howto

บางทีการแก้ปัญหาไม่ได้ อาจไม่ใช่เพราะขาดวิธีรับมือ แต่เพราะขาด “วิธีคิด”

ไม่ต้องหาร้อยวิธีแก้ แค่คิดอย่างเป็นระบบ

世界一流エンジニアの思考法

อุชิโอะ ซึโยชิ
เขียน

กมลวรรณ เพ็ญอร่าม
แปล

7 สุดยอดทักษะสำคัญที่แก้ได้ทุกโจทย์ของชาวออฟฟิศ
ด้วยวิธีคิดจากวิศวกรระดับแนวหน้าของไมโครซอฟท์
เจ้าของรางวัล IT Engineer Book Award 2025

ขายไปแล้วกว่า 100,000 เล่ม ในญี่ปุ่น

howto

ไม่ต้องหาร้อยวิธีแก้ แค่คิดอย่างเป็นระบบ

世界一流エンジニアの思考法

อุชิโอะ ซึโยชิ

เขียน

กมลวรรณ เพ็ญอร่าม

แปล

การอ่านคือรากฐานที่สำคัญ

ไม่ต้องหาร้อยวิธีแก้ แค่คิดอย่างเป็นระบบ
世界一流エンジニアの思考法

howto
ในเครือบริษัทอมรินทร์ คอร์เปอเรชั่นส์ จำกัด (มหาชน)
378 ถนนชัยพฤกษ์ (บรมราชชนนี) เขตตลิ่งชัน กรุงเทพฯ 10170
โทรศัพท์ 0-2422-9999 ต่อ 4964, 4969 E-mail: info@amarin.co.th
www.amarinbooks.com **@amarinbooks** **Amarin HOW-TO**

เลขมาตรฐานสากลประจำหนังสืออิเล็กทรอนิกส์ 978-616-18-8445-1

เจ้าของ ผู้พิมพ์ผู้โฆษณา บริษัทอมรินทร์ คอร์เปอเรชั่นส์ จำกัด (มหาชน)
กรรมการผู้อำนวยการใหญ่ ศิริ บุญพิทักษ์เกศ • กรรมการผู้จัดการ อุษณีย์ วิรัตกพันธ์
รองกรรมการผู้จัดการ ศศกร วัฒนาสุทธิวงศ์ • ที่ปรึกษา องอาจ จิระอร
ที่ปรึกษากลุ่มนอนฟิกชั่น มนทิรา ภูปากน้ำ • บรรณาธิการ ณิฌภัทร เขาแก้ว
บรรณาธิการต้นฉบับ พรรณรวี อกนิษฐาภิชาติ • ผู้จัดการฝ่ายการผลิต อมราลักษณ์ เชยกลิ่น
ศิลปกรรมปก สิริพงษ์ กิจวัตร • ออกแบบรูปเล่ม เกติพิบูล โหมดตาด • คอมพิวเตอร์ นงนุช ศรีสุขโข
พิสูจน์อักษร ปทุม ปทุมนาท, ปริณดา สร้อยคำ

คำนำสำนักพิมพ์

หนังสือเล่มนี้ถ่ายทอดประสบการณ์จริงของผู้เขียน อุชิโอะ ซึโยชิ วิศวกรไอทีจากญี่ปุ่นที่พัฒนาตนเองจนได้เข้าร่วมทีมวิศวกรซอฟต์แวร์อาวุโสของบริษัท Microsoft ประเทศสหรัฐอเมริกา

การได้ร่วมงานกับคนเก่งระดับโลกรอบตัวทำให้เขาค้นพบว่าสิ่งที่ทำให้มืออาชีพเหล่านั้นอยู่รอดและโดดเด่นในทุกสภาพแวดล้อมได้นั้น ไม่ใช่เพราะพวกเขามีพรสวรรค์อะไรเป็นพิเศษหรือฉลาดล้ำเกินคนทั่วไป แต่สิ่งหนึ่งที่พวกเขาแทบทุกคนมีเหมือนกันคือ **"วิธีคิดอย่างเป็นระบบ"** ซึ่งเป็นเครื่องมือที่ช่วยให้มองเห็นปัญหาในมุมที่กว้างขึ้น เชื่อมโยงข้อมูลได้อย่างมีเหตุผล และหาทางออกได้อย่างมีประสิทธิภาพ

ด้วยความตั้งใจจะถ่ายทอดสุดยอดทักษะที่ทำให้ทุกคนสามารถ "ควบคุมชีวิตและการทำงานได้ด้วยตนเอง" หนังสือเล่มนี้จึงไม่ได้จำกัดการใช้งานอยู่แค่เฉพาะในสายงานวิศวกรรมหรือเทคโนโลยีเท่านั้น หากแต่เป็นระบบความคิดและกระบวนการทำงานที่ได้รับการพิสูจน์แล้วว่าสามารถนำไปปรับใช้ได้กับทุกอาชีพ ทั้งในระดับบุคคล ระดับทีม ไปจนถึงระดับองค์กร ซึ่งสุดท้ายแล้ววิธีคิดเหล่านี้จะกลายเป็นแต้มต่อสำคัญในการก้าวข้ามปัญหาและเผชิญหน้ากับทุกความท้าทายในทุก ๆ ระดับได้เป็นอย่างดี

หนึ่งในเคล็ดลับที่ผู้ประสบความสำเร็จในสายอาชีพต่าง ๆ กล่าวไว้ตรงกันโดยมิได้นัดหมายคือ คนที่จะอยู่รอดไม่ใช่คนที่เก่งที่สุด แต่คือคนที่คิดเป็นที่สุด

คุณเองก็เป็นหนึ่งในนั้นได้เช่นกัน

howto

สารบัญ

บทที่ 2

ชุดความคิดที่พบในอเมริกา

— สิ่งที่ไม่เคยสังเกตเห็นตอนอยู่ญี่ปุ่น

บทที่ 3

เทคนิคจัดระเบียบข้อมูลและการจำ เพื่อสร้างพื้นที่ว่างในสมอง

— สุดยอดเคล็ดลับจากเพื่อนร่วมงานขั้นเทพ

บทที่ 4

สุดยอดเคล็ดลับการสื่อสาร

— วิธีถ่ายทอด วิธีรับฟัง และแลกเปลี่ยนความคิดเห็น

บทที่ 5

สร้างทีมเพื่อเพิ่มผลิตภาพ

— มุ่งสู่ “ผู้นำแบบผู้รับใช้” และ “ทีมแบบจัดการตนเอง”

