

# 2110521

# Software Architecture

Pittipol Kantavat (course leader)

Wiwat Vatanawood

Duangdao Wichadakul

Neungwong Tuaycharoen



# Agenda

- Course syllabus and grading policy
  - Focus on exercises and quizzes
  - ➡ – No midterm exam
  - Term project
  - Online
- Definitions of software architecture
  - 5W2H (What, Why, When, Where, Who, How, How much)



# Syllabus

- See course syllabus
- Grading policy
- In-class activities
- Working in group for this semester
- A lot of exercises and quizzes
- Tools



# Grading Policy



- Quizzes 10%
- Assignments from tutorial 15%
- Term Project 50%
- Final Exam 25%
- No Midterm Exam

# Resources

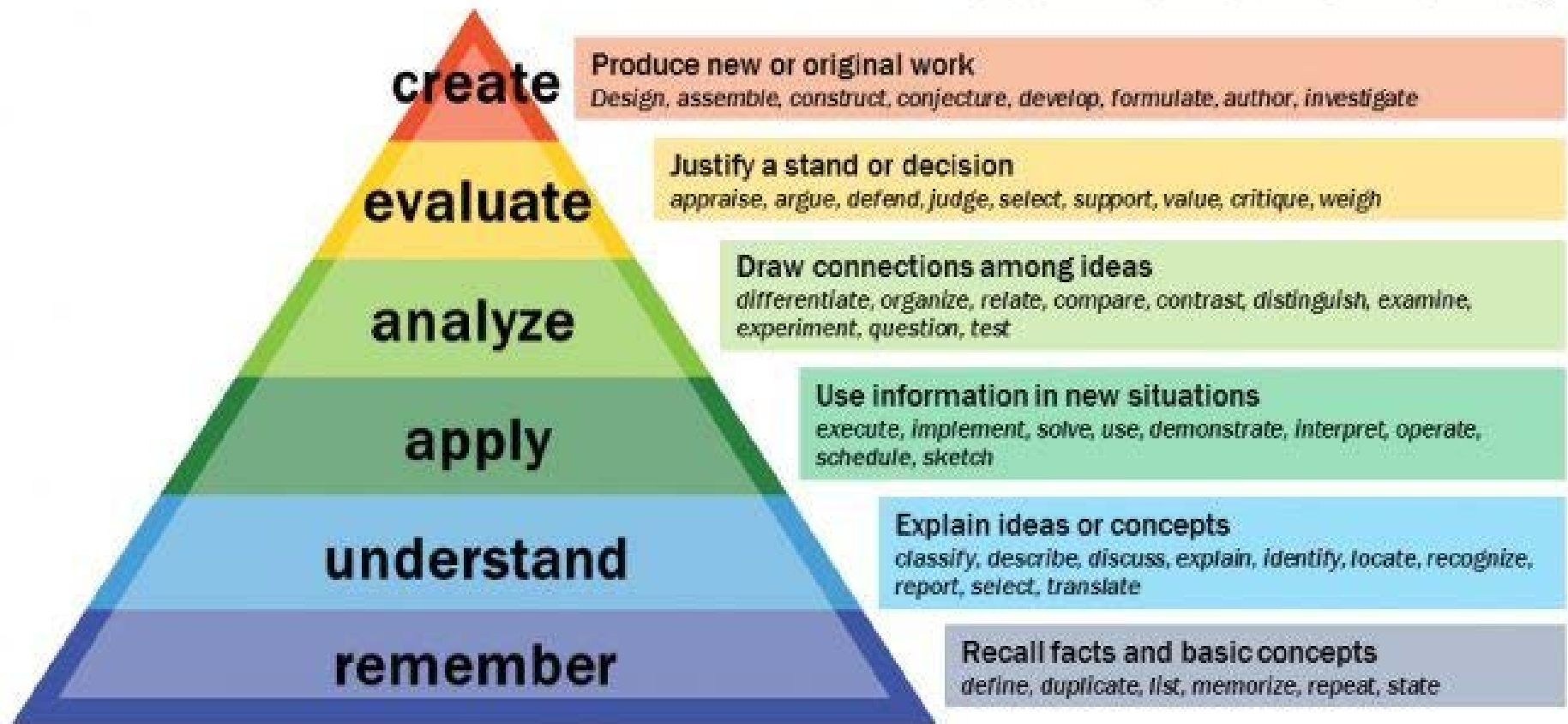
- Lecture slides & textbooks
- Hand on tutorials
- Software tools
- VDO recordings of lectures
- Notebook

# การนำเสนอเนื้อหา

- Bloom's Taxonomy in 1956 by Benjamin Bloom and his colleagues
- เนื้อหาแต่ละส่วนในวิชานี้ มุ่งหวังสัมฤทธิ์ผลแตกต่างกัน ตามระดับของ Bloom
- จำง่าย ๆ คือ Aware, Understand, Implement ก็ได้


# Taxonomy for Teaching, Learning, and Assessment in 2001

## Bloom's Taxonomy



# UML 2.5.1 Standards since Dec 2017

[HOME](#) [SITE MAP](#) [LEGAL](#) [f](#) [t](#) [in](#) [✎](#) [YouTube](#)

 **Standards Development Organization**

[GROUPS ▾](#) [CERTIFICATIONS ▾](#) [RESOURCES ▾](#) [SPECIFICATIONS ▾](#) [MEMBERSHIP ▾](#)

ABOUT THE UNIFIED MODELING LANGUAGE SPECIFICATION VERSION 2.5.1

2.5.1 • UML • SPECIFICATIONS

## UML®

## Unified Modeling Language

A specification defining a graphical language for visualizing, specifying, constructing, and documenting the artifacts of distributed object systems.

**Title:** Unified Modeling Language

**Acronym:** UML®


**Version:** 2.5.1

**Document Status:** formal ⓘ

**Publication Date:** December 2017

**Categories:** [Modeling](#) [Software Engineering](#) [Platform](#)

**IPR Mode** ⓘ [RF-Limited](#) ⓘ

  
Specification

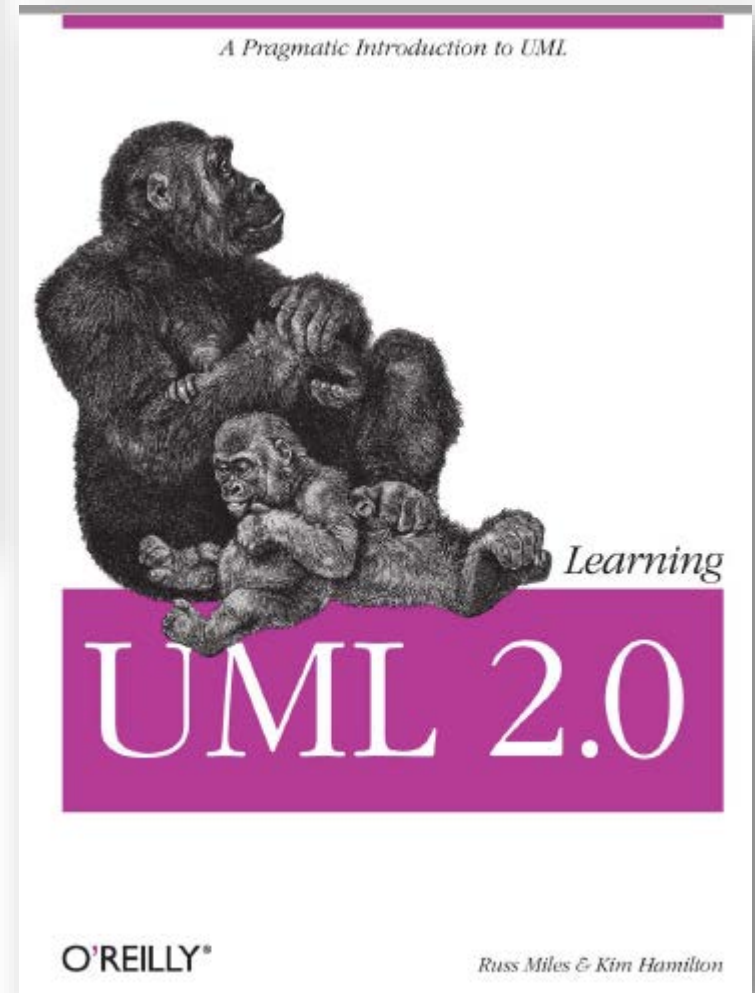
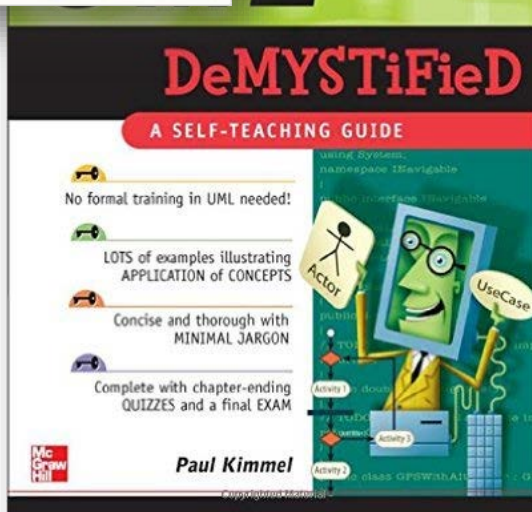
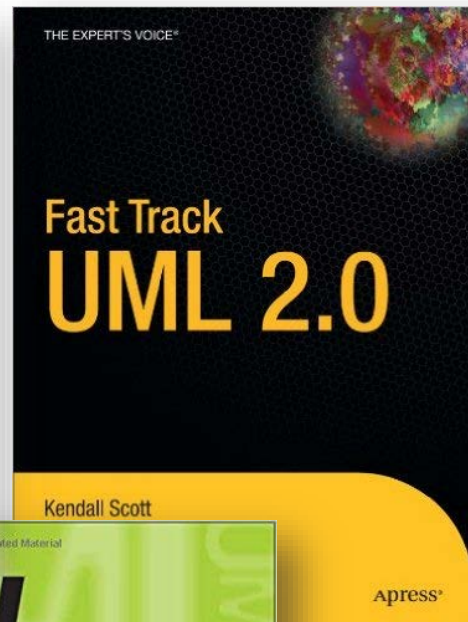
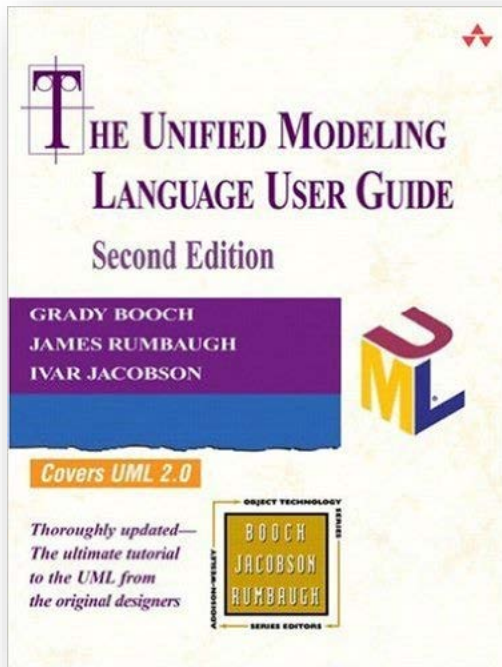
<https://www.omg.org/spec/UML>

