2 Architecture is (mostly)orthogonal กับ Functionality

เวลาเรามีFunctionalคงเดิมเราสามารถมี Architecture ได้หลายแบบ การตั้งฉากหมายถึง เราสามารถเปลี่ยนแปลง Architecture ได้โดยยังให้บริการและตอบสนองตามโจทย์แบบเดิมได้ ซึ่งArchitecture เปลี่ยนจะมีผลกระทบกับ Quality Attributes และ Architecture สร้างข้อกำจัดให้กับระบบ

สมมติว่า เรามีArchitectureอยู่2แบบ เช่น peer to peer กับ แบบ client server การต่อสองแบบมันอาจจะต่างกัน แต่ว่าสองแบบนี้อาจยังที่จะสามารถตอบโจทย์ปัญหาได้อยู่เหมือนเดิมก็ได้ แต่ทำได้แค่บางกรณี บางอย่างก็อาจจะไม่สามารถเปลี่ยนได้

3 จริงๆแล้วArchitecture constrains เป็นข้อจำกัด และมีได้หลายรูปแบบมาก ซึ่งแต่ละArchitectureการที่เรามีข้อจำกัดก็เหมือนเราได้ทำการได้ตีขอบเขตให้มัน สภาวะภายนอก หรือปัจจัยภายนอกที่เป็นข้อจำกัดและเป็นแรงกดดัน ทำให้ stakeholder เกิดความกังวล (concern) เมื่อปัจจัยภายนอกส่งผลต่อความรู้สึกจึงทำให้เกิดความต้องการ ความต้องการในลำดับแรกๆ คือ business goal หรือ business requirement จากในที่นี้อาจจะแปลว่าลดปัญหาเมื่อมีมากไป หรือช่วยให้มันเร็วขึ้น

ตัวอย่าง เช่น ต้องการจะทำระบบขนส่งรถไฟ ถูก constrains ด้วยราง เพื่อให้วิ่งได้แค่เฉพาะบนรางรถไฟ คือเราอาจจะแค่ตีทางให้มันก้ได้ การเดินทางจากต้นไปปลายก็ยังได้แบบเดิมแต่ว่าถ้าเราให้มันใช้รางในการวิ่ง มันจะทำให้เราถึงเป้าหมายได้เร็วขึ้น

4 1. High failure risk กรณีที่เกิดความผิดพลาดแล้วมันมีความเสี่ยงหรือความเสียหาย ผลกระทบจะเกิดขึ้นเยอะซึ่งการให้ความสนใจกับ Architecture ที่มีความใหญ่เล็กต่างกัน และผลกระทบหรือผลที่เกิดขึ้นต่างกัน ทำให้มีความสำคัญที่ต้องมาให้ความจำเป็นก็จะต่างกัน

2. new domain มันจะเป็นสิ่งใหม่สำหรับนักพัฒนา สมมติว่า เราอาจจะมีความคุ้นเคยกับการเขียน web app มาหลายๆครั้งแล้ว แต่พอเริ่มมาหัดเขียน mobile app ครั้งแรก เราอาจจะมาดูแล้วว่าบน android โครงสร้างในการพัฒนามันเป็นอย่างไรประกอบไปด้วยอย่างไรบ้าง มีตัวไหนที่เป็น service ของ os มีตัวที่จะต้องมาทำความเข้าใจใหม่เพราะคนละแบบกับของเดิม เหมือนเปนกับnew domain หรือ การทำ innovation ใหม่ หรือทำสิ่งใหม่ๆของผู้พัฒนา เป็นขอบเขตความรู้ใหม่ที่ไม่คุ้นเคยหรืออาจจะไม่เคยมีคนทำมาก่อน

3. Product lines คือการเป็นการ common factors คือการวางแผน Architecture มาดี เราจะต้องวางแผนว่าก่อนว่าเนี่ยเราอาจจะต้องให้ซอฟแวร์มันสามารถรันในหลายๆแพลตฟอร์มได้ โดยสร้างออกมาทีเดียวเพื่อเอาไปใช้ได้ในหลายกรณี หรือซัพพอร์ตความเปลี่ยนแปลง เพื่อให้เกิดความยืดหยุ่น