บทที่ 6

เทคนิคใช้ชีวิตประจำวัน ยกระดับคุณภาพชีวิตและงาน

— ตั้งแต่ระบบ “ไทม์บ็อกซิง” ไปจนถึงเวทเทรนนิง

บทที่ 7

เราจะอยู่รอดในยุค AI ได้อย่างไร

— ทักษะการปรับตัวต่อการเปลี่ยนแปลง และข้อเสนอแนะการเลิก “วัฒนธรรมการวิพากษ์วิจารณ์”

บทนำ

ทุกท่านที่หยิบหนังสือเล่มนี้ขึ้นมาคงสนใจเทคนิคการทำงานของวิศวกรระดับแนวหน้าที่จะ “ช่วยสร้างผลงานให้โดดเด่น” และต้องการเพิ่มผลิตภาพในแต่ละวันให้พุ่งพรวด

ปัจจุบันผมทำงานที่บริษัทไมโครซอฟท์ในตำแหน่งวิศวกรซอฟต์แวร์อาวุโส ที่เมืองซีแอตเทิล รัฐวอชิงตัน สหรัฐอเมริกา สังกัดทีม Azure Functions ทำหน้าที่พัฒนาระบบภายในบริการคลาวด์ผู้คนทั่วโลกกำลังใช้บริการที่พวกเราสร้างขึ้นในการพัฒนาโครงสร้างพื้นฐานของซอฟต์แวร์ หากเป็นเมื่อก่อนผมจะคิดว่า รู้สึกเหมือนฝันเลยที่ได้ทำงานเป็นสมาชิกคนหนึ่งของทีมนี้

พูดตรง ๆ คือ ผมไม่ใช่ “วิศวกรแนวหน้า” ครับ

ผมไม่ได้ถ่อมตัวแต่อย่างใด เพราะเป็น “คนระดับสาม”[1] จริง ๆ แต่แน่นอนว่าผมกำลังพยายามอย่างหนักเพื่อจะได้ก้าวขึ้นเป็น “ระดับ

[1] การเปรียบเทียบกับการจัดลำดับคุณภาพ โดยระดับหนึ่งคือยอดเยี่ยม ระดับสองคือปานกลาง และระดับสามคือต่ำหรือไม่ได้มาตรฐาน

หนึ่งหรือระดับแนวหน้า" ในอนาคต ที่ผมได้ตำแหน่งนี้มาไม่ใช่เพราะมีความสามารถสูงส่ง แต่ผมมาอยู่ตรงนี้ได้ด้วยความบังเอิญและจากการสั่งสมประสบการณ์เล็ก ๆ น้อย ๆ

ตั้งแต่สมัยเด็ก ๆ ผมเป็นคนที่ทำอะไรไม่ได้เรื่องสักอย่าง แต่ถึงอย่างนั้นก็ชอบคอมพิวเตอร์เอามาก ๆ จึงเล่นพ็อกเก็ตพีซีหรือ MZ-2500 ของชาร์ป โดยคัดลอกโค้ดเกมจากนิตยสารมาเล่น ทว่าเวลาที่ต้องจัดการอะไรสักอย่าง ผมต้องใช้ความพยายามมากกว่าคนทั่วไป 3 เท่า ถึงจะอยู่ใน "ระดับเดียวกับคนอื่น" ได้

ทั้งที่ไม่ได้มีพรสวรรค์อะไรเป็นพิเศษ แต่ผมก็ใฝ่ฝันอยากเป็นโปรแกรมเมอร์มาตลอดตั้งแต่เด็ก ตอนทำงานกับบริษัทแรกในญี่ปุ่น ผมกระตือรือร้นเสนอตัวว่า "อยากรับผิดชอบด้านคำสั่ง UNIX ครับ!" แต่ก็ไม่สมหวัง และต้องทำงานอยู่ฝ่ายขายถึง 5 ปี แล้ววันหนึ่งในที่สุดผมก็ได้รับโอกาสให้ทำงานในตำแหน่งวิศวกรระบบ ทว่าแต่ละวันกลับดำเนินไปอย่างไร้ความหมายเพราะผมเขียนโปรแกรมไม่เป็นเลย...

จริง ๆ แล้วพอโตขึ้น ผมได้รับการวินิจฉัยว่าเป็นโรคสมาธิสั้น (ADHD) ผมต้องทรมานอยู่หลายปีกับอาการเงอะงะ ความจำสั้น ความคิดฟุ้งซ่าน จัดระเบียบไม่ได้จนเหนื่อยล้า ดังนั้นจึงตั้งใจศึกษาวิธีเพิ่มผลิตภาพในการทำงานว่า **ต้องทำสิ่งที่ไม่ถนัดอย่างไรจึงจะมีประสิทธิภาพเทียบเท่าคนอื่น**

ด้วยเหตุนี้ ผมจึงสังเกตเห็นจุดที่มักจะถูกมองข้ามได้มากกว่าคนอื่น เช่น มือใหม่จะสะดุดตรงไหน จะเสียเวลาไปกับอะไรโดยเปล่าประโยชน์ จะทำอย่างไรเพื่อยกระดับพื้นฐานให้ดีขึ้นแบบง่าย ๆ

แม้ผมจะพึ่งพาไม่ได้ แต่พอสั่งสมประสบการณ์การทำงานมากขึ้น ก็พบว่าตัวเองมีด้านที่พอจะแสดงศักยภาพได้อยู่เหมือนกัน

ประมาณปี 2001 ผมพบวิธีการที่เรียกว่า “อไจล์” (Agile) ซึ่งช่วยเพิ่มความเร็วในการพัฒนาซอฟต์แวร์อย่างก้าวกระโดด ผมทึ่งสุด ๆ จนอยากติดอาวุธการทำงานให้ตัวเองบ้าง จากนั้นจึงเข้าร่วมคอมมิวนิตีดังกล่าวและศึกษาอย่างจริงจัง เมื่อสั่งสมประสบการณ์จนกลายเป็นโค้ชด้านอไจล์ ก็ได้รู้ว่าเราจะประสบความสำเร็จถ้า “แบ่งงานให้คนอื่นทำ” ตามตำแหน่งต่าง ๆ เช่น ที่ปรึกษา ผู้จัดการโปรเจกต์ อีแวนเจลิสต์ (evangelist: อาชีพที่อธิบายและให้ความรู้เกี่ยวกับเทรนด์และเทคโนโลยี IT ให้เข้าใจง่าย) โดยไม่โครซอฟท์ได้ว่าจ้างให้ผมเป็นอีแวนเจลิสต์ด้วย จึงมีโอกาสได้ไปบรรยายในงานประชุมระดับนานาชาติอยู่บ่อยครั้ง