Version 2.5 is formally a minor revision to the UML 2.4.1 specification



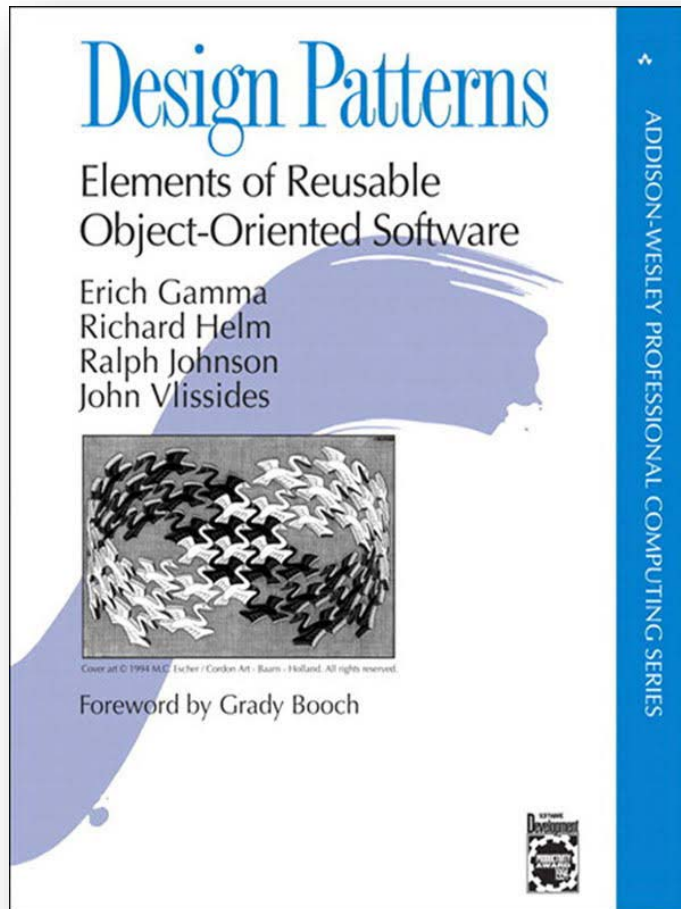


# UML 2.0



# Design Patterns (optional)

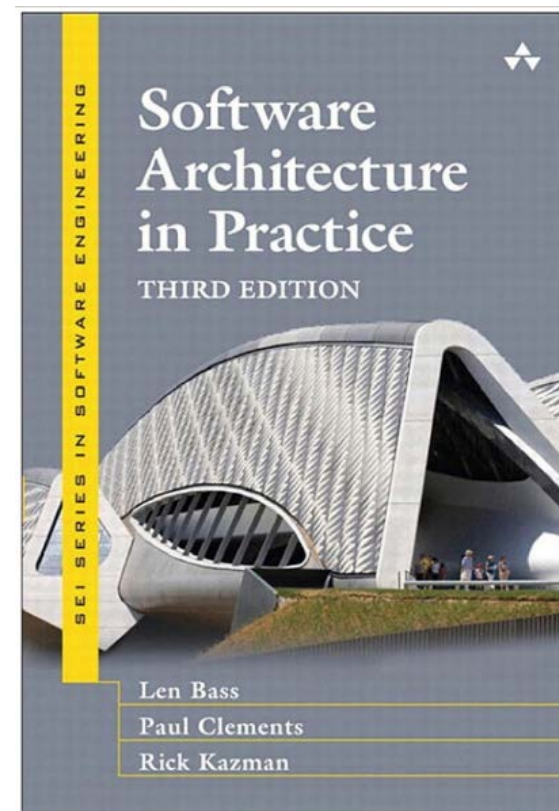
เทอมนี้ ไม่มี



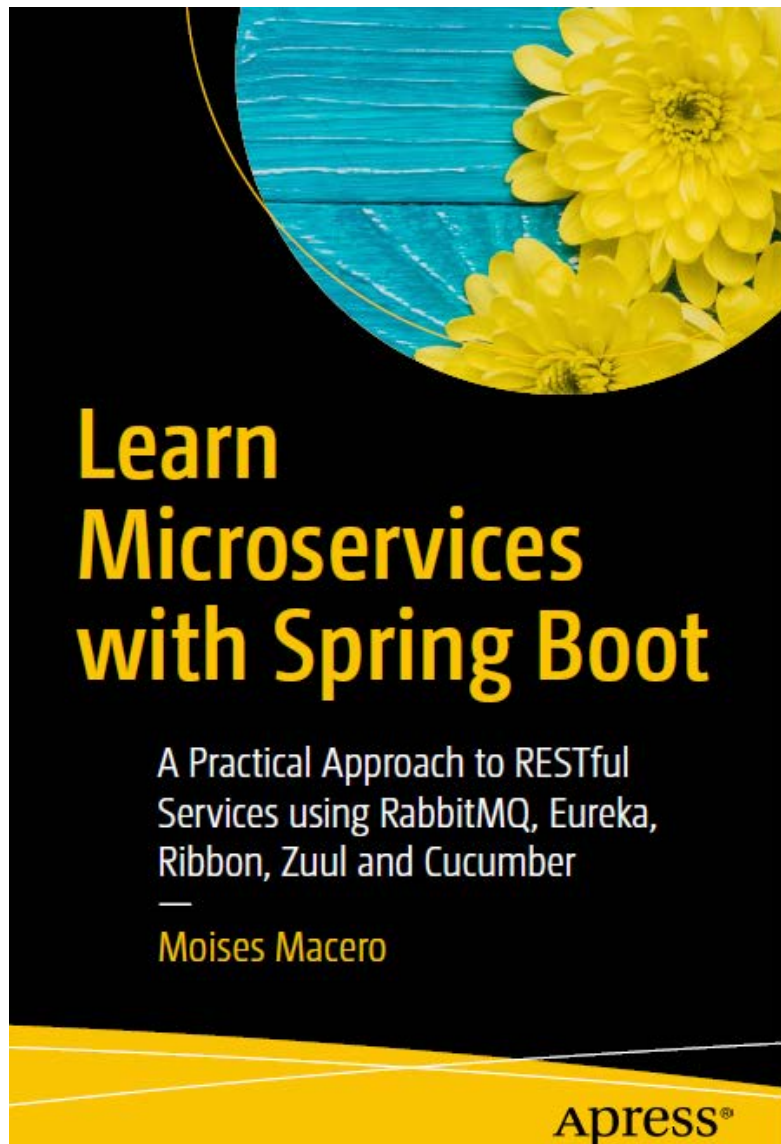
Design Patterns Made Simple

GoF Design Patterns

รวบรวมอยู่ใน  
slide

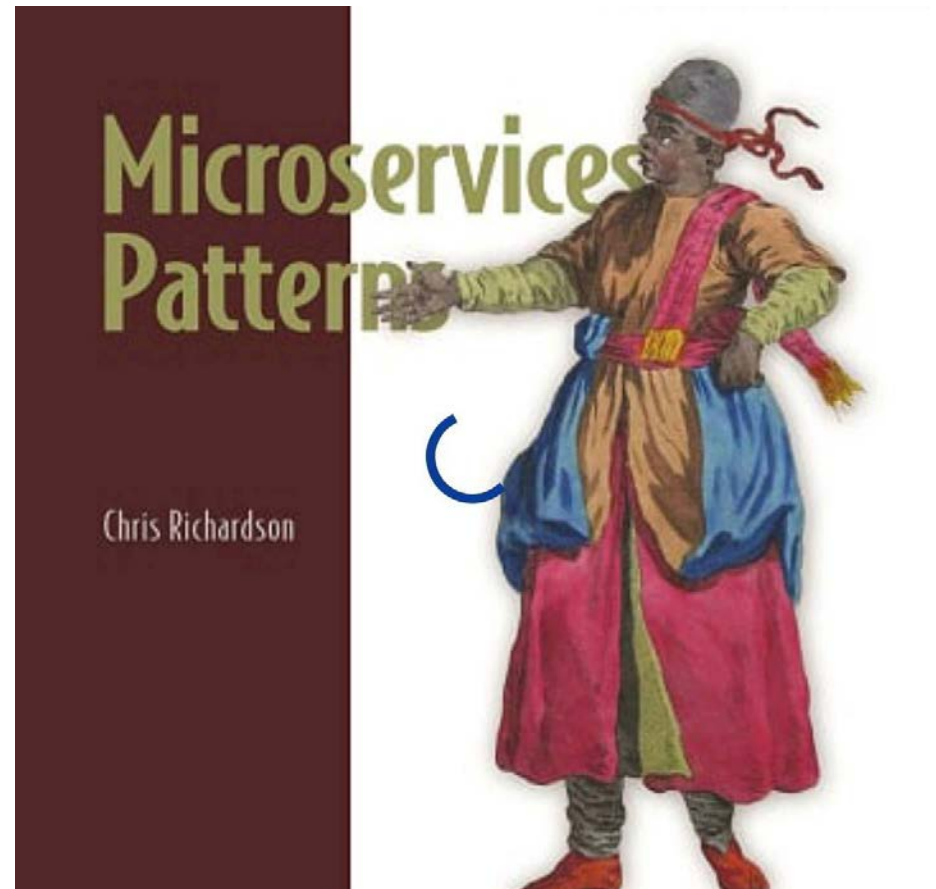






เน้น เน้น

Microservices



เข้าสู่เนื้อหาของวิชากันเลย



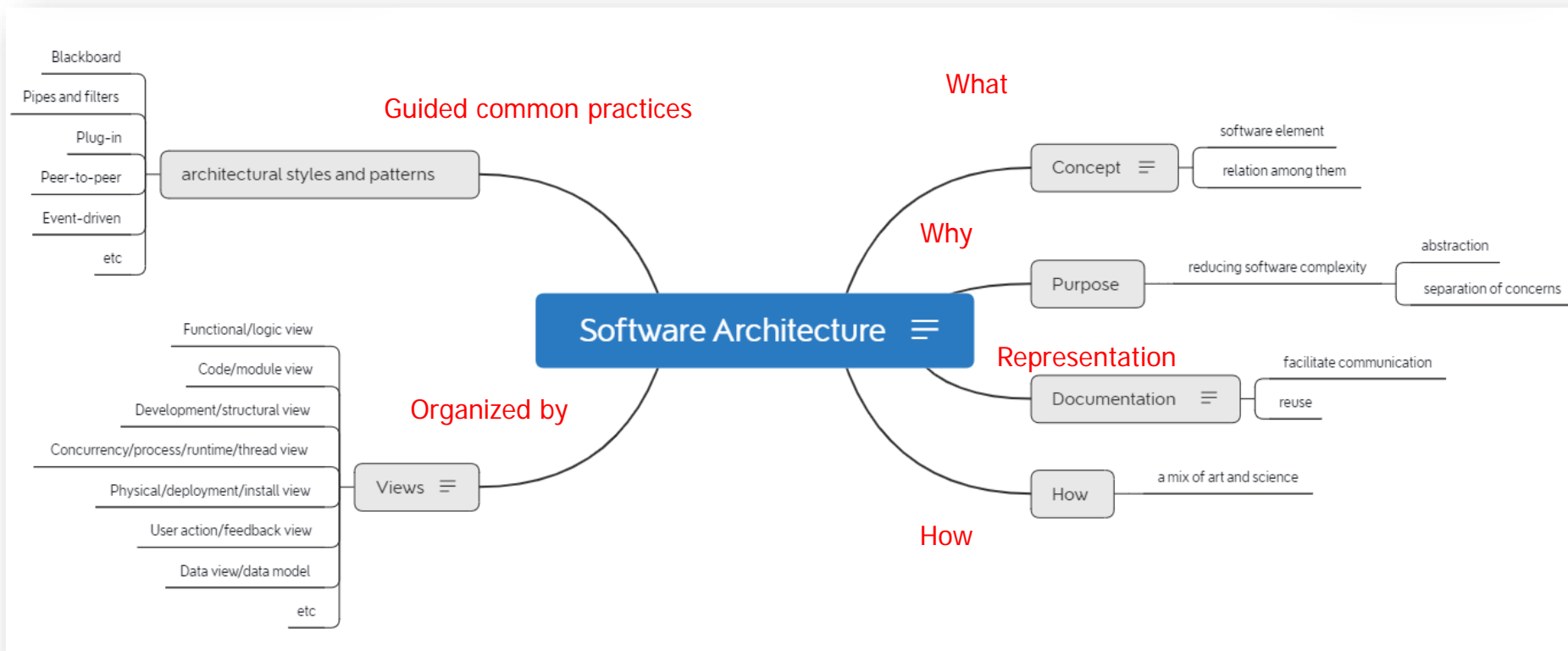
# พยายามเดินเรื่อง โดยใช้ 5W2H

What, Where,  
When, Who, Why,  
How, How much



# Sw Arch Mindmap

เราสร้างแผนภาพ เพื่อความเข้าใจให้ง่ายขึ้น



# Analogy

เปรียบเทียบ Software architecture กับ  
สิ่งที่ใกล้ตัว



# สร้างซอฟต์แวร์ขนาดใหญ่ เปรียบได้กับ สร้างตึกสูง



สร้างอาคาร 2 ชั้น แต่ละชั้นสูงไม่  
เกิน 4 เมตร ต่างกับ

สร้างอาคารสูง 100 ชั้น อย่างไร

วิศวกรเซ็นแบบ?

