รายงาน

SOFTWARE ARCHITECTURAL

จัดทำโดย

นายฐานทัพ โชตพินิช รหัสนักศึกษา 5904062630080

นางสาวณัฐธิดา หงษ์ไทย รหัสนักศึกษา 5904062630110

นายธนากร ช่วยคิด รหัสนักศึกษา 5904062630233

เสนอ

รองศาสตร์จารย์ สุชาดา รัตนคงเนตร

รายงานนี้เป็นส่วนหนึ่งของวิชา 040613344 Software Engineering

ภาคเรียนที่ 1 ปีการศึกษา 2561

ภาควิชาวิทยการคอมพิวเตอร์และสารสนเทศ คณะวิทยาศาสตร์ประยุกต์

มหาวิทยาลัยพระจอมเกล้าพระนครเหนือ

คำนำ

สถาปัตยกรรมซอฟต์แวร์ คือ การอธิบายภาพรวมของระบบซึ่งมีโครงสร้างที่เชื่อมโยงองค์ประกอบสำคัญที่มีผลต่อภาพรวมของทั้งระบบ ทั้งในด้านฟังก์ชันและคุณภาพ ซึ่งแสดงให้เห็นถึงผลกระทบ, ผลสะท้อน, คุณสมบัติและคุณลักษณะของการเชื่อมกันระหว่างองค์ประกอบเหล่านั้นโดยมุ่งเน้นจัดการและอธิบายในจุดสำคัญที่มีผลต่อความสนใจด้านธุรกิจและด้านเทคนิคและมุ่งเน้นวางกรอบแนวคิดพื้นฐานกำกับ ให้ผู้รับผิดชอบในส่วนต่างๆ ยึดปฏิบัติและต่อยอดเพื่อให้ระบบมีเอกภาพ ตอบโจทย์ภาพรวมเดียวกัน ผู้ศึกษาได้ศึกษาเรื่อง สถาปัตยกรรมซอฟต์แวร์เพื่อศึกษาหาความรู้ให้เข้าใจมากยิ่งขึ้น

รายงานฉบับนี้เป็นส่วนหนึ่งของวิชา 040613344 Software Engineering โดยมีจุดประสงค์เพื่อการศึกษาจากเรื่อง Software Architectural ซึ่งรายงานฉบับนี้มีเนื้อหาเกี่ยวกับสถาปัตยกรรมซอฟต์แวร์ การออกแบบข้อมูลสไตล์และรูปแบบของสถาปัตยกรรม การออกแบบเชิงสถาปัตยกรรม ตลอดจนการประยุกต์ใช้งาน ผู้ศึกษาหวังเป็นอย่างยิ่งว่าผู้อ่านจะได้ความรู้จาการจัดทำเป็นรูปเล่มของเรื่องสถาปัตยกรรม-ซอฟต์แวร์ที่ผู้ศึกษาได้นำเสนอไว้ในรูปเล่มรายงานนี้แล้ว

คณะผู้จัดทำ

ธันวาคม 2561

สารบัญ

คำนำ

Software Architecture สถาปัตยกรรมซอฟต์แวร์ 1

เปรียบเทียบงานสถาปัตยกรรมก่อสร้าง 1

นิยามของสถาปัตยกรรมซอฟต์แวร์ 5

ความรู้สำคัญด้าน Software Architecture 6

วัฏจักรสถาปัตยกรรมกับธุรกิจ (Architecture Business Cycle) 6

Architectural Concerns 8

ขั้นตอนของวัฏจักร 9

คุณสมบัติทางคุณภาพ (Quality Attribute) 10

การอธิบายและสื่อสารสถาปัตยกรรม (Architecture Description & Communication) 13

การออกแบบสถาปัตยกรรม (Architecture Design) 15

การประเมินสถาปัตยกรรม (Architecture Evaluation) 15

การบูรณะสถาปัตยกรรม (Architecture Reconstruction) 16

คอมโพเน้นต์สำเร็จรูป (Component-Off-The-Shelf) 17

สายการผลิตซอฟต์แวร์ (Software Product Lines) 18

ความสามารถและหน้าที่ Architect ด้าน IT/Software/Solution 18

ทักษะและหน้าที่ 19

ความรู้ด้านไอที 20

ความรู้และทักษะด้านอื่น 20

คุณลักษณะ/บุคลิก 21

แนวทางการพัฒนาและปรับปรุงความสามารถของ Architect ในองค์กร 21

Transaction Overview 22

Transaction & Transaction Processing 22

Event Source & Arrival Event 23

ACID 24

ประเภทของ transaction 24

Operation สาหรับการจัดการ transaction หลัก ๆ 25

Transaction Status 26

Granularity 26

State Transition & State Management Overview 27

Transaction Propagation 29

Simultaneous Access 30

Scheduling Strategies for Resource Arbitration 32

Transaction Orchestration 32

Local & Distributed Transaction Processing 33

Transaction Model 34

Transaction Attribute 35

Transaction Analysis 35

วิเคราะห์ Business Process & Rule 37

วิเคราะห์พฤติกรรมไคลเอนต์ (Client Behavior) 37

วิเคราะห์ Service 38

วิเคราะห์ Resource System & Connection 38

วิเคราะห์ Transaction Data & Data Transformation 39

สร้าง Scenario เพื่ออธิบาย Transaction Concern 40

Transaction Traceability 42

การออกแบบเชิงสถาปัตยกรรม 43

การจัดโครงสร้างของซอฟต์แวร์ 44

โครงสร้างแบบแบ่งปันแหล่งจัดเก็บข้อมูล (Repository Model) 44

โครงสร้างแบบแบ่งปันบริการและเซิร์ฟเวอร์ 47

โครงสร้างแบบเลเยอร์ 47

การแบ่งส่วนซอฟต์แวร์ 48

การแบ่งส่วนงานเชิงวัตถุ 48

การแบ่งส่วนซอฟต์แวร์ตามฟังก์ชันงาน 49

การควบคุมการทำงานของส่วนประกอบย่อย 50

การควบคุมแบบศูนย์กลาง 50

การควบคุมตามเหตุการณ์ 50

สถาปัตยกรรมอ้างอิง 51

ระบบประมวลผลข้อมูล 52

ระบบประมวลผลรายการข้อมูล 52

ระบบประมวลผลตามเหตุการณ์ 52

สรุป 53

ข้อมูลอ้างอิง

Software Architecture สถาปัตยกรรมซอฟต์แวร์

ซอฟต์แวร์การวางโครงสร้างสถาปัตยกรรมเพื่อให้การก่อสร้างนั้นเป็นไปตามโครงสร้างที่วางไว้ ซึ่งซอฟต์แวร์เหล่านี้จะเข้ามาช่วยงานก่อสร้างให้ดำเนินอย่างราบรื่น โดยจะใช้เทคโนโลยีที่ทันสมัยเข้ามามีส่วนร่วมในการออกแบบให้สามารถเห็นภาพโครงสร้างของสถาปัตยกรรมก่อนการสร้างได้ซึ่งจะแสดงให้เห็นในรูปแบบ 2D และ 3D ตามความสามารถของแต่ละโปรแกรม รวมถึงการวิเคราะห์โครงสร้างในอนาคตและวางแผนส่วนงานอื่นๆ ได้อย่างมีประสิทธิภาพ

ตัวอย่างโปรแกรมประเภทการออกแบบโครงสร้างสถาปัตยกรรม Architecture Software

- Bluebeam PDF Revu โปรแกรมที่สามรถแก้ไขไฟล์ PDF และทำงานร่วมกันแบบเรียลไทม์
- AutoCAD โปรแกรมออกแบบโครงสร้าง 2D 3D และมีเครื่องมือดิจิทัล
- ArchiCAD โปรแกรมการสร้างแบบจำลองข้อมูลอาคารสำหรับสถาปนิกนักออกแบบตกแต่งภายใน
- SketchUp Pro โปรแกรมออกแบบโครงสร้างแบบ 2D 3D เพื่อวางแผนโครงการในอนาคต
- Clearview InFocus โปรแกรมบัญชีการจัดการโครงการและติดตามการเรียกเก็บเงินลูกค้า

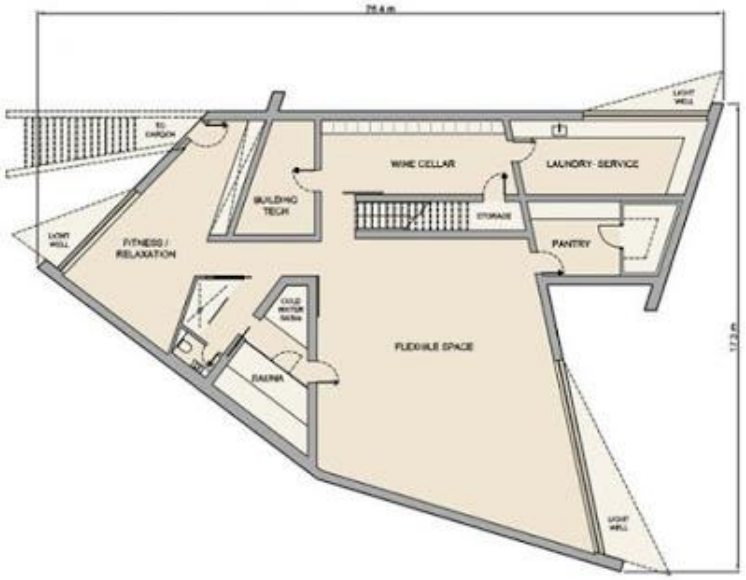
สำหรับซอฟต์แวร์ Architecture Software จะรวบรวมโปรแกรมเกี่ยวกับการออกแบบโครงสร้าง วิเคราะห์ข้อมูลทั้งปัจจุบันและอนาคตของสถาปัตยกรรมรวมถึงโปรแกรมอื่นๆที่เกี่ยวข้องทั้งโปรแกรมบัญชี ติดตามลูกค้า การชำระเงิน เหมาะสำหรับธุรกิจทุกขนาด

เปรียบเทียบงานสถาปัตยกรรมก่อสร้าง

เฉกเช่นกับสถาปัตยกรรมสิ่งก่อสร้างที่ architect ต้องมองภาพรวม ออกแบบภาพรวม เพื่อเชื่อมโยงองค์ประกอบสำคัญเข้าด้วยกัน มุ่งเน้นในจุดสำคัญทั้งในเชิงธุรกิจที่ stakeholder concern และในเชิงเทคนิคที่ทีมก่อสร้าง concern สิ่งที่ architect ต้องพิจารณาและอธิบายออกมามีอะไรบ้าง เช่น โครงสร้างอาคาร, ความสวยงามเชิงสถาปัตยกรรม, ทิศทางแสงและลมในฤดูต่างๆ, ต้นไม้ที่จะปลูกภายในและภายนอกอาคาร, ตำแหน่งบันได ลิฟต์ หน้าต่าง ประตู ท่อประปา ท่อทางเดินอากาศ ทางเดินสายไฟ, รูปแบบหลังคา, การป้องกันแรงสั่นสะเทือนเช่นจากแผ่นดินไหว เป็นต้น architect ต้องจัดวางองค์ประกอบเหล่านี้และเชื่อมโยงกันเพื่อจัดการความสัมพันธ์ระหว่างองค์ประกอบต่างๆ เช่น การจัดวางตำแหน่งลิฟต์และบันไดไม่ให้ซ้อนทับกัน, การเชื่อมต่อระหว่างระบบฉีดน้ำอัตโนมัติเวลาไฟไหม้กับระบบประปา

งานออกแบบสถาปัตยกรรม architect เน้นการออกแบบภาพรวม ไม่เน้นรายละเอียดมากนัก (ไม่ลงลึกถึงการทำ detailed design) และไม่เน้นรูปแบบการอิมพลีเม้นต์มากนัก (เป็นหน้าที่ของคนอิมพลีเม้นต์ว่าจะใช้แนวทางหรืออัลกอริธึมใด) อาจจะกล่าวได้ว่างานสถาปัตยกรรมเน้นการ ‘คุมภาพรวม’ เช่น โครงการพัฒนาระบบ ERP (Enterprise Resource Planning) ที่ประกอบด้วยระบบย่อยจำนวนมาก เช่น ระบบบัญชี ระบบการเงิน ฯ architect จะต้องคุมภาพรวมการทำงานระหว่างระบบต่างๆ หรือแม้แต่การพัฒนาระบบเพียงระบบเดียว architect ก็ต้องคุมภาพรวมของการทำงานระหว่างฟังก์ชั่นและกลไกต่างๆ

Model



Plan

Interior

Modern Prefab Villa

ในงานก่อสร้าง architect จะเป็นฝ่ายเริ่มต้นการทำงานก่อน ด้วยการออกแบบสถาปัตยกรรมที่อธิบายภาพรวมออกมา ผ่านการวาดพิมพ์เขียว (blueprint) หรือแบบแปลน ซึ่งเอาไว้ใช้กับเฉพาะวิศวกรและทีมผู้รับเหมาก่อสร้าง แต่ก่อนที่พิมพ์เขียวจะออก architect ก็ต้องวาดรูปแบบอาคารด้วยภาพวาด (วาดด้วยมือหรือใช้โปรแกรมสร้าง) เพื่อให้ลูกค้าดูและเข้าใจได้ง่าย ทั้งพิมพ์เขียวที่เอาไว้ใช้งานและรูปที่เอาไว้สื่อกับลูกค้าอาจมีหลายรูปได้ขึ้นกับจำนวนองค์ประกอบของอาคาร เช่น จำนวนชั้น จำนวนห้อง จากนั้นก็จะมาถึงงานของวิศวกรที่จะเอาพิมพ์เขียวไปใช้ต่อ เพื่อดูรายการองค์ประกอบต่างๆ ที่จะต้องสร้าง โดยงานของวิศวกรจะคำนึงถึงแนวทางในการก่อสร้างเป็นหลัก เช่น architect ระบุตำแหน่งเสาและคานของอาคาร ส่วนวิศวกรก็จะพิจารณาในแง่วัสดุ เช่น เหล็ก ปูน การลงเสาเข็ม เป็นต้น จากนั้นวิศวกรจะปรึกษากับผู้รับเหมาก่อสร้าง (ทีมก่อสร้าง) ถึงการวางลำดับการก่อสร้าง และประเด็นสำคัญในการก่อสร้าง รวมถึงเรื่องวัสดุด้วย ในขณะที่ architect ก็ยังอยู่ไม่ได้ไปไหน เพียงแต่งานเบาลง โดยยังคอยให้คำปรึกษาและติดตาม เพื่อให้อาคารสร้างตรงตามแบบที่ออกแบบไว้

โครงสร้างอาคารที่กำลังก่อสร้าง

จากรูป ลองสังเกตอาคารที่เพิ่งก่อสร้าง จะเห็นว่าสิ่งแรกๆ ที่ต้องทำคือการปรับพื้น ทำฐานรากและเสาเข็ม จากนั้นก็เชื่อมโยงเสาด้วยคาน ในขณะที่เสาก็ยังสร้างต่อเติมให้สูงขึ้นเรื่อยๆ โดยมีการสร้างคานเพิ่มขึ้นเรื่อยๆ เช่นกัน โครงสร้างอาคารก็มีอยู่เพียงเท่านี้ จากนั้นพื้น บันได จึงค่อยตามมา แล้วต่อด้วยผนังภายในการก่อสร้างมีลำดับขั้นตอน เช่น จะสร้างห้องน้ำจนเสร็จแล้วค่อยมาวางท่อประปาและท่อน้ำทิ้งไม่ได้ ต้องวางท่อประปาและท่อน้ำทิ้งก่อน หรือจะสร้างหลังคาก่อนตอกเสาเข็มก็ไม่ได้ แต่อาจสร้างห้องครัวก่อนห้องนอน หรือสร้างห้องน้ำก่อนห้องนั่งเล่นได้ สร้างอาคารจนเกือบเสร็จจึงถึงคราวของงานตกแต่งภายในหรือ “Interior Design” ที่จะมาออกแบบรายละเอียด อาทิ สี ฝ้าเพดาน พื้น ผนัง ประตู โคมไฟ ตู้ โต๊ะ ระเบียง ฯลฯ ซึ่งงานตกแต่งภายในอาจเริ่มตั้งแต่ช่วงก่อนหรือช่วงต้นของการก่อสร้างก็ได้ เพื่อกำหนดแนวทางในบั้นปลายก่อน เพื่อให้งานก่อสร้างสอดคล้องกับแนวทางการตกแต่งในตอนท้าย

อาคารภายหลังการก่อสร้างเสร็จสมบูรณ์

จากตัวอย่างงานก่อสร้างข้างต้น เสาและคานเปรียบเสมือน “อินเตอร์เฟส” ระดับสถาปัตยกรรม หรือ “Architectural Interface” เพื่อใช้เชื่อมชั้นและห้องต่างๆ และยึดโครงสร้างอาคารให้แข็งแรง จึงไม่แปลกที่งานออกแบบสถาปัตยกรรมระบบจึงให้ความสำคัญกับการออกแบบอินเตอร์เฟสเป็นอย่างมาก และมองห้องต่างๆ เป็นดั่ง ‘กล่องดำ’ งานตกแต่งภายในก็เหมือนงาน “Detailed Design” ส่วนงานของวิศวกรและทีมก่อสร้างก็เปรียบดังงานของทีมอิมพลีเม้นต์ที่มุ่งเน้นการสร้างระบบ แต่ทั้งนี้ทั้งนั้นงานก่อสร้างกับงานพัฒนาระบบไม่ได้เหมือนกันแบบแนบสนิทจริงๆ มีส่วนแตกต่างกันบ้าง เช่น งานก่อสร้างเน้นความสวยงามด้วย ขณะที่งานพัฒนาระบบเน้นฟังก์ชั่นการทำงานมากกว่า นอกจากนี้ในงานพัฒนาระบบเองยังมีความซ้อนเหลื่อมกันของกระบวนการทางสถาปัตยกรรม (Architectural Process) และกระบวนการทางวิศวกรรมซอฟต์แวร์ (Software Engineering Process) ที่คล้ายและต่างกันไปตามแต่ละวิธีการทางกระบวนการ (Software Process Methodology) ต่างๆ สำหรับมาตรฐานเองก็มีหลายมาตรฐาน

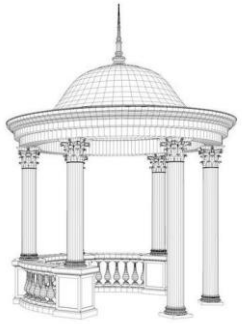
<u>นิยามของสถาปัตยกรรมซอฟต์แวร์</u>

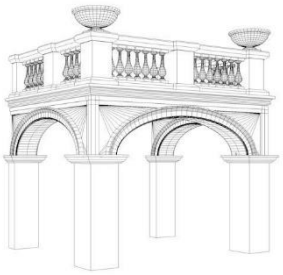
ความหมายของ “สถาปัตยกรรมซอฟต์แวร์” (Software Architecture) ยังไม่มีนิยามสั้นๆ กระชับๆ ที่จะอธิบายให้เข้าใจได้ง่ายในเวลาอันสั้นนัก บ้างก็ว่าคือ High Level Design, บ้างก็ว่าคือการอธิบายโครงสร้างของระบบ, บ้างก็ว่าคือภาพรวมของระบบ, บ้างก็ว่าคือการอธิบายการเชื่อมต่อระหว่างคอมโพเน้นต์สำคัญๆ ในระบบ... ซึ่งไม่ถูกสักเหตุผล แต่ต้องนำเหตุผลทั้งหมดมารวมกัน แต่ก็ยังไม่ชัดเจนอยู่ดี ดังนั้นจึงขอให้นิยามกับความหมายของสถาปัตยกรรมซอฟต์แวร์ไว้ง่ายๆ ดังนี้

สถาปัตยกรรมซอฟต์แวร์ คือ การอธิบายภาพรวมของระบบซึ่งมีโครงสร้างที่เชื่อมโยงองค์ประกอบสำคัญ ที่มีผลต่อภาพรวมของทั้งระบบ ทั้งในด้านฟังก์ชั่นและคุณภาพ ซึ่งแสดงให้เห็นถึงผลกระทบ, ผลสะท้อน, คุณสมบัติ และคุณลักษณะ ของการเชื่อมกันระหว่างองค์ประกอบเหล่านั้น โดยมุ่งเน้นจัดการและอธิบายในจุดสำคัญที่มีผลต่อความสนใจด้านธุรกิจและด้านเทคนิค และมุ่งเน้นวางกรอบแนวคิดพื้นฐานกำกับให้ผู้รับผิดชอบในส่วนต่างๆ ยึดปฏิบัติและต่อยอดเพื่อให้ระบบมีเอกภาพ ตอบโจทย์ภาพรวมเดียวกัน

องค์ประกอบของระบบ (system element) หมายถึง ทุกๆ โมดูล, คอมโพเน้นต์, ไฟล์คอนฟิกุเรชั่น, หน้าจอ, ไลบรารี่, ข้อมูล, เลเยอร์, เทียร์ เป็นต้น แต่องค์ประกอบสำคัญของสถาปัตยกรรม (architectural element) นั้นหมายถึงองค์ประกอบของระบบเฉพาะบางส่วนที่มีผลเท่านั้น ซึ่งเป็นส่วนที่มีผลต่อภาพรวมของทั้งระบบในเชิงฟังก์ชั่นและคุณภาพ เช่น ไม่ใช่ทุกโมดูลที่จะอยู่ในส่วนสถาปัตยกรรมอาจเป็นเฉพาะบางโมดูลเท่านั้น เช่น เป็นกลไกสำคัญของระบบ มีผลต่อคุณภาพหลักของระบบสนับสนุนฟังก์ชั่นสำคัญของระบบ และมีผลต่อธุรกิจหรือ concern ของ stakeholder สูง

ความสัมพันธ์ระหว่างองค์ประกอบสำคัญ หมายถึง ความสัมพันธ์ทั้งในเชิงไดนามิก และเชิงโครงสร้าง เช่น โมดูล “A” เรียกใช้โมดูล “E” แสดงว่า “A” มี coupling กับ “E” หาก “E” มีการเปลี่ยนแปลงจะส่งผลกระทบต่อ “A” การทำงานของโมดูล “E” สะท้อนถึงคุณภาพของโมดูล “A” ด้วย คุณสมบัติและบริการของโมดูล “E” ที่ “A” เรียกใช้ได้อาจเป็นเพียงบางส่วนที่ “E” เปิดให้ “A” เห็นและเข้าถึงได้ การทำงานร่วมกันในเชิงไดนามิกหรือการเชื่อมโยงกันในเชิงโครงสร้างระหว่างองค์ประกอบเหล่านั้น ส่วนใดที่มีผลต่อ concern ในเชิงธุรกิจและเชิงเทคนิคมากๆ ยิ่งต้องมุ่งเน้นจัดการและอธิบายให้ชัดเจน เช่น หากโมดูล ‘E’ ทำงานช้าผิดปกติ จะทำให้โมดูล “A” ทำงานช้าตามเพราะต้องรอ และจะทำให้ระบบโดยรวมเกิดปัญหาคอขวด รองรับ Request ที่เข้ามาปริมาณมากไม่ไหวจนอาจทำให้ระบบหยุดทำงาน หรืออาจทำให้มีทรานแซกชั่นถูกยกเลิกจำนวนมาก สถาปัตยกรรมจึงต้องมุ่งเน้นที่โครงสร้างความสัมพันธ์ระหว่างองค์ประกอบสำคัญต่างๆ





จากข้างต้นคือความหมายของสถาปัตยกรรมซอฟต์แวร์ แม้ว่าในบางครั้งในหนังสือเล่มนี้ผมอาจใช้คำว่า “สถาปัตยกรรมระบบ” บ้าง แต่มักใช้คำว่า “สถาปัตยกรรม” สั้นๆ มากกว่า ซึ่งก็ถือว่าคือการกล่าวถึงสิ่งเดียวกันแล้วกันครับนั่นคือ “สถาปัตยกรรมซอฟต์แวร์” เนื่องจากคนไทยเราคุ้นเคยกับการใช้คำว่า “ระบบ” มากกว่า แต่ถ้าจะใช้คำว่า “สถาปัตยกรรมระบบ” ก็ดูจะยาวไปนิด จึงขอใช้คำว่า “สถาปัตยกรรม” สั้นๆ แทน ซึ่งในความเป็นจริงแล้วทั้ง “สถาปัตยกรรมระบบ” กับ “สถาปัตยกรรมซอฟต์แวร์” นั้นมีขอบเขตที่ต่างกัน!

