Chapter 1 Introduction to Software Architecture

Software Architecture

- การออกแบบโครงของ software (แบบหยาบ ๆ) เท่าที่พอจะทำให้พัฒนา software ต่อได้
- มี 3 ส่วนประกอบคือ
  1. **Components**
  2. **Relationship** (ทั้งระหว่าง components และ components ข้างนอก)
  3. **Property**
- เน้น High level (Abstraction สูง)

Architectural design

เป็นการออกแบบในระดับ High-level

- Top-down approach
- ทำแค่ **Subsystem Design** พอ (มองแค่ก้อน ๆ ของ subsystem ห้ามลงรายละเอียด)
- จะมีอีกระดับคือ low-level approach >> ทำ Detailed design (ซึ่งลึกไปสำหรับวิชานี้)
- ถ้ามีการเปลี่ยนแปลง Scope สามารถทำได้ง่ายกว่า

> 💡 หาก **Abstraction** สูง **Detail** รายละเอียดจะยิ่งน้อย (คร่าว ๆ)

Model

แบบจำลอง (ตัวแทน) ของสิ่งที่เราสนใจ เช่น แบบจำลองของ software architecture ซึ่งสามารถเขียนหรือบรรยายได้หลายแบบ เช่น

- **Formal model** >> คนอื่นเข้าใจยาก
  - เช่น RegEx, Automata
- **Semi-formal model** >> เอาแผนภาพมาผสมกับคำอธิบาย
  - UML
- **Informal model** >> เขียนง่าย แต่รายละเอียดเยอะไป
  - เช่น พรรณาด้วยภาษาไทย

> 💡 Model คืออะไร >> แบบจำลองของสิ่งที่เราสนใจ
> Model สำคัญยังไง >> ทำให้ผู้อ่านเห็นภาพ (เพราะให้ลง Detailed เลยจะลำบาก)
> แล้วจะออกแบบ SW Arch ได้อย่างไร >> Semi-formal model (UML + คำอธิบาย)

Architectural Views

- **View** >> สิ่งที่อธิบายออกมา (สิ่งที่เห็น) จาก Viewpoints เกี่ยวกับ Software นั้น ๆ
  - เวลาทำ Software Architecture >> หนึ่งระบบสามารถมีหลาย ๆ View เพื่อใช้อธิบายได้
- แต่ละ View ประกอบไปด้วย **Components** และ **Connectors**
- View แบ่งออกเป็น 2 ประเภทใหญ่ ๆ
  1. **Design time** >> ด้วยความที่เป็นขณะดีไซน์ จึงใช้ **Subsystem Diagram**
  2. **Run time** >> ด้วยความที่สนใจขณะ deploy จึงใช้ **Component Diagram**
- วิชานี้ใช้ของ **Kruchten (4+1)**
  - **Logical view** >> Class and Package diagram
    - ทำเพื่อเอาไว้ใช้ discuss กับ end user (user ควรมาอ่านแล้วเข้าใจได้)
  - **Physical view** >> Component diagram

        - สนใจ physical topology ของระบบ
    - **Process view** >> Running components
        - สนใจเรื่อง integration, process และ scalability
    - **Development view** >> Deployment diagram
        - ให้ programmer ใช้
    - **Scenarios (Use case view)** >> Sequential diagram (use case + stories)

> 💡 ตอนออกแบบ Architecture มีได้หลาย diagram เพราะ diagram นึงก็แสดงถึงมุมมอง ๆ (view) หนึ่ง

Architectural Viewpoints

Viewpoint >> generic template ในการเขียน view

- นึกภาพง่าย ๆ ก็เหมือนกำหนดจุดที่จะให้เรายืนมองสิ่ง ๆ นึง
- เช่น Functional Viewpoint ทำให้ได้ Functional view

FR & NFR

- เรามี Software Architecture ไปเพื่อแสดงให้เห็นถึง non-functional requirements ใน design ของระบบ
- **Quality Factors** คือ set ของ NFR ที่ระบบควรมี
    - เช่น Scalability, Security, Performance, Availability, Performance, Portability
- Quality Factors บางอย่างเป็น Tradeoff กัน
    - เช่น highly secure ก็จะ Integrate ยาก
    - เช่น highly available ก็อาจจะมี lower performance
    - เช่น high performance ก็อาจจะ portable ยาก

> 💡 UML ทั่วไป เช่น class diagram, use-case diagram นอก non-functional requirement ไม่ได้

Architectural Styles

- **Architectural Styles** คือ pattern ที่มีคนเคยออกแบบมาแล้ว ทำให้ไม่ต้องเริ่มออกแบบใหม่จากศูนย์
    - เช่น 3-Tier Architecture, Microservices, Model View Controller
- มีส่วนประกอบด้วยกัน 3 ส่วน >> **Components**, **Connectors**, **Constraints**