ความลึกเสาเข็ม ฐานราก

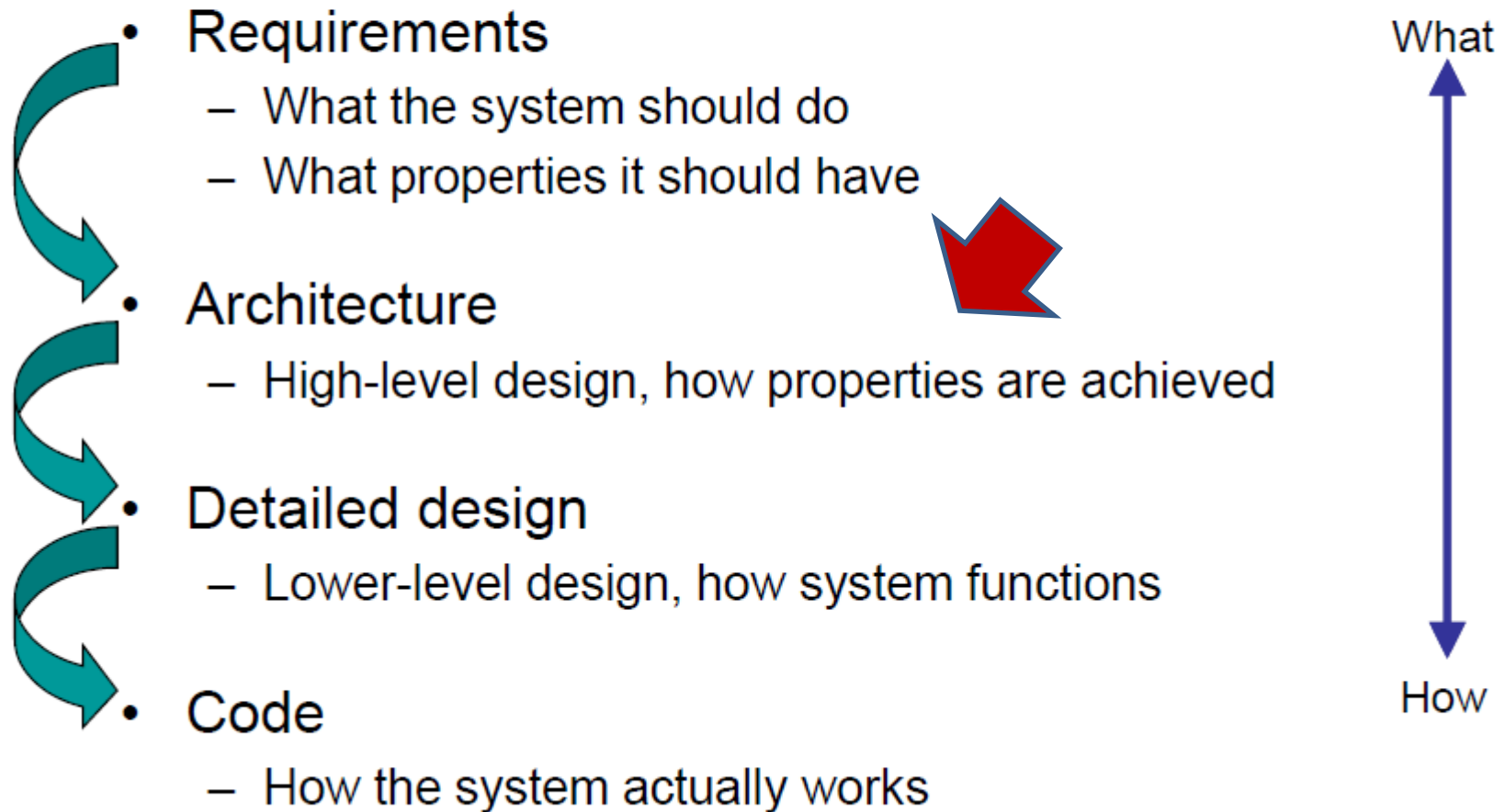
แล้วถ้าเป็น ซอฟต์แวร์??

SW Arch ถูกใช้เมื่อไร

Software Architecture  
as  
an Architectural Design

WHEN

# Where Architecture Fits



เราส่งผ่าน **input/output** ของกิจกรรมต่าง ๆ ด้วย **Model**

# Architectural Design (High level)

- Level of Abstraction
  - Top-down (High-to-Low level of abstraction) approach
- Architectural Design
  - Subsystem Design
- Complexity of the Design
  - Hiding and Omitting to Reduce the Complexity

• Detailed Design อาจจะมีรายละเอียดมากเกินไป  
– Object Design



**Model** คืออะไร มาเกี่ยวข้องกับอย่างไร

**Model** คือตัวแทนสิ่งที่เรากำลังสนใจ  
ที่เราใช้เพื่อนำเสนอหรือสื่อสารกัน  
(เพราะจัดหาของจริงมาไม่สะดวก)

**Model** หน้าตาเป็นอย่างไร



ถ้าเราสนใจสาวน้อยเทนนิส

→ Model ?

เช่น Structural model ของสาวน้อย  
แล้ว Behavioral model เป็นอย่างไร?

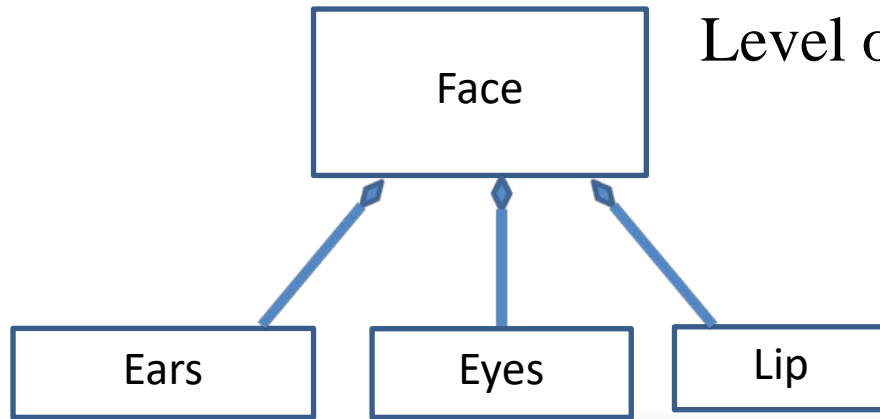
Q: Level of Abstraction: **High** or **Low**?



ถ้าเราสนใจระบบซอฟต์แวร์

→ Model ?

รู้ได้อย่างไรว่า High abstraction?



Level of Abstraction: **High**



Level of Abstraction: **Low**





Level of Abstraction: **High**

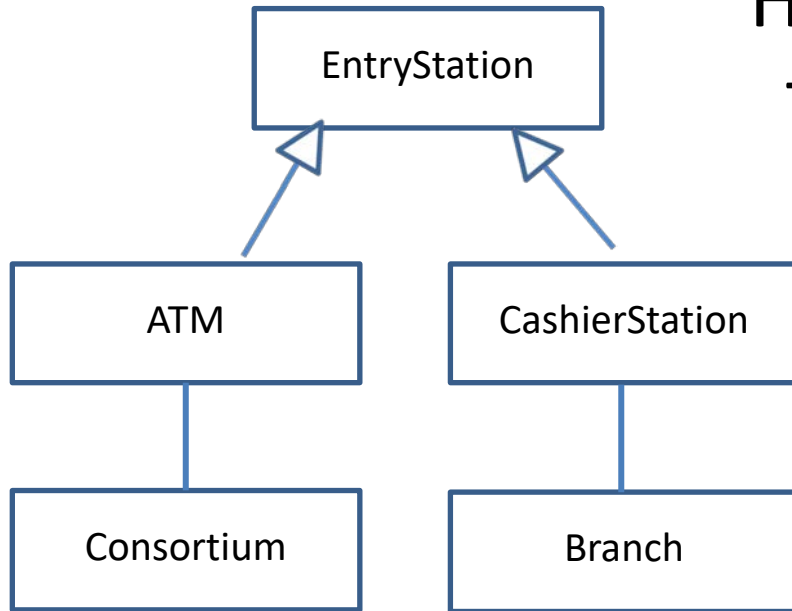


Level of Abstraction: **Low**



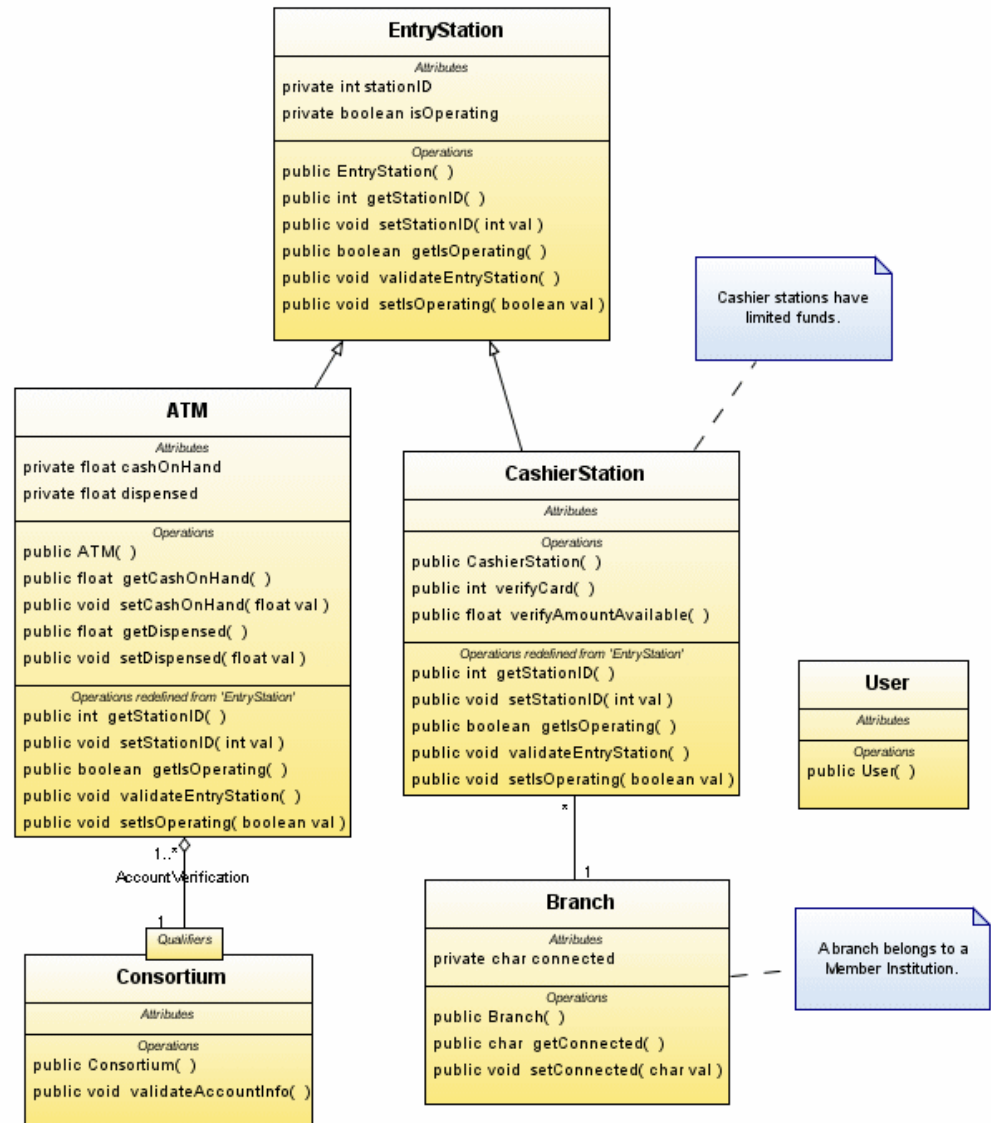
Hiding and Omitting to Reduce the Complexity of the Model

# Hiding and Omitting to Reduce the Complexity of the Model



No details of classes

Q: เวลาแก้ไขเปลี่ยนแปลง scope ภาพใดแก้ไขง่ายกว่า?