ทุกวันนี้โลกไอทีมีอะไรต่อมิอะไรที่ขึ้นต้นด้วยคำว่า “สถาปัตยกรรม” หรือลงท้ายด้วยคำว่า “Architecture” อยู่เต็มไปหมด อาทิ Enterprise Architecture, IT Architecture, Business Architecture, Information Systems Architecture, Solution Architecture, Application Architecture, Technology Architecture

ความรู้สำคัญด้าน Software Architecture

- **วัฏจักรสถาปัตยกรรมกับธุรกิจ (Architecture Business Cycle)**

  ระบบต่างๆ ที่ถูกพัฒนาขึ้นมีวัตถุประสงค์หลัก คือ เพื่อสนับสนุนกิจกรรมทางธุรกิจ ดังนั้นจึงควรทราบเสียก่อนพัฒนาระบบว่าภาคธุรกิจคาดหวังอะไรจากระบบบ้าง? ระบบต้องไปสนับสนุนกิจกรรม (operation) หรือกระบวนการทางธุรกิจ (business process) อะไรบ้าง? ใครคือผู้ที่มีส่วนได้เสียหรือมีผลต่อความสำเร็จหรือล้มเหลวของโครงการ (stakeholder) บ้าง? ระบบจะไปทำงานในสภาพแวดล้อมแบบใด? สถาปนิกผู้ออกแบบมีประสบการณ์และแบ๊กกราวด์ด้านใดมาบ้าง? สิ่งเหล่านี้เป็นดังแรงกดดัน (influence) ต่อการออกแบบสถาปัตยกรรมซอฟต์แวร์ ซึ่งจะกลายเป็นต้นแบบต่อการพัฒนาระบบ (รวมถึงการเขียนโปรแกรม) ต่อไป จนกระทั่งระบบพัฒนาเสร็จ โดยการทำงานและคุณภาพของระบบที่พัฒนาเสร็จสิ้นจะต้องสะท้อนกลับไปยังปัจจัยต่างๆ ก่อนหน้านี้ได้ว่า... ระบบตอบสนองต่อภาคธุรกิจได้ตามที่คาดหวังไว้หรือไม่?

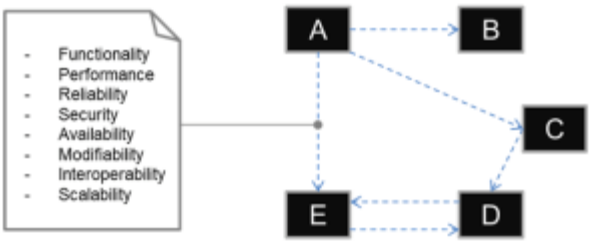
  การพัฒนาระบบแต่ไหนแต่ไรมาคนจำนวนมากคุ้นเคยแต่การสร้างซอร์สโค้ด แล้วก็คอมไพล์ ระบบสมัยก่อนมักเป็นระบบที่ไม่ใหญ่และไม่ซับซ้อนมากนัก เมื่อกาลเวลาผ่านไปเทคโนโลยีทวีความซับซ้อนยิ่งขึ้น ฮาร์ดแวร์และโครงสร้างพื้นฐานโดยเฉพาะความเร็วในการสื่อสารต่างเจริญรุดหน้าไปมาก ผลักดันให้ระบบเองเริ่มซับซ้อนยิ่งขึ้น เพื่อให้รองรับการใช้งานหลากหลายฟังก์ชั่น และผู้ใช้ที่หลากหลายยิ่งขึ้น สมัยก่อนไม่มีใครรู้จักและใส่ใจกับสถาปัตยกรรมซอฟต์แวร์กันนัก ส่วนหนึ่งเพราะยังเป็นศาสตร์ใหม่ และอีกส่วนหนึ่งที่น่าจะเป็นเหตุผลหลักคือ การที่ผู้คนยังคงสนใจแต่ซอร์สโค้ดเป็นหลัก ได้ requirement มาก็โซโล่โค้ดกันเลย ออกแบบในหัวแล้วปลดปล่อยจินตนาการออกมาเป็นซอร์สโค้ด มีงานออกแบบที่มักทำกันบ้างเช่น Data Flow, Entity-Relationship Data Model หน้าจอโปรโตไทป์ ที่เหลือก็เป็น “สเป็ก” (Specification) บรรยายฟังก์ชั่นและหน้าจอต่างๆ ซึ่งก็เป็นเพียงรายละเอียดคร่าวๆ และออกไปทางนามธรรม งานออกแบบจึงยังไม่ค่อยมีใครให้ความสำคัญนักในสมัยก่อน

ในความเป็นจริงแล้วไม่ว่าระบบจะเล็กจะใหญ่ จะเก่าจะใหม่ต่างมีสถาปัตยกรรมด้วยกันทั้งนั้น เพียงแค่ว่าได้อธิบายมันออกมาเป็นกิจลักษณะหรือไม่เท่านั้นเอง สมัยนี้การพัฒนาระบบหนึ่งๆ ต่างอุดมด้วยเทคโนโลยีอันหลากหลาย ภาษาโปรแกรมภาษาสคริปต์หลายภาษาในระบบเดียว เลเยอร์ (layer) ซับซ้อนขึ้น เทียร์ (tier) เริ่มมีหลายชั้นมากขึ้น แพลตฟอร์มในสภาพแวดล้อมระบบก็มีหลากหลาย เฟรมเวิร์กและไลบรารี่จำนวนมากถูกขยำรวมกันในระบบหนึ่งๆ แบบไม่บันยะบันยัง ผู้ใช้งานที่ทวีจำนวนมากและหลากหลายขึ้นทั้งพฤติกรรมการใช้งานและอุปกรณ์ที่ใช้ ไม่ว่าจะเป็นพีซี โทรศัพท์มือถือ แท็บเล็ต รวมไปถึงการทำงานเชื่อมต่อกันข้ามระบบก็มีมากยิ่งขึ้น resource system ก็มีมากขึ้นๆ อาทิ database server, message queue server, mail server, directory server, data warehouse ฯลฯ เหล่านี้เองที่ผลักดันให้ระบบจำต้องซับซ้อนมากขึ้นกว่าสมัยก่อนไปโดยปริยาย กลายเป็นในสภาพแวดล้อมระบบมี “อะไรต่อมิอะไร” มากมาย ซึ่งต่างเป็นปัจจัยที่ส่งผลต่อระบบทั้งนั้น ดังนั้นการสร้างสถาปัตยกรรม เพื่อใช้เป็นต้นแบบ ใช้สื่อสาร ใช้ควบคุม จึงถูกให้ความสำคัญมากยิ่งขึ้น

มุมมองเชิงสถาปัตยกรรม (Architectural View) ที่อธิบายออกมาจากงานออกแบบสถาปัตยกรรม มักเป็นนามธรรม ไม่จำเป็นต้องอธิบายส่วนต่างๆ (system element) ในระบบให้ละเอียดมากนัก แต่ต้องอธิบายให้เห็นภาพรวม และประเด็นสำคัญต่างๆ ในระบบ โดยมุ่งเน้นอธิบายพฤติกรรมและปฏิสัมพันธ์ (interaction) ของอิลิเม้นต์ต่างๆ ในระบบเป็นหลัก

อิลิเม้นต์ในระบบมีอะไรบ้าง อาทิ โมดูล คอมโพเน้นต์ ไฟล์คอนฟิกุเรชั่น หน้าจอ ไลบรารี่ที่ใช้ ข้อมูล เลเยอร์ เทียร์ เป็นต้น ส่วน “อิลิเม้นต์ที่สำคัญในระบบ” (architectural elements) จะพิจารณาเกณฑ์ง่ายๆ ว่าอิลิเม้นต์นั้นๆ

1. ต้องส่งผลต่อภาพรวมของระบบ เช่น database driver ถ้าคอนฟิกค่าผิดระบบทั้งระบบอาจติดต่อกับ database server ไม่ได้ก็ได้
2. ต้องสัมพันธ์กับสิ่งที่ stakeholder สนใจ หรือ concern เช่น stakeholder กังวลเรื่องการจัดการกับ business logic ที่เป็นความลับขององค์กร ดังนั้นสถาปัตยกรรมก็ควรเน้นตรงนี้มากหน่อย

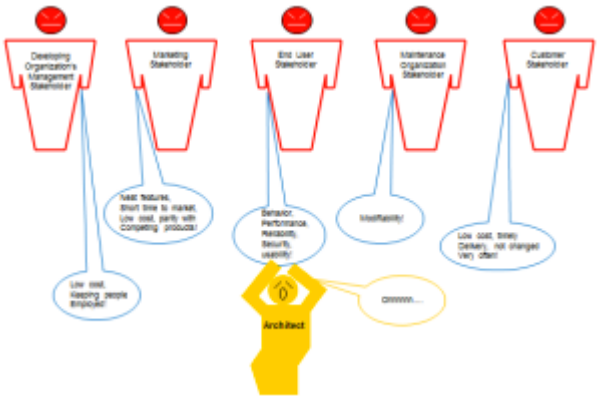


สถาปัตยกรรมซอฟต์แวร์โฟกัสที่ปฏิสัมพันธ์และพฤติกรรมระหว่าง ‘black box elements’

จากรูปเราจะพิจารณา “อิลิเม้นต์ที่สำคัญในระบบ” เป็นเสมือนกล่องดำ โดยไม่สนใจโครงสร้าง (structure) และพฤติกรรม (behavior) ภายในแต่ละอิลิเม้นต์มากนัก แต่จะมุ่งเน้นที่พฤติกรรมและปฏิสัมพันธ์ระหว่างอิลิเม้นต์เป็นหลัก จากรูปที่ 1 อิลิเม้นต์ “A” มี dependency กับอิลิเม้นต์ “E”

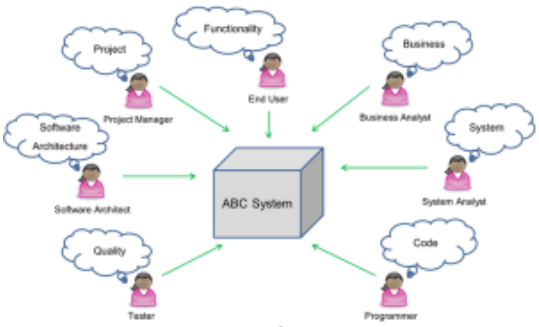
เราจะไม่สนใจกับรายละเอียดภายในของ “A” และ “E” นัก แต่สนใจที่ความสัมพันธ์ระหว่าง “A” กับ “E” ซึ่งมีประเด็นที่ต้องพิจารณากันหลายเรื่องทีเดียว อาทิ ฟังก์ชั่นการทำงาน (functionality) และคุณภาพด้านต่างๆ

- **Architectural Concerns**



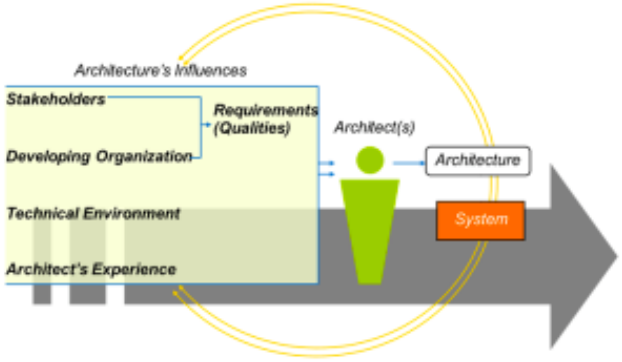
ความสนใจที่แตกต่างกันของ Stakeholder ที่ Architect ต้อง Balance Concern เหล่านี้ให้ได้

จากรูประบบจะออกมาหน้าตาอย่างไรขึ้นกับสถาปัตยกรรม ซึ่งสถาปัตยกรรมจะมีหน้าตาอย่างไรก็ขึ้นกับความสนใจของ stakeholder โดยสิ่งที่ architect จะต้องรับผิดชอบคือการ “balance concern” เหล่านี้ให้ได้ สำหรับ “balance concern” คือ การสร้างสมดุลในความต้องการของแต่ละ stakeholder ให้ได้ นั่นคือการทำให้ stakeholder ทุกคนต้อง “win-win” โดย architect ต้องเข้าใจเหตุผล ที่มาที่ไป ว่าทำไม stakeholder ถึงสนใจสิ่งเหล่านั้น ว่ามีประโยชน์ต่องานของตน สวัสดิภาพการทำงานของตน ธุรกิจของตน อย่างไร? “concern” เหล่านี้อยู่ในรูปของ requirement ที่จะถูกจำแนกเป็นประเภทต่างๆ อีกที แต่อยู่ๆ stakeholder ให้ความต้องการมา architect จะตอบเลยว่าทำได้หรือไม่ทันทีนั้นก็กระไรอยู่ ดังนั้นจึงต้องออกแบบสถาปัตยกรรมและใช้สถาปัตยกรรมมาประกอบการสื่อสารและวิเคราะห์ และอาจต้องมีการประนีประนอมกับ stakeholder เพื่อปรับ concern เหล่านั้นให้อยู่ภายในกรอบของสถาปัตยกรรมที่ สามารถนำไปสร้างได้จริง ภายใต้เงื่อนไข เช่น งบประมาณ กำลังคน ระยะเวลา และความเสี่ยงทางธุรกิจ ไม่ใช่ว่า stakeholder อยากได้อย่างไร ก็ต้องตามใจ หรือ architect จะออกแบบสถาปัตยกรรมตามอำเภอใจโดยไม่ใส่ใจ stakeholder เลยก็ไม่ได้



ทัศนคติและมุมมองที่มีต่อระบบเดียวกัน

หลายคนอาจเข้าใจผิดว่า stakeholder ต้องหมายถึงบุคคลฝั่งลูกค้าหรือผู้ใช้เท่านั้น ซึ่งจริงๆ แล้วสมาชิกในทีมพัฒนาก็ถือเป็น stakeholder ด้วยเช่นกัน จากรูปที่ 3 ในโครงการหนึ่งๆ อาจประกอบด้วย stakeholder หลายคน แต่ละคนก็รับผิดชอบบทบาทที่แตกต่างกันไป ดังนั้นผู้ที่รับผิดชอบแต่ละบทบาทจึงสามารถมีวิธีคิด มุมมอง และทัศนคติที่แตกต่างกันไปได้ ถึงแม้ว่ากำลังทำโครงการเดียวกันอยู่แท้ๆ ก็ตาม บุคคลที่แวดล้อมระบบจึงไม่จำเป็นต้องคิดเห็นตรงกันในทุกเรื่องเสมอไป ซึ่งแปรไปตามบทบาทหน้าที่รับผิดชอบ ประสบการณ์ความรู้ และผลประโยชน์จากการทำงานที่แต่ละคนพึงต้องการ architect จึงต้องมีทักษะในการสื่อสารอย่างมาก เพื่อให้เข้าใจ concern ของทุกคน ทั้งยังต้องยอมรับในความจริงและความแตกต่าง และ balance concern ให้ได้

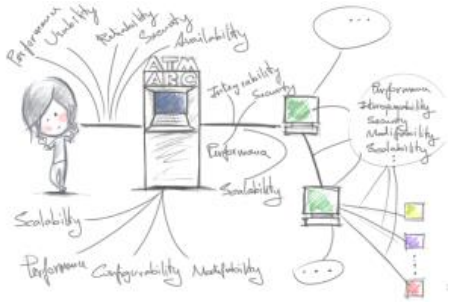


วัฏจักรสถาปัตยกรรมกับธุรกิจ

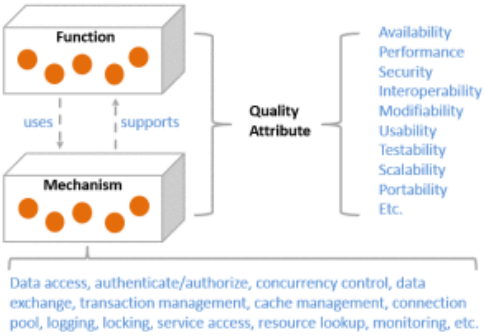
➢ **ขั้นตอนของวัฏจักร**

จากรูป Architecture's Influences คือ ปัจจัยหลักที่ถาโถมสู่ architect ประกอบด้วยความต้องการของ stakeholder ต่างๆ และ Developing Organization สภาพแวดล้อมเชิงเทคนิค และประสบการณ์ของ architect เอง เหล่านี้ถือเป็นอินพุตหลักสู่ตัว architect เอง เพื่อนำไปใช้ออกแบบสถาปัตยกรรม จากนั้นจึงนำสถาปัตยกรรมไปเป็นต้นแบบในการสร้างระบบ เมื่อระบบสร้างเสร็จแล้วก็ควรตอบสนองต่อ Architecture's Influences โดยเฉพาะความต้องการของ stakeholder ต่างๆ และ Developing Organization และเมื่อเวลาผ่านไปหลังจากเริ่มใช้งานระบบ atakeholder อาจมีความต้องการเพิ่มเติม อันเป็นผลสืบเนื่องจาก concern ด้านธุรกิจและเทคโนโลยีที่มีการเปลี่ยนแปลง เช่น แผนธุรกิจ, แผลกลยุทธ์, business rule, business process, เทคโนโลยี, ไลบรารี่ เป็นต้น ทำให้สถาปัตยกรรมและตัวระบบเองต้องปรับตาม วัฏจักรของสถาปัตยกรรมกับธุรกิจก็จะกลับมาหมุนวนรอบใหม่อีกครั้ง เป็นวัฏจักรเช่นนี้สืบเนื่องกันไป ซึ่งการบริหารสถาปัตยกรรมให้รองรับการเปลี่ยนแปลงและมีความทันสมัยอยู่เสมอเรียกว่า "Architecture Change Management" หรือบางทีเรียกว่า "Architecture Evolution Management"

➢ คุณสมบัติทางคุณภาพ (Quality Attribute)



Quality Attribute คือ คุณสมบัติเชิงคุณภาพของสิ่งต่างๆ ภายในระบบที่มีการทำงาน เช่น สถาปัตยกรรมระบบ โมดูล อ็อบเจ็คต์ หน้าจอ เป็นต้น และรวมถึงคุณภาพของระบบและระหว่างระบบภายในบริบทเดียวกัน เนื่องจากสิ่งต่างๆ เหล่านี้ต้องมีการประมวลผล มี input และมี output เกิดขึ้น การจะให้ทำงานได้มีคุณภาพที่ดีจึงมีการกำหนดคุณสมบัติเชิงคุณภาพเอาไว้ โดยควรมีคุณภาพอะไรบ้างก็ขึ้นกับบริบทของระบบนั้นๆ ว่า stakeholder ต้องการระบบที่มีศักยภาพแค่ไหน



การกำหนด Quality Attribute ได้มาจากการเก็บรวบรวมความต้องการประเภท Non-Functional Requirement โดยเจ้า Quality Attribute เองก็ยังแบ่งออกได้เป็น 3 ประเภท ได้แก่

1. System Quality คือ คุณภาพระบบ มีหลักๆ อยู่แค่ 6 ตัว ได้แก่ Availability, Modifiability, Performance, Security, Testability และ Usability ซึ่งเจ้า System Quality นี่ล่ะที่เราๆ มักเรียกอีกอย่างว่า “Non-Functional Requirement” นั่นเอง
2. Business Quality คือ คุณภาพในเชิงธุรกิจที่ระบบได้สนองต่อธุรกิจของ stakeholder มีหลายตัว มักเป็นศัพท์ธุรกิจ อาทิ Increase Net Profit, Increase Competitiveness, Short Time to Market, Reduce Paper Work เป็นต้น ถ้าสังเกตให้ดี ก็จะร้องอ๋อ…ว่ามันก็คือ Business Goal นั่นเอง แต่ต่างๆ เล็กน้อยที่ขอบเขตและ Level of Abstraction เพราะ Business Goal เป็นการมองภาพรวมของทั้งระบบหรือทั้งโครงการ แต่ Business Quality สามารถใช้มองในระดับที่ลึกในรายละเอียดกว่าได้ เช่น Business

Quality ของ Domain Layer, ของโมดูล Report Manager, ของกลไก Transaction Management, ของสถาปัตยกรรมระบบ เป็นต้น

3. Architecture Quality คือ คุณภาพของสถาปัตยกรรมระบบ ไม่ได้มองที่การทำงานแบบ System Quality แต่มองที่ประโยชน์และการนำไปใช้ มีไม่มากนัก อาทิ Conceptual Integrity, Buildability, Correctness and Completeness เป็นต้น

เราจะมาโฟกัสกันที่ System Quality หรือ Non-Functional Requirement (NFR) เป็นหลักในบทความนี้ จากนี้จะขอใช้ตัวย่อว่า "NFR" แล้วกัน สำหรับการเก็บรวบรวม NFR นั้นไม่ง่ายนัก เนื่องจากเจ้า NFR เป็นความต้องการที่ 'หิน' เอาเรื่อง เพราะหนักในเรื่องเทคนิคมากๆ มีศัพท์เทคนิคเยอะแยะ แต่ละคำแต่ละความหมายก็ยากที่จะเข้าใจได้ง่ายๆ อุปสรรคขั้นแรกที่มักพบกันได้บ่อยก็คือช่วงการ "ถาม"! เพราะ NFR มันตั้งคำถามยาก เพราะจะถาม Stakeholder ยังไงดีที่จะไม่หลุดพูดศัพท์เทคนิคปวดหัวมากมายออกไป หากยิ่งพ่นศัพท์เทคนิคยากๆ ออกไป อาจทำให้ Stakeholder ออกอาการเบลอ งง และหนักสุดที่ต้องเจอแน่ๆ คือ เราต้องเสียเวลาอธิบายคำศัพท์ให้เขาฟัง ดังนั้นผู้ที่รับผิดชอบเจ้า NFR นี้จึงต้องมีทักษะอันเยี่ยมยอดทั้งด้านเทคนิคและการสื่อสาร โดยเฉพาะต้องมีภาษาพูดและภาษาเขียนที่ดี เพื่อสื่อสารให้ผู้อื่นเข้าใจได้ง่าย

การถามเกี่ยวกับ NFR ไม่ใช่อยู่ๆ ไปถาม stakeholder ว่าอยากให้ระบบมีคุณภาพอะไรบ้าง? อยากให้คุณภาพด้าน Availability อย่างไร? อยากให้มีคุณภาพด้าน Performance แค่ไหน? อยากให้มีคุณภาพด้าน Modifiability แบบใด? อยาก.... แค่นี้ Stakeholder ก็มึนแล้ว แล้วเราก็มักจะได้ NFR มาในแบบ "เมฆๆ" คลุมเครือและกว้างเป็นทะเล

การระบุ NFR จึงมักอธิบายออกมาด้วยการบรรยายเป็นเรื่องราว ถึงสถานการณ์สำคัญขณะที่ระบบกำลังทำงาน (ต้องจินตนาการไปถึงอนาคตหน่อย) ที่ Stakeholder มีความกังวล (Concern) มาก โดยเรียกสถานการณ์สำคัญว่า "Scenario" หรือเรียกเต็มๆ ว่า "Quality Scenario" แนะนำว่าให้เรียกแบบเต็ม เพราะถ้าใช้คำว่า Scenario เฉยๆ มันดูกว้างไป เผื่อสื่อสารกับผู้อื่น อาจเข้าใจสับสนได้ เพราะในงานไอทีมี Scenario หลายประเภท เช่น Business Scenario, Test Scenario, Use Case Scenario เป็นต้น องค์ประกอบหลักๆ ของ Quality Scenario ประกอบด้วย

- Stimulus คือ สิ่งเร้าหรือสิ่งกระตุ้น ในทางไอทีอาจคุ้นกับคำว่า "Trigger Event" ที่ทำให้สถานการณ์สำคัญนั้นๆ เกิดขึ้นให้นึกถึง Storyboard หรือ Snapsho เช่น ระบบประมวลผลเพราะผู้ใช้ submit transaction
- Source of Stimulus คือ แหล่งที่มาของสิ่งเร้า หรือตัวที่ก่อให้สิ่งเร้ามันเกิดขึ้นมา จะเป็นคนหรืออุปกรณ์ก็ได้
- Artifact คือ สิ่งต่างๆ ที่เกี่ยวข้องมีผลต่อสถานการณ์สำคัญนั้นๆ เป็นอะไรก็ได้ จะระบบ โมดูล ฮาร์ดแวร์ ทรานแซกชั่น เน็ตเวิร์ก เอกสาร เป็นต้น เจ้า Artifact นี่กำหนดยาก

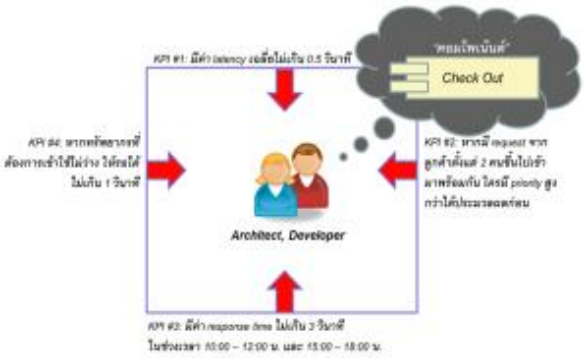
ถ้าไม่แม่นในคุณภาพนั้นๆ จริงๆ จะนึกออกยาก และจะทำให้รวบรวมได้ไม่สมบูรณ์ครอบคลุม หรืออาจคลาดเคลื่อนได้

- Environment คือ สภาพแวดล้อมขณะระบบกำลังทำงาน เช่น Degraded Mode, Normal Mode เป็นต้น
- Response คือ การตอบสนองของระบบ เมื่อสิ่งเร้าเกิดขึ้น ดังนั้น Response จะสัมพันธ์กันกับ Stimulus
- Response Measure คือ ตัววัดการตอบสนองของระบบ หรือเรียกว่าตัวชี้วัดก็ได้ เพราะคุณภาพใดๆ ต้องมีตัวชี้วัดเสมอ ซึ่งจะแตกต่างกันไปตามแต่ละคุณภาพ