5 99.0% สำหรับ Web Server (WebServer) = 0.99

90.0% สำหรับ Application Server (AppServer) = 0.9 / โอกาสล่ม 1- 0.9 = 0.1

95.0% สำหรับ Database Server (DBServer) = 0.95 / โอกาสล่ม 1- 0.95 = 0.05

หา AppServer ของระบบ ถือว่าระบบยัง available เมื่อมีอย่างน้อย 1 instance ที่ยังทำงานได้ = 1 - โอกาสล่มของ AppServer พร้อมกันทั้งหมด = 1 - (0.1)^3 = 1 - 0.001 = 0.999

หา DBServer ของระบบ ถือว่าระบบยัง available เมื่อมีอย่างน้อย 1 instance ที่ยังทำงานได้ = 1 - โอกาสล่มของ DBServer พร้อมกันทั้งหมด = 1 - (0.05)^2 = 1 - 0.0025 = 0.9975

คำนวณหา availability ของระบบทั้งหมด = 0.990 x 0.999 x 0.9975 = 0.98653

6 ค่อนข้างที่จะจำเป็น เพราะว่ายึดตามหลักความจริง เรามักจะใช้โปรแกรมหรือ platform หรือ service ต่างๆที่เป็นที่นิยม และแต่ละสิ่งพวกนี้มันจะรองรับนามสกุลไฟล์ที่แตกต่างกันหรือได้แค่บางอย่าง การที่เราจะทำงานด้วยกันได้ จึงค่อนข้างที่จำเป็นที่จะต้องมีการตกลงความเหมือนกัน หรือการทำงานที่เหมือนกันรวมไปถึง การใช้นามสกุลไฟล์ เพื่อไม่ให้เกิดปัญหา เช่นถ้าเราทำโปรเจคหนึ่งอยู่ แล้วเราเขียนโคตให้นำเข้าไฟล์นามสกุลนั้นๆ ถ้ามีคนหนึ่งส่งมาแต่เป็นคนละนามสกุล อาจจะทำให้เกิดปัญหาโค๊ตแตกได้ หรือบางทีเราต้องการใช้ library หนึ่งๆ มันอาจจะต้องกำหนดว่ารองรับแค่ไฟล์นามสกุลนี้นะ เป็นต้น เพราะฉนั้นเพื่อให้นำข้อมูลไปใช้ได้หลากหลายหรือยืดหยุ่นได้ ควรเลือกใช้มาตรฐานต่างๆ ให้ตรงกัน

7 class diagram อธิบายคุณลักษณะและการดำเนินงานของClassและข้อ จำกัด ที่กำหนดไว้ในระบบ Class diagram มันช่วย การวิเคราะห์และออกแบบมุมมองแบบคงที่ของแอปพลิเคชันและความสัมพันธ์ต่อกัน มันช่วยในการประเมิณว่า code ที่เราออกแบบนั้น มีความซับซ้อนในการทำงานไหม จนทำให้ในอนาคตเมื่อมีการแก้ไข Code นั้นแล้วอาจจะทำให้กระทบไปทั้งระบบ มันยังทำให้สามารถวิเคราะห์ความสัมพันธ์ของแต่ละโมดูลใน class ได้ เมื่อเราเห็นภาพรวมทั้งหมด ว่าอันไหนเกี่ยวข้องกับอันไหน เราจะได้ประเมินชนิดของ Coupling จากแย่ที่สุด ไป ดีที่สุดได้ เช่น Subclass Coupling : ความสัมพันธ์ระหว่าง SubClass กับ Super Class

ตัวอย่าง: Interface A มีความสัมพันธ์กับ Class B, C และ D

Temporal coupling : ใน Object มีกิจกรรม หรือหน้าที่ในการทำงานตั้งแต่ 2 อย่างเกิดขึ้นพร้อมกัน