# คำถามที่ควรตอบได้

- Model คืออะไร
- Model สำคัญอย่างไร
- Informal model → Semi formal model  
→ Formal model คืออะไร

WHAT

นิยามของ SW Arch

Computer scientist Ralph Johnson, who co-authored *Design Patterns: Elements of Reusable Object-Oriented Software*, once said:

*"Architecture is about the important stuff. Whatever that is."*

- Ralph E. Johnson



ความหมายแบบง่าย ๆ ตรง ๆ ไม่เป็นทางการนัก



# About Software Architecture

- The **extension** of software engineering discipline
- Software Architecture presents **a view** of software system as components and connectors
  - **Components** encapsulate some coherent set of functionality
  - **Connectors** realize the runtime interaction between components



Architecture เน้น Abstraction level สูง

# IEEE 1471-2000

*"Architecture is defined by the recommended practice as the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution."*

*[ANSI/IEEE Std 1471-2000, Recommended Practice for Architectural Description of Software-Intensive Systems]*

# Bass, Clements, Kazman, 2003

*“The software architecture of a program or computing system is the structure or*  
 *structures of the system, which comprise*  
*software elements, the externally visible*  
 *properties of those elements, and the*  
*relationships among them.”*

[L.Bass, P.Clements, R.Kazman, *Software Architecture in Practice (2<sup>nd</sup> edition)*, Addison-Wesley 2003]



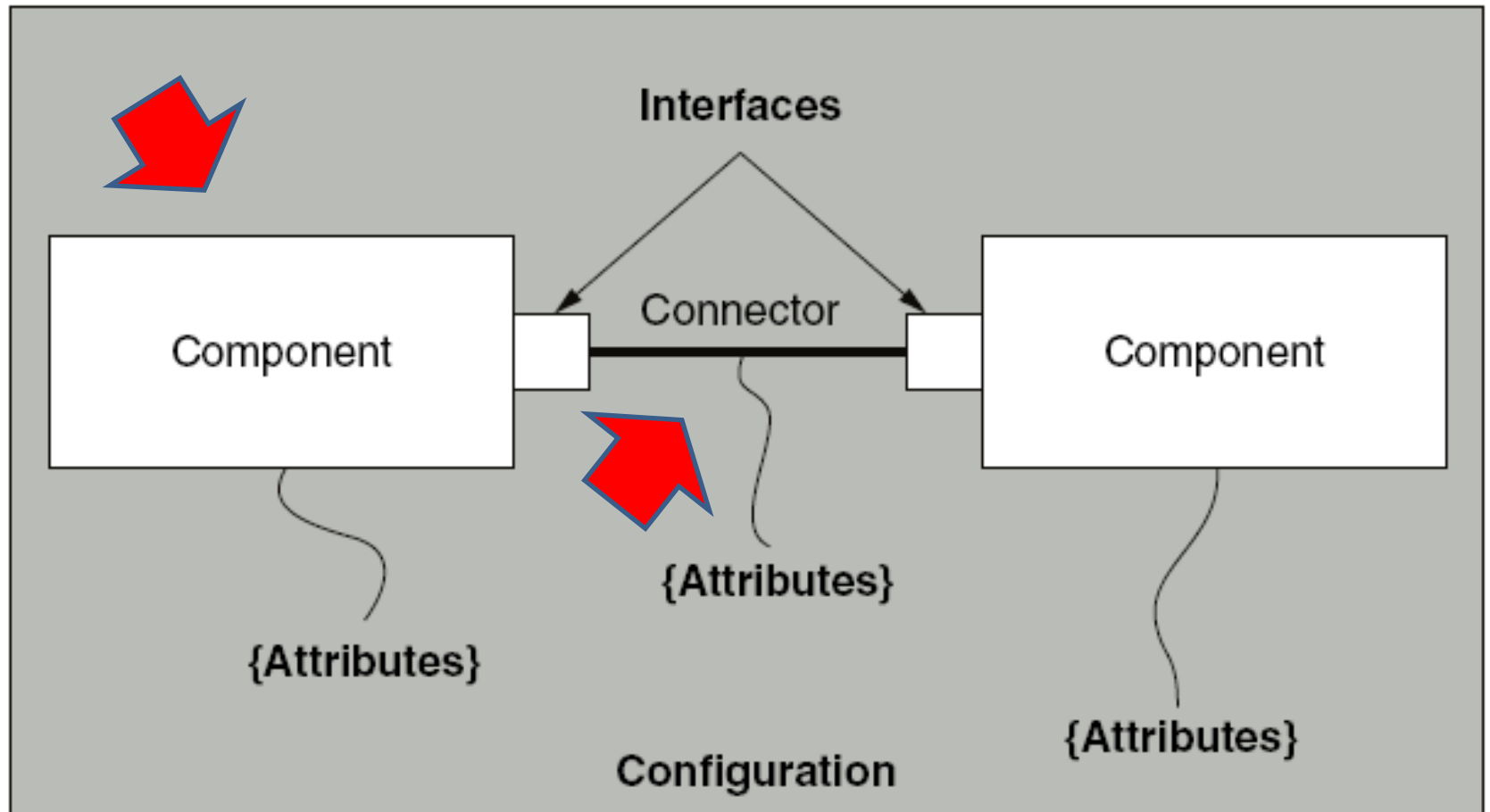


Figure 12-1. Software architecture concept.

# กล่าวได้ว่า

- สำหรับ Software Architecture เราใช้ **model** ที่เขียนด้วย
  - UML Subsystem diagram (**design-time**)
  - UML Component diagram (**run-time**)

**SW Arch ควรมีกี่ diagram?**

แต่ละ **diagram** แสดงถึงมุมมอง **View** ที่แตกต่างกัน ไม่  
จำเป็นต้องใช้แค่ **diagram** เดียว

# Worldwide Institute of Software Architect (www.wwisa.org)



ตย. องค์กรด้าน Sw Arch

- The Institute of Software Architects, Inc. is a nonprofit corporation founded to accelerate the establishment of the profession of software architecture and to provide information and services to software architects and their clients.

Copyright © 1997-2005 Institute of Software Architects, Inc. All rights reserved.

[PHILOSOPHY](#) [DESIGN](#)  
[COMPETITION](#)  
[ROLE OF SOFTWARE](#)  
[ARCHITECT](#) [DISCUSSION GROUPS](#)  
[CLIENT INFORMATION](#) [REFERRALS](#)  
[MEMBERSHIP](#) [BOOKS & TOOLS](#)



# HOW

เราออกแบบและนำเสนอ SW Arch ได้อย่างไร

# Architectural Representation and Documentation

# Software Architecture Description

= A set of practice for **expressing, communicating, and analyzing** software architecture.

- เราบันทึก Software architecture ด้วยการอธิบาย
- เรานิยมเลือกวิธีการอธิบายด้วยแผนภาพและข้อความสั้น ๆ ประกอบกัน (Semi formal model)

- ในอีกนัยหนึ่ง

Architecture Description = Architecture Modelling

นั่นเอง

# What if Software Architecture shown in source code

## Source Code Represents Architecture?

### A Bit of Modern Software...

```
SC_MODULE(producer)
{
    sc_outmaster<int> out1;
    sc_in<bool> start; // kick-start
    void generate_data ()
    {
        for(int i =0; i <10; i++) {
            out1 =i ; //to invoke slave;}
        }
    SC_CTOR(producer)
    {
        SC_METHOD(generate_data);
        sensitive << start;}};

SC_MODULE(consumer)
{
    sc_inslave<int> in1;
    int sum; // state variable
    void accumulate (){
        sum += in1;
        cout << "Sum = " << sum << endl;}
```

```
SC_CTOR(consumer)
{
    SC_SLAVE(accumulate, in1);
    sum = 0; // initialize
};

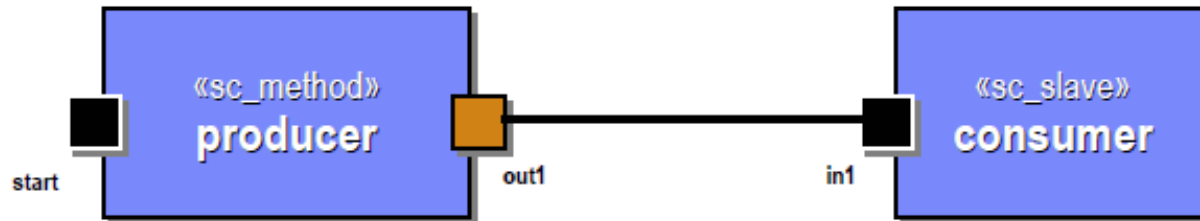
SC_MODULE(top) // container
{
    producer *A1;
    consumer *B1;
    sc_link_mp<int> link1;
    SC_CTOR(top)
    {
        A1 = new producer("A1");
        A1.out1(link1);
        B1 = new consumer("B1");
        B1.in1(link1);}}
```

ทำไม  
ใช้ยาก

Can you spot the  
architecture?