สำหรับ stimulus กับ source of stimulus ลองศึกษาทางพุทธดูก็ได้ครับในเรื่อง ปฏิจจสมุปบาท และอิทัปปัจจยตา จะช่วยให้เข้าใจลึกซึ้งขึ้นเยอะเลย หรือ ศึกษา เซน ก็ได้ครับ

สำหรับ response measure เป็นสิ่งสำคัญมากสำหรับการอธิบาย NFR หรือ Quality Attribute เนื่องจากความต้องการที่ดีควรมีตัวชี้วัด ซึ่ง response measure ก็คือตัวชี้วัดคุณภาพด้านนั้นๆ นั่นเอง

เราสามารถใช้ NFR หรือ Quality Attribute เป็นกรอบหรือ influence ให้กับทีมงานเพื่อออกแบบและพัฒนาระบบให้ได้คุณภาพตามที่กำหนด เช่น ดังรูปด้านล่าง

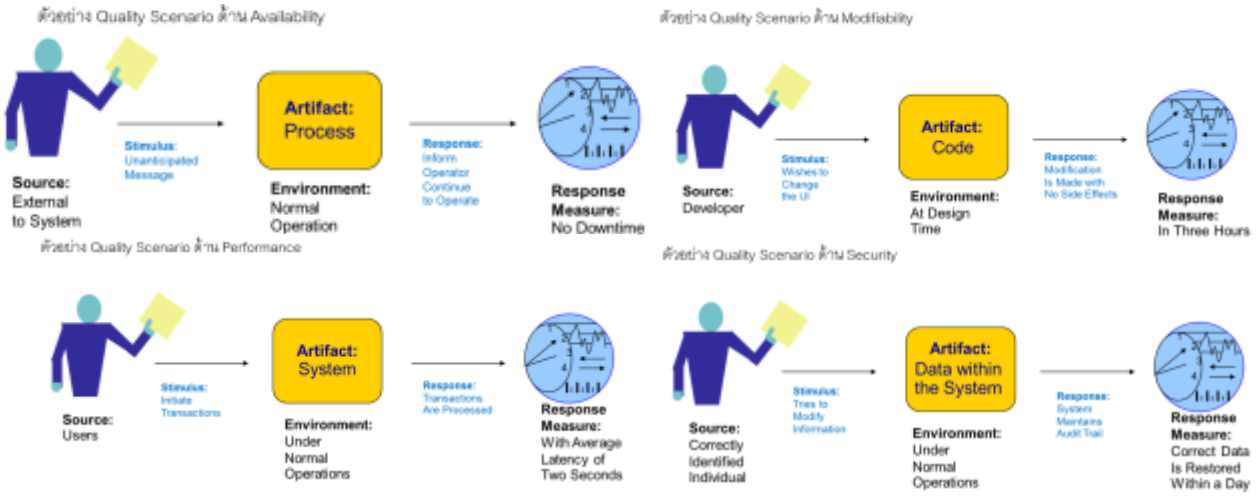


ตัวอย่างการกำหนด response measure หรือ ตัวชี้วัด (KPI) 4 ตัว ให้ architect กับ developer ออกแบบและพัฒนาคอมโพเน้นต์ “Check Out”

System Quality หรือ กล่าวได้ว่า NFR ที่พบได้ในระบบทั่วไปสมัยนี้ได้แก่

- Availability หมายถึง ความพร้อมของสิ่งนั้นๆ เช่น ระบบพร้อม (ไม่ล่ม)
- Performance หมายถึง ประสิทธิภาพ โดย performance เป็นคุณภาพที่ขึ้นกับผลิตภัณฑ์ เช่น ระบบไอที ก็สนใจเรื่องความเร็วซึ่งเกิดจากประสิทธิภาพการประมวลผลและการใช้ทรัพยากร
- Modifiability หมายถึง ความสามารถที่สิ่งนั้นๆ ปรับปรุงแก้ไขได้ง่าย สะดวก มีความยืดหยุ่น มีผลกระทบข้างเคียงน้อย
- Security หมายถึง ความปลอดภัย

- Testability หมายถึง ความสามารถในการทดสอบ เช่น โมดูลสำคัญของระบบจะอยู่ลึกลับซับซ้อนแค่ไหนก็ต้องทดสอบได้ และต้องทดสอบด้วยวิธีการที่เหมาะสม
- Usability หมายถึง การใช้งานที่ง่าย ได้ประโยชน์ สร้างประสบการณ์ที่ดีแก่ผู้ใช้ เป็นมิตรกับผู้ใช้ จึงมักเกี่ยวกับหน้าจอ หรืออินเตอร์เฟส
- Scalability หมายถึง ความสามารถในการรองรับการขยายตัวของระบบได้ ซึ่งอาจเกิดจากมีปริมาณโหลดที่เพิ่มขึ้น
- Interoperability หมายถึง ความสามารถในการทำงานร่วมกันได้ระหว่าง โมดูล หรือ ระหว่างระบบ หรือ ระหว่าง คอมโพเน้นต์ เป็นต้น โดยปราศจากข้อจำกัด เช่น ระบบบัญชีเขียนด้วย Java ต้องทำงานร่วมกับระบบการเงินที่เขียนด้วย C#.NET

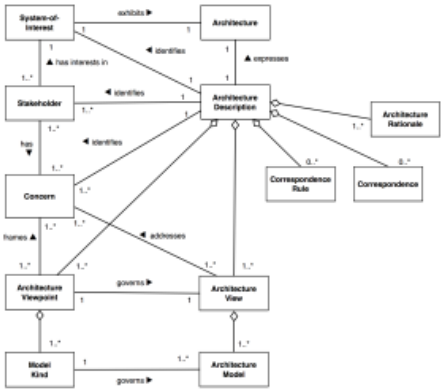


➢ **การอธิบายและสื่อสารสถาปัตยกรรม (Architecture Description & Communication)**

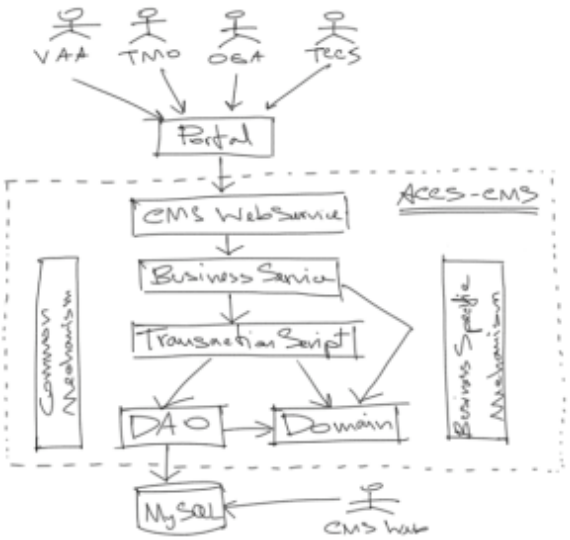
สถาปัตยกรรมที่ออกแบบและสร้างขึ้นจำเป็นต้องสื่อสารให้ stakeholder ได้ทำความเข้าใจ ซึ่งยังใช้เพื่อรับฟีดแบ็กแล้วนำมาปรับปรุงสถาปัตยกรรม ดังนั้นการอธิบายรายละเอียดของสถาปัตยกรรมออกมาจึงมีความสำคัญมาก หาก stakeholder เข้าใจคลาดเคลื่อนไปจากความตั้งใจของ architect แล้วอาจให้ดำเนินการผิดพลาด หรือไม่ได้ตามเป้าหมายที่ต้องการในโครงการปกติมักมี stakeholder ที่หลากหลาย ทั้งตำแหน่งหน้าที่, ประสบการณ์, ความรู้, ความสามารถทางภาษา

การอธิบายจึงต้องสามารถโน้มน้าวให้ stakeholder ทุกคนเข้าใจตรงกัน ดังนั้น architect จำเป็นต้องเข้าใจแนวคิด “Level of Abstraction” เป็นอย่างดี เข้าใจความแตกต่างของ stakeholder แต่ละกลุ่ม เข้าใจประเด็นสำคัญในสถาปัตยกรรมที่จะนำเสนอ โดยเลือกใช้ชุดภาษา , สัญลักษณ์รูปภาพ และรูปแบบการนำเสนอ ให้เหมาะสม และแบ่งระดับรายละเอียดของเนื้อหาให้เหมาะสม เหมือนการทำหนังสือ, หนังสือพิมพ์, นิตยสาร อาทิเช่น หัวข้อหนึ่งอาจมีผู้อ่านที่หลากหลายมีวัตถุประสงค์ในการรับรู้สาระแตกต่างกันการอธิบายสถาปัตยกรรมจะแบ่งรายละเอียดออกเป็นส่วนต่างๆ เรียกว่า

“Architectural view” หมายถึงมุมมองของสถาปัตยกรรม ที่เป็นประเด็นสำคัญที่ต้องการนำเสนอหรืออธิบายออกมา โดยสอดคล้องกับประเด็น stakeholder สนใจ (concern) อาทิ architectural process view, architectural layer view, transaction management view, integration view เป็นต้น



ISO/IEC/IEEE 42010 – Architecture Description



ตัวอย่างมุมมอง “Architecture Overview”

ตัวอย่างมุมมอง “Transaction Overview”

นอกจากนี้การอธิบายสถาปัตยกรรมไม่จำเป็นต้องทำเป็นเอกสารเสมอไป บ้านเรามักชินกับการทำเอกสารเข้าขั้นเสพติด เอกสารเป็นเพียงแค่กลวิธีหนึ่งในการสื่อสารเท่านั้น สถาปัตยกรรมของ

ระบบเล็กๆ ที่ไม่ซับซ้อนมากหรือเป็นการทำงานในทีมที่มีทีมเวิร์กสูง อาจอธิบายด้วยคำพูดและวาดรูปง่ายๆ ก็ยังได้

- **การออกแบบสถาปัตยกรรม (Architecture Design)**

การออกแบบสถาปัตยกรรมก็คือการหา solution ที่ตอบโจทย์ความต้องการหลัก โดยเฉพาะ Business Goal, Architectural Concern, Quality Attribute, Functionalityจากความต้องการด้านต่างๆ โดยเฉพาะความต้องการตัวที่มีผลต่อระบบและสถาปัตยกรรมมากๆ (Architecture Requirement) ซึ่งอยู่ในโซนปัญหา (problem area) ก็จะไหลผ่านกระบวนการพัฒนาไปสู่ระบบที่พัฒนาเสร็จสิ้น (concrete system) ความต้องการเหล่านั้นต้องไหลผ่านการแก้ปัญหาที่อยู่ในโซนแนวทางแก้ไข (solution area) เสียก่อน

solution ที่ใช้แก้ปัญหาทางสถาปัตยกรรมอาทิเช่น Architectural, Architectural Pattern, Architectural Framework การออกแบบสถาปัตยกรรมก็คือการเลือก solution ที่มีอยู่มาปรับใช้หรือสร้าง solution ขึ้นใหม่ ที่ต้องตอบโจทย์ความต้องการที่มีผลต่อระบบและสถาปัตยกรรมให้ได้การออกแบบสถาปัตยกรรมเป็นกระบวนการแบบวนซ้ำ (iterative) โดยเริ่มจากคิด solution คร่าวๆ ระดับไอเดียก่อน (คล้ายเริ่มจากการสเก็ตช์ภาพก่อน แล้วค่อยลงรายละเอียด) ซึ่งตลอดเวลาของการออกแบบจะต้องพิจารณาความต้องการหลัก (และโดยเฉพาะ Quality Attribute) ประกอบไปด้วยเสมอ

สถาปัตยกรรมที่ออกแบบต้องนำเสนอภาพรวมและโฟกัสให้เห็นประเด็นสำคัญ แล้วสื่อสารกับ stakeholder ต่างๆ รวมถึงทีมงาน โดยมุ่งหมายให้สถาปัตยกรรมเป็นต้นแบบหรือพิมพ์เขียวสำหรับการทำงานในส่วนต่างๆ ต่อไป อาทิ detail design, เขียนโปรแกรม, testing, maintain ระบบ

แนวทางที่ใช้ในการออกแบบสถาปัตยกรรมมีมากมาย อาทิ Attribute-Driven Design, Model-Driven Architecture, Domain-Driven Design เป็นต้น

- **การประเมินสถาปัตยกรรม (Architecture Evaluation)**

การที่ Architecture Requirement จะผ่าน solution มาอย่างราบรื่นตอบโจทย์ทางธุรกิจและเทคนิคหลักได้หรือไม่ขึ้นอยู่กับ solution ที่เลือกมาใช้ ดังนั้นจึงจำเป็นต้องมีการประเมิน solution เหล่านั้นอย่างรอบคอบนั่นเอง

การประเมิน solution ที่นำมาใช้แก้ปัญหาทางสถาปัตยกรรม สามารถเริ่มประเมินได้ตั้งแต่ solution ยังเป็นแค่ไอเดียคร่าวๆ ไปจนถึง solution ได้รับการออกแบบรายละเอียดแล้ว แต่ยิ่งประเมินเร็วก็ยิ่งลดความเสี่ยงเร็ว ดังนั้นผู้ประเมินควรมีความรู้ในหลักการพื้นฐานของ solution เหล่านั้นอย่างดี

การประเมินจำเป็นต้องประเมินหลายมิติควบคู่กันไป อาทิ ประเมินทั้งด้านคุณภาพและความคุ้มค่าของ solution (ซึ่งอาจรวมถึงประเมินด้านกฎหมายและแผนฯ ด้วยในบางกรณี) ดังนั้น Architecture Evaluation จึงถือเป็นเครื่องมือสำคัญหนึ่งในการทำการศึกษาความเป็นไปได้ หรือ ‘Feasibility Study’

นอกจากนี้ในการประเมิน solution ต่างๆ จึงต้องทำควบคู่กันไปกับการทำ Proof-of Concept หรือ POCสำหรับการประเมินสถาปัตยกรรมแตกต่างจากการทดสอบ (testing) เพราะเราควรประเมิน solution ให้ได้ก่อนนำไปอิมพลีเม้นต์ เพราะหากอิมพลีเม้นต์แล้วจะมาดูว่ามันเวิร์กไหม อย่างนี้เรียกว่า test แล้วล่ะ แต่…การประเมินอาจใช้ testing มาเป็นเครื่องมือช่วยก็ได้ ด้วยการนำ solution ไปสร้าง prototype (เขียนโค้ด mock up) แล้วสร้าง test case ขึ้นมาเพื่อทดสอบสมมติฐาน

Proof-of Concept

- **การบูรณะสถาปัตยกรรม (Architecture Reconstruction)**

การรื้อ แกะ แคะ แงะ สถาปัตยกรรม เหมารวมไปถึงการทำซ้ำ/การลอกเลียนแบบ (ก๊อปปี้) หรือผมชอบเรียกว่า hacking architecture design ซึ่งส่วนตัวคิดว่าในอนาคตน่าจะเป็นอีกอาชีพที่น่าสนใจมากๆครับ เพราะคนไทย จีน ไต้หวัน ฮ่องกง มีพรสวรรค์ด้านนี้อยู่ใน DNA มาแต่ครั้งบรรพกาลแล้วจุดประสงค์ของการทำ Architecture Reconstruction มีหลายเหตุผล อาทิ

- เพื่อจัดทำเอกสารสถาปัตยกรรมของระบบขึ้นมาใหม่
- เพื่อวิเคราะห์และเรียนรู้การทำงานของระบบ
- เพื่อระบุองค์ประกอบของระบบว่าส่วนใดที่สามารถ re-use ได้บ้าง ส่วนใด re-use ไม่ได้
- เพื่อระบุส่วนที่ต้องปรับปรุง
- เพื่อการวิวัฒน์ (evolve) สถาปัตยกรรมให้ทันสมัย

Architecture Reconstruction เป็นกระบวนการที่มีการดำเนินการในลักษณะวนซ้ำ (iterative) ทำไปเรื่อยๆ จนกว่าจะบรรลุเป้าหมาย มีกิจกรรมที่เกี่ยวข้องจำนวนมาก โดยมักไม่สามารถทำให้เป็นอัตโนมัติตั้งแต่ต้นจนจบได้ แต่กิจกรรมบางอย่างอาจทำให้เป็นอัตโนมัติได้บ้าง ซึ่งมักต้องใช้เครื่องมือ (tool) ช่วย อาทิการใช้เครื่องมือประเภท reverse engineering, de-compiler เป็นต้น

การจะเป็น architect ระดับสุดยอดได้ก็ต้องฝึก “รื้อ แกะ แคะ แงะ” สถาปัตยกรรมของคนอื่นบ่อยๆ คล้ายกับ programmer ที่จะเขียนโปรแกรมเก่งๆ ก็ต้องแกะโค้ดโปรแกรมอื่นบ่อยๆ บางทีการแกะดีไซน์ของคนอื่นเป็นสิ่งที่เราๆ ก็คุ้นเคยอยู่แล้ว กรณีที่เรียบง่ายและใกล้ตัวที่สุด อาทิ การแต่งตัวตามแฟชั่น, การแต่งสวนตามคอลัมน์ในนิตยสาร, การดัดแปลงรถให้คล้ายกับรถแข่งของนักแข่งชื่อดัง, การซื้อหนังสือแบบบ้านสำเร็จรูปเพื่อเลือกแบบมาดัดแปลงแล้วออกแบบใหม่ให้ได้บ้านแบบที่ชอบ เพียงแต่การจะแกะ software architecture นั้นผู้ปฏิบัติย่อมต้องมีพื้นฐานที่แข็งแกร่งพอควร

เพราะสถาปัตยกรรมบางระบบที่ซับซ้อนมากๆ โดยเฉพาะระบบปิด อาจต้องใช้กำลังภายในมากพอควร ซึ่งนี่ก็คือความท้าทาย เพราะหาก “reconstruct” สถาปัตยกรรมจนชำนาญแล้ว ก็จะก้าวไปสู่งานด้าน Architecture Evaluation เพื่อไปเป็นผู้ประเมินสถาปัตยกรรมได้ไวขึ้น

- **คอมโพเน้นต์สำเร็จรูป (Component-Off-The-Shelf)**

ในการพัฒนาระบบสมัยใหม่เป็นลักษณะการออกแบบคอมโพเน้นต์แล้วนำมาประกอบ (assemble) กันเป็นระบบที่เรียกว่า “Component-Based System” หรือการออกแบบเซอร์วิสแบบในสถาปัตยกรรม SOA และ Cloud Computing เมื่อทราบว่าระบบต้องมีคอมโพเน้นต์อะไรบ้างแล้ว ต้องวิเคราะห์เพื่อประเมินว่าคอมโพเน้นต์ใดจะพัฒนาเองคอมโพเน้นต์ไม่ควรพัฒนาเองโดยเลือกใช้ COTS (Commercial/Component Off-the-Shelf)

การได้คอมโพเน้นต์มาใช้งาน (acquisition) มีหลายทางเลือก อาทิ ได้มาด้วยการ outsource ให้ผู้อื่นผลิตให้, ซื้อคอมโพเน้นต์สำเร็จรูปมาใช้, ดาวน์โหลดคอมโพเน้นต์ที่เป็นโอเพ่นซอร์สมาใช้ architect ต้องเป็นผู้ประเมินคอมโพเน้นต์ที่ออกแบบ โดยวิเคราะห์อินเตอร์เฟสและ Quality Attribute ของคอมโพเน้นต์ในสถาปัตยกรรม เพื่อระบุข้อกำหนด (specification) และเกณฑ์คัดเลือกคอมโพเน้นต์ (qualification) เพื่อป้องกันปัญหา “interface mismatch” หรือ ความไม่เข้ากัน หรือเข้ากันได้แต่ไม่ครอบคลุม quality attribute และ functionality ที่กำหนดนอกจากนี้ architect ยังต้องประเมินเบื้องต้นเกี่ยวกับ ค่าใช้จ่าย, ระยะเวลา, องค์ความรู้ที่เกี่ยวกับคอมโพเน้นต์นั้นๆ และทักษะของผู้ที่จะมาอิมพลีเม้นต์คอมโพเน้นต์นั้นๆ รวมไปถึงความเสี่ยงในการอิมพลีเม้นต์ เพื่อเป็นข้อมูลตั้งต้นให้ project manager ประมาณการณ์และวางแผนโครงการต่อไป

การสร้างระบบไอทีทุกวันนี้เราไม่ควรสร้างทุกชิ้นส่วนขึ้นมาเอง เสียทั้งเวลา, งบประมาณ และกำลังคน ให้นึกถึงกฎ 80/20 ของ Pareto เพราะงานสถาปัตยกรรมที่ดีควร “Do More with Less” หรือ “Less is More” นั่นเอง ความบ้าพลังของผู้บริหารและนักไอทีไม่น้อยที่มีค่านิยม “ชอบทำทุกชิ้นส่วน” เอง นำมาซึ่งการสูญเสีย ให้นึกถึงอาคาร, ห้างสรรพสินค้า, เครื่องจักร สมัยนี้ที่ออกแบบ, วางแผน และผลิตเสร็จเร็วมาก อาคารใหญ่ๆ สมัยนี้เพียงไม่ถึงปีก็สร้างเสร็จแล้ว ขณะที่ระบบไอทีบางระบบที่ไม่ได้ใหญ่อะไรนักกลับใช้เวลากว่าปี! ผมพบผู้บริหารหลายรายมักมีทัศนคติว่าอะไรทำเองได้ก็น่าประหยัดกว่า… ซึ่งบางครั้งมันไม่ใช่! เช่น การเอาคนเงินเดือนเกือบแสนมานั่งวาดรูปโมเดล แทนที่จะเอาเด็กๆ เงินเดือนน้อยๆ มาวาดรูป แล้วเอาคนเงินเดือนแพงๆ มาตั้งโจทย์ให้เด็กๆ วาดรูปแล้วตัวเองมานั่งวิเคราะห์รูปนั้นแทน น่าจะคุ้มค่ากว่า

➢ **สายการผลิตซอฟต์แวร์ (Software Product Lines)**

Software Product Lines (SPL) คือ สายผลิตภัณฑ์ซอฟต์แวร์ในลักษณะ “กลุ่มผลิตภัณฑ์” (product family) หมายถึง มีระบบหลายระบบที่มีการแชร์บาง architectural element ร่วมกัน ยกตัวอย่าง เช่น Microsoft Office ที่ประกอบด้วย Word, Excel, PowerPoint, Access ฯ ที่มีการแชร์กลไกการทำงานบางอย่างร่วมกัน เป็นต้นกลุ่มผลิตภัณฑ์เหล่านี้ต้องมี “core asset base” หรือสถาปัตยกรรมกลาง (Product Line Architecture) เพื่อแชร์องค์ประกอบบางอย่างระหว่างกันได้ อาทิ คอมโพเน้นต์, ไลบรารี่, คอนฟิกกุเรชั่น เป็นต้น

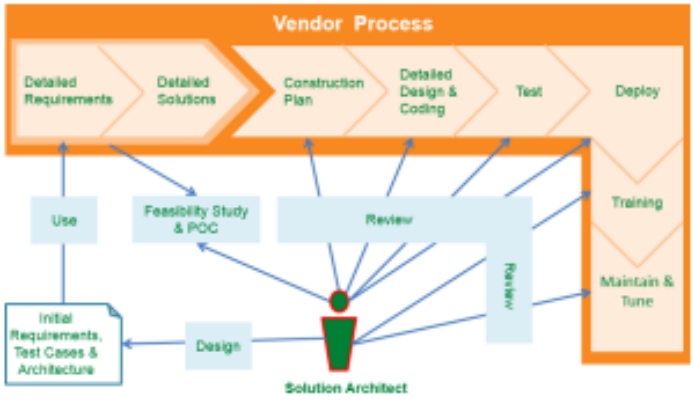
SPL เป็นการมุ่งเน้นการ re-use ในเชิงกลยุทธ์ (strategic reuse) ซึ่งไม่ได้เน้นแค่ด้านเทคนิคเท่านั้น แต่เน้นธุรกิจและการวางแผนด้วย แนวทางการ re-use ของ SPL จึงแตกต่างจาก Object-Orientation อยู่มากในแง่ขอบเขตและวิธีปฏิบัติ

SPL เป็นการหยิบยืมแนวคิด ‘สายการผลิต’ (Product Line) มาจากอุตสาหกรรมการผลิต ที่มีมายาวนาน ตัวอย่างผลิตภัณฑ์ที่มี Product Line อาทิ เครื่องยนต์, โทรศัพท์มือถือ, คอมพิวเตอร์, เครื่องใช้ไฟฟ้า, เฟอร์นิเจอร์, หรือแม้แต่แม็คโดนัลด์, เซเว่นอีเลฟเว่น

SPL เน้นเรื่องการ re-use เพื่อลดต้นทุนและระยะเวลาการผลิตลง จึงมีประโยชน์มากกับการพัฒนาซอฟต์แวร์เชิงพาณิชย์, การพัฒนาระบบขนาดใหญ่อย่าง enterprise system ที่ประกอบด้วยระบบย่อยจำนวนมาก, การพัฒนาเซอร์วิสจำนวนมากอย่างในสถาปัตยกรรมแบบ SOA เป็นต้น แต่...เรามักพบว่าการจะใช้ SPL ให้สัมฤทธิ์ผลต้องอาศัยการวางแผนและ...มองการณ์ไกล ซึ่งขัดแย้งกับแนวทางการพัฒนาระบบไอทีในบ้านเรามาก ที่มักนึกถึงไอทีเมื่อเวลาเจอปัญหา แต่กลับไม่พยายามมองให้เห็นปัญหาในอนาคต จึงมักพบได้บ่อยทั้งองค์กรภาครัฐและเอกชนที่มักเอาไอทีมาแก้ปัญหาเฉพาะหน้าหรือแค่เพียงช่วงเวลาหนึ่ง

ความสามารถและหน้าที่ Architect ด้าน IT/Software/Solution

การพัฒนาความสามารถด้าน software architecture ซึ่งรวมไปถึงด้าน IT architecture และ solution architecture งานทั้ง 3 ด้านนี้คล้ายคลึงกัน ต่างกันที่ขอบเขตและระดับความลึกของงาน นอกจากนี้ยังได้ไปบรรยายและเป็นที่ปรึกษาให้กับหลายองค์กร



ความสามารถ (ความรู้ + ทักษะ + คุณลักษณะ) ของ Architect ด้าน IT (IT Architect) หรือ ด้าน Software (Software Architect) หรือ ด้าน Solution (Solution Architect) มีมากมายมหาศาลครับ เพราะงานด้าน architecture ต้องรู้กว้าง มีความสามารถหลากหลาย ดังนั้นอย่าเพิ่งตกใจกันครับ ส่วนจะรู้ลึกไปด้านไหนก็ขึ้นกับความสนใจส่วนตัวและขึ้นกับ domain ของงานที่ทำครับ

สำหรับ architect นั้นมี 2 แบบใหญ่ๆ คือ architect แนวกว้าง (cross domain architect) หรือ architect แนวนอน ที่ต้องรู้กว้าง เก่งแบบกว้างๆ ไปทำระบบอะไรก็ได้ไม่ได้ชำนาญใน domain ใด domain หนึ่ง ไปทำงานในองค์กรในอุตสาหกรรมหรือธุรกิจประเภทไหนก็ได้ หรือระบบหรือเทคโนโลยีอะไรก็ได้ (ผมเองก็เป็น architect ประเภทนี้ครับ เพราะผมเป็นฟรีแลนซ์ ทำงานอิสระ เจอลูกค้าแบบไหนก็ได้) ส่วน architect แบบที่สอง คือ architect แนวลึก (domain architect) หรือ architect แนวตั้ง ที่ต้องรู้ลึกใน domain ใด domain หนึ่ง เช่น ชำนาญด้านระบบ banking ชำนาญด้านเทคโนโลยี .NET ชำนาญระบบประเภท web application เป็นต้น

สำหรับ architect ประเภทนี้ถ้าเลือกสนใจทำ domain ใดแล้ว การจะมาเปลี่ยนไป domain อื่นในภายหลัง เช่น ย้ายงาน อาจเหนื่อยหน่อย แต่ถ้าเป็นคนที่รักการเรียนรู้และปรับตัวง่ายก็ไม่น่ากังวล

ทักษะและหน้าที่

- Analyze, design and maintain IT solution
- Define architecture landscape
- Analyze, design and maintain solution architecture, focus on structure and interoperation of business, data, application, service, technology, infrastructure
- Listen to and make understanding all key stakeholder’s aspect
- Define and clarify constraints, concerns, business goals, system features, use cases, quality attributes, mechanisms
- Find win-win solution for all key stakeholders
- Collaborate with a variety of stakeholders (product development, operation, infrastructure, development, vendor, etc.)
- Define architecture principles to shape the implementation
- DELIVER SOLUTION!

