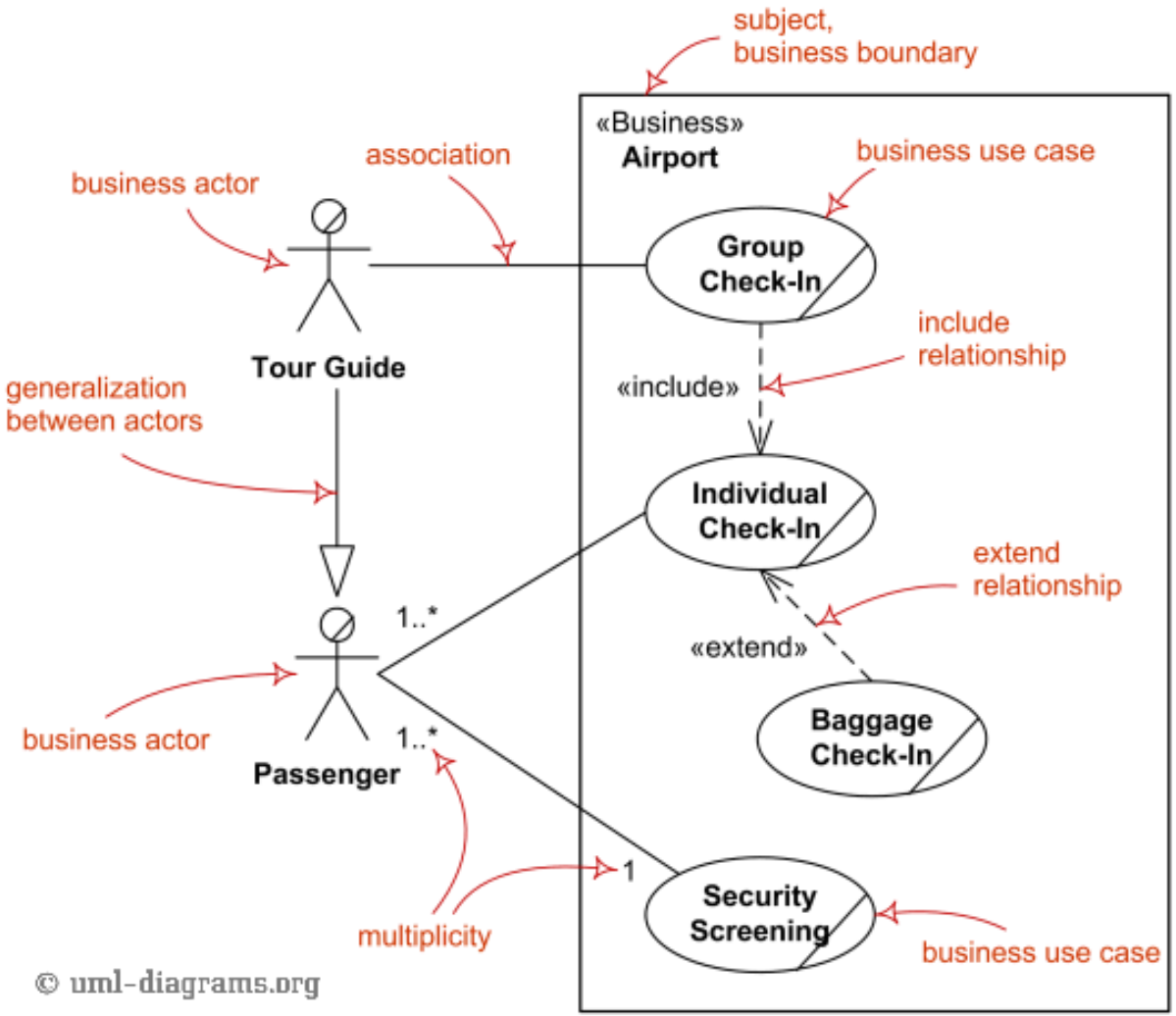
UML Component Diagram



UML Sequence Diagram



UML Use Case Diagram



More Theory vs. Practice

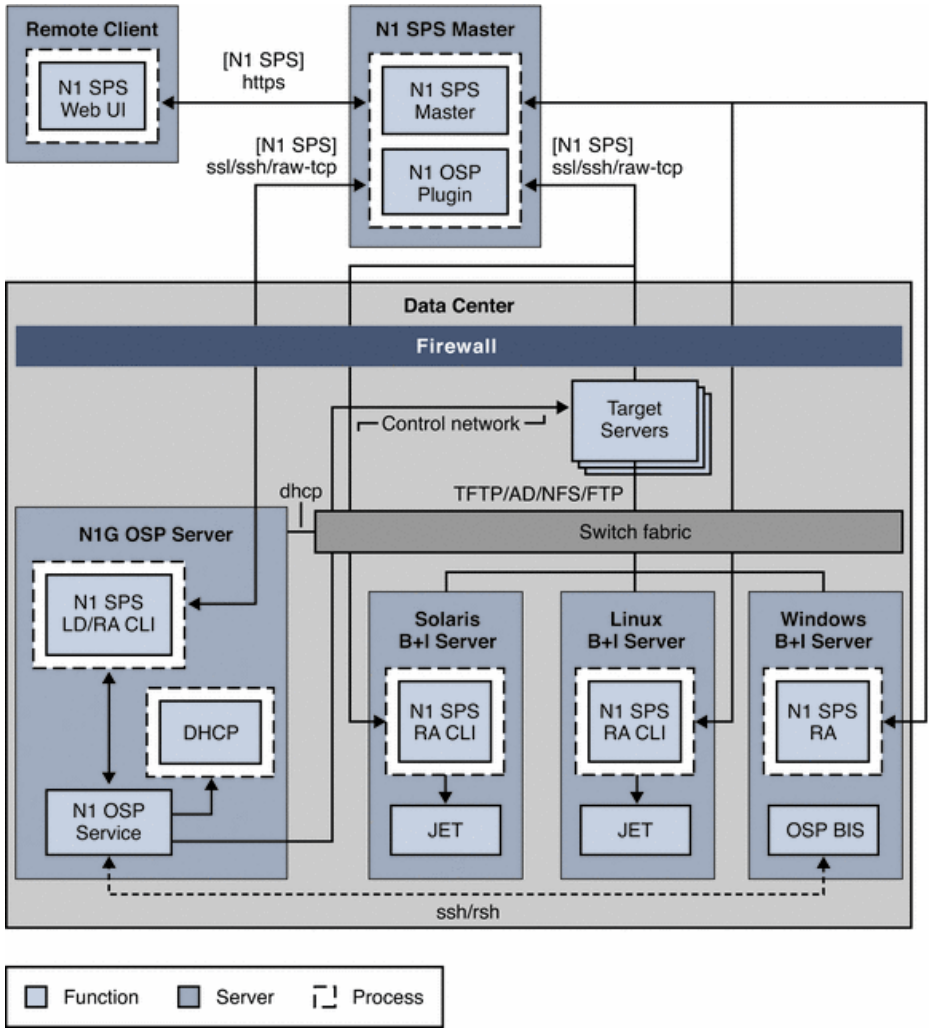
Ok, that is the theory

Let's look at some real examples

Generally:

- Only one diagram; includes hardware and software components
- Boxes: usually don't follow UML
- Lots of lines, arrows
  - Some of which are labeled
- Explanatory text

Oracle: Sun OS provisioning Plug-in



Oracle: text accompanying diagram

The figure describes the following relationships among the OS provisioning components and uses slightly abbreviated terminology:

Remote Client – The N1 SPS remote client runs the browser interface and command-line interface. The remote client can be a separate system from the Master Server.

N1 SPS Master – The N1 SPS Master Server is the main processing engine of the N1 SPS software.

N1 OSP Plug-In – The OS provisioning plug-in is installed on the Master Server. The plug-in provides functionality to install operating systems on different hardware platforms that support different protocols.

N1 OSP Server – The OS provisioning control server, usually referred to as the OS provisioning server, is the main processing engine of the OS provisioning plug-in. The OS provisioning server runs the OS provisioning service (N1 OSP Service), which orchestrates the OS provisioning activities. The OS provisioning server controls the target hosts through a control network using appropriate network management protocols (such as IPMI, ALOM,LOM, RSC, ILO, and terminal server). These protocols over the control network are used to automate the power, boot, and console services.