# ถ้าเราใช้ UML จะง่ายในการเข้าใจภาพรวม

...and its UML 2.0 Model



**Can you see it now?**



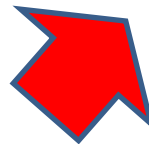
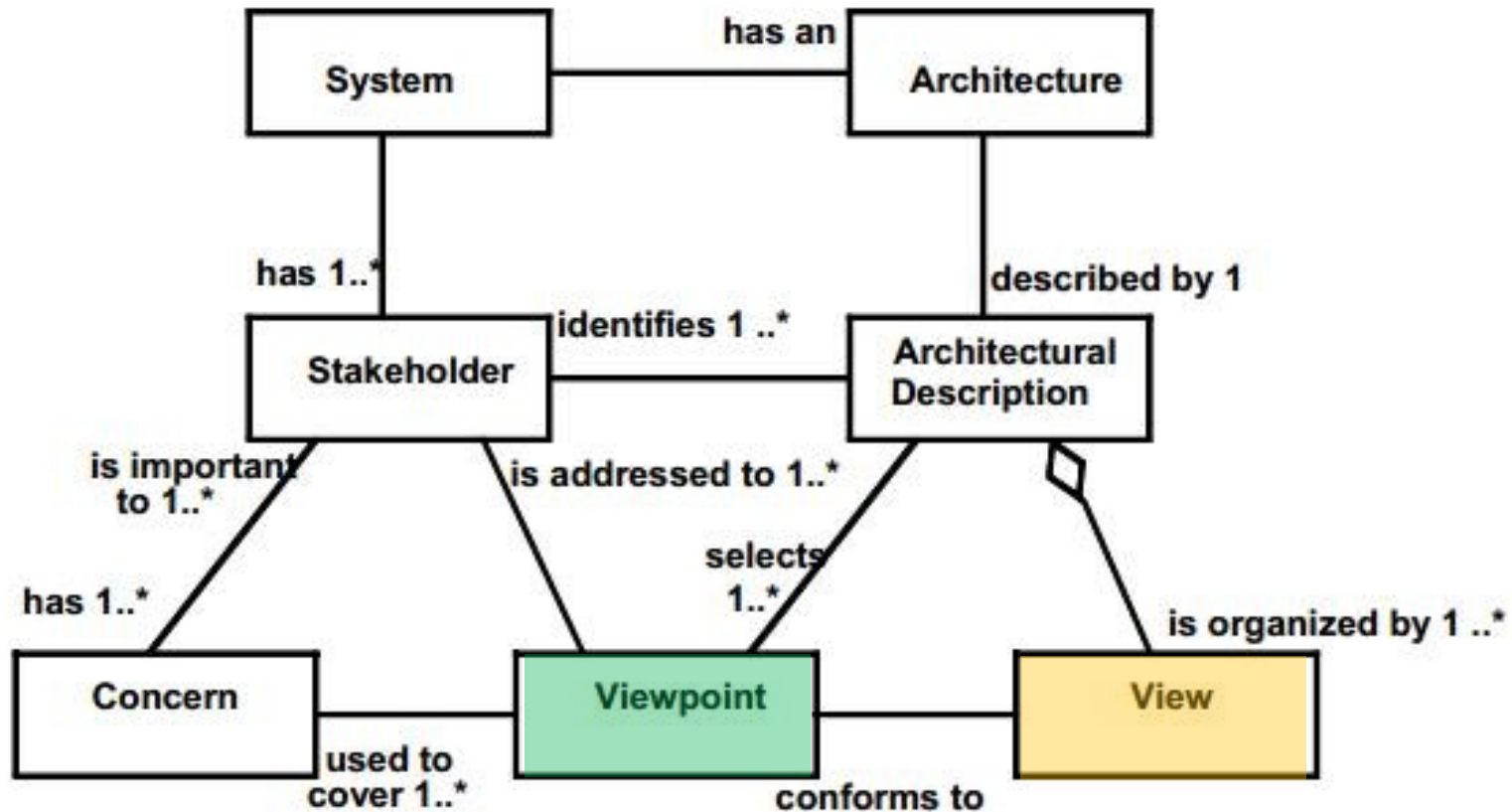
# Architectural Views vs. Architectural Viewpoints



# Architecture Viewpoints

- Software architecture descriptions are **organized into Views**.
- Each view addresses a set of **system concerns**
- Each view is drawn following the conventions of its viewpoint.
- A **viewpoint is a specification** that describes the notations, modeling techniques to be used in a view to express the architecture.
- **Viewpoints are generic templates**, while views are what you actually see.

# From ISO 42010 *Systems and software engineering*



# Examples of Viewpoints

- Functional viewpoint
- Logical viewpoint
- Information/Data viewpoint
- Module viewpoint
- Component-and-connector viewpoint
- Requirements viewpoint
- Security viewpoint
- Physical/Deployment/Installation viewpoint
- Etc.

**Viewpoint** = แม่แบบวิธีเขียน **view** สำหรับมุมมองนั้น  
เช่น ที่ **Functional viewpoint** จะได้ **Func.view**, etc.

# Architectural Views

- Many possible “views” of architecture
  - Implementation structures
    - Modules, packages
    - Modifiability, Independent construction, ...
  - Run-time structures
    - Components, connectors
    - Interactions, dynamism, reliability, ...
  - Deployment structures
    - Hardware, processes, networks
    - Security, fault tolerance, ...

# How many views?

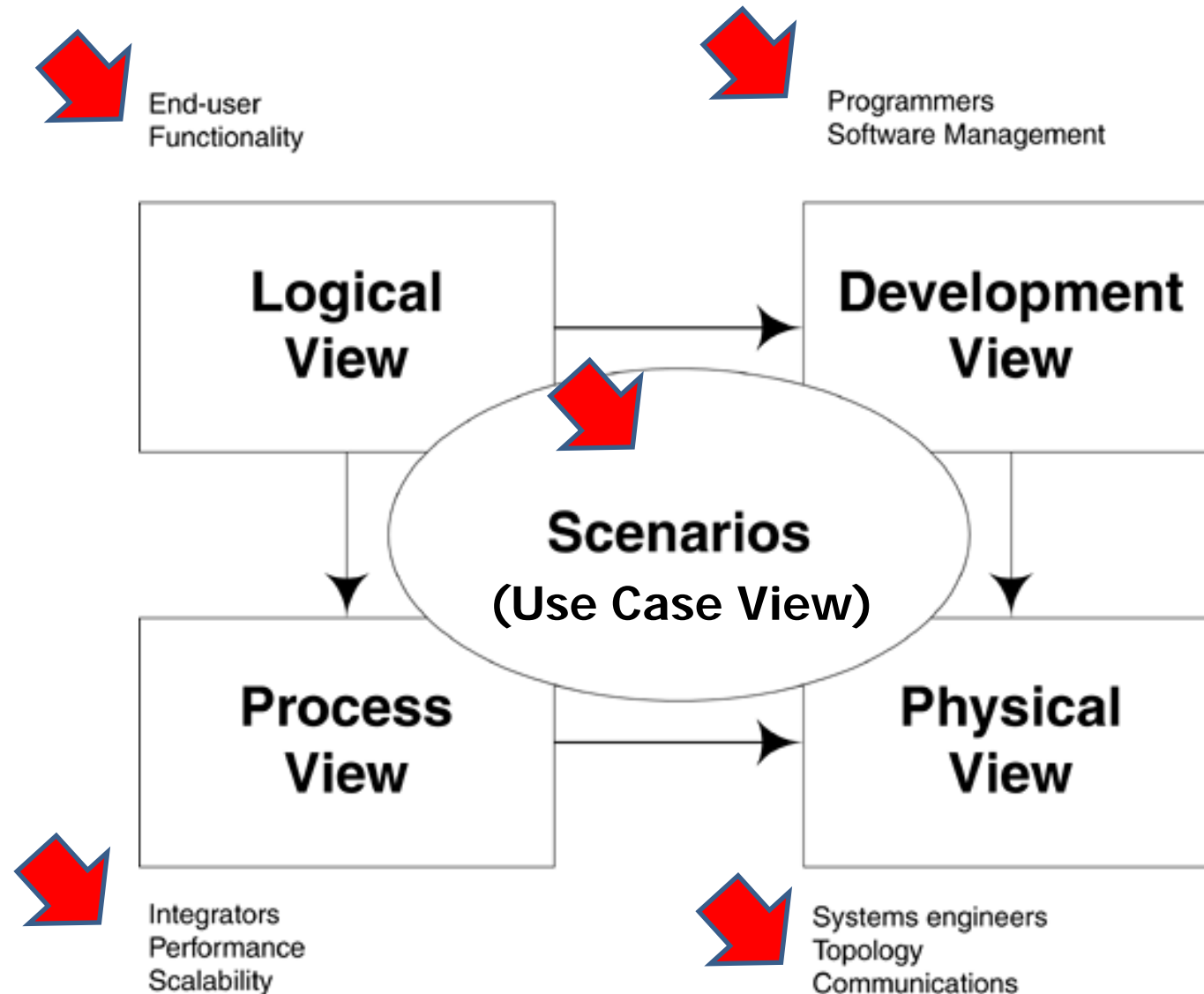
- 5 by Taylor et al.: Logical, Physical, Deployment, Concurrency, Behavioral
- 3 by Bass et al.: Component & Connector, Module View, Behavior
- **4+1 by Kruchten**: Logical, Physical, Process, Development, and Scenarios



# 4+1 Architecture Views

- The term “architecture views” rose to prominence in Philippe Kruchten’s 1995 paper on the 4+1 View Model
- This presented a way of describing and understanding an architecture based on the following four views:
  - Logical View
  - Process View
  - Development View
  - Physical View
  - Tied together by Use Case View

# The 4+1 Architectural View by Krutchen





# 4+1 Architectural Views

- **The logical view**  
Structural breakdown of computational, communicational and behavioral responsibilities. In UML the logical view is modelled using **Class, Object, State machine and Interaction diagrams (e.g. Sequence diagrams)**.
- **The process view**  
This describes the concurrent processes within the system. It encompasses some non-functional requirements such as performance and availability. In UML, **Activity diagrams** - which can be used to model concurrent behavior - are used to model the process view.
- **The development view**  
The development view focusses on software modules and subsystems. In UML, **Package and Component diagrams** are used to model the development view.

# 4+1 Architectural Views (cont.)

- **The physical view**

The physical view describes the physical deployment of the system. For example, how many nodes are used and what is deployed on what node. Thus, the physical view concerns some non-functional requirements such as scalability and availability. In UML, **Deployment diagrams** are used to model the physical view.

- **The use case view**

This view describes the functionality of the system from the perspective from outside world. It contains diagrams describing what the system is supposed to do from a black box perspective. This view typically contains **Use Case diagrams**. All other views use this view to guide them

# Architecture not so easy

- Unfortunately, this is not an easy task, and the risks and costs associated with selecting an inappropriate technology are high
- Software architecture is the set of design decisions which, if made incorrectly, may cause your project to be cancelled

การออกแบบ architecture ง่ายจริงหรือ

# Three Examples of Software Architecture

ระบบที่มี **Functional req.** เหมือนกัน  
แต่มี **Non-functional req.** ต่างกัน

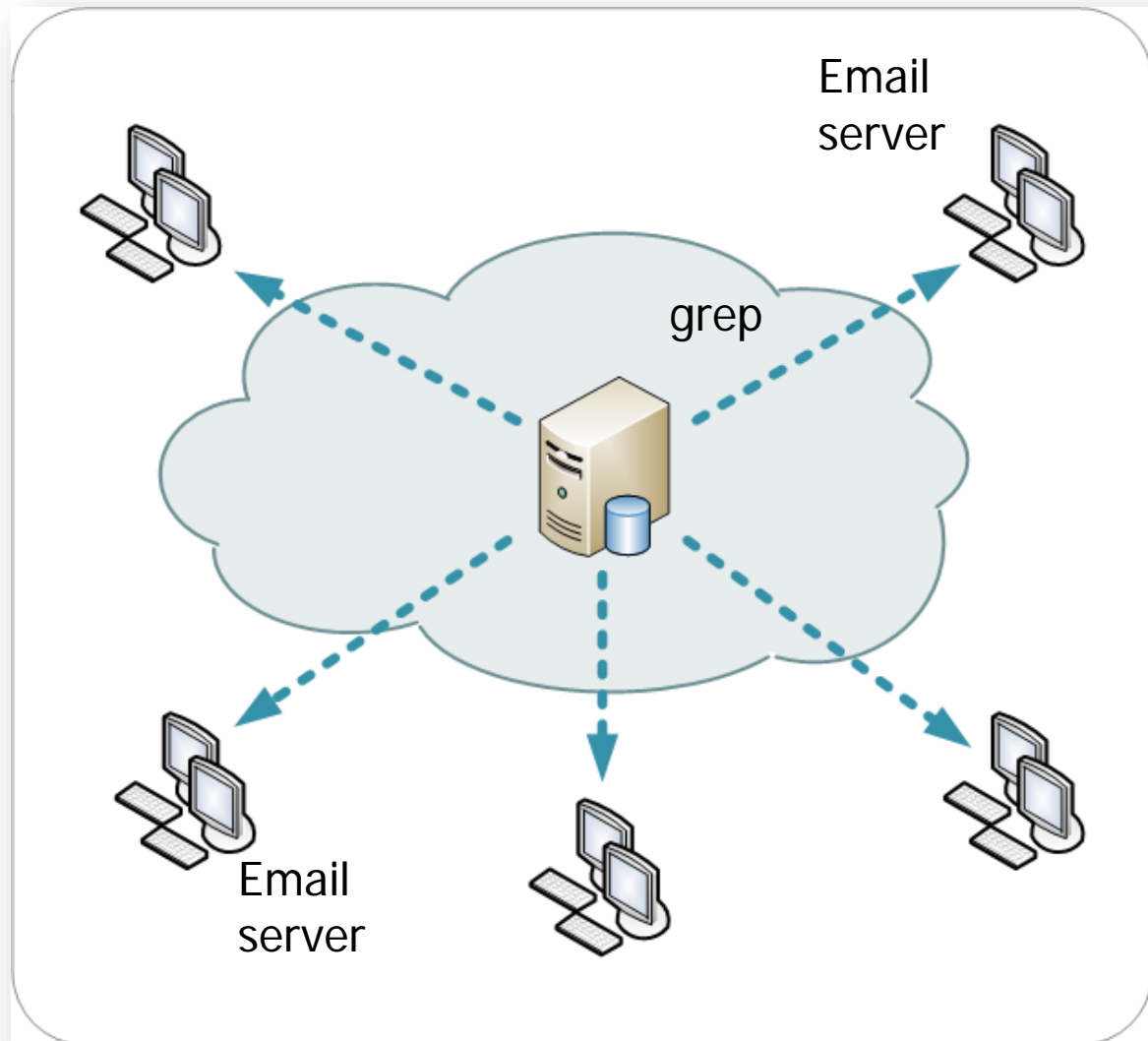
# Simple Email Search App

- Many **email servers** with log files
- App search each email servers **using grep** query commands
- Need **programmer** to adjust the grep query
- **Low cost and easy** for initial use case
- If Huge number of servers → **High overhead** from sequentially searching

# Version 1: Local log files

- **Version 1: Local log files.**
- The first version of the program was simple. There were already dozens of email servers generating log files.
- Rackspace wrote a script that would use ssh to connect to each machine and execute a grep query on the mail log file.
- Engineers could control the **search results by adjusting the grep query.**
- This version initially worked well, but over time the number of searches increased and the overhead of running those searches on the **email servers** became noticeable.
- Also, it required an engineer, rather than a support tech, to perform the search.