ตัวอย่าง: Object E มี Method GetActiveUser กับ GetLastLogin ที่ถูกเรียกใช้งานพร้อมกันทุกๆ 1 วินาที

เพราะในการเขียน OOP ยังมีการเขียนโปรแกรมแบบ Procedural ประกอบด้วย เพราะ ถ้าเรายึดหลัก OOP เพียงอย่างเดียว เวลาเขียนโปรแกรมจริงๆคงมี Class จำนวนมหาศาล ระบบที่ดีความออกแบบให้มี Coupling ต่ำ (พยายามแยกให้ Object ต่างๆเป็นอิสระจากกันมากที่สุด)

8 การเพิ่มความสามารถconcurrency ของระบบ หรือความพร้อมกันซึ่งมีอยู่เมื่อใดก็ตามที่หลายหน่วยงานดำเนินการพร้อมกัน เป็นความจริงที่แพร่หลายและหลีกเลี่ยงไม่ได้ของชีวิตในระบบและวิศวกรรมซอฟต์แวร์ เพิ่มความซับซ้อนของระบบและซอฟต์แวร์อย่างมาก ซึ่งส่งผลกระทบโดยตรงต่อการทดสอบ ภาวะพร้อมกันนำไปสู่พฤติกรรมที่ไม่ได้กำหนดไว้และข้อบกพร่องของการทำงานพร้อมกันหลายประเภทที่ต้องใช้วิธีการเฉพาะในการเปิดเผย

ภาวะพร้อมกันมีอยู่ในรูปแบบหลักสามรูปแบบ ซึ่งแต่ละรูปแบบมีผลกระทบต่อการทดสอบต่างกันไป

ความแตกต่างระหว่างการทำงานพร้อมกันจริงและเสมือนนี้มีการแบ่งส่วนการทดสอบ โดยเฉพาะอย่างยิ่งเกี่ยวกับการทำงานพร้อมกันจริงในระบบแบบกระจายและ SoS การทำงานพร้อมกันที่เกิดขึ้นจริงนั้นซับซ้อนกว่าการทำงานพร้อมกันเสมือนอย่างมาก ทำให้ทั้งความสามารถในการควบคุมและการมองเห็นลดลง ซึ่งเป็นองค์ประกอบหลักสองประการของความสามารถในการทดสอบ ข้อบกพร่องของการทำงานพร้อมกันที่ก่อให้เกิดการละเมิดคำสั่งและไม่สามารถจัดการกับความผิดพลาด การขัดข้อง และการรีบูตได้อย่างถูกต้องนั้นพบได้บ่อยในระบบแบบกระจายมากกว่าในระบบโหนดเดียว

เช่น การเพิ่มขีดความสามารถ concurrency ของระบบมักจะทำให้เกิดพฤติกรรมที่ไม่กำหนดขึ้นเองได้ ซึ่งการดำเนินการด้วยอินพุตเดียวกันอาจนำไปสู่ผลลัพธ์ที่แตกต่างกัน ดังนั้นพฤติกรรมดังกล่าวจึงกลายเป็นความน่าจะเป็น และการทดสอบเดียวกันอาจให้ผลลัพธ์ที่แตกต่างกัน และ การสื่อสารระหว่างส่วนประกอบที่ทำงานพร้อมกันมักจะต้องมีการซิงโครไนซ์พฤติกรรมของพวกเขา อย่างไรก็ตาม จำนวนกรณีทดสอบเพิ่มขึ้นอย่างรวดเร็วด้วยจำนวนส่วนประกอบที่ทำงานพร้อมกัน เนื่องจากเวลาและลำดับของการแทรกระหว่างการดำเนินการที่เกิดขึ้นอาจเกิดขึ้นได้หลายวิธี