ความรู้ด้านไอที

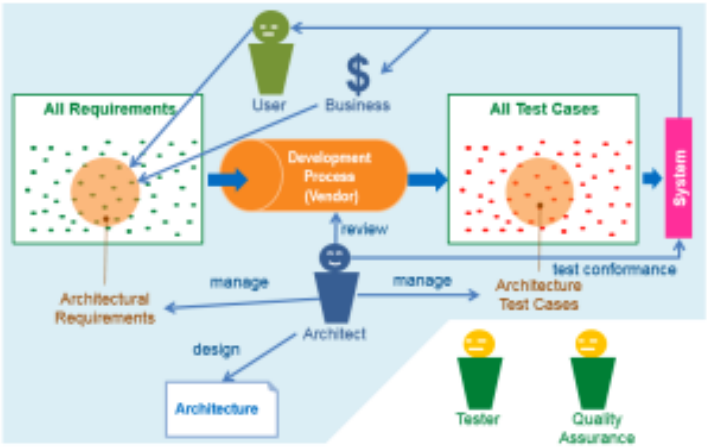
- Object-Oriented Analysis and Design
- UML (Unified Modeling Language)
- Design principles
- Software development principles
- Non-Functional Requirements, Quality Attributes
- Enterprise architecture, Software architecture
- Solution analysis, design and management
- Software process: CMMI, Agile
- Design Patterns, Architectural Patterns
- IT security
- Business technology

ความรู้และทักษะด้านอื่น

- Communication
- Business thinking
- Transform idea to picture by drawing/modeling
- Feasibility study and proof-of-concept
- Documentation
- Consulting and coaching
- Strategy
- Risk and change management
- Political and social issues handling

คุณลักษณะ/บุคลิก

- Good leadership
- High accountability
- Eager to learn new things
- Good attitude and EQ
- Thinking เช่น :
  - Creative thinking (การคิดเชิงสร้างสรรค์)
  - Integrated thinking (การคิดเชิงบูรณาการ)
  - Lateral thinking (การคิดนอกกรอบ หรือ การคิดแนวข้าง)
  - Positive thinking (การคิดบวก)
  - Strategic thinking (การคิดเชิงกลยุทธ์)
- Good human relations
- Good personality
- Good decision making



แนวทางการพัฒนาและปรับปรุงความสามารถของ Architect ในองค์กร

- จัดแบ่งกลุ่มหรือประเภทโซลูชั่นที่จะมีใช้ในองค์กร แล้วกำหนดเป็น “domain” แล้วจึงกำหนดคุณสมบัติ: ความสามารถและหน้าที่ ของ solution architect ในแต่ละ solution domain แล้วจึงคัดเลือกผู้ที่มีความเหมาะสมที่จะ “ปั้น” ให้เป็น solution architect
- กำหนดระดับของ solution architect ตามความสามารถ (ไม่ใช่อายุและวุฒิการศึกษา)
- กำหนดระดับความสามารถ (competency) ซึ่งประกอบด้วย ความรู้, ทักษะ และ คุณลักษณะ จากนั้นกำหนด career path, job description, learning path แล้วกำหนดตัวชี้วัดสำหรับ solution architect แต่ละระดับ และกำหนดให้กับสำหรับ solution architect แต่ละคน

Transaction Overview

การออกแบบทรานแซกชั่น เป็นขั้นเริ่มต้นของกระบวนการการทำงานที่สำคัญส่วนหนึ่งเลยทีเดียว เนื่องจาก หากมีการออกแบบที่ดี จะสามารถช่วยในการจัดการ บริหาร หรือ ควบคุมได้อย่างง่ายดาย และยังสามารถให้ประสิทธิภาพในการทำงานได้อย่างเต็มที่อีกด้วย ผู้จัดทำจะขอเล่าพื้นฐานและหลักการคร่าว ๆ ที่เกี่ยวข้องกับการออกแบบและจัดการทรานแซกชั่น เพื่อเกริ่นนำก่อนที่จะเข้าสู่ส่วนของการวิเคราะห์และออกแบบ ทรานแซกชั่นกันต่อในบทถัดไป

- **Transaction & Transaction Processing**

  Transaction หรือคำในภาษาไทยคือ ‘ธุรกรรม’ มีความหมายว่า กิจกรรมที่เกี่ยวกับการทำนิติกรรม สัญญา หรือการดำเนินการใด ๆ ที่เกี่ยวข้อง หรือที่มักเข้าใจกันโดยทั่วไปว่าเป็น กิจกรรมที่มีการซื้อขายแลกเปลี่ยนกันเกิดขึ้น ในทางไอทีสามารถอธิบายได้ว่า คือ กิจกรรมที่มีการประมวลผลและใช้ทรัพยากรประกอบการประมวลผล โดยวัตถุประสงค์เพื่อให้ได้รับผลลัพธ์ที่ผูกพันกับข้อมูลที่นำเข้าและความรับผิดชอบของกิจกรรม ผลลัพธ์ของทรานแซกชั่นอยู่บนพื้นฐานแบบ ‘all-or-nothing’

  Transaction Processing คือ การประมวลผลข้อมูล โดยแบ่งการประมวลผลเป็นส่วนๆ โดยการประมวลผลแบ่งเป็น operation ย่อยๆ ซึ่งเรียกว่า ‘transaction’ โดย

  - ผลลัพธ์ต้องเป็นอย่างใดอย่างหนึ่งระหว่าง สาเร็จ หรือ ไม่สาเร็จ
  - การประมวลผลทรานแซกชั่น จึงเป็นการควบคุมและรับประกันการประมวลผลว่า
  - หากไม่มีข้อผิดพลาดเกิดขึ้น ผลลัพธ์จะเป็นผลสาเร็จ
  - หากมีข้อผิดพลาดเกิดขึ้นระหว่างประมวลผล ข้อมูลที่ถูกเปลี่ยนแปลงหรือเสียหายจะถูกย้อนสถานะ (rollback) กลับไปยังจุดเริ่มต้นก่อนประมวลผล โดยไม่ส่งผลเสียหาย
  - การประมวลผลทรานแซกชั่น ยังรับประกันถึงการควบคุมความสอดคล้องและความถูกต้องของข้อมูลที่ถูกใช้ประกอบการประมวลผล (data consistency)

➢ **Event Source & Arrival Event**

Event Source คือ จุดกำเนิดของเหตุการณ์ หรือ กิจกรรม อธิบายได้ว่าเป็นสิ่งที่สร้างเหตุการณ์ (event) ขึ้นมา ซึ่งเหตุการณ์ส่งผลให้เกิดการใช้ทรัพยากร ตามมา เช่น เมื่อเกิดการประมวลผลก็จะเกิดการใช้ ทรัพยากร ในการออกแบบ ซึ่งเราต้องวิเคราะห์ตั้งแต่จุดที่มีการร้องขอการใช้ทรัพยากรจากไคลเอนต์มาถึงระบบ หรืออย่างน้อยมาถึงจุดที่ทาให้เกิดการ begin ทรานแซกชั่น

Arrival Event คือ การเกิดขึ้นของเหตุการณ์ ซึ่งจาเป็นต้องทราบว่ามีความถี่เท่าไหร่ เป็นประเภทไหน จะได้นำไปวิเคราะห์และคาดการณ์ได้ โดยแบ่งเป็นประเภทเหตุการณ์ คือ

- **Periodic event** คือ event ที่มีความแน่นอนของ event เกิดเป็นประจำ สม่ำเสมอ คาดการณ์ได้ล่วงหน้า เช่น ในทุกสิ้นเดือนระบบฝาก / ถอนของธนาคารจะทำงานหนัก, พนักงานแผนกบัญชีมักล็อกอินเข้าระบบในช่วง 8:00-8:15 ของทุกวัน, ทุก 15 วินาที มีการ backup transaction state ตลอดทั้งวัน, ลูกค้าประเภท B เมื่อกดรหัสบัตร ATM เสร็จแล้วจะถอนเงินด้วยการเลือกตัวเลขบนหน้าจอทันที
- **Sporadic event** คือ event ที่มีความแน่นอนในช่วงเวลาหนึ่ง เช่น admin ของระบบมักเข้ามาตรวจสอบสถานการณ์ทำงานของระบบในช่วง 7:00-8:00 กับ 17:00-18:00 นอกเหนือเวลานี้จะเข้ามาเป็นครั้งคราว หรือขึ้นกับสถานการณ์, ลูกค้าประเภท Z มักเข้ามาซื้อสินค้าผ่านเว็บในช่วงกลางคืน แต่ช่วงเช้าและกลางวันก็มีเข้ามาบ้าง, เมื่อมีการอัพเดตโปรโมชั่นใหม่ ลูกค้ามักเข้ามาสั่งซื้อสินค้ามากขึ้นในช่วง 3 วันแรก แต่บางโปรโมชั่นก็ไม่ประสบความสำเร็จ
- **Stochastic event** คือ event ที่คาดการณ์ล่วงหน้าไม่ได้ เช่น เจ้าหน้าที่ไม่ทราบว่าลูกค้าที่เข้าใช้ระบบ internet banking แต่ละคน เมื่อล็อกอินผ่านแล้วจะทำอะไรต่อไป, ทีมงานไม่ทราบได้ว่า request ที่ส่งมาจากจุดจำหน่ายบัตรคอนเสิร์ต 8,500 สาขาทั่วประเทศ จะค้นหาสถานะที่นั่งในรอบใดราคาใด, ราคาหุ้นหลายตัวที่ถูกปั่นราคาจนมีราคาสูงขึ้นและลดลงอย่างรวดเร็วได้ทำให้ตลาดหลักทรัพย์มีความผันผวนหนักซึ่งกระทำโดยนักปั่นราคาที่มีทุนมหาศาลทำตัวเสมือนเจ้ามือในวงพนัน

➢ **ACID**

คุณสมบัติที่เกี่ยวกับการจัดการ transaction ในส่วนของการเข้าถึง resource เพื่อให้การทำงานมีความน่าเชื่อถือและมีประสิทธิภาพ

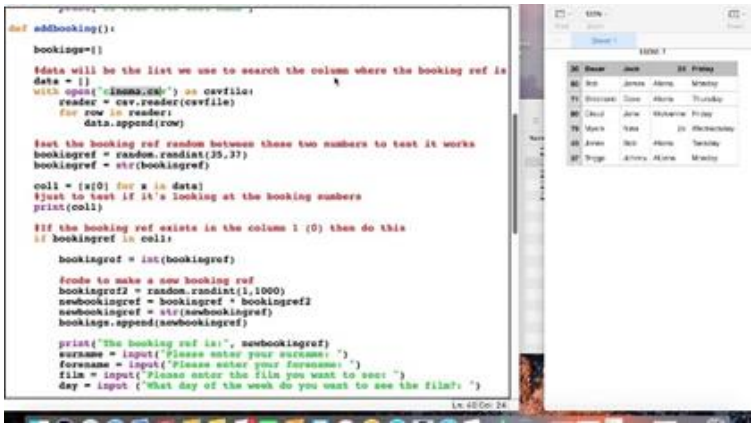
- A – Atomicity คือ หน่วย (atom) ของ transaction โดย transaction ควรแบ่งเป็น atom หรือแบ่งเป็น transaction เล็กๆ ย่อยๆ ซึ่งแต่ละ atom มีการจัดการ transaction เป็นของตัวเอง หรือขึ้นกับการจัดการ ผลลัพธ์คือสำเร็จทั้งหมดหรือไม่ก็ rollback ทั้งหมด
- C – Consistency คือ การจัดการความน่าเชื่อถือ ความถูกต้อง และความสอดคล้องของ resource เมื่อเกิดเหตุการณ์ที่มีหลายงานเข้าถึง resource เดียวกันพร้อมๆ กัน
- I – Isolation คือ การป้องกันการผิดพลาดจากการเข้าถึง resource พร้อมๆ กัน เป็นลักษณะของการจัดการด้านการ lock resource และเพื่อป้องกันการการเกิด dead-lock
- D – Durability คือ ระยะเวลาของ atom หรือ transaction ที่ประมวลผล สังเกตุได้ว่าหาก atom มีขนาดใหญ่จะส่งผลต่อระยะเวลาที่นานขึ้น

➢ **ประเภทของ transaction**

การแบ่งประเภทของ transaction มีหลายแบบ แต่รูปแบบใหญ่ ๆ ที่นิยมใช้งานกัน คือ

- แบ่งตามประเภทการประมวลผล เช่น
  - Local transaction processing
  - Distributed transaction processing
- แบ่งตามประเภทโมเดลของทรานแซกชั่นตาม Transaction Service Specification โดย OMG (Object Management Group)
  - Flat transaction
  - Nested transaction
- แบ่งตามประเภทโมเดลของทรานแซกชั่นตามแนวทางของภาษาโปรแกรมนั้น ๆ ตัวอย่างในภาษา Java เช่น
  - Local transaction model
  - Programmatic transaction model
  - Declarative transaction model

ใช้วิธีวาดรูป transaction model หลังจากวิเคราะห์ความต้องการประเภทต่างๆ และวิเคราะห์ปัจจัยต่างๆ แล้ววิเคราะห์รูปโมเดลต่างๆ แล้วจำแนกว่าในระบบที่กำลังพัฒนานี้น่าจะแบ่งทรานแซกชั่นออกเป็น กี่ประเภท กี่กลุ่ม เช่น แบ่งตามระดับขนาด(สั้น-ยาว), ระดับความซับซ้อน, ความถี่ในการเข้าใช้ resource และ shared resource, ปริมาณ resource system ที่เข้าใช้ ฯลฯ

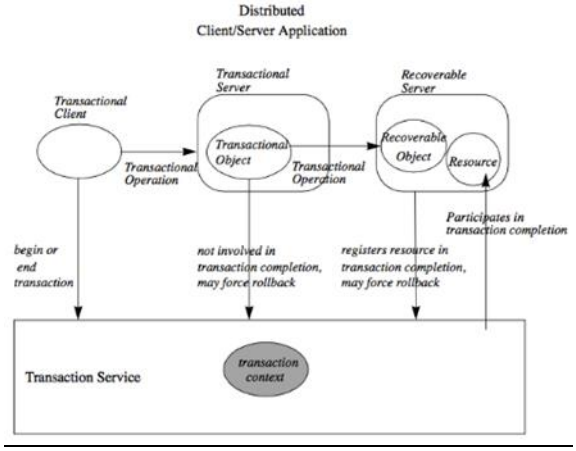


ตัวอย่าง – ทรานแซกชั่นในรูปแบบซอร์สโค้ดในภาษา Python

➢ **Operation สาหรับการจัดการ transaction หลัก ๆ**

- **Begin** คือ การเริ่มต้นทรานแซกชั่น และเป็นการแจ้งให้ transaction manager รับรู้และพร้อมจัดการ เป็นการเริ่มต้น (สร้าง) unit of work พร้อมกับสร้าง transaction session เริ่มจองหน่วยความจำ เพื่อเตรียมเก็บ transaction data หรือ temporary data
- **Commit** คือ การจบหรือปิดทรานแซกชั่น เมื่อประมวลผลทรานแซกชั่นเสร็จ
- **Rollback** คือ การย้อนสถานะของ transaction data กลับไปสู่จุดเริ่มต้น หรือจุดที่ได้มาร์ก check point ไว้
- **Continue** คือ การตัดสินใจทำทรานแซกชั่นต่อ โดยไม่ rollback หรือ rollback เฉพาะบางจุดที่กำหนด ในกรณีที่อาจเกิด error ขึ้น
- **Get Status** คือ การแจ้งสถานะของทรานแซกชั่น เช่น active, committed, committing, marked rollback, no transaction, prepared, preparing, rolled back, rolling back, unknown

องค์ประกอบหลักในการจัดการทรานแซกชั่นตาม Transaction Service Specification โดย OMG



➢ **Transaction Status**

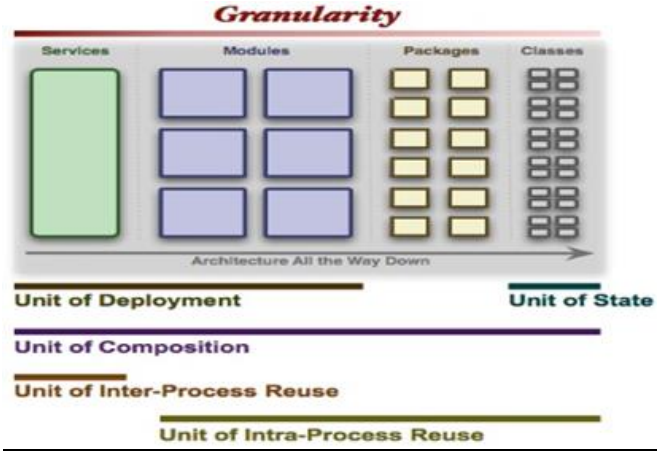
เราสามารถทราบสถานะการทำงานของทรานแซกชั่นได้ด้วยการเขียนโปรแกรมที่มีการเรียกใช้ API ที่มี (กรุณาตรวจสอบภาษาโปรแกรม / เฟรมเวิร์ก / ไลบรารี่ ที่ใช้ เนื่องจากในแต่ละ ภาษาโปรแกรม / เฟรมเวิร์ก / ไลบรารี่ มีรูปแบบการเรียกใช้งานที่แตกต่างกัน แต่โดยปกติมีให้เรียกใช้ได้อยู่แล้ว) หรือดูจากซอฟต์แวร์ประเภท transaction process monitoring (เป็นซอฟต์แวร์เฉพาะทาง ที่ใช้ในการดู)

ตัวอย่างสถานะทั่วไปของทรานแซกชั่น :

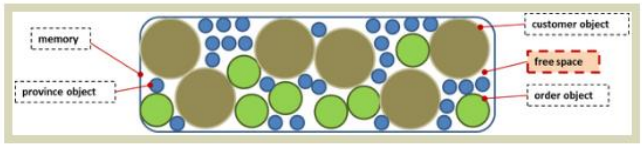
- Active คือ ทรานแซกชั่นกาลังประมวลผลอยู่
- Committed คือ ทรานแซกชั่น commit เสร็จแล้ว
- Committing คือ ทราบแซกชั่นที่กาลัง commit แต่ยังไม่เสร็จ
- Marked rollback คือ ทรานแซกชั่นถูกระบุว่าจะถูก rollback
- No transaction คือ ไม่มีการสร้างทรานแซกชั่นเซสชั่น
- Prepared คือ ทรานแซกชั่นเตรียมเสร็จแล้ว สิ่งที่ต้องเตรียม เช่น ทรานแซกชั่นเซสชั่น การเชื่อมต่อกับ transaction manager ฯลฯ
- Preparing คือ ทรานแซกชั่นกาลังเตรียมพร้อม
- Rolled back คือ ทรานแซกชั่นถูก rollback แล้ว
- Rolling back คือ ทรานแซกชั่นกาลัง rollback
- Unknown คือ ไม่ทราบสถานะของทรานแซกชั่น

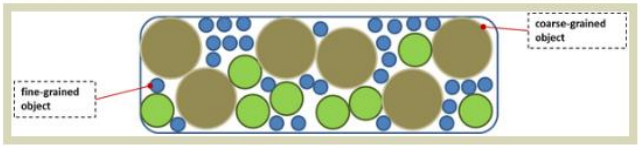
➢ **Granularity**

การเน้นในด้านของขนาดของสิ่งต่างๆ เป็นหน่วยๆ และยังเกี่ยวข้องกับปริมาณของสิ่งนั้นๆ Granularity มีผลต่อ performance มาก



- **Life Cycle** – วัฏจักรสิ่งใดๆ โดยทั่วไปคือการสร้าง (construct), การถูกใช้ (action/use), การทำลาย (destroy) อาจมีขั้นตอนเพิ่มเติมอื่น เช่น pre-condition หรือ initialization
- **Impact** – ขนาดของสิ่งใดๆ มีผลต่อเนื้อที่หรือพื้นที่, การคงอยู่ของสิ่งนั้นสั้น-ยาวมีผลต่อระยะเวลาและการถือครองเนื้อที่หรือพื้นที่
- **State** – สิ่งใดๆ อาจมีการเปลี่ยนแปลงสถานะ หรือเคลื่อนที่ได้ เช่น
  - Swap-out จากสถานะ ready ในหน่วยความจาสู่สถานะ block ในฮาร์ดดิสก์
  - Swap-in จากสถานะ block ในฮาร์ดดิสก์สู่สถานะ running ในหน่วยความจำ
- **Size** – ขนาดของสิ่งใดๆ จะทราบได้ต้องมีตัวเปรียบเทียบ สิ่งที่มีขนาดเล็กๆ เรียกว่า Fine-Grain เช่น fine-grained object สิ่งที่มีขนาดใหญ่ๆ เรียกว่า Coarse-Grain เช่น coarse-grained object

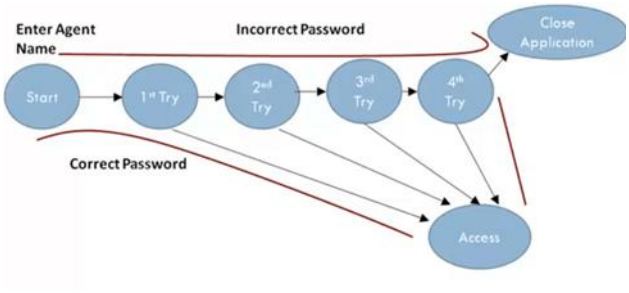




➢ **State Transition & State Management Overview**

State หรือ สภาวะภายใน ณ ช่วงเวลาขณะหนึ่งของ ‘สิ่งๆ หนึ่ง’ มีองค์ประกอบหลัก เช่น ข้อมูล,storage, เวลา, action, event, transition, lifecycle เป็นต้น นอกจากนี้อาจมี Worker, and Business Rule/Logic ประกอบด้วยได้ สิ่งใดๆ โดยทั่วไปประกอบด้วย capability และ property ทั้ง capability และ property สามารถมี state ได้ 20