อย่างไรก็ตาม “งานที่ต้องลงมือทำเอง” กลับไม่สำเร็จสักอย่าง แต่เพราะผมใฝ่ฝันอยากเป็น “โปรแกรมเมอร์” มานาน จึงลองท้าทายทำงานนี้อยู่หลายครั้ง แต่ทุกครั้งคนอื่น ๆ มักจะบอกว่า “ถ้าเป็นตำแหน่งผู้จัดการโปรเจกต์ละก็ คุณอุชิโอะมีพรสวรรค์อยู่นะ น่าจะไปทำงานนี้” หรือ “ทำไมไม่เป็นผู้เชี่ยวชาญด้านอีแวนเจลิสต์ไปเลยล่ะ”

ทว่างานที่ผมอยากทำจากใจจริงคือ “โปรแกรมเมอร์” ที่พัฒนาซอฟต์แวร์ ความใฝ่ฝันอันแรงกล้าไม่เคยดับสูญ แม้จะรู้ตัวดีว่าไม่มีพรสวรรค์ แต่ถึงอย่างไรผมก็อยากเป็นให้ได้ ดังนั้นจึงหมั่นเพียรพยายามและปรับเปลี่ยนจนได้ย้ายไปทำงานที่ไมโครซอฟท์ในวัย 44 ปี จนถึงตอนนี้ก็เป็นเวลา 4 ปีแล้วที่ผมมาทำงานที่อเมริกาซึ่งเป็นศูนย์กลางด้านนี้ในฐานะวิศวกรคลาวด์ชาวญี่ปุ่นที่มีอยู่เพียงไม่กี่คน

ทีมที่ผมทำอยู่กำลังพัฒนาระบบคลาวด์ระดับโลก จึงมีแต่เหล่าวิศวกรระดับแนวหน้าของโลกโดยแท้ ผมทึ่งกับภาพที่พวกเขาพัฒนาบริการจนกลายเป็นข่าวดังได้อย่างไม่ขาดตกบกพร่องพร้อมกับทำงานอย่างสนุกสนานไปด้วย

ทำไมพวกเขาถึงทำงานได้เหนือชั้นขนาดนี้

อันที่จริงใช่ว่าทุกคนในทีมจะมีไหวพริบมากกว่าคนทั่วไป แล้วก็ใช่ว่าจะมีความจำระดับอัจฉริยะด้วย

แต่โดยหลักการแล้ว **"ชุดความคิด" (mindset) ต่างหากคือสิ่งที่สร้างผลิตภาพสูงให้ออกมาเป็นรูปเป็นร่าง** ไม่ใช่ทั้งเทคนิคเล็ก ๆ น้อย ๆ หรือเคล็ดลับใด ๆ แต่เป็นความจริงที่ว่าประสิทธิภาพเหนือชั้นล้วนเกิดจากชุดความคิด พอผมได้ทำงานกับพวกเขาจึงเข้าใจแก่นแท้ดังกล่าว และสไตล์การทำงานก็เปลี่ยนไปมากด้วย เรียกว่าทำให้ "ชีวิตเปลี่ยนไป" จริง ๆ

แม้ที่ญี่ปุ่นผมแทบจะไม่เป็นที่ยอมรับในฐานะ "โปรแกรมเมอร์" **แต่การเลียนแบบชุดความคิดของพวกเขาแล้วนำมาทำตามอย่างใส่ใจ ก็ทำให้ผมได้ทำงานที่อเมริกาและเป็นส่วนหนึ่งของทีมในฝันที่เจ๋งที่สุดในโลก** นับเป็นปาฏิหาริย์สำหรับวิศวกรระดับสามที่เคยมีความเชื่อผิด ๆ มาตลอดหลายปีว่าตัวเองทำไม่ได้

ในหนังสือเล่มนี้ ผมจะถ่ายทอดเทคนิคต่าง ๆ ว่าผมทำอย่างไรถึงได้ฝึกฝนชุดความคิดจากเหล่าวิศวกรแนวหน้าจนสามารถเพิ่มประสิทธิภาพและคุณภาพในการทำงานจนผลักดันตัวเองขึ้นมาอยู่แถวหน้าได้สำเร็จ

หวังว่าจะเป็นประโยชน์ไม่เพียงต่อคนที่ตั้งเป้าจะประสบความสำเร็จในฐานะวิศวกรเท่านั้น แต่ยังรวมถึงคนที่ผิดหวังกับความ

ไร้ประสิทธิภาพของตัวเอง คนที่อยากเก่งขึ้น และคนที่อยากเพิ่มมูลค่าให้ตัวเองในตลาดโลกเหมือนกับตัวผม

ปัจจุบันจะเห็นว่าหลายอุตสาหกรรมในญี่ปุ่นกำลังประสบกับภาวะ “ขาดทุนมหาศาล” เนื่องจาก “ผลิตภาพต่ำ” เรื่องนี้ไม่ใช่แค่ความล่าช้าจากการปรับตัวสู่ยุคดิจิทัล แต่ยังมีที่ว่างอีกมากให้ปรับปรุงวิธีคิดและการสร้างทีมซึ่งถือเป็นหัวใจสำคัญของผลิตภาพ

ประเด็นสำคัญที่ซ่อนอยู่ในหนังสือเล่มนี้คือ ความคาดหวังอยากให้ญี่ปุ่นเร่งนำเทคโนโลยีและกระบวนการใหม่ล่าสุดอย่างอไจล์และ DevOps มาใช้ให้ทัดเทียมกับอเมริกา

โดยเฉพาะอย่างยิ่งสำหรับนักธุรกิจและองค์กรที่มุ่งหวังจะนำเทคโนโลยีมาใช้ให้รวดเร็วเทียบเท่าบริษัทระดับโลก ผมจึงอยากให้คุณได้เรียนรู้ชุดความคิดแบบวัฒนธรรมตะวันตกเพื่อเร่งเพิ่มผลิตภาพ