9 สำหรับองค์กรหลายแห่ง การรักษาความปลอดภัยมักทำให้เสียความสามารถในการใช้งาน ยิ่งพื้นที่ของงานมีความสำคัญมากเท่าไร ก็ยิ่งต้องได้รับการปกป้องที่เข้มงวดมากขึ้นเท่านั้น ซึ่งมักจะทำให้ผู้ปฏิบัติงานเสียดสีกันมากขึ้น

การเพิ่มรักษาความปลอดภัยใดๆ ที่ขัดขวางการใช้งานหรือทำให้ประสิทธิภาพการทำงานช้าลง มักจะทำให้ผู้คนมีความปลอดภัยน้อยลง ถ้าเพิ่ม Security มากขึ้น มันจะทำให้เกิดความยิ่งซับซ้อนและจะทำให้ Usability ใช้งานได้ยากมากขึ้น มันมีความเกี่ยวข้องกันอย่างใกล้ชิด และมักจะมีปฏิสัมพันธ์ในทางที่ไม่ดี ในแง่ความรู้สึก คุณลักษณะด้านความปลอดภัยเป็น 'อุปสรรค' สำหรับสิ่งอื่น ดังนั้นมันเป็นสิ่งสำคัญที่คุณจะต้องลบขั้นตอนเพิ่มเติมที่ไม่จำเป็นออกไป เพื่อไม่ให้ผู้ใช้ท้อถอย

เหตุของการเปลี่ยนแปลงเพราะว่าการโจมตีด้าน Security มีมากขึ้นอย่างมาก ทำให้นักพัฒนาต้องเพิ่ม Security ให้กับระบบ และสิ่งนั่นเองก็ทำให้เกิดปัญหาความยากให้กับผู้ใช้งานได้

10 SOA มักจะมีการปรับใช้ในองค์กรขนาดใหญ่ที่มีความซับซ้อน มีทั้งบริการต่างๆที่หลากหลายในองค์กร และต้องการให้มาทำงานร่วมกันได้ เป็นการ compose แอพลิเคชั่นจากความหลากหลายของเซอร์วิส

ซี่งโจทย์บอกว่า บริการสารสนเทศของแต่ละบริษัทมีความหลากหลาย การเชื่อมต่อเพื่อให้ระบบสามารถทำงานร่วมกันได้จึงมีความซับซ้อนสูง โดย SOA มาช่วยปรับใช้ในงานที่มีความซับซ้อน

โดยแต่ละบริษัทมีบริการหลากหลาย โดยไม่จำเป็นต้องเป็นต้องทราบรายละเอียดว่าบริการปลายทางดังกล่าวตั้งอยู่ที่ใด สามารถใช้งานเซอวิสโดยไม่จำเป็นต้องรู้ imprement detail ของ service provider แต่ต้องเข้าใจและใช้ได้ และทำการ combine การสร้าง application เข้าด้วยกัน

11 ใน Publish-subscribe pattern นั้นจะมีการทำงาน 3 ส่วนคือ Publisher, Subscriber และคนกลางโดยที่ Publisher คือผู้ส่ง message ต่าง ๆ โดยจะส่งไปที่คนกลางส่วนที่ไว้รับ message ต่าง ๆ เช่น Broker, Message broker หรือพวก Event bus , Queue messaging หรือ Middleware แน่นอนว่า Publisher นั้นจะไม่รู้จักกับ Subscriber เลย โดยเอาข้อมูลมาจากคนกลาง โดย คนกลางจะมีช่องทางให้ไปลงทะเบียนว่า Subscriber แต่ละตัวอยากจะทำงานกับอะไร ก็ไปลงทะเบียนไว้

เมื่อมี message หรือ event เข้ามาก็นำไปทำงานได้เลย ดังนั้นทั้ง Publisher และ Subscriber จึงไม่ต้องรู้จักกันเลย

จากโจทย์มีข่าวสารใหม่เพิ่มเติมตลอดเวลา และผู้ใช้บริการโทรศัพท์มือถือแต่ละคนมีความสนใจข่าวสารไม่เหมือนกัน คือเราไม่มีทางรู้ข้อมูลความสนใจของSubscriber แต่เราสามารถใช้วิธีแบบ Publish-subscribe pattern โดยเอาข้อมูลจากคนกลางได้และรองรับการส่งข่าวให้ผู้ใช้แต่ละรายตามที่แต่ละคนสนใจเท่านั้น