- State ของ capability คือ สภาวะของแต่ละขั้นตอนการทางาน หรือการเปลี่ยนแปลงขณะประมวลผล
- State ของ property คือ สภาวะของคุณสมบัติของสิ่งนั้นๆ
- State Transition คือ การเปลี่ยนจากสถานะหนึ่งไปอีกสถานะหนึ่ง
- เกิดขึ้นแบบ automatic
- เกิดขึ้นแบบ manual หรือมี stimulus ไปกระทำ



ตัวอย่าง State Transition Diagram ของการป้อนรหัสผ่านเพื่อเข้าสู่ระบบของโปรแกรม

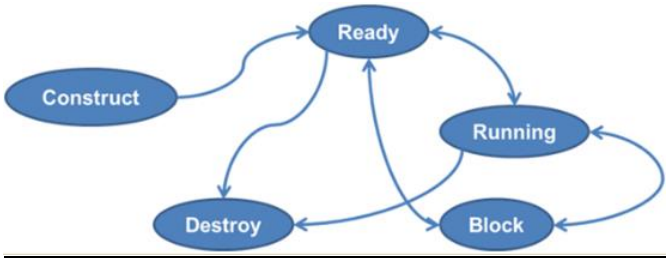
- S1 = หน้าใดๆ ที่มีปุ่ม 'Login'
- S2 = แสดงหน้าจอรอรับข้อมูล
- S3 = แสดงแสดงผลลัพท์ที่ได้จากการประมวลผลข้อมูล
- S4 = แสดงหน้าจอเข้าสู่ระบบของโปรแกรมได้
- S5 = แสดงข้อผิดพลาดของข้อมูลและผิดโปรแกรม

State Management

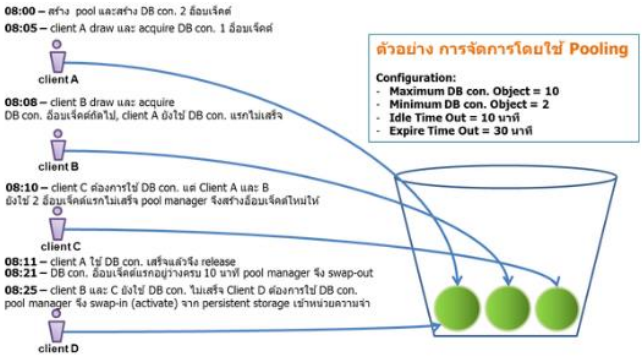
การจัดการกับสภาวะของสิ่งนั้น ๆ เช่น อ็อบเจ็คต์, session, temporary data ฯลฯ State ของสิ่งใดๆ มักต้องมีที่อยู่ (storage) ปกติมักคือหน่วยความจำ State จึงมี lifecycle และจำเป็นต้องมีการจัดการเพื่อใช้หน่วยความจำหรือ storage ให้คุ้มค่า ไม่เปลือง

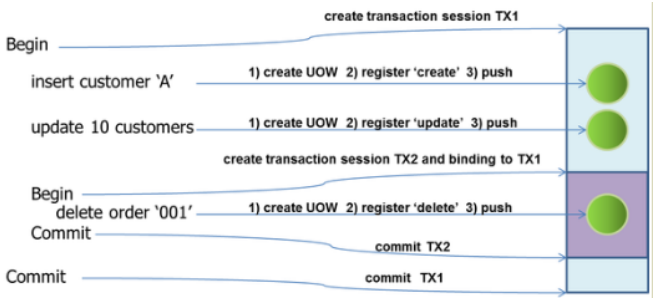
Swap-In / Activation / Object De-Serialization คือ การโหลด state กลับสู่หน่วยความจำ

Swap-Out / Passivation / Object Serialization คือ การโหลด state เก็บลง persistent storage



State Management จึงมักจำเป็นต้องสร้าง State Machine เป็นกลไกเพื่อทำหน้าที่



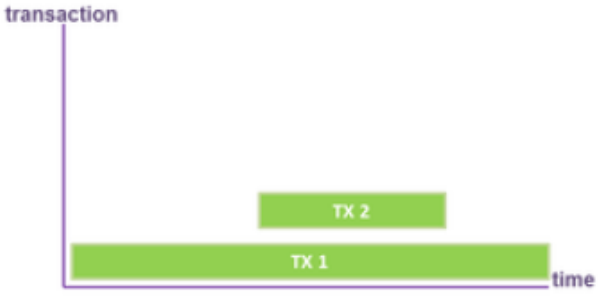


ปกติเมื่อทรานแซกชั่นเริ่มทำงาน (เมื่อ begin หรือ prepare) กลไกการจัดการทรานแซกชั่น หรือ transaction manager จะสร้าง session ขึ้นมาเรียกว่า transaction session หรือ transaction context เพื่อเก็บสถานะการทำงานและรายละเอียดของทรานแซกชั่นนั้นๆ ดังนั้นหากทรานแซกชั่นมีขนาดใหญ่และใช้เวลาประมวลผลนานกว่าจะเสร็จ ก็จะกินเนื้อที่ในหน่วยความจำมากและนาน ส่งผลเสียต่อ performance การออกแบบทรานแซกชั่นที่ดีคือ ควรทำให้ทรานแซกชั่นมีขนาดเล็กที่สุดเท่าที่จะทำได้และประมวลผลให้เสร็จเร็วที่สุดเท่าที่จะทำได้ และยิ่งทรานแซกชั่นมีการเปลี่ยน state มากเท่าไรก็ยิ่งส่งผลเสียต่อ performance เพราะต้องคอยเก็บ state ซึ่งโดยปกติจะสร้าง state ใหม่ ไม่ write ทับ state เดิม เพื่อประโยชน์ในการ rollback/undo ได้ ยกเว้นการออกแบบและจัดการบางกรณีที่ให้ write ทับ state เดิมเลย ซึ่งจะไม่สามารถ rollback/undo หรือดู state ย้อนหลังได้

- **Transaction Propagation**

การส่งต่อการจัดการทรานแซกชั่นจากต้นทางไปยังปลายทาง หรือจาก caller method ไปยัง calling method เพื่อเชื่อมโยงการจัดการทรานแซกชั่น การส่งต่ออาจเป็นแบบ

- explicit propagate (โดยการเพิ่มพารามิเตอร์)
- implicit propagate (ไม่ต้องเพิ่มพารามิเตอร์)
  - TX 1 ใช้ transaction attribute 'Required' และ TX 2 ใช้ 'Requires New'



Unit of Work, Transaction, Nested Transaction

ทรานแซกชั่นหนึ่งสามารถประกอบด้วยทรานแซกชั่นย่อยๆ ได้ หนึ่งหน่วย (atom) หรือหนึ่งบล็อกของ ทรานแซกชั่นคือ เริ่มจากตำแหน่ง begin จนถึง commit

ในหนึ่งหน่วยหรือหนึ่งทรานแซกชั่นก็คือ หนึ่ง unit of work หนึ่ง unit of work สามารถสัมพันธ์กับ operation เช่น create, update, delete, read, read only โดยต้องระบุ operation ที่เกี่ยวข้องให้ชัดเจน และองค์ประกอบอื่น เช่น Registered user, Registered date/time, Commit, Rollback

การประมวลผลทรานแซกชั่นก็คือ ทาทีละ unit of work หากเกิดข้อผิดพลาดก็ rollback ทีละ unit of work แต่ละ unit of work จัดเก็บอยู่ในโครงสร้างข้อมูลแบบ stack (ในทางปฏิบัติหรือระบบจัดการจริงอาจไม่ใช้ stack ก็ได้ แต่ประยุกต์อัลกอริธึมแบบ stack คือ ล่าสุดออกก่อน) แต่ละ unit of work ก็คือ ทรานแซกชั่นเซสชั่นนั่นเอง

- **Simultaneous Access**

Simultaneous access คือ การเข้าถึง resource เดียวกันพร้อมกันจากหลายทรานแซกชั่น ทาให้เกิดการช่วงชิงกันเข้าใช้ resource หรือเรียกว่า resource contention ทำให้ต้องมีการตัดสินว่าใครจะได้เข้าใช้ resource หรือเรียกว่า resource arbitration การเข้าถึง resource เดียวกันพร้อมกันคราวละมากกว่า 1 ทรานแซกชั่นสามารถทำได้ การจัดการสลับหรือเข้าจังหวะเพื่อเข้าใช้ resource เรียกว่า synchronization

หากต้องการจัดการการช่วงชิงกันเข้าใช้ resource และต้องการจัดการความสอดคล้องและความถูกต้องของข้อมูล จึงต้องล็อก (lock) ข้อมูลเพื่อป้องกันความผิดพลาด โดยการล็อกมีหลายระดับ และมีหลายประเภท เช่น

- จำแนกตาม Architectural Pattern:
- Optimistic Offline Lock คือ การล็อกแบบในแง่ดี – ใครยืนยันทรานแซกชั่นหรือส่ง commit ก่อนได้ล็อกข้อมูล, การล็อกเกิดขึ้นเมื่อ confirm request จากใครก็ตามมาถึง ขณะที่ตอนใช้ resource เดียวกันกันโดยหลายทรานแซกชั่นจะยังไม่มีการล็อกเกิดขึ้น
- Pessimistic Offline Lock คือ การล็อกแบบในแง่ร้าย – ใครล็อกข้อมูลก่อนได้ commit, การล็อกเกิดขึ้นเมื่อทรานแซกชั่นแรกที่เข้าถึงข้อมูลก่อนเพื่อต้องการอัพเดต
- Coarse-Grained Lock คือ การล็อกข้อมูลเป็นกลุ่ม
- จำแนกตาม Isolation Level

Isolation Level	Dirty Read	Unrepeatable Read	Phantom Read
Read Uncommitted	Yes	Yes	Yes
Read Committed	No	Yes	Yes
Repeatable Read	No	No	Yes
Serializable	No	No	No

ตัวอย่างการเซ็ต Isolation Lock ใน java.sql.Connection ในภาษา Java

Field Summary	
static int	**TRANSACTION_NONE** A constant indicating that transactions are not supported.
static int	**TRANSACTION_READ_COMMITTED** A constant indicating that dirty reads are prevented; non-repeatable reads and phantom reads can occur.
static int	**TRANSACTION_READ_UNCOMMITTED** A constant indicating that dirty reads, non-repeatable reads and phantom reads can occur.
static int	**TRANSACTION_REPEATABLE_READ** A constant indicating that dirty reads and non-repeatable reads are prevented; phantom reads can occur.
static int	**TRANSACTION_SERIALIZABLE** A constant indicating that dirty reads, non-repeatable reads and phantom reads are prevented.

setTransactionIsolation

```
void setTransactionIsolation(int level)
                             throws SQLException
```

Attempts to change the transaction isolation level for this `Connection` object to the one given. The constants defined in the interface `Connection` are the possible transaction isolation levels.

Note: If this method is called during a transaction, the result is implementation-defined.

Parameters:

`level` - one of the following `Connection` constants:
`Connection.TRANSACTION_READ_UNCOMMITTED`,
`Connection.TRANSACTION_READ_COMMITTED`,
`Connection.TRANSACTION_REPEATABLE_READ`, or
`Connection.TRANSACTION_SERIALIZABLE`. (Note that
`Connection.TRANSACTION_NONE` cannot be used because it specifies that transactions are not supported.)

Throws:

`SQLException` - if a database access error occurs or the given parameter is not one of the `Connection` constants

See Also:

`DatabaseMetaData.supportsTransactionIsolationLevel(int)`, `getTransactionIsolation()`

ในกลไกการจัดการทรานแซกชั่นหรือภาษาโปรแกรมหรือไดรเวอร์โดยทั่วไป มักใช้ default isolation คือ Repeatable Read บางตัวก็ใช้ Read Committed ตัวที่เกิดโอเวอร์เฮดสูงสุดแต่ชัวร์สุดคือ Serializable เพราะเข้าใช้คนเดียวเลย ซึ่งอาจทาให้เกิดคอขวดได้สูงมาก ส่วนตัวที่อันตรายที่สุดคือ Read Uncommitted แต่ถึงอันตรายก็ไม่ได้หมายความว่าไม่ควรใช้ เพราะจะใช้ isolation level แบบไหนนั้นขึ้นกับลักษณะงาน และควรออกแบบและเลือกให้สอดคล้องกัน business process กับ business rule

Scheduling Strategies for Resource Arbitration

การประมวลผลหลายงานพร้อมๆ กันเกิดขึ้นเร็วมาก เพราะซีพียูประมวลผลรวดเร็วมาก ซึ่งเป็นการใช้การเข้าจังหวะ (synchronization) ในลักษณะการสลับกันเข้า-ออก

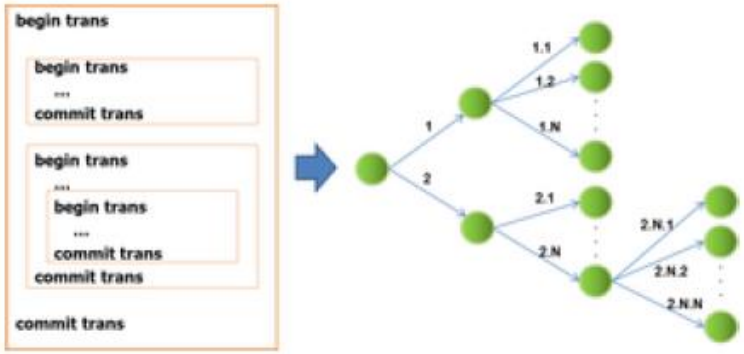
- เอางานเข้ามาใช้ทรัพยากร (swap-in)
- เอางานออกจากการใช้ทรัพยากร (swap-out)
- เลือกเอางานใหม่ที่จะเอาเข้ามาใช้ทรัพยากร (scheduling)

การ swap-in และ swap-out เป็นการใช้ I/O ยิ่งมีการสลับถี่ยิ่งใช้ I/O สูง Synchronization จึงเป็นการเข้าจังหวะของ swap-in, swap-out และ scheduling ให้เหมาะสมลงตัว มิฉะนั้นอาจเกิดการชนกัน หรือการแย่งกันหรือเกี่ยงกันเข้า ซึ่งอาจนาไปสู่การเกิด dead-lock โดยปกติจะไม่มีการให้ตั้งแต่ 2 งานขึ้นไปเข้าถึงทรัพยากรเดียวกันพร้อมกันได้

Scheduling Strategies หรือ Scheduling Algorithms เช่น First-In / First-Out หรือ Queue, Last-In / Last-Out หรือ Stack, Least Recently Use, Most Frequently Use, Semaphore, Round-Robin ฯลฯ

Transaction Orchestration

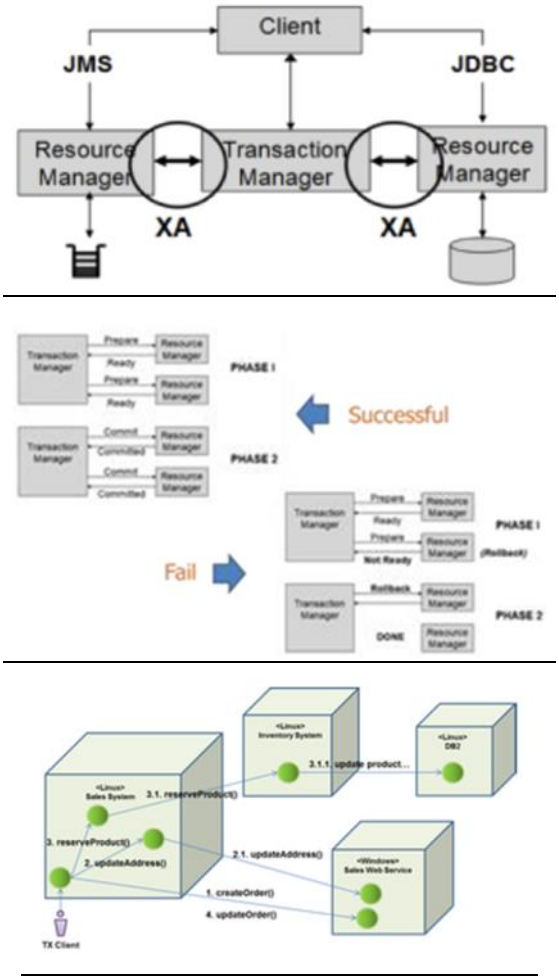
การควบคุมลำดับหรือ flow ของทรานแซกชั่นในการทางานแต่ละขั้นตอน มักใช้ Transaction Script Pattern [PoEAA] ในการจัดการ ซึ่งเป็นเสมือนเลเยอร์บางๆ ใน architecture ทาหน้าที่ควบคุมทรานแซกชั่น บางที่เรียกว่า Transaction Orchestrator ซึ่งสิ่งสำคัญคือไม่ควรให้เข้าถึง resource เองโดยตรง เทคนิคนี้เป็นการออกแบบในแนว functional design ไม่ใช่ OO design เพราะการจัดการทรานแซกชั่นบางครั้งก็ต้องการ performance มากๆ ซึ่ง OO มี performance ที่แย่ เพราะ OO เน้น modifiability, reusability มากกว่าเน้น performance ดังนั้นในบางจุดของระบบเราต้องตัดสินใจว่าจะใช้ design principle แบบไหน จะใช้แบบเดียวทั้งระบบก็ได้ หรือบางจุดใช้แบบ functional บางจุดใช้แบบ OO ดังนั้นจึงควรใช้ให้เหมาะสม ข้อผิดพลาดที่พบได้บ่อยครั้งโดยเฉพาะกับทีมที่ใช้ OO และภาษาโปรแกรมแบบ OO คือมักใช้แนวคิดการออกแบบและเขียนโปรแกรมแบบ OO ทั้งระบบ ซึ่งไม่จาเป็นครับ ต้องเลือกให้เหมาะสมดีที่สุด ตรงไหนเน้น OO มาก ตรงไหนเน้น OO น้อย ตรงไหนใช้แนว functional ไปเลย เช่น ในระบบขนาดใหญ่อย่าง enterprise system ในจุดที่มีการทำ transaction orchestration หนักๆ จึงควรใช้แนว functional ผสม OO



Local & Distributed Transaction Processing

- Local Transaction Processing คือ การจัดการทรานแซกชั่นโดย resource system ที่เข้าใช้
- Distributed Transaction Processing คือ การจัดการทรานแซกชั่น ซึ่งในแต่ละขั้นตอนอาจประกอบด้วยทรานแซกชั่นย่อย หรือ operation ที่ต้องเข้าใช้ resource system ที่แตกต่างกัน ทาให้ทรานแซกชั่นใหญ่ที่อยู่นอกสุดต้องจัดการและควบคุมการประมวลผลทรานแซกชั่นใน resource system ต่างๆ ในอีกนัยหนึ่งก็คือ ทรานแซกชั่นที่ประกอบด้วย local transaction processing หลายส่วนที่กระจายไปยัง resource system ที่แตกต่างกัน โดยทรานแซกชั่นกลางที่อยู่นอกสุดต้องประสานงานเพื่อควบคุมความถูกต้องและความสอดคล้องของข้อมูลได้

Distributed Transaction Processing บางทีเรียกว่า Two Phase Commit โดย driver ที่ใช้เชื่อมต่อกับ resource system มักเรียกว่า Two Phase Commit Driver หรือ XA Driver



ตัวอย่าง Distributed Transaction Processing

Transaction Model

Local Transaction Model คือ การจัดการทรานแซกชั่นเกิดขึ้นภายใน resource system ที่เข้าใช้ เช่น database server โดยไม่ได้จัดการที่ระดับแอพพลิเคชั่น หรือกล่าวอีกทางหนึ่งคือเป็นการจัดการที่ระดับ connection ที่เชื่อมต่อไปยัง resource system นั้นๆ เช่น

```
public void updateTradeOrder(TradeOrderData order)
   throws Exception {
   DataSource ds = (DataSource)
       (new InitialContext()).lookup("jdbc/MasterDS");
   Connection conn = ds.getConnection();
   conn.setAutoCommit(false);
   OrderDAO orderDao = new OrderDAO();
   TradeDAO tradeDao = new TradeDAO();
   try {
      //SQL and Connection Logic in DAO Classes
      orderDao.update(order, conn);
      tradeDao.update(order, conn);
      conn.commit();
   } catch (Exception e) {
      logger.fatal(e);
      conn.rollback();
      throw e;
   } finally {
      conn.close();
   }
}
```

Programmatic Transaction Model คือ การจัดการทรานแซกชั่นที่ระดับแอพพลิเคชั่น ด้วยการจัดการในระดับซอร์สโค้ดโดยนักพัฒนา ซึ่งเป็นการจัดการทรานแซกชั่นจริงๆ ไม่ใช่เน้นจัดการ connection ในแบบ local transaction model

```
public void updateTradeOrder(TradeOrderData order)
   throws Exception {
   UserTransaction txn = sessionCtx.getUserTransaction();
   txn.begin();
   try {
      TradeOrderDAO dao = new TradeOrderDAO();
      dao.updateTradeOrder(order);
      txn.commit();
   } catch (ApplicationException e) {
      log.fatal(e);
      txn.rollback();
      throw e;
   }
}
```

Declarative Transaction Model คือ การจัดการทรานแซกชั่นแบบไม่ต้องเขียนโปรแกรมควบคุม ทำโดยการคอนฟิก หรือจัดการผ่านตัวกลาง

- ข้อดี
  - ปรับเปลี่ยนรูปแบบการจัดการทรานแซกชั่นสะดวก
  - ไม่ต้องแก้โค้ด
- ข้อเสีย
  - ต้องมีความเข้าใจการจัดการทรานแซกชั่นในแบบต่างๆ ที่ผ่านการคอนฟิก
  - transaction attribute อาจมีความซับซ้อนมากได้
  - การจัดการขึ้นกับ Transaction API และ middleware ที่ใช้

Transaction Attribute

- Required คือ ต้องประมวลผลภายใต้การจัดการทรานแซกชั่น, ถ้าไม่มีจะสร้างเอง
- Not Required คือ ไม่ต้องการประมวลผลภายใต้การจัดการทรานแซกชั่น, ถ้ามีจะไม่สนใจ จะประมวลผลของตัวเอง โดยไม่มีการจัดการทรานแซกชั่น
- Requires New คือ ต้องการประมวลผลภายใต้การจัดการทรานแซกชั่น โดยต้องการจัดการเอง, จะสร้างใหม่สาหรับของตัวเอง
- Mandatory คือ ต้องการประมวลผลภายใต้การจัดการทรานแซกชั่นเสมอ, ถ้าไม่มีจะแจ้ง exception
- Support คือ รองรับการประมวลผลภายใต้การจัดการทรานแซกชั่น, ถ้ามีอยู่แล้วก็จะไปใช้ด้วย
- Not Support คือ ไม่รองรับการประมวลผลภายใต้การจัดการทรานแซกชั่น, ถ้ามีจะแจ้ง exception
- Read Only คือ การอ่านข้อมูลอย่างเดียว ซึ่งช่วยประหยัดการจัดการทรานแซกชั่น

Transaction Analysis

การออกแบบทรานแซกชั่นควรเริ่มต้นด้วยการวิเคราะห์ก่อน ซึ่งเราสามารถเริ่มวิเคราะห์ทรานแซกชั่นได้เร็วมากตั้งแต่เฟส requirement หรือช่วงต้นๆ ของโครงการเลยทีเดียว นั่นหมายถึง ไม่ต้องรอให้เก็บ requirement ให้จบ หรือในระบบขนาดใหญ่อย่าง enterprise system หรือ business system ที่ซับซ้อน ก่อนเริ่มทา requirement ควรทา business process modeling & analysis ก่อน นั่นคือ ทำความเข้าใจกับ business process และรายละเอียดต่างๆ ที่เกี่ยวข้องก่อน ในขณะที่วิเคราะห์ทำความเข้าใจ business process ก็ควรวิเคราะห์ทำความเข้าใจกับ business transaction ด้วย

เราอาจแบ่งทรานแซกชั่นออกเป็น 2 ขอบเขตใหญ่ๆ ได้แก่

- **Business transaction**

  นับตั้งแต่กระบวนการเริ่มต้น อาทิ ในระบบโอนเงินออนไลน์ เริ่มต้นจากผู้ใช้เลือกเมนู เลือกบัญชีต้นทาง ระบบก็จะดึงข้อมูลเงินคงเหลือมาแสดง เลือกบัญชีปลายทาง ระบบก็จะตรวจสอบธนาคารและบัญชีปลายทาง จากนั้นผู้ใช้ก็ป้อนจานวนเงินที่ต้องการโอน แล้วก็ยืนยันการโอน ระบบก็จะส่งรหัส OTP มาให้ทางโทรศัพท์มือถือ แล้วจึงป้อนเพื่อยืนยันทางหน้าจอ จึงจบกระบวนการ จากตัวอย่าง จะเห็นว่ามีการส่ง request และรับ response ไปกลับกันหลายรอบ แสดงว่า business transaction นี้เป็นแบบ stateful transaction โดย business transaction เริ่มต้นที่การเริ่มใช้แบบฟอร์ม และจบลงด้วยการตรวจสอบรหัส OTP