ทุกวันนี้เครื่องมือต่าง ๆ ในการทำงานพัฒนาอย่างรวดเร็วจนตามไม่ทัน คำถามจึงอยู่ที่ว่ามนุษย์จะรับมือเทคโนโลยีอย่างไร และจะนำมาใช้ให้เกิดประโยชน์สูงสุดได้อย่างไร

อาจกล่าวได้ว่า ทักษะที่เป็นหัวใจสำคัญของหนังสือเล่มนี้คือ **“วิธีคิดเพื่อเอาตัวรอดในยุค** AI”

ผมเชื่อว่า หากคุณเข้าใจและนำความรู้ในหนังสือเล่มนี้ไปปรับใช้อย่างเชี่ยวชาญ คุณจะสามารถทำงานร่วมกับคนได้หลากหลาย คิดอย่างเป็นระบบ สื่อสารได้อย่างราบรื่น แม้แต่ในวงการซอฟต์แวร์ที่เต็มไปด้วยแรงกดดันและการเปลี่ยนแปลงที่รวดเร็ว ก็ยังสามารถโดดเด่นออกมาได้อย่างแน่นอน

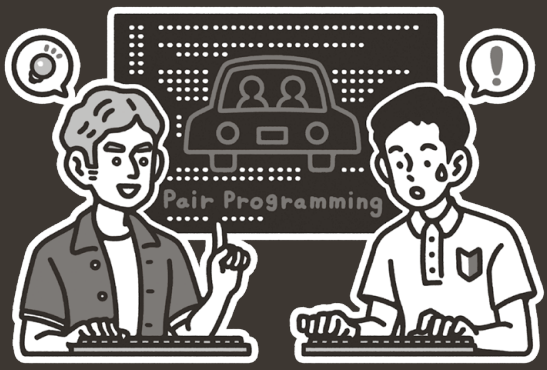
แม้ตัวอย่างที่ยกมาในเล่มอาจเป็นประเด็นเฉพาะทางเสียมาก แต่หวังว่าแก่นแนวคิดนี้จะเป็นเคล็ดลับที่ช่วยเพิ่มผลิตภาพให้คุณได้ ไม่ว่าจะอยู่ในวงการไหนหรือสายอาชีพใดก็ตาม

ขอให้ทุกท่านได้เข้าใกล้ “ความฝัน” ของตัวเองมากยิ่งขึ้นครับ

บทที่ 1

วิศวกรแนวหน้าของโลกต่างจากเราตรงไหน

— ความลับของการมีผลิตภาพสูง



ความแตกต่างเรื่อง “ผลิตภาพสูง”

สิ่งแรกที่ผมประหลาดใจเมื่อได้เข้าร่วมทีมพัฒนาบริการคลาวด์ชื่อ Azure Functions ของไมโครซอฟท์คือ การมี “ผลิตภาพสูง” แบบเหนือชั้น **ซึ่งไม่ได้มีคนเก่งเพียงบางส่วนที่พยายามอย่างโดดเด่น แต่ทุกคนในทีมต่างมีผลิตภาพสูงผิดปกติเป็นพื้นฐานอยู่แล้ว**

วิธีพัฒนาไม่มีกฎเกณฑ์มาตรฐานเหมือนในญี่ปุ่น โดยพื้นฐานทุกคนมีความรู้ด้านวิทยาการคอมพิวเตอร์กันอยู่แล้ว จึงเป็นระบบที่แต่ละคนจะต้องคิด ลงมือทำ และตัดสินใจด้วยตัวเอง

ในบริการคลาวด์ของไมโครซอฟท์ เมื่อคลิกไอคอนเว็บไซต์ที่เรียกว่าพอร์ทัล (portal) แล้วตั้งชื่อให้มัน จะสามารถสร้างสภาพแวดล้อมสำหรับรันโปรแกรมได้อย่างง่ายดาย หากเป็นเมื่อก่อน แค่เตรียมเซิร์ฟเวอร์อาจใช้เวลาถึงครึ่งปี แต่ปัจจุบันเราสามารถพัฒนาแอปพลิเคชันและสร้างเว็บบริการแบบเซิร์ฟเวอร์เลส (serverless) ได้

จากมุมผู้ใช้งานอาจเห็นว่า “ไอคอน” ดูใช้ง่ายสุด ๆ แต่ภายในนั้นคือระบบย่อยขนาดเล็กรวมกันอย่างซับซ้อน เรียกว่า “ไมโครเซอร์วิส” เนื่องจากเป็นแพลตฟอร์มที่องค์กรใหญ่ ๆ ทั่วโลกใช้งาน หากระบบนี้

หยุดทำงานจะส่งผลกระทบอย่างร้ายแรงต่อระบบของแต่ละบริษัท ดังนั้นจึงเป็นระบบที่ต้องมีความน่าเชื่อถือเป็นอย่างมาก

ตอนเข้าร่วมทีมครั้งแรกในเดือนเมษายน ปี 2020 ผมตกใจมาก เมื่อรู้ว่าส่วนที่เปิดเผยแบบสาธารณะใน GitHub[1] เป็นเพียงปลายยอดภูเขาน้ำแข็ง เพราะมีโครงสร้างขนาดยักษ์ที่ซับซ้อนสร้างอยู่ภายในนั้นด้วย

ถึงแม้ Azure Functions จะเป็นระบบที่ละเอียดอ่อนขนาดนั้น แต่ทุกวันยังมีวิศวกรจำนวนมากคอยเพิ่มฟีเจอร์ อัปเดต และดีบั๊กด้วยความเร็วอย่างน่าเหลือเชื่อ ในบริษัทของญี่ปุ่นที่ผมเคยทำงานเมื่อก่อนไม่ค่อยกล้าเพิ่มฟีเจอร์หรืออัปเดตระบบที่กำลังใช้งาน แต่ที่นี่เราอัปเดตกันอยู่เรื่อย ๆ

จู่ ๆ ผมก็นึกถึงสิ่งที่เพื่อนร่วมงานเคยพูดขึ้นได้ว่า

“พอเข้าทีมโปรดักต์แล้ว ให้เตรียมใจไว้เลยว่า ในปีแรกคุณจะไม่มีประโยชน์อะไรเลย”

