Pipes and Filters คือ pipe เป็นท่อที่ใช้สำหรับส่งต่อข้อมูลไปยัง filter โดย filter จะทำหน้าที่รับข้อมูลมาประมวลผล และส่งข้อมูลไปยังท่อถัดๆไป

- **การใช้งาน:** แต่ละส่วน filter จะมีท่อนำเข้าและท่อผลลัพธ์ ซึ่งทำงานโดยอ่านข้อมูลจากท่อนำเข้าและส่งผลลัพธ์ไปที่ท่อผลลัพธ์ การประมวลผลเกิดขึ้นโดยการทำการแปลงใน filter และผลลัพธ์ถูกส่งไปยังส่วนถัดไป
- **เหมาะสำหรับ:** ระบบที่มีการประมวลผลข้อมูลเป็นลำดับแบบท่อที่ต้องการการแปลงและประมวลผลตามลำดับที่เกี่ยวข้อง ตัวอย่างเช่น การใช้ในโปรแกรม Unix shell, การเขียนโปรแกรมฟังก์ชัน, และระบบกระจาย

Data Abstraction and Object-Oriented Organization คือ
แนวคิดในการออกแบบโปรแกรมที่เน้นการห่อหุ้มข้อมูลและการดำเนินการในรูปแบบของ abstract data types หรือ objects (วัตถุ)
ซึ่งเป็นตัวอย่างของคอมโพเนนท์ที่เรียกว่า managers เพื่อรักษาความสมบูรณ์ของทรัพยากร วัตถุทำงานร่วมกันผ่านการเรียกใช้ฟังก์ชันและกระบวนการ

- การใช้งาน: ในระบบนี้ ข้อมูลและการทำงานกับข้อมูลถูกจัดในรูปของ abstract data types หรือ objects โดยจะตอบสนองกับ objects อื่นๆ ผ่านการเรียกใช้งาน function
- **เหมาะสำหรับ:** ระบบที่มีความซับซ้อนและต้องการการจัดการข้อมูลและการทำงานที่หลากหลาย เพราะ Data Abstraction and Object-Oriented Organization ช่วยในการแบ่งปัญหาออกเป็นส่วนๆ ที่สามารถจัดการแยกกันได้ และมีความยืดหยุ่นในการพัฒนาและบำรุงรักษา

Event-based, Implicit Invocation Pattern คือ แนวคิดในการผสานรวมระบบโดยไม่ใช้การเรียกโปรแกรมโดยตรง แต่ใช้การประกาศ event และการ register ใน event นั้น ๆ เพื่อให้ระบบเรียกใช้งานโปรแกรมที่ register เมื่อ event ถูกประกาศ

- การใช้งาน: ทุกครั้งที่ event เกิดขึ้น, ส่วนต่าง ๆ ในระบบที่สนใจจะ register เพื่อตอบรับ event นั้นๆ และระบบจะเรียกใช้งานโปรแกรมหรือกระบวนการที่ register เมื่อ event ถูกประกาศ
- **เหมาะสำหรับ:** เหมาะสำหรับระบบที่ต้องการความยืดหยุ่นและการตอบรับต่อเหตุการณ์ได้อย่างมีประสิทธิภาพ เช่น ระบบที่มีโครงสร้างที่ซับซ้อนและต้องการการเชื่อมโยงส่วนต่าง ๆ ของระบบให้สามารถทำงานร่วมกันได้ ระบบที่ต้องการการปรับปรุงโปรแกรมหรือเพิ่มเติมตลอดเวลาโดยไม่กระทบต่อโครงสร้างหลักของระบบ หรือระบบที่ต้องการการแสดงผลแบบ Interactive และการรับรองความถูกต้องของข้อมูล. การใช้งานแบบ Event-based, Implicit Invocation ช่วยลดความซับซ้อนในการเชื่อมโยงและการปรับปรุงระบบอย่างยืดหยุ่น

Layered Systems คือ ระบบที่ถูกจัดเป็นลำดับชั้นหรือระดับที่แตกต่างกัน โดยแต่ละชั้นจะให้บริการกับชั้นที่อยู่ด้านบน
และทำหน้าที่เป็นลูกค้าต่อชั้นที่ต่ำกว่า ในลำดับที่เหลือ
การสื่อสารระหว่างชั้นจะถูกกำหนดโดยโปรโตคอลที่กำหนดว่าชั้นนี้จะทำงานอย่างไรกับชั้นที่ต่ำกว่าและสูงกว่า

- การใช้งาน: ระบบที่ต้องการการออกแบบที่มีระดับความสำคัญและทำให้ง่ายต่อการจัดการ มีประโยชน์ในการพัฒนาแบบขั้นบันได รวมถึงการสนับสนุนการเพิ่มฟีเจอร์และการปรับปรุง
- เหมาะสำหรับระบบ: ระบบที่มีโครงสร้างซับซ้อนและต้องการการจัดการที่เป็นลำดับ เช่น ระบบโปรโตคอลการสื่อสารที่มีชั้นแต่ละชั้นทำหน้าที่ในระดับที่แตกต่างกัน ระบบฐานข้อมูลที่มีชั้นสำหรับการจัดการข้อมูล และระบบปฏิบัติการที่มีการจัดลำดับการทำงานของระบบ