# Sequential file of email logs



# Centralized DB of Email Logs

- Active email log files are not accumulated
- History of email data are deleted
- Volume of log data is increasing
- Sequential searching becomes slower
- Easy searching UIs are needed



# Version 2: Central Database

- The second version addressed the drawbacks of the first one by moving the log data off of the email servers and by making it searchable by support techs.
- Every few minutes, each email server would send its recent log data to a central machine where it was loaded into a relational database. Support techs had access to the database data via a web-based interface.
- Rackspace was now handling hundreds of email servers, so the volume of log data had increased correspondingly. Rackspace's challenge became how to get the log data into the database as quickly and efficiently as possible.
- The company settled on bulk record insertion into merge tables, which enabled loading of the data in two or three minutes. Only three days worth of logs were kept so that the database size would not hinder performance.
- Over time, this system also encountered problems. The database server was a single machine and, because of the constant loading of data and the query volume, it was pushed to its limit with heavy CPU and disk loads. Wildcard searches were prohibited because of the extra load they put on the server.
- As the amount of log data grew, searches became slower. The server experienced seemingly random failures that became increasingly frequent. Any log data that was dropped was gone forever because it was not backed up. These problems led to a loss of confidence in the system.

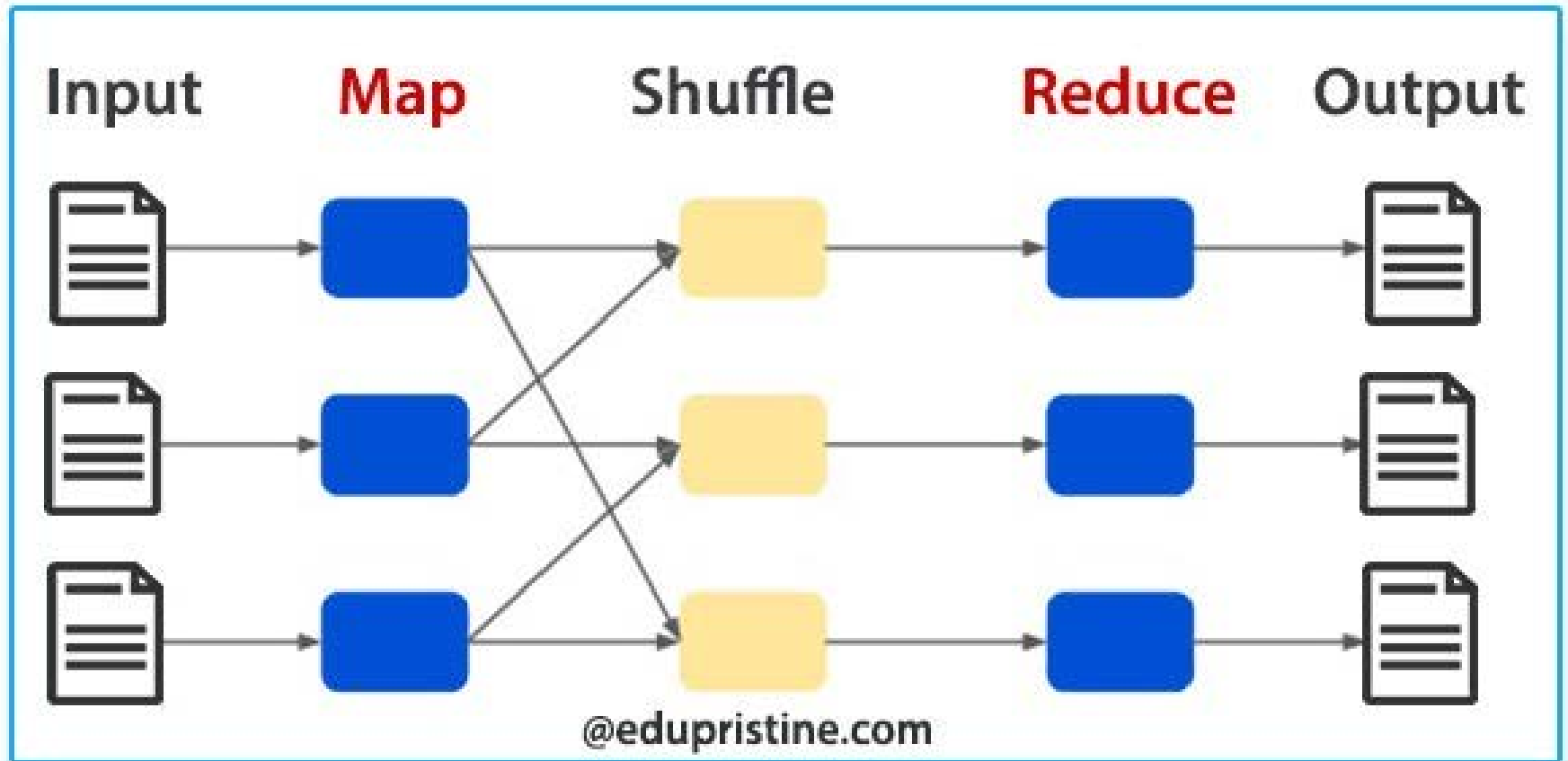
# Centralized DB of email logs



# Version 3: Indexing Cluster

- The third version addressed the drawbacks of the second by saving log data into a distributed file system and by parallelizing the indexing of log data. Instead of running on a single powerful machine, it used ten commodity machines.
- Log data from the email servers streamed into the Hadoop Distributed File System, which kept three copies of everything on different disks. In 2008, when Rackspace wrote a report on its experiences, it had over six terabytes of data spanning thirty disk drives, which represented six months of search indexes.
- Indexing was performed using Hadoop, which divides the input data, indexes (or “maps”) in jobs, then combines (or “reduces”) the partial results into a complete index.
- Jobs ran every ten minutes and took about five minutes to complete, so index results were about fifteen minutes stale. Rackspace was able to index over 140 gigabytes of log data per day and had executed over 150,000 jobs since starting the system.
- As in the second system, support techs had access via a web interface that was much like a web search-engine interface. Query results were provided within seconds.

# Map-Reduce Server of Email logs



What if  
Representing Architecture  
using  
Formal Model

# Architectural Description Language (ADL)

[https://en.wikipedia.org/wiki/Software\\_architecture\\_description#cite\\_note-3](https://en.wikipedia.org/wiki/Software_architecture_description#cite_note-3)

# ADL

- An **architecture description language (ADL)** is any means of expression used to describe a software architecture (**ISO/IEC/IEEE 42010**).
- Many special-purpose ADLs have been developed since the 1990s, including
  - AADL (SAE standard),
  - Wright (developed by Carnegie Mellon),
  - **Acme** (developed by Carnegie Mellon),
  - xADL (developed by UCI),
  - Darwin (developed by Imperial College London),
  - **ArchC**
- **UML** is an alternative ADL

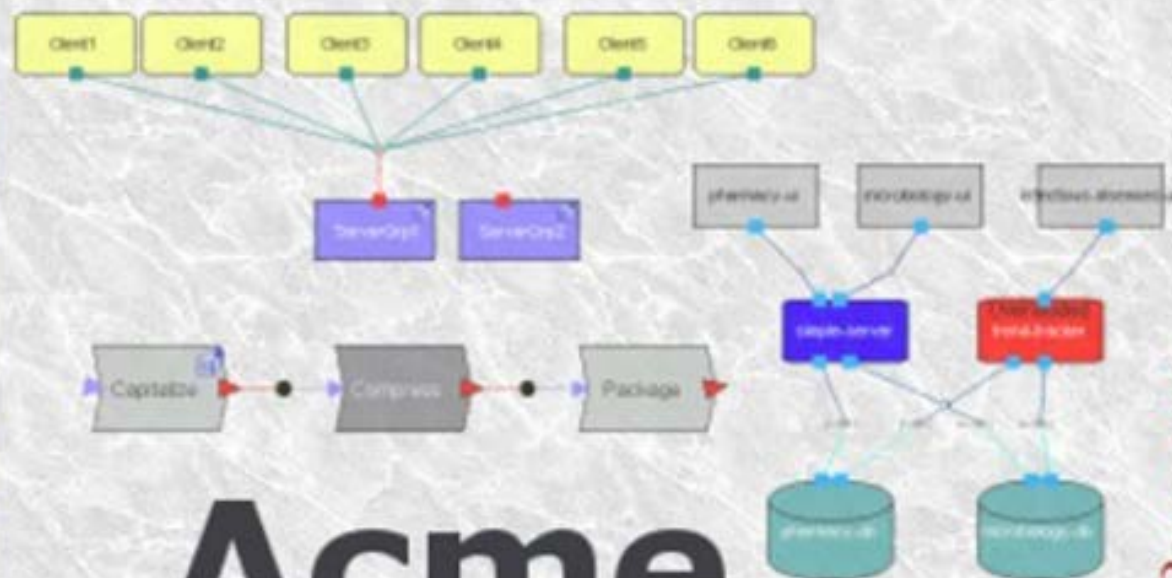
# Acme – Carnegie Mellon University

- First version released in 1997
- Similar to IBM Rational for UML
- Two components:
  - Acme ADL
  - AcmeStudio
- Can act as a vehicle for standardizing elements used across multiple ADLs



ABLE

Carnegie Mellon



# Acme STUDIO

# ArchC

- Specialized for processor architecture description.
- Allows users to generate assemblers and simulators for processors from:
  - Architecture Resources: User provides processor resource info from programmer's manual
  - Instruction Set Architecture: User enters information about each instruction such as format and syntax, behavior, and info for decoding
- Tailors the modeling environment to your processor

# ArchC Example

```
AC_ARCH(mips){
    ac_mem      MEM:5M;
    ac_cache    l2cache("dm", 256, 32, "lru", "wb", "wal");
    ac_icache    icache("dm", 16, 4, "wt", "war");
    ac_dcachecache("2w", 32, 4, "lru", "wb", "wal");

    ac_regbank  RB:34;
    ac_pipe     pipe = {IF, ID, EX, MEM, WB};
    ac_wordsize 32;

    ac_format IF_ID_Fmt = "%apc:32";
    ac_format ID_EX_Fmt = "%rd:5 %rs:32 %rt:32 %imm:32";

    ac_reg<IF_ID_Fmt> IF_ID;
    ac_reg<ID_EX_Fmt> ID_EX;
    ...
    ARCH_CTOR(mips){
        set_endian("big");

        icache.bindsTo( l2cache );
        dcachecache.bindsTo( l2cache );
        l2cache.bindsTo( MEM );
    }
};
```

Fig. 1. MIPS AC\_ARCH resource declaration.

```
AC_ISA(mips){

    ac_format Type_R="%op1:6 %rs:5 %rt:5 %rd:5 0x00:5 %op2:6";

    ac_format Type_I="%op1:6 %rs:5 %rt:5 %imm:16";

    ac_instr<Type_R> add;
    ac_instr<Type_I> load;

    ISA_CTOR(mips){
        add.set_asm("add %reg,%reg,%reg," rd,rs,rt);
        add.set_decoder(op1=0x00, op2=0x20);

        load.set_asm("lw %reg,%imm(%reg)",rt,imm,rs);
        load.set_decoder(op1=0x23);

    };
};
```

Fig. 2. MIPS ISA Description.