ทั้ง ๆ ที่ระบบภายในซับซ้อนถึงขั้นนั้นและเปลี่ยนแปลงเร็วมาก แต่ตัวผมกลับทำงานช้ากว่าคนรอบข้าง ทำอะไรเงอะงะและต้องใช้เวลานาน ปัญหาเร่งด่วนของผมคือต้องพัฒนาตัวเองให้ได้เกณฑ์มาตรฐานของทีม **ความรู้สึกว่า “ฉันทำบางอย่างได้” มีน้อยมากจนเห็นคุณค่าในตัวเองต่ำ** สิ่งเหล่านี้เป็นปัญหาอันดับหนึ่งที่ผมอยากแก้ไขในชีวิต

[1] แพลตฟอร์มการพัฒนาซอฟต์แวร์ที่ผู้คนทั่วโลกสามารถจัดเก็บและเผยแพร่โค้ดโปรแกรมและข้อมูลการออกแบบของตัวเอง

วันหนึ่งผมมีโอกาสทำงานกับเพื่อนร่วมงานชื่อพอล เขาเป็นสมาชิกทีมพัฒนาสถาปัตยกรรม (การออกแบบระบบสารสนเทศ) มาตั้งแต่ช่วงแรกเริ่ม เรียกได้ว่าเป็นผู้ชายที่ฉลาดที่สุดเท่าที่ผมเคยเจอมาในชีวิตนี้ ไม่ว่าจะทำอะไร เขาก็รวดเร็วและรอบคอบ ไม่ว่าจะเกิดปัญหาแบบไหนก็แก้ไขได้ทันที ความสามารถทางเทคนิคก็ยอดเยี่ยมจนน่าทึ่ง แถมเขายังเขียนบทความวิชาการใหม่ๆ ไว้หลายฉบับอีกด้วย

ตอนที่ผมเริ่มทำงานกับเขา ผู้จัดการเคยบอกกับผมด้วยท่าทางภูมิใจว่า “ดีใจไหมที่ได้ทำงานกับวิศวกรแนวหน้าของโลก” ผมตอบทันทีว่า “แน่นอนครับ” นี่แสดงให้เห็นว่าเขาเป็นวิศวกรที่ได้รับการยกย่องมากขนาดไหนในบริษัท

พอลหัวไวกว่าผมมากและยังความจำดีสุดๆ แต่ขณะเดียวกันผมก็เชื่อมั่นว่า

“เขาฉลาดที่สุดเท่าที่เคยเจอมาในชีวิต แต่ถ้าเทียบกับตัวเราแล้วก็ใช่ว่าสมองของเขาจะมีประสิทธิภาพต่างกันจนถึงขั้นตามไม่ทัน”

ในหน้าต่อๆ ไป ผมจะพาคุณไปตามรอยกระบวนการคิดของเขาที่แม้แต่ผมก็ยังทำตามได้

วิธีแก้ปัญหาสุดทึ่ง ของวิศวกรระดับแนวหน้า

วันหนึ่งโปรแกรมของผมทำงานติดขัด แน่นอนว่าผมจะแก้เองก็ได้ แต่จากประสบการณ์ที่ผ่านมา ผมคงต้องใช้เวลาหาสาเหตุหลายชั่วโมงหรือเป็นวัน

ผมสงสัยว่าหากเป็นพอลจะคิดแบบไหน จึงตัดสินใจขอให้เขาช่วย “**เขียนโปรแกรมเป็นคู่**” (pair programming) วิธีนี้เป็นเทคนิคการเขียนโปรแกรมร่วมกัน 2 คนโดยใช้คอมพิวเตอร์ 1 เครื่อง นับเป็น 1 กลุ่ม ซึ่งช่วยให้ได้เรียนรู้เทคนิคของกัน รวมทั้งทำความเข้าใจและแชร์ที่มาที่ไปของโค้ดได้อย่างมีประสิทธิภาพ

ผมจึงบอกพอลว่า

“โปรแกรมผมมีปัญหา ผมคิดว่าตัวเองน่าจะมีข้อบกพร่องบางอย่างเลยทำงานได้ไม่ค่อยดี ช่วยเขียนโปรแกรมเป็นคู่หน่อยได้ไหม ผมอยากเรียนรู้ว่าถ้าเป็นคุณจะทำยังไง”

เขาตอบกลับมาว่า “ผมไม่รู้ว่าคุณอยากรู้เรื่องอะไร แต่ได้นะ”

แล้วช่วงปรึกษาหารือก็เริ่มต้นขึ้น

ผมเห็น “ล็อก” (log) ที่แสดงว่าโปรแกรมของตัวเองมีปัญหา ล็อก คือบันทึกข้อมูลว่าภายในโปรแกรมทำงานอย่างไร หากดูสิ่งนี้แล้วจะตรวจสอบได้ว่าโปรแกรมทำงานถูกต้องไหม และมีปัญหาเกิดขึ้นหรือเปล่า

สำหรับผมในเวลาปกติ เมื่อเจอปัญหาในล็อก นั่นจะเป็นช่วงได้ลองผิดลองถูกว่า “ตรงนี้น่าจะเป็นสาเหตุ” “ไม่สิ อาจจะเป็นบั๊กตรงนั้น” ผมจะเดาจุดบกพร่องและแก้ไขโดยย้ายโปรแกรมไปยังเซิร์ฟเวอร์แล้วรันให้ทำงาน จากนั้นดูล็อกอีกครั้ง แต่กว่าจะแสดงผลก็ใช้เวลาประมาณ 15 นาที ดังนั้นแค่ผมทดลองแก้ปัญหาด้วยตัวเองเพียงไม่กี่วิธีก็ใช้เวลาค่อนข้างนานแล้ว

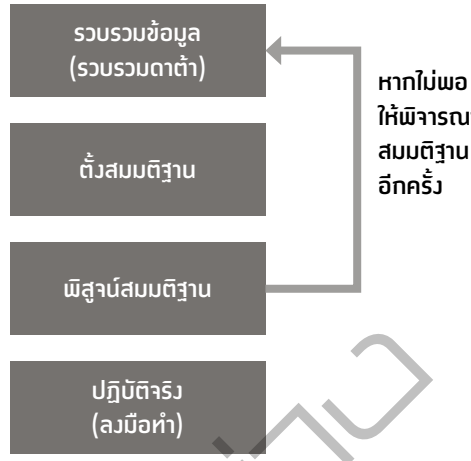
แต่สิ่งที่พอลทำตรงข้ามกับผมโดยสิ้นเชิง **เขาดูแค่ล็อกแรกอันเดียว แล้วเริ่มตั้ง “สมมติฐาน” โดยที่ยังไม่ได้ลงมือทำแม้แต่นิดเดียว**