12 pipe filter หนึ่งในลักษณะพื้นฐานในสถาปัตยกรรมซอฟต์แวร์ที่ปรากฏซ้ำแล้วซ้ำอีกได้

Filters คือมีสามารถจัดการได้ด้วยตัวเอง, เป็นอิสระจาก filter อื่น ๆ และทำงานเป็น stateless โดยทั่วไป stateless filter ควรดำเนินการได้แค่หนึ่ง task เท่านั้นลำดับขั้นตอนการทำงานในแต่ละ task ควรได้รับการจัดการโดย handle filter ได้มากกว่าหนึ่งตัว

มี filter สี่ประเภทภายในสถาปัตยกรรมนี้:

Producer จุดเริ่มต้นของกระบวนการ outbound หรือบางครั้งเรียกว่า source

Transformer รับ input หรือเลือกดำเนินการแปลงข้อมูลบางส่วนหรือทั้งหมดจากนั้นส่งต่อไปยัง output pipe

Tester รับ input โดยทำการทดสอบตามเงื่อนไขที่กำหนด จากนั้นทำการสร้าง output ตามเงื่อนไขการทดสอบโปรแกรมเมอร์ที่ใช้งานจะรับรู้ว่าสิ่งนี้คล้ายกับการ reduce

Consumer จุดสิ้นสุดสำหรับการไหลของ pipeline บางครั้งผลลัพธ์ขั้นสุดท้ายของ output จะถูกนำไปเก็บยังฐานข้อมูลหรืออาจแสดงผลลัพธ์ขั้นสุดท้ายบนหน้าจอผ่าน UI ไปยังผู้ใช้งาน

โดยจากโจทย์สิ่งที่ต้องการคือ การทดสอบ และการติดตั้ง องค์กรจึงตัดสินใจสร้างระบบที่ช่วยจัดการกระบวนการซ้ำๆ และมีการไหลข้อมูลแบบทิศทางเดียว (undirectional) และความเรียบง่ายของแต่ละ pipe และ filter สามารถนำมากลับมาใช้ร่วมกันได้

13 peer to peer ในการใช้งานที่มีการกระจายศูนย์โดยจากหลายๆโฮสต์ แต่ละโหนดหรือโฮสต์เราให้ความสำคัญเท่าๆกัน พวกโหนดนี้ก็จะมีทรัพยากรเป็นของตัวเอง สามารถแชร์กันเองได้ โดยจะเห็นจากโจทย์ว่า โดยไม่ต้องผ่านระบบส่วนกลางขององค์กร แต่ผู้ใช้งานสามารถรับและส่งต่อข่าวที่ตัวเองได้รับไปยังผู้ใช้งานด้วยกันเองได้ มีปฏิสัมพันธ์กันเองระหว่างลูกค้า ก็เหมือนการที่แต่ละโหนดมีการส่งข้อมูลกันเองต่อกันเองได้

14 map reduce ประมวลข้อมูลที่เป็นจำนวนมากด้วยเวลาจำกัดหรือเวลาอันสั้น และรันได้อย่างมีประสิทธิภาพ ส่วนของ map เป็นฟังก์ชั่นทำงานได้แบบกระจายศูนย์หรือขนาด และช่วยให้ได้ output ในเวลาจำกัด และ divided across many machines ซึ่งจากโจทย์ ต้องการเก็บรวบรวมสถิติการส่งต่อข่าวของลูกค้าแบบ near real-time และมีปริมาณข้อมูลจำนวนมากโดยเป็นลักษณะการกระจายศูนย์