- **System transaction**

  เกิดขึ้นเมื่อระบบรับ OTP ที่ป้อนโดยผู้ใช้ และเริ่ม begin transaction จริงๆ จังหวะนี้ และจบการทางานทันทีภายใน request เดียว ไม่ต้องมีการรับ/ส่ง request/response กันหลายรอบ แสดงว่า system transaction เป็นแบบ stateless transaction แต่จังหวะ business transaction อาจมีการพักข้อมูลชั่วคราวที่ระดับ user session ใน web application ก็ได้

การวิเคราะห์ทรานแซกชั่นจึงไม่ควรเริ่มต้นที่ system transaction เพราะจะทาให้ไม่เข้าใจที่มาที่ไปอย่างละเอียดครบถ้วนจริงๆ และการวิเคราะห์และออกแบบทรานแซกชั่นจาเป็นต้องวิเคราะห์และทาความเข้าใจ business process กับ business rule เสมอ (แต่มักพบว่าข้อมูลเหล่านี้มักมาไม่ถึงมือคนออกแบบและโปรแกรมเมอร์ ส่วน tester ก็ต้องทราบข้อมูลเหล่านี้ด้วยเช่นกัน เพื่อประโยชน์ต่อการออกแบบ test scenario/test case)

หากสังเกตให้ดีจะพบว่าเราสามารถวิเคราะห์และออกแบบทรานแซกชั่นได้ตั้งแต่ต้นๆ โครงการกันเลยทีเดียว วางจังหวะตรงไหนจะ begin-commit-rollback business transaction จังหวะไหนจะ begin-commit-rollback system transaction หรือพูดง่ายๆ คือ วิเคราะห์ business process กับ requirement กันไป ก็วิเคราะห์และออกแบบทรานแซกชั่นเบื้องต้นพร้อมกันไปได้เลย หากติดขัดอะไรตรงไหนก็ปรับเปลี่ยนกันตั้งแต่เนิ่นๆ ได้ทันท่วงที ทาให้บริหารความเสี่ยงได้ดี นอกจากนี้ยังควรพิจารณาเรื่อง topology ของฮาร์ดแวร์และเน็ตเวิร์กประกอบด้วย ว่าใช้เครื่องกี่เครื่อง แต่ละเครื่องทำอะไรบ้าง แบ่งสถาปัตยกรรมระบบเป็นกี่เลเยอร์ และแบ่ง tier เป็นกี่ tier เป็นต้น เพราะมีผลต่อ performance มากๆ และอย่าโยกการวิเคราะห์และออกแบบทรานแซกชั่น โดยเฉพาะจังหวะ system transaction ไปให้โปรแกรมเมอร์ทำนะครับ เพราะหากโปรแกรมเมอร์ไม่มีข้อมูล business process กับ business rule และไม่ชำนาญใน domain knowledge ของระบบงานนั้นๆ หรือไม่ชำนาญด้านการออกแบบทรานแซกชั่นดีพอ

และยิ่งหากวิเคราะห์และออกแบบทรานแซกชั่นด้วยการวาดรูป business process model ด้วย BPMN (Business Process Model Notation) จะยอดเยี่ยมมาก เพราะช่วยให้เข้าใจง่าย แถม modeling tool หลายยี่ห้อสามารถทำ process simulation ได้ด้วย เป็น animation วิ่งกันให้เห็นจะๆ กันไปเลย ยกตัวอย่างเช่นยี่ห้อ VisualParadigm ส่วนของฟรีก็มีเช่น BPMN (bpmn.org)

การวิเคราะห์ทรานแซกชั่นในรายละเอียดที่จะกล่าวถึงต่อไปนี้ จาเป็นต้องมีข้อมูลที่เพียบพร้อมมาประกอบการวิเคราะห์ จึงสามารถใช้หัวข้อต่างๆ เอาไปถามหรือให้คนเก็บ requirement ไปเก็บเพิ่มได้ เพราะหากเก็บมาไม่สมบูรณ์พอ ก็มีผลต่อการวิเคราะห์อย่างมาก ซึ่งปกติที่ทำๆ กันโดยส่วนมากก็ไม่ค่อยจะเก็บกันครบเท่าไร เพราะงาน requirement คือจุดอ่อนของการพัฒนาและทีมพัฒนาในองค์กรจานวนมากเลยทีเดียว และจากที่ผมพบในองค์กรจานวนมากคือ บุคลากรที่รับผิดชอบด้าน requirement มักมีความรู้และทักษะ ไม่เหมาะสมกับงานสักเท่าไหร่ เนื่องจากงาน requirement เป็นงานที่ต้องอาศัยความรู้และทักษะทั้งด้าน soft skill, เทคนิค และ การจัดการ

วิเคราะห์ Business Process & Rule

วิเคราะห์ business process ทำให้ทราบ

- Business actor
- Business worker
- Business location
- Business activity
- Business rule
- ทำให้เข้าใจขั้นตอนการปฏิบัติงานในมุมมองของผู้ใช้ยิ่งขึ้น
- ฯลฯ

การวิเคราะห์ business rule ทำให้เข้าใจเงื่อนไขหรือตรรกะทางธุรกิจ ซึ่งมีผลต่อการประมวลผล

วิเคราะห์พฤติกรรมไคลเอนต์ (Client Behavior)

หากไม่มี event เกิดขึ้น ระบบก็จะไม่เกิดการประมวลผล โดย Event เกิดจาก client ซึ่ง event มีหลายประเภท เช่น periodic, sporadic, stochastic

การทำความเข้าใจกับ event สามารถใช้การวิเคราะห์พฤติกรรมของ client ได้ โดยต้องดูว่า client มีพฤติกรรมในการใช้ระบบอย่างไร ลักษณะใด นอกจากควรทำความเข้าใจกับพฤติกรรมของ client ควรพิจารณาปัจจัยอื่น เช่น สถานที่ของ client หรือ client location, ประเภทของ client, ปริมาณของ client, client device (เช่น พีซี, โทรศัพท์มือถือ, ตู้ kiosk ฯลฯ), บทบาท หรือ role ของ client, ความต้องการ หรือ concern ของ client, ฯลฯ

Actor/Worker	Action	Time	Rate (เฉลี่ย)
ลูกค้าที่เป็นสมาชิก			20,000 คน
ลูกค้าที่เป็นสมาชิก	ดูสินค้า	08:00 – 11:30	8,000 คน/ช.ม.
		11:30 – 14:00	5,000 คน/ช.ม.
		14:00 – 20:00	7,000 คน/ช.ม.
		20:00 – 08:00	3,500 คน/ช.ม.
ลูกค้าที่เป็นสมาชิก	ซื้อสินค้า (ต้อง login ก่อน check out)	08:00 – 11:30	3,500 คน/ช.ม.
		11:30 – 14:00	1,500 คน/ช.ม.
		14:00 – 20:00	1,800 คน/ช.ม.
		20:00 – 08:00	750 คน/ช.ม.
บุคคลทั่วไป	ดูสินค้า	00:00 – 24:00	15,000 คน/ช.ม.
ลูกค้าที่เป็นสมาชิก และ บุคคลทั่วไป	ชำระค่าโทรศัพท์มือถือ (DTAC)	00:00 – 24:00	2,000 คน/ช.ม.
	ชำระค่าโทรศัพท์มือถือ (True Move)	00:00 – 24:00	1,000 คน/ช.ม.
	ชำระค่าโทรศัพท์มือถือ (AIS)	00:00 – 24:00	3,500 คน/ช.ม.

Actor/Worker	Action	Time	Rate (เฉลี่ย)
พนักงานขาย ประจำจุดขาย			2 คน/จุดขาย, มีทั้งหมด 200 จุดทั่วประเทศ
พนักงานขาย ประจำจุดขาย	ดูสินค้า	10:00 – 21:00	20 ครั้ง/คน/จุดขาย/ช.ม.
	ซื้อสินค้า (รับชำระสินค้า)	10:00 – 21:00	15 ครั้ง/คน/จุดขาย/ช.ม.
ระบบ CRM	ดึงข้อมูลพฤติกรรมผู้ใช้ เฉพาะลูกค้าที่เป็นสมาชิก และ บุคคลทั่วไป เฉพาะการดูสินค้า และ ซื้อสินค้า	01:00	1 ครั้ง/วัน

ตัวอย่าง client behavior ซึ่งในความเป็นจริงอาจเก็บให้ละเอียดมาก/น้อยกว่านี้ก็ได้

<u>วิเคราะห์ Service</u>

ต้องระบุรายละเอียดเกี่ยวกับเซอร์วิสที่เกี่ยวข้องกับทรานแซกชั่น เช่น มีเซอร์วิสประเภทใดบ้าง, พฤติกรรมของเซอร์วิสและการให้บริการมีลักษณะอย่างไร, ใครเป็น client ของเซอร์วิส, เซอร์วิสไปเรียกใช้ใครต่อ มี coupling/dependency กับใครบ้าง หรือกับ resource ใดบ้าง, เซอร์วิสอยู่บนเครื่องเดียวกัน หรือกระจายตัวไปอยู่ตรงไหนบ้าง และเมื่อทราบว่ามีการใช้เซอร์วิสอะไรบ้าง จะได้วิเคราะห์ต่อว่าทรานแซกชั่นจะไปเรียกใช้เซอร์วิสเหล่านั้นอย่างไร

ตัวอย่าง เซอร์วิสโดยทั่วไป เช่น

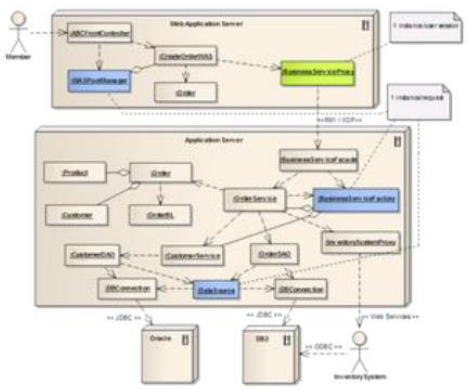
- User Screen Service
- Web Application Service
- Web Service
- Business Service
- Domain Service : Business Logic, Data Logic
- Resource Service เช่น database service, message queue service, SMS service, email service, directory service
- Architectural Mechanism Service / Common Service / Non-Functionality เช่น
- Proxy, Façade, Front Controller, Filter, Adapter, Exception Handling, Authentication, Authorization, Logging, Transaction Management, I/O, Network, Distribution, Data Exchange ฯลฯ

<u>วิเคราะห์ Resource System & Connection</u>

ระบุว่ารายละเอียดของ resource ที่ระบบและทรานแซกชั่นต้องใช้ เช่น มี resource อะไรบ้าง, มี resource ประเภทใดบ้าง, resource อยู่ที่ไหนบ้าง, ระบบมีพฤติกรรมการใช้ resource อย่างไร, Resource มีพฤติกรรมอย่างไร ลักษณะใด, constraint เกี่ยวกับ resource มีหรือไม่ อะไรบ้าง, ขอบเขตของอานาจของระบบหรือคอมโพเน้นต์ ในการเข้าใช้ resource มีแค่ไหน(Autonomy for Resource Access) นอกจากนี้ยังต้องทราบค่าคอนฟิกของ resource นั้นๆ ด้วย เพื่อจะได้ออกแบบการเรียกใช้ resource ให้มีประสิทธิภาพที่สุด หลักในการจำง่ายๆ คือ acquire resource ให้เร็วที่สุด, ถ้าจะล็อกก็ต้องรีบล็อกให้เร็ว, ใช้ให้เสร็จเร็วๆ, ปลดล็อกเร็วๆ ยิ่งถ้าเป็น

Shared resource ยิ่งต้องออกแบบการเรียกใช้ให้ดียิ่งขึ้น เพื่อป้องกันการเกิด latency time (เวลาหน่วง) ซึ่งมีผลต่อ response time ทาให้ช้าได้

<u>ตัวอย่าง 1</u>



ตัวอย่าง 2

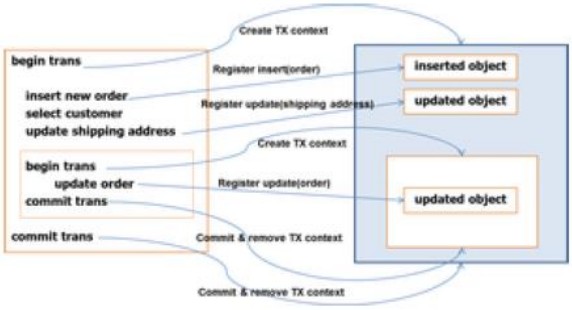
Resource	Type	Scope	Connection
ABCFrontController	Stateless Object	1 instance/thread/request	HTTP, Pointer
CreateOrderWAS	Stateless Object	1 instance/thread/request	Pointer
BusinessServiceProxy	Stateful Object	1 instance/user session	Pointer, RMI/IIOP
BusinessServiceFacade	Stateless Object	1 instance/thread/request	RMI/IIOP, Pointer
BusinessServiceFactory	**Stateless Object**	**1 instance/system**	**Pointer**
DataSource	**Stateless Object**	**1 instance/system**	**Pointer**
DBConnection	Reusable Database connection	is used/leased at a time, is released when finish using, is closed/remove when expired	Pointer, JDBC
Oracle	Database server	24/7	JDBC
InventorySystemProxy	Stateful Object	1 instance/user session	Pointer, Web Services
Inventory System	System	24/7	Web Services

วิเคราะห์ Transaction Data & Data Transformation

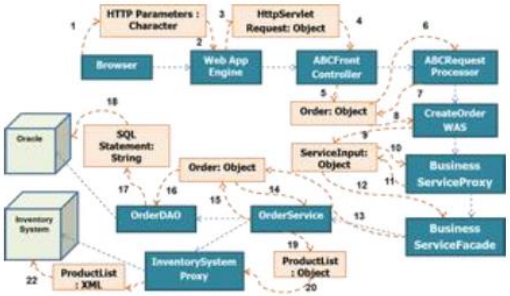
การวิเคราะห์ transaction data / process data ทาให้ทราบ เช่น มีการใช้ข้อมูลอะไร, ชนิด/ฟอร์แมตอะไร มีลำดับการใช้ข้อมูลอย่างไร, มี data granularity อย่างไร, ระบุการใช้ transaction session และ unit of work วิเคราะห์ data transformation ทาให้ทราบ เช่น การ convert/exchange ข้อมูล, จุดที่มีการ convert/exchange ข้อมูล, การไหลของข้อมูล

ประโยชน์ของการวิเคราะห์ transaction data กับ data transformation เพื่อช่วยให้จัดการ performance ของการประมวลผลทรานแซกชั่น เพราะ transaction data ถ้ามีขนาดใหญ่หรือมีจานวนมาก ก็จะเปลืองหน่วยความจาและ I/O รวมถึงเน็ตเวิร์ก ส่วน data transformation ถ้ามีมากก็จะเกิดโอเวอร์เฮดในการประมวลผลซึ่งเปลืองซีพียู และถ้าข้อมูลใหญ่ก็เปลืองเหมือน transaction data การเลือกใช้ฟอร์แมตหรือ data type อะไรก็ต้องเลือกให้เหมาะสม โปรแกรมเมอร์หรือคนออกแบบสมัยนี้ชอบใช้อะไรที่มันง่ายต่อการเขียนโปรแกรม ซึ่งมักง่ายไม่ดีๆ ควรเลือกให้เหมาะสมดีกว่า

ตัวอย่าง 1



ตัวอย่าง 2



ในตัวอย่างที่ 2 Browser ส่งข้อมูลออกมาเป็นพารามิเตอร์อยู่ในเฮดเดอร์ของโปรโตคอล HTTP จากนั้น Web App Engine ก็ครอบ type ด้วยการใช้ HttpServletRequest ลักษณะครอบ type หรือ encapsulate type แบบนี้ไม่ค่อยเปลือง performance เท่าไร จากนั้นก็ส่งไปให้ ABCFrontController ทาการดึงข้อมูลออกมา convert เป็นอ็อบเจ็คต์รวมถึงมีการแปลง data type ให้กับฟิลด์ต่างๆ จังหวะเริ่มเปลือง performance แล้ว จากนั้นก็ส่งไปให้ ABCRequestProcessor เพื่อเลือกว่าจะส่งให้ WAS ตัวไหน จากนั้นเลือกได้ก็ส่งไปให้กับ CreateOrderWAS เพื่อทำ application logic จากนั้นก็ครอบ type ด้วยการเอา ServiceInput มาครอบ แล้วส่งไปให้ BusinessServiceProxy เพื่อส่ง ServiceInput ออกไปนอกเครื่องผ่านเน็ตเวิร์กไปให้กับ BusinessServiceFacade ที่ทำหน้าที่ถอดอ็อบเจ็คต์จาก ServiceInput ออกมา แล้วส่งอ็อบเจ็คต์ Order ไปให้ OrderService เพื่อทำ business logic และ ทรานแซกชั่น จากนั้นก็ส่งอ็อบเจ็คต์ Order ไปให้ OrderDAO เพื่อดึงฟิลด์ต่างๆ มาสร้างเป็นคำสั่ง SQL ที่เป็น String แล้วส่งผ่าน database driver ไปให้ Oracle ประมวลผล จากนั้น call ก็จะกลับมาที่ OrderService เพื่อทำขั้นตอนต่อไปโดยถอดอ็อบเจ็คต์ ProductList ออกมาจากอ็อบเจ็คต์ Order เพื่อส่งให้ InventorySystemProxy เพื่อนำไป convert เป็น XML ซึ่งมีโอเวอร์เฮดพอสมควรหากข้อมูลมีขนาดใหญ่ จากนั้นจึงส่งไปให้กับระบบ InventorySystem

จากรูปในตัวอย่างข้างบน จุดที่น่าสนใจที่เราควรฝึกตั้งข้อสันนิษฐานว่าน่าจะมีผลต่อ performance ซึ่งมีอยู่ 3 จุด ได้แก่ จังหวะสร้างอ็อบเจ็คต์ Order, จังหวะสร้างคำสั่ง SQL และจังหวะแปลงอ็อบเจ็คต์ ProductList เป็น XML แล้วเราก็เอาข้อสันนิษฐานนี้ไปสร้าง test case เพื่อทดสอบ ถ้าผลออกมาไม่ดีก็ทำการปรับเปลี่ยน เพื่อให้ได้ผลที่เหมาะสม

<u>สร้าง Scenario เพื่ออธิบาย Transaction Concern</u>

วิเคราะห์ความต้องการด้านต่างๆ และ concern ของ stakeholder เพื่อระบุจุดที่น่าสนใจที่มีผลต่อการจัดการทรานแซกชั่น โดยพิจารณาจากความเสี่ยง, performance, และการเข้าใช้ resource

วิเคราะห์และอธิบายรายละเอียดด้วยการจาลองสถานการณ์ (scenario) ซึ่งประกอบด้วยคุณสมบัติ เช่น

- Transaction name
- Relevant use cases
- Relevant non-functional requirements
- Relevant business process and rules
- Event source and event behavior
- Process Flow
- Transaction scope within process
- Concerned resources

อาจมีคุณสมบัติอื่นเพิ่มเติมอีกได้ Non-functional requirements หรือ quality attributes ที่มักมีผลต่อทรานแซกชั่น เช่น

- performance, reliability, availability, modifiability

➢ interoperability, integrability, configurability, scalability, testability, security

ตัวอย่าง 1

Attributes	Description
Transaction name	ดูสินค้า
Relevant use cases	View Product
Relevant non-functional requirements	Performance, reliability, availability
Event source and event behavior	ลูกค้าที่เป็นสมาชิก, ในช่วงเวลา 08:00 – 11:30
Process Flow	ABCFrontController -> ViewProductWAS -> ... -> Oracle
Transaction scope within process	ViewProductService -> Oracle
Concerned resources	• Object: ProductCatalog, Product, ProductDAO, BusinessServiceProxy, BusienssServiceFacade, ViewProductService, DataSource, DBConnectionPoolManager, DBConnectionPool, DBConnection • Others: network, Oracle

ตัวอย่าง 2

Attributes	Description
Transaction name	ซื้อสินค้า
Relevant use cases	Check Out
Relevant non-functional requirements	Performance, reliability, availability, scalability, configurability
Event source and event behavior	ลูกค้าที่เป็นสมาชิก, ในช่วงเวลา 08:00 – 11:30
Process Flow	ABCFrontController -> CreateOrderWAS -> ... -> Oracle -> InventorySystem -> Oracle
Transaction scope within process	OrderService -> Oracle -> InventorySystem -> Oracle
Concerned resources	• Object: OrderService, DataSource, DBConnectionPoolManager, DBConnectionPool, DBConnection, InventorySystemProxy • Others: network, Oracle, InventorySystem

ตัวอย่าง 3

Attributes	Description
Transaction name	รายการสินค้าที่ถูก check out โดยสมาชิกมากกว่า 1 คน มีรายการสินค้าชิ้นเดียวกันตรงกัน
Relevant use cases	Check Out
Relevant non-functional requirements	Performance, reliability, availability, scalability, configurability
Business process and rules	• Receive order process • rules: สร้างใบสั่งซื้อเปล่า -> กำหนดที่อยู่จัดส่ง -> ล็อกสินค้าตามรายการสั่งซื้อ -> อัพเดตใบสั่งซื้อ • request ของใครถึงข้อมูลสินค้าก่อนก็ได้ไป(ได้ซื้อ / ล็อกก่อน)
Event source and event behavior	ลูกค้าที่เป็นสมาชิกมากกว่า 1 คนเข้ามาซื้อพร้อมกัน, ในเวลาเดียวกัน ภายในช่วงเวลา 08:00 – 11:30
Transaction scope within process	• ควบคุมทรานแซกชั่นที่ OrderService • Locking type ของ InventorySystem ในส่วนสถานะสินค้า(ปริมาณสินค้าคงเหลือ) คือ pessimistic lock, row lock, exclusive lock mode กรณีเขียน, read committed • Scheduling strategy ของ InventorySystem ในส่วนที่ใช้กับเร็กคอร์ดของตารางฐานข้อมูลที่เก็บข้อมูลสถานะสินค้า(ปริมาณสินค้าคงเหลือ) คือ queue • InventorySystem: หากอัพเดตสินค้าสำเร็จจะส่งค่า 1, ไม่สำเร็จส่ง 0 กลับมา
Concerned resources	• Object: InventorySystemProxy • Others: network, InventorySystem, DB2 (ที่ InventorySystem ใช้)

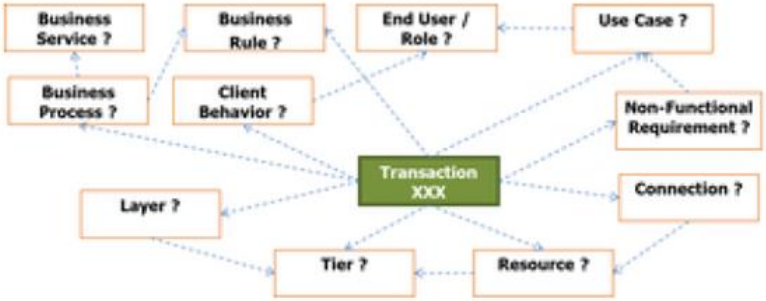
การสร้าง transaction scenario ช่วยให้อธิบายรายละเอียดทรานแซกชั่นได้ชัดเจนขึ้น ช่วยให้ผู้ใช้ลูกค้า ผู้ออกแบบ และทีมงานเข้าใจตรงกันยิ่งขึ้น สาหรับการวิเคราะห์และออกแบบทรานแซกชั่นสามารถทำได้หลายจุดใน development process เช่น ออกแบบไปแล้วก็กลับมาวิเคราะห์ใหม่ได้เพื่อปรับปรุง หรือเขียนโปรแกรมไป

แล้วทดสอบเบื้องต้นแล้ว ก็กลับมาแก้งานออกแบบใหม่ได้ เป็นต้น ซึ่งเป็นหลักของ Iterative Process นั่นเอง หรือจะเรียกว่าเป็นการทา transaction refactoring ก็ได้ ทามันอยู่เรื่อยๆ วนไปวนมา optimize ไปเรื่อยๆ จนกว่าทดสอบแล้วได้ค่าที่พอใจ

Transaction Traceability

การตรวจสอบย้อนกลับ เมื่อนามาประยุกต์กับการออกแบบและจัดการทรานแซกชั่น ทำให้ได้ประโยชน์หลายอย่าง เช่น

- การตัดสินใจในการควบคุมทรานแซกชั่นว่าจะจัดการอย่างไรจึงเหมาะสม
- การวิเคราะห์ผลกระทบ (impact analysis) และ การบริหารความเสี่ยง
- บริหารการเปลี่ยนแปลง (change management)
- การพิจารณาการออกแบบตามหลัก ACID