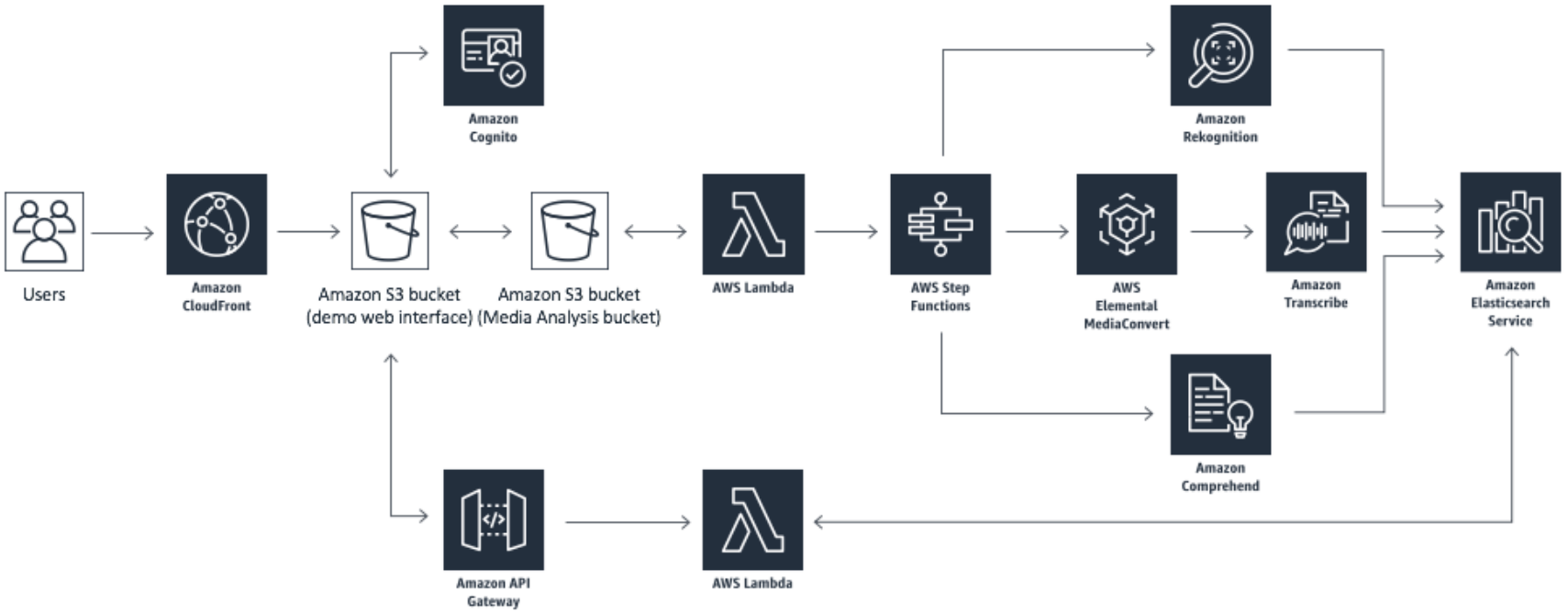
The OS provisioning server supports extensive network topologies (multiple subnets, VLANs, and so on). The OS provisioning server has a bundled DHCP server to serve relevant IP addresses and other boot specific information to target hosts .

Boot and Install Servers – Three servers are shown supporting OS specific Boot and Install servers:

- Solaris B + I Server – The Solaris boot and install server uses the JumpStart™ Enterprise Toolkit (JET) to automate the installation of the Solaris distribution media and installation profile.
- Linux B + I Server – The Linux boot and install server uses the Linux Kickstart technology.
- Windows B + I Server – The Windows boot and install server uses Windows Remote Installation Services (RIS) technology.

The boot and install servers have OS-specific boot and install services for automation and monitoring purposes. You have to set up the Linux and Windows boot and install servers outside of the OS provisioning plug-in. For Linux systems, you have to install the N1 SPS Remote Agent (RA) manually. For Solaris systems, the OS provisioning plug-in installs and configures the RA.

An architecture using AWS



Text accompanying previous diagram

(um, there wasn't any)

Big assumption in previous diagram: you know all the AWS components

In Practice (What you should do)

Use your own style of boxes and lines

- Be clear and consistent
- You may nest boxes inside boxes, but be clear on meaning
- Layout must be intuitive

For connectors:

- Be clear and consistent
- Describe what they mean, maybe have a “legend”
- Connectors are generally not well documented; do better!

Sequence diagrams:

- Follow UML. There are many free tools for it

Text

- Write more than you think you need
- Explain diagrams by using text
- Explain rationale behind important decisions

Use cases

- Follow standard use case style
- Use case maps are often not very useful