15 MVC ใช้เพื่อแยกส่วนซอฟต์แวร์ในส่วน ตรรกะเนื้อหา (domain logic) ได้แก่ความเข้าใจในระบบของผู้ใช้ และส่วนการป้อนข้อมูลและแสดงผล (GUI) ซึ่งช่วยให้การพัฒนา การทดสอบ และการดูแลรักษาซอฟต์แวร์ แยกออกจากกัน

MVC มักจะพบได้ในเว็บแอปพลิเคชันโดย วิว จะเป็น HTML หรือ XHTML ที่สร้างโดยแอปพลิเคชันนั้น ส่วนคอนโทรลเลอร์รับค่า GET หรือ POST เข้ามา แล้วเลือกติดต่อกับโมเดลในส่วนที่เกี่ยวข้องเพื่อตอบสนอง โมเดลซึ่งมี business rules จะทำการจัดการตามคำร้องขอนั้นๆ จะสร้างการแสดงผลที่เหมาะสมแก่ผู้ใช้ ปรับปรุงเปลี่ยนแปลงได้เสมอตามความต้องการ

โดยจากโจทย์ความต้องการของผู้บริหารและบริษัทคู๋ค้ามีการเปลี่ยนแปลงเรื่อยๆ จึงมีความจำเป็นต้องปรับเปลี่ยนรูปแบบการนำเสนออยู่บ่อยครั้ง และนำเสนอจัดทำ visualization แบบ Web Application

16 คือข้อกำหนดที่มีผลต่อ สถาปัตยกรรมของระบบคอมพิวเตอร์ที่สามารถวัดได้ ซึ่งอาจประกอบด้วยข้อกำหนดทั้งซอฟต์แวร์และฮาร์ดแวร์ เป็นชุดย่อยของข้อกำหนดซึ่งเป็นชุดย่อยที่ส่งผลต่อสถาปัตยกรรมของระบบในลักษณะที่สามารถระบุได้

ความสำคัญ ใช้วัดผลต่อสถาปัตยกรรมของระบบ เป็นข้อกำหนดที่มีบทบาทสำคัญในการกำหนดสถาปัตยกรรมของระบบ ข้อกำหนดดังกล่าวต้องการความเอาใจใส่เป็นพิเศษ ข้อกำหนดบางข้อไม่ได้มีความสำคัญเท่าเทียมกันในเรื่องสถาปัตยกรรม

สามารถหา ASR ได้จาก User Requirements สัมภาษณ์จากผู้มีส่วนได้ส่วนเสียกับโปรเจค หาจากเป้าของธุรกิจ

17 ATAM มีทั้งหมด 4 Phases ได้แก่

Phase 0 คือ ช่วงของการให้ความร่วมมือ และการเตรียมความพร้อม โดยจะมีหัวหน้าทีม และผู้มีอำนาจในการตัดสินโครงการประชุมกันเพื่อหารายละเอียด

Phase 1 & Phase 2 คือ ช่วงของการประเมิน และวิเคราะห์ของ ATAM ประกอบด้วย 9 ขั้นตอน แบ่งเป็น

- ขั้นที่ 1-6 อยู่ใน Phase 1 โดยมีทีมประเมิน และผู้มีอำนาจตัดสินใจโครงการ (ทีมสถาปัตยกรรม, ผูู้จัดการโครงการ, ผู้สนับสนุนโครงการ)

- ขั้นที่ 7-9 อยู่ใน Phase 2 จะเริ่มต่อจาก Phase 1 โดยจะมีผู้มีส่วนได้ส่วนเสียทั้งหมดนำเสนอ

Phase 3 คือ ช่วงของการติดตามงาน ทีมประเมินผลจะจัดทำ และส่งรายงานขั้นสุดท้าย และรายงานจะส่งแสดงให้กับผู้มีส่วนได้ส่วนเสียหลัก เพื่อให้แน่ใจว่าไม่มีข้อผิดพลาด หลังจากตรวจสอบรายงานจะถูกส่งให้ผู้มอบหมายให้การประเมิน

18 Phase 1 step 5 : สร้าง QA ในรูปของ Utility tree ให้ requirement เป็นรูปธรรม