Common Architecture Styles

เพื่อเเสดงให้เห็นถึงพื้นที่สมบูรณ์ของตัวเลือกสถาปัตยกรรม การใช้งาน การเลือกสไตล์ เเละเพื่อให้เข้าใจถึงความเเตกต่างระหว่างสไตล์ จะช่วยให้กรอบการทำงานเพื่อให้การทำงานรักษาโครงสร้างของระบบไว้ ซึ่งจะมี components มาเพื่อทำให้เข้าใจง่ายขึ้น

**Pipes and filters**

คือรูปแบบการทำงานข้อมูลที่ต้องการการประมวลผลที่มีความซับซ้อนทำเป็นที่ขั้นตอน ซึ่งจะมีเป็นท่อ (pipe) ที่มีข้อมูลส่งผ่านซึ่งจะมีตัวกรองข้อมูล ( filter ) ในการกรองข้อมูลจะมีการประมวลผลข้อมูลเเละส่งไปยังอีกตัวกรองนึง

**Data Abstraction and Object-Oriented Organization**

คือเป็นส่วนสำคัญในการออกเเบบเเละการพัฒนาซอฟต์เเวร์ที่เน้นไปที่การจัดการข้อมูลเเละโครงสร้างของโปรเเกรม

* Data Abtraction ใช้เป็นการลดความซับซ้อนของโค้ดเเละทำให้โปรแกรมมีความเข้าใจเเละบำรุงง่าย ใช้ ใช้กับงาน
* Object-Oriented Organization เป็นหลักการซ่อนข้อมูล การสืบทอด การอธิบายโครงสร้างให้เข้าใจง่ายเเละให้เห็นเป็นรูปภาพ

**Event-based, Implicit Invocation**

เป็นเเนวคิดที่เกี่ยวข้องกับการโต้ตอบของโปรแกรมต่อเหตุการณ์

* Event-based โปรแกรมที่ตอบสนองกับเหตุการณ์ที่เกิดขึ้น เเละเก็บข้อมูลเพื่อเป็นตัวจัดการกับเหตุการณ์กับเหตุการณ์ ใช้งานกับเหตุการณ์ที่เกิดขึ้นที่จะส่งกระทบในระบบ
* Implicit Invocation โปรแกรมไม่รู้จักหรือการระบุชื่อของตัวอย่าง

การใช้งานจะเป็นการใช้งานร่วมกันช่ววยเพิ่มความยืดหยุ่นของงานเเละการปรับปรุงในการออกเเบบเเละการพัฒนาระบบซอฟต์แวร์ ส่วนใหญ่จะใช้กับการพัฒนาที่ต้องการการตอบสนองกับเหตุการณ์ที่เกิดขึ้นเเบบไม่เเน่นอนหรือไม่สม่ำเสมอ

**Layered Systems**

เป็นหนึ่งในสไตล์การออกแบบที่พบมากในโลกของซอฟต์แวร์ สไตล์นี้มักถูกใช้ในการสร้างระบบที่มีความซับซ้อน โดยการแบ่งแยกฟังก์ชันหรือบทบาทต่าง ๆ ของระบบออกเป็นชั้น (Layer) หรือเลเยอร์ต่าง ๆ โดยแต่ละเลเยอร์จะมีความรับผิดชอบและหน้าที่ที่แตกต่างกัน โดยทั่วไปแล้วมักจะมีแบ่งเป็นหลายเลเยอร์ เช่น:

1. Presentation Layer: เลเยอร์ที่รับผิดชอบในการแสดงผลและโต้ตอบกับผู้ใช้ มักจะเป็นส่วนของโปรแกรมหรืออินเทอร์เฟซที่มองเห็นได้ของระบบ

2. Business Logic Layer: เลเยอร์ที่รับผิดชอบในการประมวลผลทางธุรกิจหรือระบบ ซึ่งมักจะประกอบด้วยการตรวจสอบข้อมูล ประมวลผลตามกฎระเบียบ หรือการดำเนินการทางธุรกิจอื่น ๆ

3. Data Access Layer: เลเยอร์ที่รับผิดชอบในการเชื่อมต่อกับแหล่งข้อมูล เช่น ฐานข้อมูล และประมวลผลคำสั่ง SQL เพื่อเรียกใช้ข้อมูล

**Repositories**

การจัดเก็บข้อมูลที่ใช้ในระบบซอฟต์แวร์ โดยทั่วไปแล้ว Repositories จะเป็นส่วนหนึ่งของแอพพลิเคชันที่ใช้ในการจัดเก็บและการจัดการข้อมูล เพื่อให้สามารถเข้าถึงข้อมูลได้อย่างมีประสิทธิภาพและง่ายต่อการบริหารจัดการ

1. Centralized Repository (Repositories แบบจัดเก็บในส่วนกลาง): ในแบบนี้ มี Repository หรือฐานข้อมูลใหญ่ที่ใช้เก็บข้อมูลทั้งหมด ซึ่งแอพพลิเคชันทุกตัวจะเข้าถึงข้อมูลผ่าน Repository เดียว

2. Distributed Repositories (Repositories แบบกระจาย): ในแบบนี้ มี Repositories หลายๆ ส่วนที่มีการจัดเก็บข้อมูลแยกกันตามภูมิภาคหรือพื้นที่ต่างๆ โดยแต่ละส่วนอาจมีการจัดเก็บข้อมูลและการจัดการข้อมูลของตนเอง ซึ่งในบางกรณีอาจมีการซิงโครไนซ์ข้อมูลระหว่าง Repositories เพื่อให้ข้อมูลเป็นความสมบูรณ์ทั่วทั้งระบบ