Repositories คือ รูปแบบของสถาปัตยกรรมที่ประกอบด้วยสองประเภท: โครงสร้างข้อมูลที่เป็นจุดศูนย์แทนสถานะปัจจุบัน
และชุดของคอมโพเนนต์ที่เป็นอิสระทำงานกับตัวจัดเก็บข้อมูลหลัก
การปฏิสัมพันธ์ระหว่างตัวจัดเก็บและคอมโพเนนต์ภายนอกจะแตกต่างกันไปขึ้นอยู่กับระบบ

- การใช้งาน: จัดการแหล่งความรู้แยกต่างหากที่ทำงานร่วมกับคลังข้อมูลกลาง
  เพื่อปรับปรุงระบบโดยให้สามารถปรับเปลี่ยนและปรับใช้ได้อย่างยืดหยุ่นตามสถานะปัจจุบันของระบบ
- เหมาะสำหรับระบบ: ระบบที่ต้องการการแบ่งปันข้อมูลในรูปแบบที่สามารถเข้าถึงได้จาก component ต่าง ๆ โดยอิสระ

Table Driven Interpreters คือ Interpreter ถูกออกแบบให้เป็นเหมือน virtual machine ที่ทำหน้าที่แปลและรันโปรแกรมในรูปแบบของ pseudo-code หรือตารางคำสั่ง ตารางที่ใช้ในการแปลมีหน้าที่เก็บข้อมูลที่ใช้ในกระบวนการแปลภาษา

- การใช้งาน: ใช้เพื่อสร้างเครื่องมือเสมือนที่ทำหน้าที่ในการประมวลผลโปรแกรมในรูปแบบที่ไม่ได้รับการคอมไพล์แบบเต็มรูปแบบ (interpreted), และมีความยืดหยุ่นในการปรับแต่งคำสั่งหรือพฤติกรรมโปรแกรม
- เหมาะสำหรับระบบ: ระบบที่ต้องการแปลงคำสั่งหรือโค้ดจากภาษาหรือรูปแบบหนึ่งเป็นอีกภาษาหรือรูปแบบหนึ่ง

Other Familiar Architectures มี

Distributed Processes คือ กระบวนการที่ทำงานและมีการทำงานต่อกันใน Distributed Systems

- **การใช้งาน:** ทำงานบนหลายเครื่องคอมพิวเตอร์ทำงานร่วมกันเพื่อทำภารกิจหนึ่ง
- **เหมาะกับระบบ:** ระบบกระจายที่ต้องการการจัดระเบียบกระบวนการแบบหลายกระบวนการ

Main Program/Subroutine Organizations คือ ลักษณะการออกแบบระบบโดยให้โครงสร้างของระบบนั้นประกอบด้วย main program ซึ่งเป็นตัวควบคุมหลักและ subroutines ที่เป็นโค้ดย่อยที่ถูกเรียกใช้โดย main program เพื่อทงานที่เฉพาะเจาะจง โดย main program มักจะมีลูปควบคุมที่เรียกใช้ subroutines ตามลำดับที่กำหนด

- **การใช้งาน:** เริ่มต้นจากการกำหนดการทำงานหลักของระบบใน **main program** ซึ่งเป็นตัวควบคุมหลักของโปรแกรม และทำการเรียก **subroutines เพื่อทำงานที่ได้รับมอบหมาย**
- **เหมาะกับระบบ:** ที่มีโครงสร้างแบบขั้นตอนและไม่มีความซับซ้อนมาก

Domain-Specific Software Architectures คือ
แนวคิดในการพัฒนารูปแบบโครงสร้างซอฟต์แวร์ที่ถูกพัฒนาขึ้นเพื่อให้เหมาะสมกับโดเมนหรือประเภทของแอปพลิเคชันที่เฉพาะเจาะจง

- **การใช้งาน:** ใช้ในกระบวนการพัฒนาซอฟต์แวร์ การใช้ Domain-specific software architectures
  ช่วยให้สามารถสร้างโครงสร้างซอฟต์แวร์ได้อย่างรวดเร็วและเป็นระบบ
- **เหมาะกับระบบ:** ที่มีความซับซ้อนและเฉพาะเจาะจงในโดเมนหนึ่ง เช่น ระบบที่เกี่ยวข้องกับวิทยาศาสตร์, การแพทย์,
  หรืออุตสาหกรรมที่มีความซับซ้อน

State Transition Systems คือ รูปแบบการออกแบบที่พบบ่อยในระบบที่มีลักษณะการทำงานแบบ reactive หรือตอบสนองต่อเหตุการณ์ ระบบประกอบด้วยเซตของสถานะ (states) และเซตของการเปลี่ยนสถานะที่มีชื่อ (named transitions)
ที่ทำให้ระบบเคลื่อนไหวจากสถานะหนึ่งไปสู่อีกสถานะหนึ่ง