ควรทำ transaction traceability ในทุกโครงการ โดยเฉพาะในโครงการขนาดใหญ่ หรือการออกแบบระบบขนาดใหญ่ เพื่อให้ทราบว่าแต่ละทรานแซกชั่นเกี่ยวข้องกับอะไรบ้าง เวลามีอะไรเปลี่ยนแปลงเกิดขึ้นจะได้วิเคราะห์การเปลี่ยนแปลงและความเสี่ยงได้เร็ว และจะได้ปรับทรานแซกชั่นได้เร็วด้วย

<u>การออกแบบเชิงสถาปัตยกรรม</u>

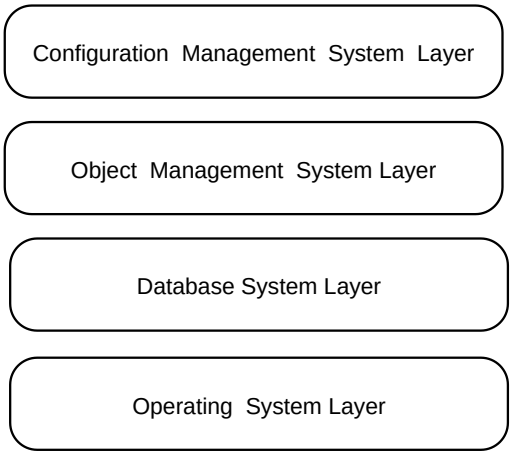
การออกแบบซอฟต์แวร์สำหรับงานวิศวกรรมซอฟต์แวร์นั้น ควรเริ่มต้นจากการออกแบบในระดับสูง คือ ในภาพรวมของซอฟต์แวร์ทั้งระบบในแต่ละด้าน ได้แก่ การจัดโครงสร้างของซอฟต์แวร์ การแบ่งส่วนซอฟต์แวร์ และการควบคุมการทำงานของส่วนประกอบย่อย รวมเรียกว่า การออกแบบเชิงสถาปัตยกรรม หลังจากออกแบบเชิงสถาปัตยกรรมแล้ว ทีมงานทำการออกแบบในรายละเอียดของแต่ละส่วนประกอบต่อไป

- **การออกแบบเชิงสถาปัตยกรรม**

การออกแบบเชิงสถาปัตยกรรม หมายถึง การกำหนดลักษณะโครงสร้างของระบบหรือซอฟต์แวร์ในมุมมองระดับบน เป็นการแสดงให้เห็นส่วนประกอบต่างๆ ของซอฟต์แวร์ภายใต้โครงสร้างสถาปัตยกรรมรูปแบบ ใดๆ กล่าวโดยสรุป การออกแบบเชิงสถาปัตยกรรม ก็คือ การเลือกรูปแบบของสถาปัตยกรรมให้กับซอฟต์แวร์

ปัจจุบันโครงสร้างสถาปัตยกรรมซอฟต์แวร์ได้ถูกคิดค้นขึ้นมาหลายรูปแบบ ไม่ว่าจะเป็นสถาปัตยกรรมแบบ Client / Server, Layer หรือ Three Tier ซึ่งต่างก็มีมุมมองการนำเสนอที่แตกต่างกันและจัดสรรให้แต่ละส่วนประกอบของซอฟต์แวร์อยู่ในตำแหน่งต่างๆ แตกต่างกันไปตามรูปแบบของโครงสร้าง ดังนั้น เพื่อให้แน่ใจว่าออกแบบส่วนอื่นๆ ของซอฟต์แวร์จะรองรับสถาปัตยกรรมรูปแบบเดียวกัน จึงจำเป็นต้องกำหนดรูปแบบโครงสร้างสถาปัตยกรรมและจัดสรรแต่ละส่วนประกอบของซอฟต์แวร์ขึ้นมาก่อน เพื่อใช้เป็นการออกแบบรายละเอียดในแต่ละส่วนของซอฟต์แวร์ต่อไปนั่นเอง

ยกตัวอย่างเช่น การออกแบบซอฟต์แวร์ระบบจัดการสินค้าคงคลัง อันดับแรกทีมงานจะต้องออกแบบเชิงสถาปัตยกรรมนั่นคือ ทีมงานต้องเลือกรูปแบบสถาปัตยกรรมให้กับซอฟต์แวร์ โดยเมื่อสรุปความคิดเห็นร่วมกันแล้วตัดสินใจเลือก รูปแบบ Layer ซึ่งมีการจัดสรรส่วนต่างๆ ของซอฟต์แวร์ ดังรูปต่อไปนี้



รูปแสดงตัวอย่างการออกแบบเชิงสถาปัตยกรรม

จากรูป เป็นเพียงตัวอย่างการออกแบบเชิงสถาปัตยกรรมเฉพาะการจัดโครงสร้างของซอฟต์แวร์ แต่การออกแบบเชิงสถาปัตยกรรมนั้น เป็นกิจกรรมที่รวมไปถึงการออกแบบอีก 2 ส่วน ได้แก่ การออกแบบวิธีการแบ่งส่วนซอฟต์แวร์ และการออกแบบวิธีการควบคุมการทำงานส่วนประกอบย่อย กล่าวได้ว่าการออกแบบเชิงสถาปัตยกรรม ประกอบไปด้วยกิจกรรมออกแบบทั้งหมด 3 ส่วน ดังรูป

รูปแสดงกิจกรรมออกแบบเชิงสถาปัตยกรรม

สิ่งที่ได้จากการออกแบบเชิงสถาปัตยกรรม คือ เอกสารการออกแบบเชิงสถาปัตยกรรมหรือรายละเอียดสถาปัตยกรรมซอฟต์แวร์นั่งเอง ซึ่งรวมถึงแบบจำลองของการออกแบบเชิงสถาปัตยกรรม ที่จะต้องแสดงให้เห็นถึงการจัดโครงสร้างความสัมพันธ์ระหว่างระบบย่อยที่รวมกันเป็นระบบใหญ่ และโครงสร้างภายในระบบย่อยที่รวมกันเป็นโมดูล โดยแบบจำลองของการออกแบบเชิงสถาปัตยกรรมที่ได้ จะนำเสนอมุมมองของสถาปัตยกรรมในแต่ละด้าน ดังนี้

1. **แบบจำลองเชิงโครงสร้าง** แสดงให้เห็นโครงสร้างของส่วนประกอบย่อยหรือคอมโพเนนท์แต่ละส่วนของซอฟต์แวร์
2. **แบบจำลองไดนามิก** แสดงการเปลี่ยนแปลงพฤติกรรมเมื่อได้รับการกระตุ้นจากเหตุการณ์ภายนอก
3. **แบบจำลองส่วนประสาน** แสดงให้เห็นบริการของแต่ละส่วนประกอบย่อยผ่าน Public Interface
4. **แบบจำลองแสดงความสัมพันธ์** แสดงความสัมพันธ์ระหว่างส่วนประกอบย่อย เช่น Data Flow
5. **แบบจำลองแสดงการกระจาย** แสดงให้เห็นกระจายส่วนประกอบย่อย ไปทำงานบนคอมพิวเตอร์แต่ละส่วน

การจัดโครงสร้างของซอฟต์แวร์

การจัดโครงสร้างของซอฟต์แวร์ หรือ “การจัดโครงสร้างของระบบ” เป็นส่วนแรกที่ต้องออกแบบเพื่อแสดงให้เห็นโครงสร้างภายนอกของซอฟต์แวร์ เนื่องจากอาจส่งต่อโครงสร้างภายในของซอฟต์แวร์ ซึ่งถูกออกแบบให้สอดคล้องกับโครงสร้างภายนอก (แต่ไม่เสมอไป) หากทีมงานออกแบบโครงสร้างภายนอกเป็นขั้นสุดท้ายแล้วย่อยส่งผลกระทบต่อโครงสร้างภายในของซอฟต์แวร์อย่างแน่นอน

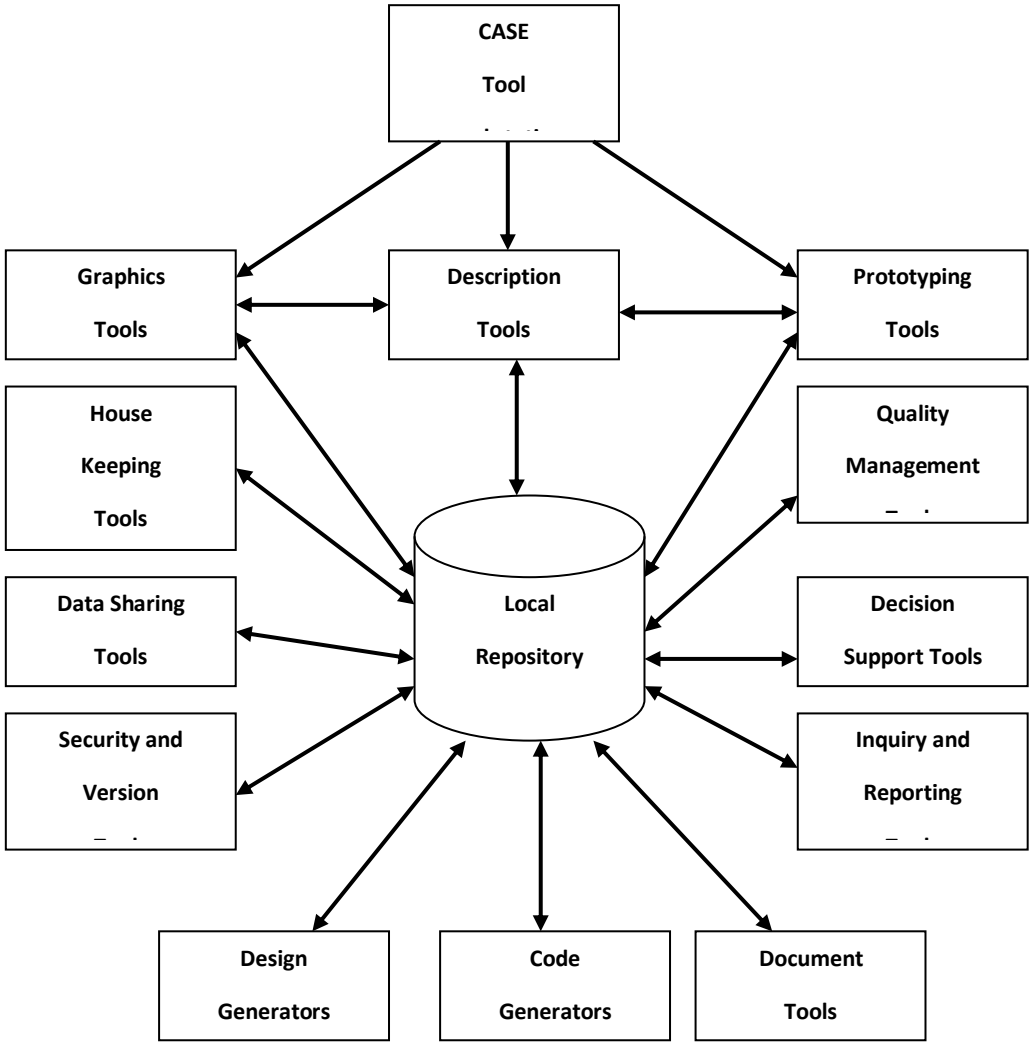
การจัดโครงสร้างของซอฟต์แวร์นั้นมีหลายรูปแบบ ทีมงานสามารถเลือกเฉพาะโครงสร้างที่เหมาะสมกับซอฟต์แวร์ที่กำลังผลิต อย่างไรก็ตาม หากซอฟต์แวร์นั้นเป็นซอฟต์แวร์ขนาดใหญ่ อาจใช้โครงสร้างมากกว่า 1 รูปแบบในการนำเสนอซึ่งในที่นี้จะนำเสนอรูปแบบโครงสร้างของซอฟต์แวร์ทั้งหมด 3 รูปแบบ ได้แก่ 1) แบบแบ่งปันแหล่งจัดเก็บข้อมูล 2) แบบแบ่งปันบริการและเซิร์ฟเวอร์ และ 3) แบบเลเยอร์

- **โครงสร้างแบบแบ่งปันแหล่งจัดเก็บข้อมูล (Repository Model)**

เป็นรูปแบบที่ส่วนประกอบย่อยของซอฟต์แวร์หรือระบบย่อยที่ทำงานร่วมกันเพื่อให้เกิดเป็นซอฟต์แวร์จะต้องมีการแลกเปลี่ยนข้อมูลระหว่างกันอย่างมีประสิทธิภาพ ด้วยวิธีการต่อไปนี้

1. จัดเก็บข้อมูลที่จะใช้ร่วมกันไว้ในฐานข้อมูล และทุกระบบย่อยสามารถเข้าถึงได้ เรียกรูปแบบดังกล่าวว่า “Repository Model”
2. แต่ละระบบย่อยมีฐานข้อมูลของตนเอง และแลกเปลี่ยนข้อมูลระหว่างกันด้วยการส่ง Message

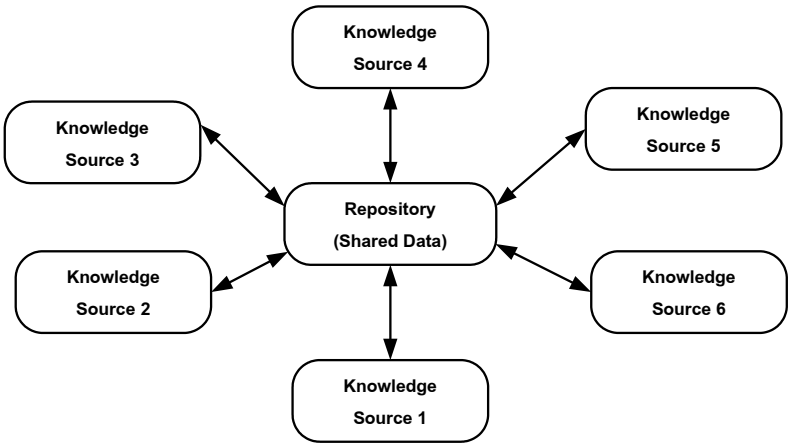
ลักษณะดังกล่าว คือ ซอฟต์แวร์ที่มีการแบ่งปันการใช้แหล่งจัดเก็บข้อมูลร่วมกัน โดยซอฟต์แวร์ที่มีโครงสร้างดังกล่าวส่วนมากจะเป็นซอฟต์แวร์ที่ต้องใช้ข้อมูลจำนวนมาก มีกระบวนการอื่นเรียกใช้ ยกตัวอย่าง เช่น ระบบสารสนเทศเพื่อการจัดการมCAD System และ CASE Tool เป็นต้น แสดงตัวอย่างสถาปัตยกรรม CASE Tool



รูปแสดงตัวอย่างโครงสร้างสถาปัตยกรรม Repository ของระบบ CASE Tool

จากรูป จะเห็นว่า ส่วนประกอบย่อยของซอฟต์แวร์ที่เป็น CASE Tool นั้น จะมีการแบ่งปันข้อมูลหรือใช้ข้อมูลร่วมกัน ซึ่งแหล่งจัดเก็บข้อมูลของ CASE Tool จะเรียกว่า Repository โดย Repository ในรูปแบบข้างต้นจะมีแบบจำลองข้อมูลที่มีโครงสร้างแน่นอน แต่หากเป็นโครงสร้างที่ไม่แน่นอน เช่น ไม่สามารถคาดเดาปัญหาที่เกิดขึ้นได้ องค์การอาจเลือกใช้ Shared Repository อีกรูปแบบหนึ่ง เรียกว่า “Backboard Model”

Backboard Model เป็นอีกรูปแบบหนึ่งของการแบ่งข้อมูล ที่เหมาะกับปัญหาแบบไม่มีโครงสร้างหรือโครงสร้างข้อมูลของ Repository มีลักษณะไม่แน่นอน เมื่อมีข้อมูลเข้ามาในระบบจะต้องอาศัย Trigger เป็นตัวกระตุ้นการทำงานของระบบย่อย ดังนั้น Backboard จึงมีองค์ประกอบ 3 ส่วน ได้แก่ ตัว Backboard แหล่งองค์ความรู้ และควบคุมโดยในตัว Backboard จะบรรจุข้อมูลในการแก้ปัญหาไว้ตามลำดับขั้นการตัดสินใจ ส่วนแหล่งองค์ความรู้จะแยกเป็นส่วนๆ แบ่งปันกันโดยผ่าน Backboard เมื่อ Backboard เปลี่ยนสถานะ แหล่งองค์ความรู้จะตอบสนองต่อ Backboard ทันที แสดงโครงสร้างดังรูป



รูปแสดงโครงสร้างสถาปัตยกรรม Repository แบบ Backboard

ข้อดี

1. เป็นวิธีการแบ่งเป็นข้อมูลจำนวนมากที่ดีวิธีหนึ่ง โดยที่ไม่จำเป็นต้องส่งข้อมูลโดยตรงระหว่างระบบย่อย
2. ระบบย่อยที่มีหน้าที่ผลิตข้อมูลไม่จำเป็นต้องทราบถึงวิธีเรียกใช้ข้อมูลของระบบย่อยอื่น
3. เครื่องมือในการสำรองข้อมูล ระบบรักษาความปลอดภัย การควบคุมการเข้าถึงข้อมูล และการกู้คืนระบบเป็นความรับผิดชอบส่วนกลางที่เรียกว่า Repository Manager ทำให้ควบคุมการทำงานง่ายขึ้น

ข้อเสีย

1. ระบบย่อยอื่นที่จะร่วมแบ่งปัน Repository จะต้องมีแบบจำลองข้อมูลในรูปแบบเดียวกับ Repository หรือไม่ก็ต้องสนับสนุนแบบจำลองดังกล่าว จึงจะสามารถทำงานร่วมกันได้ บางครั้งทำให้ประสิทธิภาพในการทำงานของระบบย่อยหรือเครื่องมืออื่นลดลง อันเนื่องมาจากการปรับแบบจำลองข้อมูลให้เข้ากัน
2. เนื่องจากมีข้อมูลจำนวนมากใน Repository ทำให้การปรับปรุงหรือแก้ไขแบบจำลองของ Repository ต้องใช้ค่าใช้จ่ายจำนวนมาก
3. ถึงแม้ว่าจะมี Repository Manager เพื่อจัดการกับเครื่องมือสำรองข้อมุลและเครื่องมืออรรถประโยชน์อื่นๆ ก็ตาม แต่เครื่องมือเหล่านั้น อาจมีนโยบายการทำงานที่แตกต่างกัน แต่เมื่อ Repository Manager ต้องควบคุมเครื่องมือเหล่านี้ วิศวกรออกแบบจึงต้องพยายามหานโยบายที่เป็นกลางที่สุด เพื่อที่จะให้กับเครื่องมือเหล่านี้ให้เหมือนกันได้

4. การกระจายการทำงานของ Repository ไปยังคอมพิวเตอร์ต่างชนิดกันได้ยาก เนื่องจากอาจเกิดปัญหาข้อมูลซ้ำซ้อนและไม่สอดคล้องกัน

➢ **โครงสร้างแบบแบ่งปันบริการและเซิร์ฟเวอร์**

Client / Server เป็นรูปแบบหนึ่งที่ได้รับความนิยมสูงสุดในปัจจุบัน ด้วยการจัดให้ระบบต้องประกอบไปด้วยบริการใดๆ เตรียมไว้ให้กับเครื่องคอมพิวเตอร์ 2 ชนิด คือ Server และ Client องค์ประกอบหลักของ Client / Server Model จึงต้องประกอบไปด้วย

1. **Server** คือ เครื่องคอมพิวเตอร์แม่ข่าย ซึ่งจะต้องมีซอฟต์แวร์สำหรับจัดการเครื่องแม่ข่าย ภายในจะประกอบด้วยระบบย่อยต่างๆ ไว้คอยบริการเครื่องลูกข่าย
2. **Client** คือ เครื่องคอมพิวเตอร์ลูกข่าย ที่เรียกใช้บริการต่างๆ จากเครื่องแม่ข่าย โดยที่เครื่องลูกข่ายจะสามารถติดตั้งซอฟต์แวร์ใดๆ เพื่อทำงานบนเครื่องได้ และขณะที่เรียกใช้บริการจากเครื่องแม่ข่าย เครื่องลูกข่ายสามารถรันโปรแกรมใดๆ ในเวลาเดียวกันได้
3. **Network** คือ เครือข่ายที่นอกจากเครื่องคอมพิวเตอร์แล้ว ยังประกอบไปด้วยอุปกรณ์เครือข่ายหลายชนิด เพื่อช่วยให้เครื่องลูกข่ายสามารถติดต่อและเรียกใช้บริการจากเครื่องแม่ข่ายได้ การเชื่อมต่อมีหลายลักษณะ ที่ไม่จำเป็นต้องมีเครื่องลูกข่ายหรือแม่ข่ายเพียงเครื่องเดียว เครื่องลูกข่าย 1 เครื่องสามารถติดต่อกับเครื่องแม่ข่ายมากกว่า 1 เครื่องได้ และเครื่องแม่ข่าย 1 เครื่องก็สามารถให้บริการกับเครื่องลูกข่ายหลายเครื่องได้เช่นกัน และแน่นอนว่าเครื่องแม่ข่าย 1 เครื่องสามารถให้บริการหลายอย่างแก่เครื่องลูกข่ายได้

ข้อดี

1. สามารถกระจายการประมวลผลได้
2. การเพิ่ม Server หรือประสานการทำงานของ Server และการอัพเกรด Server ทำได้ง่ายโดยไม่กระทบต่อส่วนอื่นของระบบ

ข้อเสีย

1. กรณีมีเครื่องแม่ข่ายหลายเครื่อง จะต้องติดตั้งระบบจัดการความซ้ำซ้อนของข้อมูลด้วย
2. การเข้าถึงบริการของเครื่องแม่ข่ายในระบบทำได้ยาก เพราะต้องผ่านโปรโตคอล เช่น HTTP เป็นต้น

➢ **โครงสร้างแบบเลเยอร์**

เป็นโครงสร้างที่แสดงให้เห็นการจัดโครงสร้างระบบย่อยหรือประกอบต่างๆ ในมุมมองแบบระดับชั้น แต่ละระดับชั้นคือ ส่วนประกอบย่อยที่รับผิดชองการทำงานในแต่ละด้านของซอฟต์แวร์ มีลักษณะคล้ายกับชุดคำสั่งของเครื่องจักร จึงเรียกโครงสร้างดังกล่าวได้อีกชื่อหนึ่งว่า “Abstract Machine Model”

แต่ละเลเยอร์มีบริการจัดเตรียมไว้เพื่อทำงานร่วมกับเลเยอร์ด้สนนอก ที่จะเรียกใช้บริการตลอกเวลาเมื่อมีการประมวลผล (มีลักษณะคล้ายกับ Client) ในบางระบบทุกเลเยอร์จะสามารถติดต่อกันได้ทั้งหมด แต่บางระบบเลเยอร์จะถูกจำกัดการติดต่อขึ้นอยู่กับนโยบายของระบบ เลเยอร์ด้านนอกสุด จะทำหน้าที่ติดต่อกับผู้ใช้ซอฟต์แวร์ ส่วนเลเยอร์ด้านในสุดเป็นระบบปฏิบัติการที่สนับสนุนการทำงานของส่วนประกอบอื่น

สำหรับเลเยอร์ในระดับกลาง คือ Utility และ Application Layer นั้น จะให้บริการอรรถประโยชน์และฟังก์ชันงานธุรกิจต่างๆ

ข้อดี

1. สามารถเปลี่ยนซอฟต์แวร์เป็นชนิดที่อยู่ในเลเยอร์เดียวกันได้ ตราบเท่าที่ยังไม่มีการเปลี่ยน Interface ในเลเยอร์ที่อยู่ติดกันเท่านั้นที่จะได้รับผลกระทบและต้องแก้ไข
2. สามารถเตรียมแพลตฟอร์มสำหรับระบบที่ใช้โครงสร้างดังกล่าวได้หลายแพลตฟอร์ม เนื่องจากการขึ้นต่อกันของระบบย่อยนั้นจะมีลักษณะเรียงต่อกันไปยังด้านนอกสุด ยกเว้นเลเยอร์ด้านในสุดที่อาจต้องปรับเปลี่ยน เช่น ระบบปฏิบัติการ และฐานข้อมูล เป็นต้น

ข้อเสีย

1. การจัดโครงสร้างระบบเลเยอร์อาจทำได้ยาก เนื่องจากเลเยอร์ด้านในจะต้องเตรียมบริการหลายอย่างที่เลเยอร์ด้านนอกทุกเลเยอร์ต้องการ (ในบางระบบ)
2. ประสิทธิภาพการทำงานอาจลดลง เนื่องจากต้องส่งคำสั่งผ่านส่วนประสานหลายระดับ

<u>การแบ่งส่วนซอฟต์แวร์</u>

หลังจากที่ทีมงานตัดสินใจเลือกรูปแบบโครงสร้างสถาปัตยกรรมของระบบหรือซอฟต์แวร์ได้แล้ว สิ่งต่อไปที่จะต้องตัดสินใจ ก็คือ การแบ่งส่วนซอฟต์แวร์ ซึ่งในที่นี้จะกล่าวถึงการแบ่งซอฟต์แวร์ออกเป็นส่วนย่อยในลักษณะที่เรียกว่า “โมดูล” ดังนั้น การออกแบบเชิงสถาปัตยกรรมในด้านนี้ จึงเป็นการออกแบบที่แสดงให้เห็นส่วนประกอบของซอฟต์แวร์ออกเป็นโมดูล มีหลายรูปแบบ จึงเรียกว่า “รูปแบบการแบ่งส่วนซอฟต์แวร์ออกเป็นโมดูล” ก่อนอื่นในที่นี้ขอทำความเข้าใจกับคำว่า “ระบบย่อย” และ “โมดูล” ดังนี้