“ที่ล็อกนี้แสดงขึ้นมา ผมเดาว่าน่าจะเกิดบางอย่างขึ้นข้างในพอดูให้ดีแล้ว สิ่งที่เราควรเช็กคือ...” เขาพึมพำพร้อมกับเปิดซอฟต์แวร์ดูฐานข้อมูล จากนั้นเขียนคำสั่งไปยังระบบแค่คำสั่งเดียว แล้วพูดขึ้นมาว่า

“น่าจะเป็นตรงนี้”

เหลือเชื่อว่าผลลัพธ์ของคำสั่งชี้ไปยังสาเหตุที่แท้จริงของปัญหาในโปรแกรมของผมได้อย่างแม่นยำ โดยที่เขาลงมือแค่ครั้งเดียว

ปัญหาที่ผมต้องใช้เวลาเช็กโค้ดนั่นนี่อยู่หลายชั่วโมง เขากลับแก้ได้ในชั่วพริบตาราวกับใช้เวทมนตร์



รูปที่ 1 จงพิสูจน์สมมติฐานก่อนลงมือทำ

ผมซ็อกมาก เขาแทบไม่มีประสบการณ์เกี่ยวกับโค้ดเบส (codebase) นั้นเลย ผมต่างหากที่รู้เรื่องโค้ดที่เขียนในโปรแกรมนั้นมากกว่า

ในโลกซอฟต์แวร์มีคำพูดที่ว่า **โปรแกรมเมอร์ที่เก่งกับไม่เก่งแตกต่างกันถึง 25 เท่า** เหตุการณ์นี้ทำให้ผมได้เจอกับตัว แต่ก็ใช่ว่าสิ่งที่เขาทำจะเป็นไปไม่ได้สำหรับมนุษย์คนอื่น ผมจึงนึกถึงสิ่งที่บาร์ราเพื่อนร่วมงานเคยพูดไว้

“เวลาตรวจสอบปัญหา ไม่ควรลงมือทันที ใส่หลาย ๆ คำสั่งเพื่อลองผิดลองถูก แต่เราต้องดูล็อกแล้วคาดเดาเองก่อนว่าน่าจะเกิดอะไรขึ้น จากนั้นถึงพิสูจน์ด้วยการสร้างคำสั่งให้ตรงกับที่คาดเดาไว้”

จงอย่าลงมือทันที อันดับแรกให้หาข้อเท็จจริง (ข้อมูล) สักอย่างก่อน → ตั้งสมมติฐานหลายๆ ข้อ → ลงมือพิสูจน์สมมติฐานนั้น

พูดอีกอย่างคือ อย่าลองผิดลองถูกสุ่มสี่สุ่มห้า แต่ให้ใช้แบบจำลองความคิดของตัวเองมาตั้งสมมติฐานและพิสูจน์ความถูกต้อง ผมจะอธิบายการสร้างแบบจำลองความคิดอย่างละเอียดในภายหลัง แต่ขั้นตอนทั้งหมดนี้จะช่วยประหยัดเวลาในการลองทุกความเป็นไปได้ และทำให้เกิดประสิทธิภาพอย่างเหนือชั้น

นั่นเป็นช่วงเวลาที่ผมตระหนักชัดว่า ผลิตภาพของการเขียนโปรแกรมคือ “การใช้แรงสมอง” ดังนั้นผลิตภาพที่แตกต่างกันราวฟ้ากับเหวจึงอาจขึ้นอยู่กับวิธีการใช้ความคิดแบบนี้นี่เอง

ลองผิดลองถูกคือ “ความเลวร้าย”

ในญี่ปุ่น การแก้ปัญหาด้วยการลองผิดลองถูกด้วยตัวเองก่อนเป็นสิ่งที่มักจะได้รับคำชื่นชมจากคนรอบข้าง แต่สำหรับวิศวกรไอที การลองผิดลองถูกไม่ใช่เรื่องดีเลย ผมขอยกตัวอย่างอีกสักเรื่องครับ

วันหนึ่งตอนที่ผมกำลังทำงานระหว่างกลับญี่ปุ่นชั่วคราว เกิดบางอย่างแปลกๆ ขึ้นในแพลตฟอร์มที่ควบคุมล็อก ทั้งที่พอดูไฟล์ล็อกก็เห็นว่าทุกอย่างทำงานปกติดี แต่ข้อมูลจากระยะไกลหรือโทรมาตร (telemetry) กลับไม่มาที่พอร์ทัลของคลาวด์ (เว็บไซต์ที่แสดงขึ้นเป็นอันดับแรก)

พอผมแชร์หน้าจอให้สมาชิกในทีมดูผ่านโปรแกรม Microsoft Teams พร้อมกับบอกว่า “ดูสิ มันไม่ขึ้นใช่ไหม” แต่ไม่รู้ทำไมตอนนั้นข้อมูลโทรมาตรกลับเข้าพอร์ทัลเฉยเลย หลังจากนั้นไม่ว่าผมจะลองอีกกี่ครั้งก็ยังไม่ขึ้นเหมือนเดิม ผมจึงแชตไปขอความช่วยเหลือจากสมาชิกคนอื่นอีกรอบ ทันทีที่ได้รับคำแนะนำว่า “ลองใช้ Fiddler (เครื่องมือมอนิเตอร์เน็ตเวิร์ก) วิเคราะห์สิ” ผมรู้สึกเหมือนโดนสาป พอหันไปดูพอร์ทัลอีกครั้ง ข้อมูลโทรมาตรกลับแสดงขึ้นพอดี

หลังจากลองผิดลองถูกอยู่หลายวิธี หลายไอเดียตีกันไปหมด เช่น อาจจะเกิดปัญหาเพราะข้อมูลถูกตัดในช่วงเวลาที่มีการส่งข้อมูลจำนวนมากหรือเปล่า หรือเป็นที่การกรองข้อมูลจากฝั่งพอร์ทัล ผมจึงปรับสคริปต์ (ภาษาเขียนโปรแกรมแบบง่าย) ส่งออกล็อกให้ไปที่ไฟล์เท่านั้น จำกัดการรันสคริปต์ให้เหลือไม่กี่รูปแบบ...ผมพยายามลองทุกวิธีเพื่อแก้ปัญหาแล้ว