# UML Shortcomings as an ADL

- “[A]n ADL for software application focuses on the high-level structure of the overall application rather than the implementation details of any specific source module”
- Less formalized than ADLs
- No notion of entity restriction (styles)
- Largely a documenting language

ทำไมต้องมี Software Architecture

# Architecture

WHY

- Architecture addresses non-functional requirements

Use case diagram บอก Non-functional req. ได้ไหม?  
Class diagram บอกได้ไหม?



จงอธิบายความแตกต่างระหว่าง

# Functional vs. Non-functional

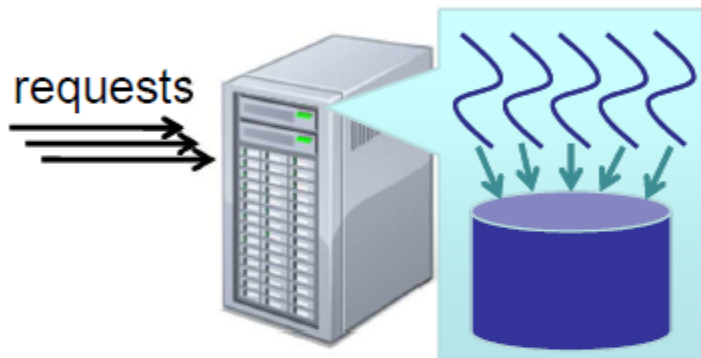
**Non-functional** มักจะมองไม่เห็นในการใช้งาน  
แรก ๆ แต่ปรากฏเมื่อเวลาผ่านไป ในการบำรุงรักษา

**Martin fowler** เรียกมันว่า **Internal quality**

<https://martinfowler.com/articles/is-quality-worth-cost.html>

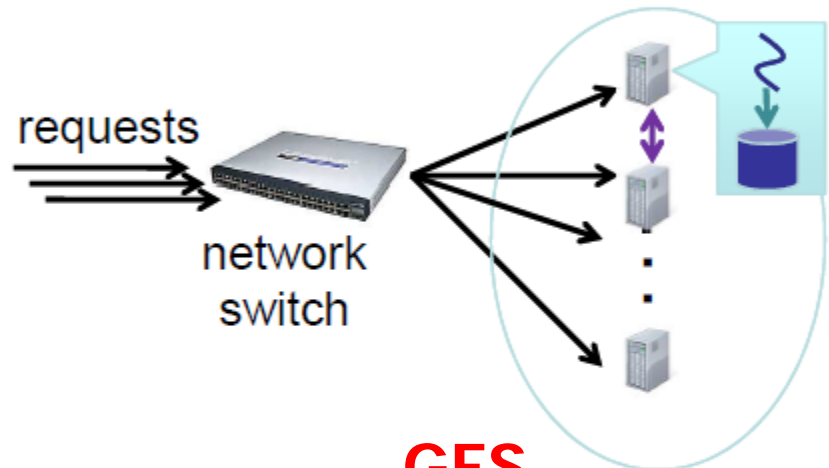
# Two Architectures for Web Search

Altavista search engine



Big server

Google search engine



**GFS**

Cluster of commodity servers

จงแจกแจงให้ความแตกต่าง

How does architecture affect system **properties**?

- Modifiability / ease of change
- Consistency of results
- System cost
- Scalability of system
- Reliability of system

ทั้งสองระบบนี้ทำ **function** ได้เท่ากันแต่  
**Function = web search**  
**Non-funct. = ...**



WHY

# Quality Factors (some non-functional reqs.)

# Quality Factors

Most of a software architect's work is focused on designing software systems **to meet a set of quality** factor requirements.

# General software quality factors

- Scalability
- Security
- Performance
- Modifiability
- Availability
- Integration

Quality ที่พบบ่อยบางส่วนเท่านั้น

# Design Trade-offs



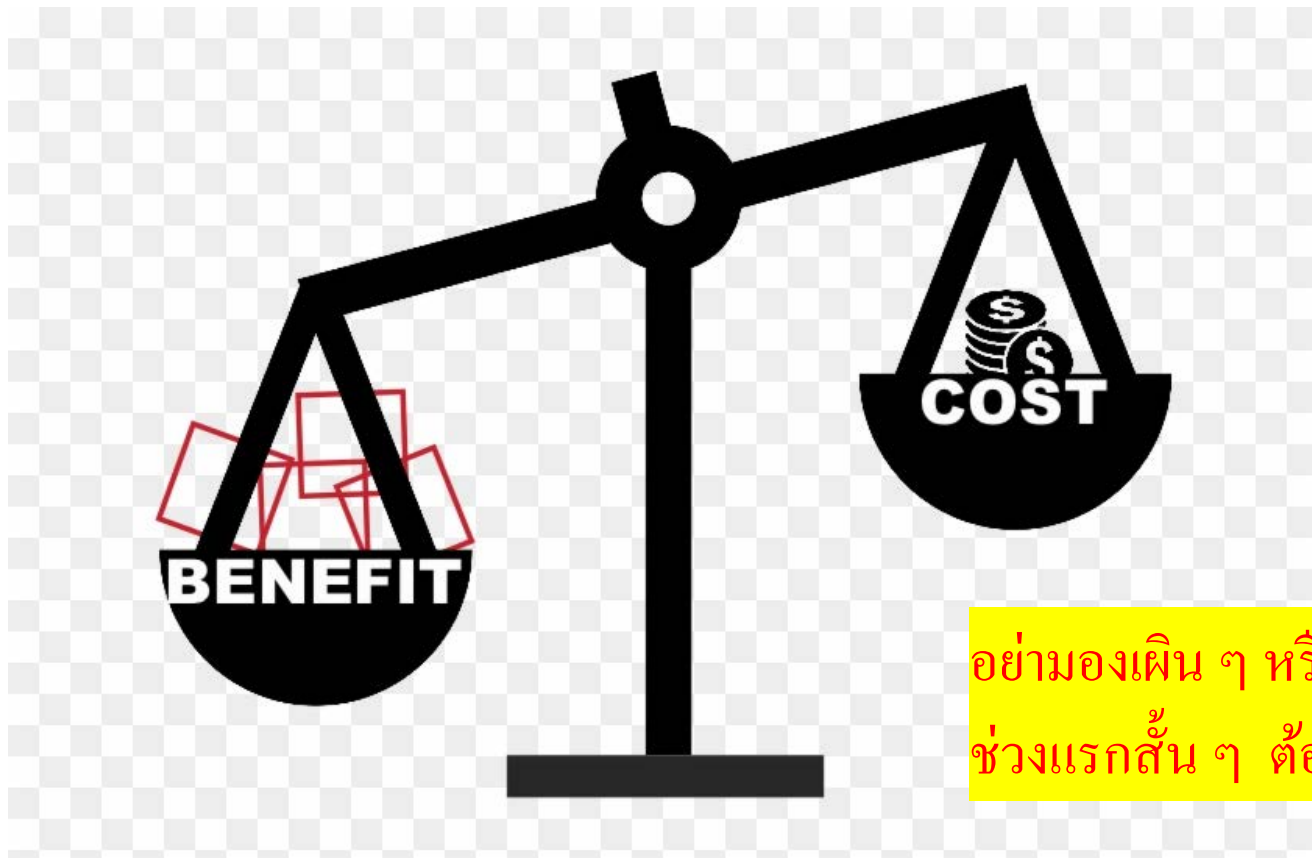
- Quality factors บางคู่ไม่แปรตามกัน
- For example, a **highly secure** system may be difficult or **impossible to integrate** in an open environment.
- A **highly available** application may trade-off **lower performance** for greater availability.
- An application that requires **high performance** may be tied to a particular platform, and hence **not be easily portable**

เลือก quality factors อย่างเหมาะสม

# When is architecture important?

- Small solution space
  - เช่น making a faster car is often no harder than adding a bigger engine
- High failure risk
  - เช่น People might die if your hospital system fails
- Difficult quality attributes
  - เช่น making email with quick performance that supports millions of users is hard
- New domain
- Product lines
  - เช่น Some sets of products share a common architecture

# Cost Benefit Analysis for SW Arch



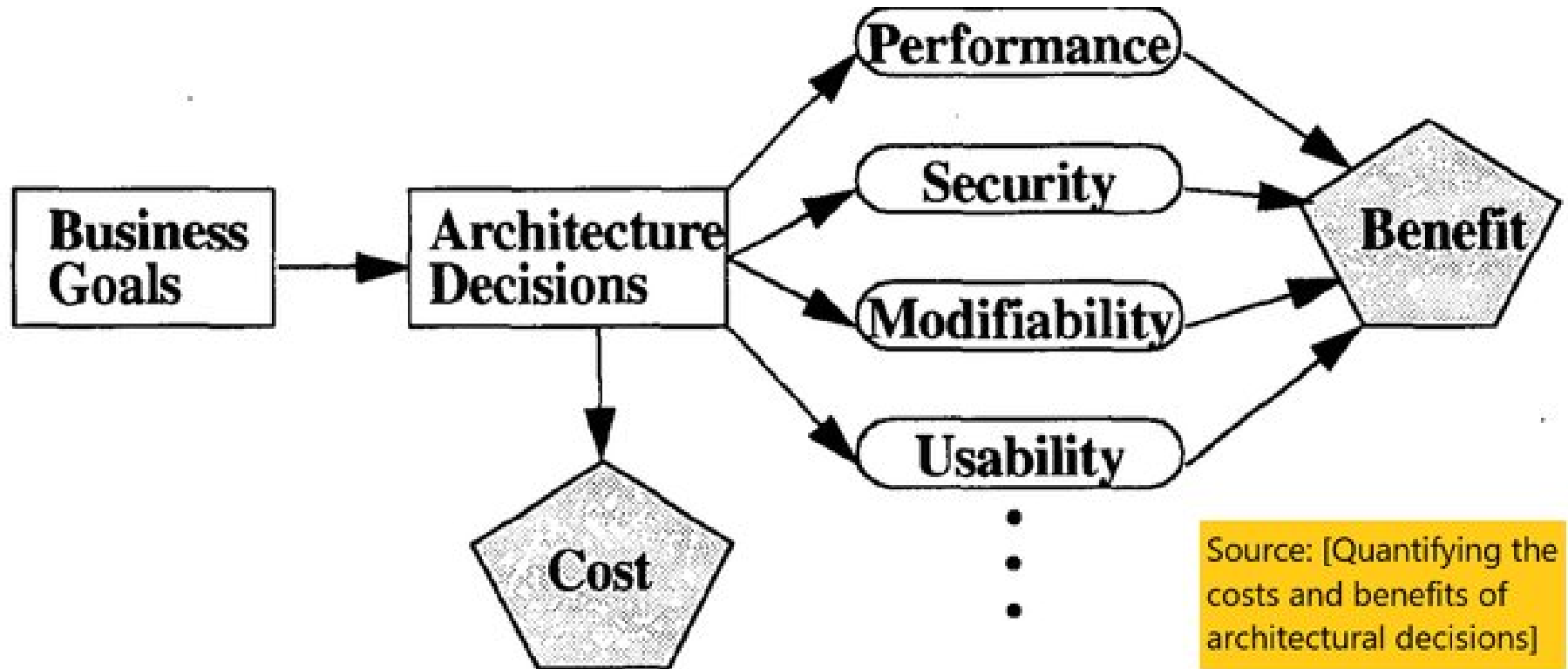
อย่ามองเผิน ๆ หรือมองแค่  
ช่วงแรกสั้น ๆ ต้องมองยาว ๆ

Organizations tend to consider costs only in terms of building a system, and they fail to consider the costs associated with maintaining and upgrading it.