1. ระบบย่อย คือ ระบบที่เป็นส่วนหนึ่งของระบบทั้งหมด จึงมีคุณลักษณะเช่นเดียวกับระบบทุกประการ แต่เมื่อเป็นส่วนหนึ่งของระบบทั้งหมดแล้ว ระบบย่อยจะสามารถทำงานได้โดยไม่ต้องพึ่งพาบริการจากระบบย่อยอื่น โดยจะประกอบด้วยโมดูล Interface เพื่อใช้ในการติดต่อสื่อสารกับระบบย่อยอื่น
2. โมดูล คือ ส่วนประกอบของซอฟต์แวร์หรือระบบ ที่ได้จัดเตรียมบริการตั้งแต่ 1 อย่างขึ้นไปไว้ให้กับโมดูลอื่นได้เรียกใช้ กล่าวโดยสรุปแล้ว โมดูล ก็คือ กลุ่มการทำงานหรือกลุ่มของชุดคำสั่งหรือก็คือโปรแกรมย่อย

การแบ่งส่วนซอฟต์แวร์ออกเป็นโมดูล ที่นำเสนอในหัวข้อนี้ มี 2 รูปแบบ ได้แก่ การแบ่งส่วนซอฟต์แวร์เชิงวัตถุ และการแบ่งส่วนซอฟต์แวร์ตามฟังก์ชันงาน

- **การแบ่งส่วนงานเชิงวัตถุ**

การแบ่งส่วนงานเชิงวัตถุ จะแบ่งระบบหรือซอฟต์แวร์ออกเป็นกลุ่มของอ็อบเจ็กต์ที่สัมพันธ์กัน ซึ่งกลุ่มของอ็อบเจ็กต์ดังกล่าวคือลักษณะหนึ่งของโมดูล

กลุ่มของอ็อบเจ็กต์ที่สัมพันธ์กันนั้น จะต้องมีการขึ้นต่อกันน้อย คือ ข้อมูลที่ส่งระหว่างอ็อบเจ็กต์ต้องเป็นข้อมูลเดี่ยว(ไม่ซับซ้อน) อ็อบเจ็กต์จะสามารถทำงานได้อย่างอิสระ และไม่ส่งผลกระทบต่อกันเมื่อมีการ

เปลี่ยนแปลงการทำงานภายในอ็อบเจ็กต์จะต้องกำหนด Interface ไว้เพื่อให้บริการต่ออ็อบเจ็กต์อื่นที่ต้องการเรียกใช้บริการนั้น Interface จึงเป็นช่องทางสื่อสารระหว่างอ็อบเจ็กต์

ข้อดีของการแบ่งส่วซอฟต์แวร์ด้วยวิธีเชิงวัตถุ จะเกิดขึ้นก็ต่อเมื่อความสัมพันธ์ระหว่างอ็อบเจ็กต์เป็นแบบขึ้นต่อกันน้อยที่สุด ซึ่งจะทำให้อ็อบเจ็กต์สามารถทำงานได้อิสระ เมื่อมีการเปลี่ยนแปลงการทำงานภายในอ็อบเจ็กต์หนึ่งจึงไม่ส่งผลกระทบต่ออ็อบเจ็กต์อื่น นอกจากนี้ การใช้อ็อบเจ็กต์เป็นตัวแทนของสรรพสิ่งในโลกความเป็นจริงนั้นจะทำให้ทีมงานมีความเข้าใจในโครงสร้างของแต่ละสิ่งที่สนใจได้ง่ายขึ้นตามความเป็นจริง และแต่ละสิ่งสามารถอยู่ในสถานการณ์หรือระบบแตกต่างกันได้ ดังนั้น จึงสามารถนำอ็อบเจ็กต์ไปใช้ในอีกระบบหนึ่งได้ ซึ่งก็คือ การ Reuse นั้นเอง อย่างไรก็ตาม แม้ว่าการเปลี่ยนแปลงการทำงานภายในอ็อบเจ็กต์หนึ่งจะไม่ส่งผลกระทบต่ออ็อบเจ็กต์อื่นก็ตาม แต่หากต้องเปลี่ยน Interface ของอ็อบเจ็กต์ใด จะส่งผลกระทบต่ออ็อบเจ็กต์ที่เรียกใช้บริการจากอ็อบเจ็กต์นั้นทันที

- **การแบ่งส่วนซอฟต์แวร์ตามฟังก์ชันงาน**

การแบ่งส่วนซอฟต์แวร์ชนิดนี้ จะแบ่งตามหน้าที่การประมวลผลข้อมูลของกระบวนการ ซึ่งแต่ละกระบวนการจะได้รับข้อมูลที่นำมาแล้วทำหน้าที่ประมวลผลเพื่อเปลี่ยนรูปข้อมูลดังกล่าวให้เป็นผลลัพธ์ กระบวนการเปลี่ยนรูปข้อมูลสามารถดำเนินการได้ทั้งแบบขนานและแบบเรียงลำดับ สำหรับระบบงานที่ถูกออกแบบให้ทำงานบนระบบปฏิบัติการยูนิกส์ จะเรียกการแบ่งส่วนซอฟต์แวร์ลักษณะนี้ว่า “Pipe and Filter” โดยจะเรียก กระแสข้อมูลหรือสตรีมข้อมูล ว่า Pipe เป็นเสมือนท่อข้อมูลนำเข้า แต่ละ Filter จะทำงานอย่างอิสระ โดยไม่สนใจการทำงานของ Filter อื่น ข้อมูลที่ส่งเข้าไปยัง Filter นั้นจะมีคำสั่งต่างๆ รวมอยู่ด้วย สำหรับแบบจำลอง Pipe and Filter นั้นจะมีหลายรูปแบบแตกต่างกันออกไป แต่จะต้องมีลำดับการประมวลผลแบบเดียวกันคือ แบบเรียงลำดับ

ทีมงานสามารถใช้การแบ่งซอฟต์แวร์ลักษณะดังกล่าว เพื่อนำเสนอโครงสร้างของซอฟต์แวร์ในอีกมุมมองหนึ่งเพิ่มเติมจากมุมมองของโครงสร้างได้

ข้อดี

1. สนับสนุนการนำกระบวนการแปลงสภาพข้อมูลไปใช้ซ้ำได้
2. เป็นวิธีที่ทำให้ทีมงานเข้าใจง่าย
3. การเพิ่มกระบวนการใหม่ทำได้ง่าย คือ เพิ่มในลักษณะเรียงลำดับต่อเนื่องกันไปเรื่อยๆ

ข้อเสีย

1. ไม่เหมาะกับระบบงานที่ต้องการประมวลผลแบบโต้ตอบทันที
2. ข้อมูลที่จะส่งไปตาม Pipe จะต้องมีรูปแบบเดียวกัน
3. การที่ Filter สามารถทำงานได้อิสระ อาจทำให้เกิดการทำหน้าที่เพื่อประมวลข้อมูลเดียวกันซ้ำซ้อนกัน อาจส่งผลให้โค้ดโปรแกรมซับซ้อนและประสิทธิภาพการทำงานลงได้

<u>การควบคุมการทำงานของส่วนประกอบย่อย</u>

การที่ส่วนประกอบย่อยติดต่อสื่อสารเพื่อทำงานร่วมกันได้นั้น ต้องถูกควบคุมจากส่วนควบคุมการทำงานของระบบเพื่อดูแลการทำงานของส่วนประกอบย่อยเหล่านั้นให้ถูกต้องและมีประสิทธิภาพ ซึ่งเมื่อมีการกำหนดรูปแบบการจัดโครงสร้างและการแบ่งส่วนซอฟต์แวร์แล้ว ทีมงานจะต้องกำหนดรูปแบบการควบคุมการทำงาน ขงส่วนประกอบย่อยเหล่านั้นด้วย และสร้างเป็นแบบจำลองขึ้นเพื่อนำเสนอมุมมองการควบคุมการทำงานของซอฟต์แวร์ ซึ่งแบบจำลองดังกล่าวจะแสดงให้เห็นทิศทางการควบคุมการทำงาน ระหว่างส่วนประกอบย่อย

รูปแบบควบคุมการทำงานส่วนประกอบย่อยของซอฟต์แวร์มี 2 รูปแบบ ได้แก่ การควบคุมแบบศูนย์กลาง และการควบคุมตามเหตุการณ์

- **การควบคุมแบบศูนย์กลาง**

ลักษณะการควบคุมแบบศูนย์กลาง จะมีระบบย่อย 1 ระบบ ทำหน้าที่ควบคุมการทำงานของทุกระบบย่อย หรือสามารถแบ่งภาระการควบคุมไปยังระบบอื่นได้ แต่ต้องคืนสถานการณ์ควบคุมการทำงานกลับมา การควบคุมแบบศูนย์กลางมี 2 ลักษณะ ได้แก่

- **Call and Return Model**

การควบคุมการทำงานจะมีลักษณะเป็นแบบ Top-down คือ จะจัดให้ระบบย่อยที่ทำหน้าที่ควบคุมระบบย่อยอื่นที่อยู่ในระดับสูงสุด เพื่อเรียกใช้งานระบบย่อยอื่น ซึ่งถูกเรียกว่า “Subroutine” ที่อยู่ในระดับชั้นถัดลงมาด้านล่าง การควบคุมการทำงานลักษณะนี้จะทำให้เกิด โครงสร้างการควบคุมแบบเรียงลำดับ เช่นลักษณะแบบ โปรแกรมหลัก และ โปรแกรมย่อย นั่นเอง ซึ่งพบมากในภาษาโปรแกรมมิ่งเชิงโครงสร้าง เช่น Pascal, Ada เป็นต้น

- **Manager Model**

มีลักษณะการทำงานเหมือนเจ้านายกับลูกน้อง โดยมีระบบย่อย 1 ระบบทำหน้าที่เป็นเจ้านาย โดยทำหน้าที่เรียกใช้งานลูกน้อง (ระบบย่อยอื่นๆ) พร้อมกับประสานงานกับระบบอื่นด้วย ในกระบวนการใดๆ ระบบย่อยสามารถทำงานพร้อมๆกันได้ หรือสามารถทำงานเรียงลำดับตามเงื่อนไขที่ได้รับก็ได้ การควบคุมลักษณะนี้ พบได้จากระบบที่ทำงานแบบ เวลาจริง ที่ไม่เข้มงวดเรื่องระยะเวลามากนัก

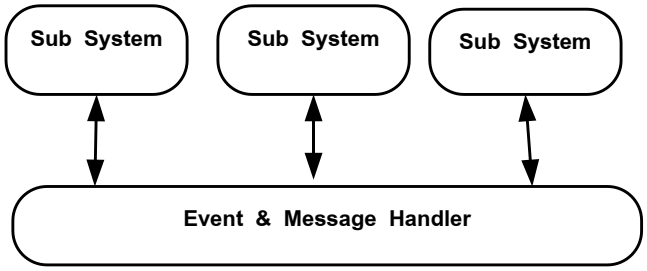
- **การควบคุมตามเหตุการณ์**

การควบคุมลักษณะนี้ การทำงานของระบบย่อยจะเกิดขึ้นเมื่อมีเหตุการณ์ใดๆ เกิดขึ้น ลักษณะของการทำงานที่ตอบสนองต่อเหตุการณ์จะแตกต่างกันออกไปตามชนิดของเหตุการณ์ที่เป็นตัวกระตุ้น ซึ่งคำว่าเหตุการณ์ ในที่นี้เป็นไปได้ทั้งในรูปของสัญญาณข้อมูลเลขฐานสอง และสัญญาณข้อมูลที่ได้รับมาจาก GUI ที่ใช้รับข้อมูลประเภทต่างๆ ระบบที่ทำงานตามเหตุการณ์ จะเรียกว่า “Event-driven System” มีหลายลักษณะ เช่น ซอฟต์แวร์ที่ช่วยเขียนโปรแกรมในลักษณะ Visual หรือ ระบบสร้างกฎเกณฑ์ ซึ่งมีค่าเงื่อนไขที่เป็นจริงทำหน้าที่เป็นเหตุการณ์กระตุ้นให้เกิดการทำงานอย่างใดอย่างหนึ่ง เป็นต้น

ในหัวข้อนี้จะนำเสนอการควบคุมตามเหตุการณ์ 2 รูปแบบ ได้แก่ เหตุการณ์แบบกระจาย และเหตุการณ์แบบขัดจังหวะ

- **เหตุการณ์แบบกระจาย (Broadcast Model)**

เหตุการณ์ลักษณะนี้ จะถูกกระจายไปยังระบบย่อยทั้งหมด ระบบย่อยใดที่ถูกโปรแกรมให้สามารถรองรับเหตุการณ์นั้นได้ จะเป็นผู้รับผิดชองการทำงานเพื่อตอบสนองเหตุการณ์ดังรูป



รูปแสดงแบบจำลองเหตุการณ์แบบกระจาย

Broadcast Model คือ สามารถเพิ่มระบบย่อยใหม่ได้ง่าย เพียงเพิ่มข้อมูลของระบบย่อยใหม่เข้าไปใน Event Handler และยังสามารถ Implement ระบบย่อยแบบกระจายได้อีกด้วย อย่างไรก็ตาม ข้อเสียของ Broadcast Model คือ ระบบย่อยอาจเก็บ Event ที่ซ้ำซ้อนกันและอาจทำให้เกิดการทำงานที่ขัดแย้งกันได้

- **เหตุการณ์แบบขัดจังหวะ (Interrupt Model)**

เหตุการณ์ลักษณะนี้ จะใช้โดยเฉพาะกับระบบเวลาจริง เนื่องจากจะมีส่วนที่เรียกว่า Interrupt คอยจัดสรรเหตุการณ์ไปยังระบบย่อยที่สามารถประมวลผลเหตุการณ์นั้นได้ โดนที่ Interrupt แต่ละชนิดจะถูกกำหนดให้เรียกใช้ระบบย่อยที่จะตอบสนองต่อเหตุการณ์นั้นไว้แล้ว นอกจากนี้ Interrupt ยังมีการเก็บค่าแอดเดรส (Address) ที่อยู่ในหน่วยความจำของระบบย่อยแต่ละระบบไว้ด้วย ดังนั้น Interrupt ได้รับเหตุการณ์ซึ้งตรงกับชนิดที่ตนเองรับผิดชอบ จะกระตุ้นให้ระบบย่อยทำงานได้อย่างรวดเร็ว จึงเหมาะสำหรับระบบแบบ Real Time

สถาปัตยกรรมอ้างอิง

สถาปัตยกรรมอ้างอิง คือสถาปัตยกรรมของซอฟต์แวร์ที่อ้างอิงมาจากกลุ่มของระบบงานต่างๆ ที่มีอยู่ทั้งในโลกของความเป็นจริง แสดงออกมาในรูปแบบจำลองของสถาปัตยกรรมหรือโครงสร้างเฉพาะระบบงานนั้นๆ

จากหัวข้อทั้งหมดที่กล่าวมาข้างต้น เป็นแบบจำลองสถาปัตยกรรมทั่วไปที่สามารถนำไปประยุกต์ใช้กับระบบงานใดก็ได้ตามต้องการ แต่สำหรับแบบจำลองสถาปัตยกรรมอ้างอิงนั้น จะสามารถนำไปใช้กับระบบงานเฉพาะที่นำเสนอไว้ในแบบจำลองเท่านั้น องค์กรใดที่ต้องดำเนินงานในระบบงานเฉพาะดังกล่าว สามารถนำแบบจำลองสถาปัตยกรรมนั้นไปใช้เป็นโครงสร้างพื้นฐานของการพัฒนาระบบได้ โดยโครงสร้างของระบบจะเหมือนกัน แต่ต่างกันที่รายละเอียดการดำเนินงานของแต่ละองค์กร สถาปัตยกรรมอ้างอิงจึงเรียกได้อีกอย่างหนึ่งว่า “Domain Specific Architecture”

ตัวอย่างของระบบงานเฉพาะดังกล่าว ก็คือ ระบบงานพื้นฐานของระบบธุรกิจ โดยแบ่งเป็น 3 ประเภท ได้แก่

- **ระบบประมวลผลข้อมูล**

เป็นระบบที่ยึดข้อมูลเป็นศูนย์กลาง จะมีการเก็บข้อมูลไว้ประมวลผล ณ ช่วงเวลาใดเวลาหนึ่งอย่างสม่ำเสมอ โดยไม่รบกวนการทำงานของผู้ใช้ ซึ่งก็คือ การประมวลผลแบบกลุ่มนั่นเอง ระบบประเภทนี้จะต้องประมวลผลข้อมูลจำนวนมาก ที่เกี่ยวข้องกับงานบริหารเป็นหลัก เช่น บัญชีเงินเดือน ระบบจัดทำบิล ระบบบัญชี เป็นต้น

- **ระบบประมวลผลรายการข้อมูล**

เป็นระบบที่มีฐานข้อมูลเป็นศูนย์กลางการประมวลผลคำร้องขอข้อมูลและการอัพเดทข้อมูลจากผู้ใช้ทั้งหมด ส่วนใหญ่เป็นระบบธุรกิจที่ต้องมีการอัพเดทข้อมุลทันทีที่มีการทำรายการ และผู้ใช้หลายคนสามารถใช้งานระบบพร้อมกันได้ เช่น ระบบธนาคาร ระบบอีคอมเมิร์ซ ระบบสารสนเทศ และระบบจองตั๋ว

- **ระบบประมวลผลตามเหตุการณ์**

เป็นประเภทของระบบที่มีขอบเขตกว้างมาก กล่าวคือ ทุกระบบที่การทำงานของระบบจะเกิดขึ้นเมื่อมีเหตุการณ์มากระตุ้น จะจัดว่าเป็นระบบประมวลผลตามเหตุการณ์ ซึ่งเหตุการณ์อาจอยู่ในรูปของคำสั่งจากผู้ใช้หรืออยู่ในรูปของตั๋วตัวแปรกระตุ้น ระบบดังกล่าวส่วนใหญ่จะเป็นโปรแกรมที่ทำงานบนเครื่องคอมพิวเตอร์ส่วนบุคคล เช่น โปรแกรมประมวลผล โปรแกรมกระดาษคำนวณอิเล็กทรอนิกส์ โปรแกรมนำเสนองาน หรือโปรแกรมตกแต่งภาพ เป็นต้น

สถาปัตยกรรมดังกล่าว สามารถนำมาใช้ประโยชน์ได้หลายอย่าง เช่น นำมาใช้เป็นจุดเริ่มต้นในกระบวนการออกแบบใช้เป็นรายการหลักในการตรวจสอบการทำงาน ใช้เป็นเกณฑ์ในการจัดโครงสร้างทีมงาน หรือใช้ประเมินว่าส่วนใดควรสร้างแบบ Reuse เป็นต้น

นอกจากนี้ ยังมีสถาปัตยกรรมอีกรูปแบบหนึ่งที่ใช้เพื่อการอ้างอิงอย่างแท้จริง กล่าวคือ เป็นสถาปัตยกรรมที่มีลักษะเป็นนามธรรม แสดงให้เห็นเพียงกรอบโครงสร้างของระบบ (ซอฟต์แวร์) เท่านั้น เป็นเสมือนการแนะนำโครงสร้างทั่วไปของระบบให้แก่ทีมงานออกแบบได้ทราบ สถาปัตยกรรมอ้างอิงลักษณะนี้จะมีขอบเขตกว้างกว่าสถาปัตยกรรมอ้างอิงที่กล่าวถึงข้างต้น เช่น สถาปัตยกรรม OSI ที่ใช้สำหรับการเชื่อมต่อระบบที่มีเครือข่ายต่างชนิดกัน หรือสถาปัตยกรรม ECMA CASE Environment สำหรับ CASE Tool เป็นต้น

<u>สรุป</u>

- **การออกแบบเชิงสถาปัตยกรรม** หมายถึง การกำหนดลักษณะโครงสร้างของระบบหรือซอฟต์แวร์ในมุมมองระดับบน เป็นการแสดงให้เห็นส่วนประกอบต่างๆ ของซอฟต์แวร์ภายใต้โครงสร้างสถาปัตยกรรมรูปแบบใดๆ กล่าวโดยสรุป การออกแบบเชิงสถาปัตยกรรม ก็คือ การเลือกรูปแบบของสถาปัตยกรรมให้กับซอฟต์แวร์ ซึ่งวิศวกรรมซอฟต์แวร์ควรดำเนินการเป็นลำดับแรกของการออกแบบทั้งหมด เนื่องจากสถาปัตยกรรมที่เลือกไว้จะใช้เป็นกรอบการออแบบในส่วนอื่นๆ ของซอฟต์แวร์ที่จะต้องทำงานบนสถาปัตยกรรมเดียวกัน การออกแบบเชิงสถาปัตยกรรม จะเป็นการกำหนดโครงสร้างของซอฟต์แวร์ในทั้งหมด 3 มุมมอง ได้แก่ กำหนดรูปแบบการจัดโครงสร้างของซอฟต์แวร์ การแบ่งส่วนซอฟต์แวร์ และการควบคุมการทำงานของส่วนประกอบย่อย
- **การจัดโครงสร้างของซอฟต์แวร์** นั้นมีหลายรูปแบบ ได้แก่ โครงสร้างแบบแบ่งปันแหล่งจัดเก็บข้อมูล เป็นโครงสร้างที่จัดให้ระบบย่อยใช้ข้อมูลในแหล่งจัดเก็บข้อมูลที่เรียกว่า “Repository” ร่วมกัน โดยมีกระบวนการสร้างข้อมูลโดยสร้างข้อมูลโดยเฉพาะ เช่น CASE Tool เป็นต้น โครงสร้างแบบแบ่งปันบริการและเซิร์ฟเวอร์ เป็นโครงสร้างที่จัดให้มีเครื่องแม่ข่าย 1 เครื่อง เพื่อให้บริการแก่เครื่องลูกข่ายที่อยู่ต่างสถานที่กันจำนวนมาก และโครงสร้างแบบเลเยอร์ เป็นโครงสร้างที่จัดให้ระบบย่อยหรือส่วนประกอบต่างๆ มีการติดต่อสื่อสารแบบระดับชั้น แต่ละระดับชั้นคือ ส่วนประกอบย่อยที่รับผิดชอบการทำงานในแต่ละด้านของซอฟต์แวร์
- หลังจากกำหนดรูปแบบการจัดรูแบบการจัดโครงสร้างของซอฟต์แวร์แล้ว ทีมงานจะต้องกำหนดรูปแบบ **การแบ่งส่วนซอฟต์แวร์** โดยแต่ละส่วนย่อยของซอฟต์แวร์จะเรียกว่า โมดูล ซึ่งมีการแบ่งส่วนทั้งหมด 2 รูปแบบ ได้แก่ การแบ่งส่วนซอฟต์แวร์เชิงวัตถุ จะแบ่งระบบหรือซอฟต์แวร์ออกเป็นกลุ่มของอ็อบเจ็กต์ที่สัมพันธ์กัน ซึ่งกลุ่มของออ็อบเจ็กต์ดังกล่าวก็คือ ลักษณะหนึ่งของโมดูล และการแบ่งส่วนตามฟังก์ชันงาน จะแบ่งตามหน้าที่การประมวลผลข้อมูลของกระบวนการ ซึ่งแต่ละกระบวนการจะได้รับข้อมูลที่นำมาแล้วทำหน้าที่ประมวลผลเพื่อเปลี่ยนรูปข้อมูลกล่าวให้เป็นผลลัพธ์ เรียกได้อีกอย่างหนึ่งว่า “Pipe Filter”
- ส่วนสุดท้ายที่ทีมงานควรออกแบบ คือ รูปแบบการควบคุมการทำงานของส่วนประกอบย่อย แบ่งออกเป็น 2 รูปแบบ ได้แก่ การควบคุมแบบศูนย์กลาง จะมีระบบย่อย 1 ระบบ ทำหน้าที่ควบคุมการทำงานของทุกระบบย่อย การควบคุมศูนย์กลางมี 2 ลักษณะ ได้แก่ Call and Return และ Manager Model การควบคุมรูปแบบที่ 2 คือ การควบคุมตามเหตุการณ์ การทำงานของระบบย่อยจะเกิดขึ้นเมื่อมีเหตุการณ์ใดๆ เกิดขึ้น ลักษณะของการทำงานที่ตอบสนองต่อเหตุการณ์จะแตกต่างกันออกไปตามชนิดเหตุการณ์ที่เป็นตัวกระตุ้น

ข้อมูลอ้างอิง

1. ณรงค์ จันทร์สร้อย. (2013, 26 กนัยายน) แหล่งที่มาสืบค้นจาก https://minimallife.wordpress.com/2013/09/26/software-architecture
2. พรฤดี เนติโสภากุล. (2549). วิศวกรรมซอฟต์แวร์ ทฤษฎี หลักการ และการประยุกต์ใช้ Software Engineering