แต่หลังจากเสียเวลาทั้งวัน **การลองผิดลองถูกทั้งหมดจบลงด้วยความล้มเหลว** ทันใดนั้นผมก็นึกขึ้นได้ว่ามีคนหนึ่งในทีมเคยบอกว่า “ข้อมูลแสดงผลถูกต้องแล้วนะ” จากนั้นเธอก็ทดสอบแบบแมนวลโดยไม่ใช้สคริปต์

ผมจึงลองทำแบบแมนวลโดยไม่ใช้สคริปต์ดูบ้าง ปรากฏว่ามันสามารถทำงานได้อย่างสมบูรณ์แบบ

ปัญหาอยู่ที่ “สคริปต์” นั่นเอง! ด้วยเหตุนี้ผมจึงระบุที่มาของอาการแปลก ๆ และแก้ปัญหาได้เสร็จสมบูรณ์

ผมแก้ปัญหาได้แล้ว หลังจากที่ใช้เวลาไปหนึ่งวันเต็ม ๆ

เมื่อทบทวนดู ถึงผมจะใช้เวลาทั้งวัน แต่กลับ “ไม่มีอะไรพัฒนาเลย” **เพราะแค่หาคำตอบโดยลองทำตามรูปแบบต่าง ๆ ที่นึกออกนอกจากเสียเวลาแล้ว ก็ยังไม่ได้เรียนรู้อะไรใหม่ ๆ**

หากเกิดปัญหาเดิมขึ้นซ้ำ ๆ การจดบันทึกไว้น่าจะมีประโยชน์สำหรับการแก้ปัญหาครั้งถัดไป แต่หากปัญหาไม่เหมือนเดิมแม้เพียงนิดเดียวย่อมหมดหนทาง น่าเสียดายที่หากเจอปัญหาเดิมขึ้นมาอีกเป็นไปได้มากว่าผมจะยังคงเสียเวลาเท่าเดิมอยู่ดี

เพื่อนร่วมงานที่เป็นโปรแกรมเมอร์ซึ่งแก้ปัญหาได้ในพริบตาเคยแนะนำผมว่า **อย่าลองผิดลองถูก แต่ "วิเคราะห์ด้วย Fiddler" แทนสิ**

หากใช้ Fiddler จะตรวจสอบได้ว่า Application Insights ส่งข้อมูลแบบใดออกมา แต่ผมไม่รู้ว่าใช้อย่างไร จึงเก็บไว้ใช้เป็นทางเลือกสุดท้ายเมื่อวิธีอื่นไม่ได้ผลแล้วจริง ๆ

สิ่งที่ผมควรทำคือ มอนิเตอร์ด้วย Fiddler เป็นอันดับแรก เพราะจะเห็นได้ทันทีว่าข้อมูลโทรมาตรไม่มาจากฝั่ง WSL2[2]

หากผมเลือกแนวทางนั้นจะไม่เพียงหาสาเหตุของปัญหาได้อย่างรวดเร็ว แต่ยังได้ลองวิธีการ "ใช้ Fiddler ร่วมกับ WSL2" ด้วย พูดง่าย ๆ ว่า เป็นโอกาสได้เรียนรู้วิธีใช้เครื่องมือที่สามารถนำมาใช้ได้ในหลากหลายกรณี ไม่จำกัดแค่ในบางสถานการณ์เท่านั้น จึงทำให้ "ผลิตภาพในอนาคต" ดีขึ้นได้แน่นอน

เหตุการณ์นี้ทำให้ผมตระหนักว่า การลองผิดลองถูกเท่าที่นึกขึ้นได้ถือเป็น "ความเลวร้าย"

[2] ย่อมาจาก Windows Subsystem for Linux 2 คือ ฟีเจอร์ที่พัฒนาโดย Microsoft ทำให้ผู้ใช้ Windows สามารถรันระบบปฏิบัติการ Linux ภายใน Windows ได้โดยไม่ต้องใช้การจำลองเครื่องเสมือน (virtual machine) หรือใช้ระบบปฏิบัติการ Linux แยกต่างหาก

ฉลาดแค่ไหน ก็ต้องใช้เวลา “ทำความเข้าใจ” อยู่ดี

“จงอย่าลงมือทันที แต่ตั้งสมมติฐานและเลือกวิธีการก่อน จากนั้นจึงลงมือทำ” เรามายึดกฎเหล็กนี้เป็นพื้นฐานแล้วนำไปประยุกต์ใช้กับทุกเรื่องเพื่อ “ผลิตภาพในอนาคต” กันเถอะ

ระหว่างที่กินมื้อกลางวันกับเพื่อนร่วมงานรุ่นน้อง 2 คน (อายุงานยังน้อย แต่เก่งสุด ๆ แถมหัวดีมาก) เราคุยกันถึงตอนที่เริ่มทำโปรเจกต์ใหม่

สถาปัตยกรรมในทีมเรามีความซับซ้อนมาก จึงมีบางคนทำวิดีโอสำหรับแต่ละไมโครเซอร์วิสไว้ใช้ภายในแผนก โดยจัดเป็นชุดเพื่อให้ง่ายต่อการศึกษาแต่ละหัวข้อ ผมสนใจวิธีที่พวกเขาใช้ทำความเข้าใจการออกแบบระบบสารสนเทศที่ซับซ้อนเช่นนั้น

“เฮ้อ ดูวิดีโอไปตั้ง 10 รอบแล้ว แต่ก็ยังยากอยู่ดี นี่ดูทวนซ้ำหลายรอบเลย ตรงไหนไม่เข้าใจจะกดหยุดแล้วจดโน้ตไว้”

เพื่อนร่วมงานอีกคนพูดเสริมว่า “ก็ต้องงั้นแหละ ฉันเองยังดูซ้ำไม่รู้กี่รอบเหมือนกัน”

ความคิดเห็นที่ตรงกันของสองวิศวกรเทคนิคคนเก่งทำให้ผมตาสว่าง

ปกติแล้ว หากดูแค่ครั้งเดียวผมจะไม่สามารถเข้าใจได้เลย เพราะมันยาก เลยไม่ค่อยเข้าหัวเท่าไร ผมจึงใช้วิธีทำความเข้าใจทีละน้อยด้วยการรันโปรแกรมหรือไม่ก็ดีบั๊ก (การรันซอฟต์แวร์จริงเพื่อดูสถานะภายใน) ผมเคยคิดว่าคนเก่ง ๆ นั้นน่าอิจฉา เพราะเข้าใจข้อมูลเหล่านั้นได้ในทีเดียว แต่กลับคิดผิดโดยสิ้นเชิง