3. Domain-specific Repositories (Repositories แบบเฉพาะด้านหรือโดเมน): ในแบบนี้ มี Repositories ที่เน้นการจัดเก็บข้อมูลที่เฉพาะเฉพาะในโดเมนหรือหัวข้อที่เฉพาะเจาะจง เช่น Repositories สำหรับจัดเก็บข้อมูลลูกค้า ข้อมูลการทำธุรกรรมการซื้อขาย เป็นต้น

4. Data Access Layer (DAL): คือชั้นหรือส่วนของระบบที่ใช้ในการเข้าถึงฐานข้อมูลหรือ Repositories ต่างๆ โดยทั่วไปมักจะมีการแยกชั้นหรือการเขียนโค้ดที่ใช้ในการเข้าถึงข้อมูลออกเป็นชั้นของตนเอง เพื่อให้ง่ายต่อการบริหารจัดการและการปรับปรุง

**Table-driven interpreters**

เป็นแนวคิดในการออกแบบและสร้างตัวแปลภาษาที่ใช้ตารางหรือข้อมูลที่แปลงเป็นรูปแบบของตารางเพื่อช่วยในการแปลงและประมวลผลข้อมูลที่ต้องการตรวจสอบหรือแปลงโดยอัตโนมัติ

* การออกแบบ: ตัวแปลภาษาใช้โครงสร้างข้อมูลแบบตาราง เพื่อระบุกฎและการกระทำที่ต้องการในขณะที่แปลงหรือประมวลผลข้อมูล
* การทำงาน: เมื่อมีข้อมูลเข้ามา ตัวแปลภาษาจะดำเนินการตรวจสอบตามตารางเพื่อดำเนินการต่อไปตามกฎและการกระทำที่กำหนด
* ความยืดหยุ่น: การใช้ตารางทำให้ง่ายต่อการเปลี่ยนแปลงกฎหรือการกระทำโดยไม่ต้องแก้ไขโค้ดตัวแปลภาษาโดยตรง
* การบำรุงรักษา: การแก้ไขข้อผิดพลาดหรือการปรับปรุงสามารถทำได้โดยการแก้ไขหรือเพิ่มเติมในตารางโดยไม่ต้องมีการเปลี่ยนแปลงในโค้ดหลักของตัวแปลภาษา

Other familiar architecture

1. RESTful Architecture : ออกแบบระบบให้ทำงานตามหลักของ REST (Representational State Transfer) โดยใช้ HTTP เป็นโพรโตคอลสำหรับการสื่อสาร มีการใช้งานแบบ stateless และการจัดการข้อมูลผ่านการใช้งานของ HTTP methods เช่น GET, POST, PUT, DELETE

2. Serverless Architecture : สถาปัตยกรรมที่ไม่ต้องกังวลเกี่ยวกับการจัดการเซิร์ฟเวอร์เอง แต่จะใช้บริการ cloud platform เพื่อจัดการด้านพื้นฐานและขีดจำกัดทรัพยากรให้เอง เช่น AWS Lambda, Azure Functions

3. Event-Driven Architecture : ออกแบบระบบให้ทำงานตามเหตุการณ์ที่เกิดขึ้น โดยมีการส่งและรับเหตุการณ์ระหว่างส่วนประกอบของระบบ เพื่อให้การทำงานเป็นไปตามลำดับและสอดคล้องกันได้

4. Containerized Architecture (สถาปัตยกรรมในคอนเทนเนอร์): ใช้คอนเทนเนอร์เป็นหน่วยของการจัดการแอปพลิเคชัน ทำให้ง่ายต่อการเคลื่อนย้ายและปรับปรุงระบบ รวมถึงลดปัญหาในการจัดการสภาพแวดล้อมของแอปพลิเคชัน

5. Model-View-Controller (MVC) Architecture : แบ่งแยกโครงสร้างของแอปพลิเคชันออกเป็น 3 ส่วนหลักคือ Model (โมเดล) เป็นข้อมูลและธุรกรรมทางธุรกิจ View (วิว) เป็นส่วนที่แสดงผลให้ผู้ใช้เห็น และ Controller (คอนโทรลเลอร์) เป็นส่วนที่ควบคุมและประมวลผลข้อมูล

"Heterogeneous Architecture" หมายถึง การสร้างระบบซอฟต์แวร์ที่ประกอบไปด้วยส่วนต่าง ๆ ที่มีลักษณะและเทคโนโลยีที่แตกต่างกัน ซึ่งส่วนต่าง ๆ เหล่านี้อาจมาจากแหล่งที่มาที่แตกต่างกันหรือถูกสร้างขึ้นด้วยเครื่องมือและเทคโนโลยีที่ไม่เหมือนกัน เช่น ซอฟต์แวร์ที่เขียนด้วยภาษาโปรแกรมมิ่งแตกต่างกัน เทคโนโลยีฐานข้อมูลที่แตกต่างกัน หรือแม้กระทั่งแพลตฟอร์มการเปิดเข้าถึงข้อมูลที่ไม่เหมือนกัน เป็นต้น

**Heterogeneous Architecture**

การสร้างระบบซอฟต์แวร์ที่ประกอบด้วยส่วนต่าง ๆ ที่มีลักษณะและเทคโนโลยีที่แตกต่างกันและมาจากแหล่งที่มาที่แตกต่างกัน โดยมีจุดเด่นในการใช้เทคโนโลยีที่เหมาะสมสำหรับแต่ละส่วนและความยืดหยุ่นในการขยายขนาดและปรับปรุงระบบให้เข้ากับความเปลี่ยนแปลงของธุรกิจได้อย่างมีประสิทธิภาพ