# CBAM from SEI

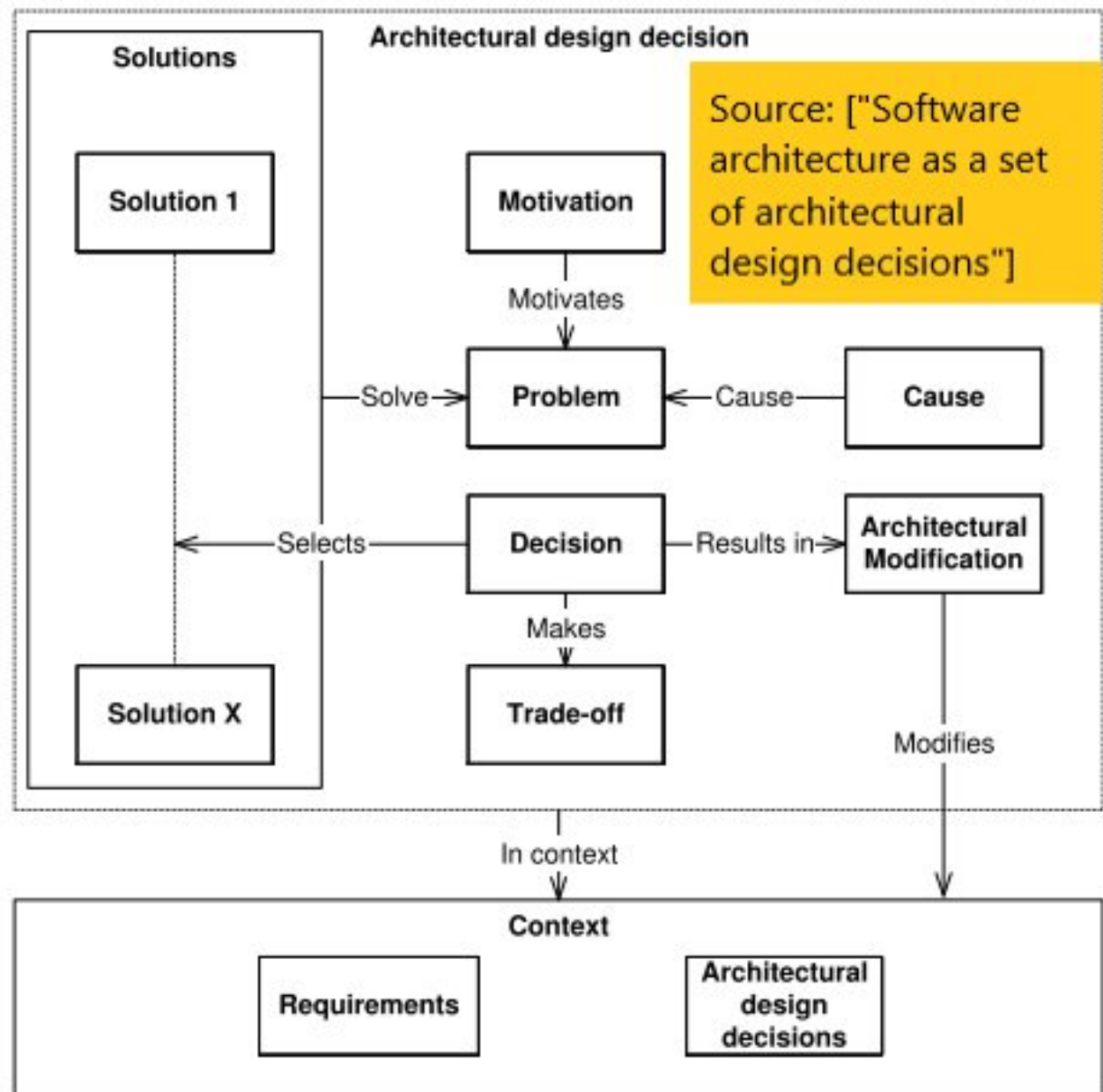
- Cost Benefit Analysis Method (CBAM)
- CBAM process consists of the following steps
  1. Choose **scenarios** and architectural strategies
  2. Assess **quality attribute** benefits
  3. **Quantify the benefits** of architectural strategies
  4. **Quantify the costs and schedule** implications of the architectural strategies
  5. **Calculate** the desirability of each option
  6. **Make** architectural design **decisions**.

# The Context of CBAM





วิทยานิพนธ์ของคุณ  
Manoj  
Mahabaleshwar  
(2020)



เราออกแบบ Arch ตามใจชอบได้ไหม?

ดีไหม ถ้ามีใครมาแนะนำวิธีการออกแบบ ไม่ต้อง  
เริ่มจากศูนย์?

# Architectural Styles



- Architectural styles are high-level design
- GoF's design patterns are low-level design



# What is a Software Architectural Style?

- Architectural Style =  
Components + Connectors + Constraints
- Primarily,
  - Components: computational elements
  - Connectors: interactions between components
  - Constraints: how components and connectors may be combined

# Questions About Architectural Styles

- What is the **design vocabulary**? 
- What are the allowable patterns?
- What is the underlying computational model?
- What are the essential invariants?
- What are common examples?
- What are **advantages and disadvantages**? 
- What are common specializations?

# Examples of Architectural Styles

- Call and return
- Data Abstraction
- Implicit Invocation (Call back)
- Layered
- Pipe and Filter
- Repository (Blackboard or Hub&Spokes)
- Tiers (3-tiers)
- Model-View-Controller
- Service Oriented Architecture (SOA)
- Web Services
- Microservices
- Etc.



## 5W2H method

Type	5W2H	Description	Countermeasure
Subject Matter	What?	What is being done? Can this task be eliminated?	Eliminate unnecessary tasks
Purpose	Why?	Why is this task necessary? Clarify the purpose.	
Location	Where?	Where is it being done? Does it have to be done there?	Change the sequence or combination
Sequence	When?	When is the best time to do it? Does it have to be done then?	
People	Who?	Who is doing it? Should someone else do it? Why am I doing it?	
Method	How?	How is it being done? Is this the best method? Is there some other way?	Simplify the task
Cost	How much?	How much does it cost now? What will the cost be after improvement?	Select an improvement method

# Software Architecture

- What – Definition
- Why – Quality factors (non-functional req.)
- When – High level design
- Where – Dev Office
- Who – SW Architect
- How – Document to communicate (model, styles)
- How many – Cost



# คำสำคัญ

- Software Architecture
- Model, Level of Abstraction
- Architectural Views
- Architectural Viewpoints
- Architectural Styles (Patterns)
- Architectural Representation/Documentation
- Architectural Decision Record (ADR)

# สิ่งเหล่านี้กำหนดว่าเราจะต้อง **model** อย่างไร

- Constraints
- Risks
- Non-functional requirements
- เช่น นำระบบ **Online shopping** ที่เรียนได้มา และเพิ่มเงื่อนไขเหล่านี้
  - Scalability
  - Security
  - Cost constraints
  - Connecting to the Legacy systems
  - 24/7 Availability
  - ...

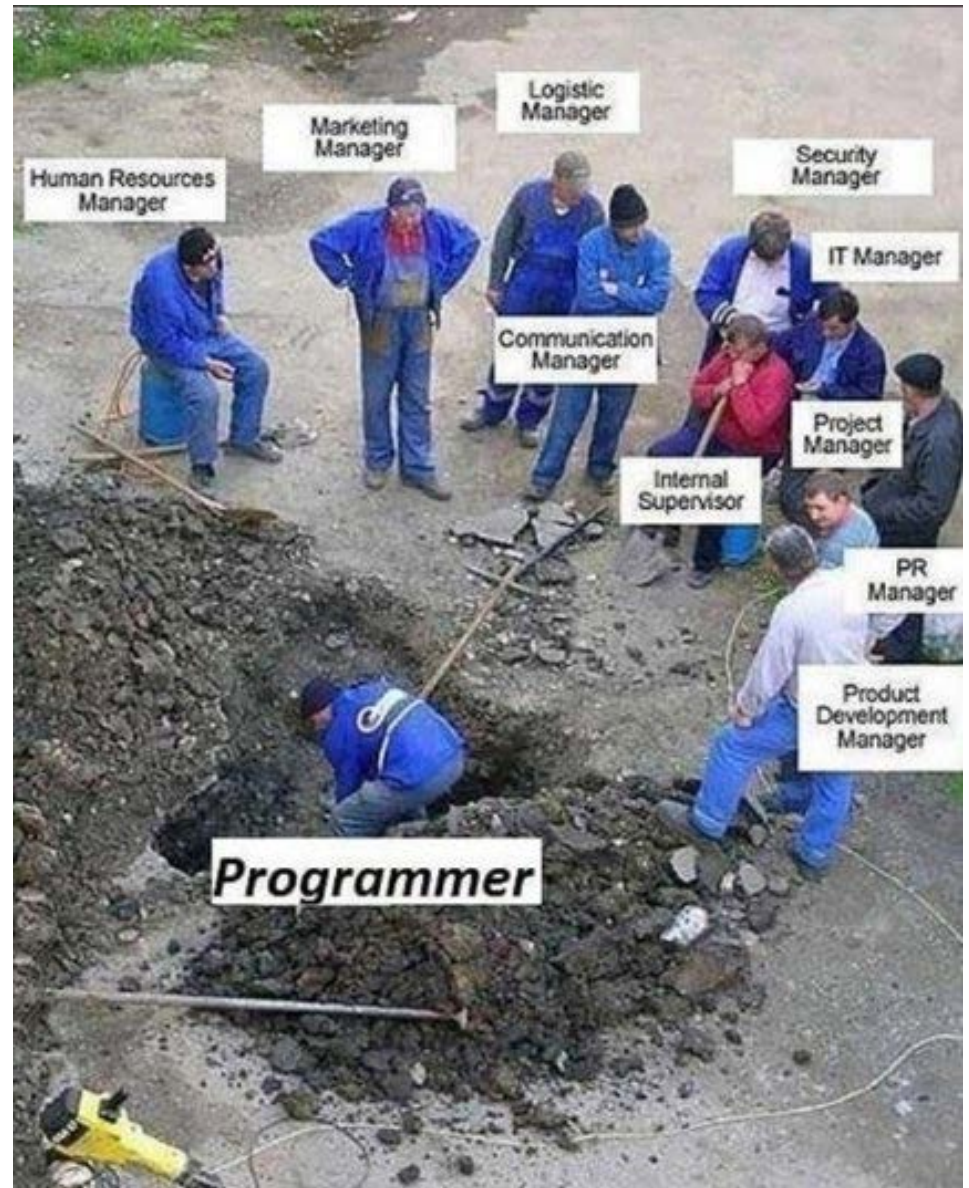
# UML for Software Architecture

- Subsystem diagram
  - Subsystem Interface (provided, required) also called “ball and socket interface”
- Component diagram
- High level sequence diagram
- Deployment diagram

# Roles in Software Development Project

- CEO
  - CIO
  - Project Champion
  - Project Manager
  - Business Analyst
  - Designer
  - Programmer
  - Tester
  - QA Officers
- .....?? Software Architect ??

... อยู่ไหน



# การบ้าน

- List 50 Keywords พร้อมอธิบายสั้นๆ ส่งบน MCV