- **การใช้งาน:** ใช้ในการแสดงและออกแบบระบบที่ต้องการตอบสนองต่อเหตุการณ์หรือสถานการณ์ต่าง ๆ ได้ เช่น ระบบควบคุม,
  ระบบอัตโนมัติ, หรือการจัดการสถานะของระบบ
- **เหมาะกับระบบ:** ระบบที่ต้องการการจัดแบ่งที่เชื่อมโยงกับการเปลี่ยนสถานะ

Process Control Systems คือ ระบบที่ถูกออกแบบมาเพื่อควบคุมและปรับการทำงานของกระบวนการ โดยระบบนี้จะใช้ Feedback Loop รับข้อมูลจากเซ็นเซอร์ที่ตรวจวัดสถานะหรือข้อมูลจากสภาพแวดล้อม
และนำข้อมูลนี้มาปรับค่าของระบบเพื่อควบคุมสภาพแวดล้อมให้ตรงตามเป้าหมาย

- **การใช้งาน:** ใช้ในงานที่ต้องการควบคุมกระบวนการผลิตอย่างต่อเนื่อง เช่น ในอุตสาหกรรม, การผลิต, หรือระบบที่ต้องการควบคุมอุณหภูมิ, ความดัน, หรือปริมาณวัสดุต่าง ๆ ในกระบวนการผลิต
- **เหมาะกับระบบ:** ระบบที่ต้องการการควบคุมแบบตอบสนองที่ไดนามิกในสภาพแวดล้อมทางกายภาพ

Heterogeneous Architectures คือ การผสมผสานระบบที่มีลักษณะแตกต่างกันมารวมกันในระบบหนึ่ง ๆ
ซึ่งทำให้ระบบมีความหลากหลายทางสถาปัตยกรรม

การใช้งาน มี 3 แบบ

- **Hierarchy:** การใช้สไตล์แต่ละอันภายในองค์ประกอบของระบบ
- **Connectors:** การแยกคอนเน็กเตอร์เป็นลำดับชั้น ทำให้สามารถนำไปใช้ภายในโครงสร้างอื่น ๆ ได้
- **Mixture of Styles:** การให้ส่วนหนึ่งของระบบใช้สไตล์ที่แตกต่างกันได้ ตลอดจนการให้ component ทำงานผ่านคอนเน็กเตอร์แบบต่าง ๆ เพื่อให้ได้ระบบที่มีลักษณะการทำงานที่หลากหลาย

เหมาะกับระบบ: ระบบที่มีความซับซ้อนและต้องการผสานรวมความสามารถและลักษณะการทำงานจากสไตล์ต่าง ๆ
เพื่อให้ได้ระบบที่ตอบสนองต่อความต้องการหลากหลาย หรือระบบที่มีความต้องการในการผสานรวมและใช้ประโยชน์จากคอนเน็กเตอร์ที่แตกต่างกัน เพื่อให้รองรับความหลากหลายของการสื่อสารและการเชื่อมโยงระหว่าง component

An Interpreter Using Different Idioms for the Components

Rule-base system วิธีการทำงานของระบบที่ใช้กฎในการแก้ปัญหา โดยระบบนี้ใช้กฎที่ถูกระบุโดยผู้เชี่ยวชาญเพื่อแก้ปัญหา และต้องมีตัวตีความ (interpreter) เพื่อทำให้กฎเหล่านี้สามารถทำงานได้ โดยลักษณะพื้นฐานของ rule-base มีดังนี้

- Pseudo code ที่จะดำเนินการ
- Interpreter ตัวตีความ (หัวใจหลัก)
- สถานะการควบคุม interpreter
- สถานะของโปรแกรมที่รันอยู่

Rule-base system ให้ความสำคัญกับบริบท การเพิ่มกลไลพิเศษเพื่อสร้างความสะดวกจะทำให้เกิดความซับซ้อน ซึ่งทำให้ original interperter หายไปผ่านการไหลข้อมูล

อย่างไรก็ตาม ตัวแบบของ interpreter ถูกค้นพบใหม่ จากมุมมองนี้ การเพิ่มขึ้นทำให้ดูง่าย และอธิบายได้เข้าใจกว่า เช่น

- Knowledge base ยังเป็นโครงสร้างที่เรียบง่าย แค่เพิ่มโครงสร้างย่อยมาแยกสิ่งที่ใช้กับไม่ใช้
- Rule Interperter เป็นหัวใจที่ถูกขยายด้วยลักษณะการตีความ โดยที่ตัวตีความนี้ได้รับการนำไปใช้เป็นตัวตีความตามรูปแบบตาราง.
- Rule and element ข้อมูลถูกนำมาใช้เป็นตัวเปลี่ยนแปลงกฎและข้อมูล ให้เป็นการเรียกใช้งานที่มีลำดับ
- Working Memory แทนสถานะปัจจุบันของโปรแกรมที่ทำงานบนเครื่องจำลอง

Interface ที่ค้นพบใหม่ไม่เป็นเปลี่ยนไปจากเดิม เพียงแค่มีเส้นสองเส้นที่ interpreter control เรียกใช้งานเพิ่มขึ้นมา