ไม่ว่าจะเป็นคนฉลาดแค่ไหน ก็ต้องใช้เวลาทำความเข้าใจอยู่ดี สาเหตุที่เราเห็นว่าคนหัวดีเข้าใจได้เร็วเป็นเพราะพวกเขาใช้เวลาสั่งสมพื้นฐานด้วยวิธีนี้มา หน่วยความจำในสมองจึงมีบริบทของสิ่งที่เข้าใจอยู่แล้ว

สำหรับผม คิดว่าความเข้าใจไม่สามารถเกิดขึ้นได้โดยสมบูรณ์ตั้งแต่แรก แต่ต้องค่อย ๆ ซึมซับไปทีละน้อย ยิ่งไปกว่านั้น ผมมักวิตกกังวลเสมอว่า “จะต้องเพิ่มผลิตภาพให้ได้” “จะทำให้เร็วขึ้นได้ยังไงนะ” และจดจ่ออยู่กับการสร้างผลลัพธ์

แต่ก็ตลกดีที่ **“การพยายามทำให้ได้เร็ว ๆ” สุดท้ายมักจะจบลงด้วยผลิตภาพที่แย่กว่าเดิม**

ยกตัวอย่างจากการเขียนโปรแกรม หากเราค้นหาคำตอบจาก Stack Overflow (เว็บไซต์คอมมิวนิตีเกี่ยวกับเทคนิคการเขียนโปรแกรม) แล้วทำสิ่งที่ใกล้เคียงกับการคัดลอกและวางก็จะได้ “ผลลัพธ์” ดีกว่า แต่วิธีนี้จะต้องคอยค้นหาทุกครั้งเพราะไม่เข้าใจพื้นฐาน จึงแก้ปัญหาไม่เก่ง และสุดท้ายก็ไม่มีประสิทธิภาพ ปัจจุบันหากเราถาม Bing หรือ ChatGPT ก็จะบอกวิธีแก้ไขมาให้เลย แต่หากเรา “เข้าใจ” และ “นำมาใช้ได้จริง” ย่อมไม่จำเป็นต้องทำถึงขนาดนั้น

หากลงมือทำทั้งที่ยังไม่เข้าใจมากพอ ถึงพยายามไปก็ไร้ประโยชน์ แถมไม่ได้ความรู้อะไรติดตัวด้วย การลองผิดลองถูกแบบสุ่มสี่สุ่มห้ามักทำให้ลืมง่ายและจดจำอะไรไม่ได้

ดังนั้น “การพยายามเร่งรีบทำบางอย่างให้ได้เร็ว ๆ” กลับยิ่งส่งผลให้เราห่างไกลจากความเข้าใจอย่างลึกซึ้ง

นำแนวคิด “ใช้เวลาทำความเข้าใจ” ไปทำจริง

“ความเข้าใจ” ในการเรียนรู้คืออะไรกันแน่ แม้จะมีหลายระดับ แต่ผมคิดว่า “องค์ประกอบ 3 ประการของความเข้าใจ” มีดังต่อไปนี้

- **เข้าใจโครงสร้างของสิ่งนั้นและอธิบายให้คนอื่นฟังได้**
- **ดึงมาใช้ได้ทันทีทุกที่ทุกเวลา**
- **นำความรู้ที่ได้มาไปปรับใช้ได้**

เวลาทำความเข้าใจไมโครเซอร์วิสใด ๆ หากเราเข้าใจแก่นแท้ว่ามีโครงสร้างที่ประกอบขึ้นจากกลไกแบบนี้จึงทำให้บริการนี้ใช้งานได้ ไม่ได้เข้าใจแบบผิวเผิน ย่อมอธิบายอย่างละเอียดและไม่ซับซ้อนให้คนอื่นเข้าใจได้ง่าย หรือหากไปพูดในวงการอื่น เราก็อธิบายจากมุมมองอื่นและนำเสนอด้วยการเปรียบเทียบได้ว่าสรุปแล้วมันก็คือบริการแบบนี้ ๆ นะ

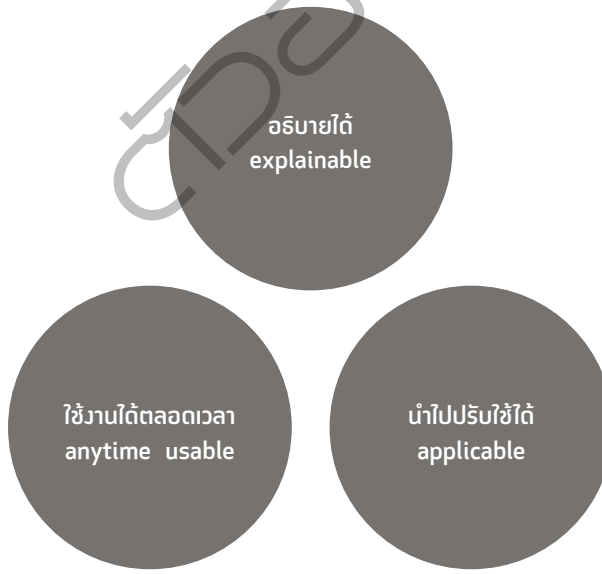
ความรู้ที่เกิดจากการเข้าใจโครงสร้างพื้นฐานสามารถนำไปใช้ได้ทันทีทุกที่ทุกเวลา โดยไม่ต้องดูหนังสืออ้างอิงหรือค้นกูเกิล และนำไปปรับใช้ได้ในหลากหลายสาขา ซึ่งไม่เพียงแก้ปัญหาในประเด็นที่

คล้ายกันได้เท่านั้น แต่ยังแยกปัญหาที่ซับซ้อนออกเป็นหลายปัจจัย และสร้างสรรค์สิ่งใหม่ ๆ ได้ เช่น ดูว่าหากเอาบริการนี้มาใส่ตรงจุดนี้ จะสร้างอะไรขึ้นมาได้หรือไม่

ผมจะอธิบายให้เห็นภาพขึ้น

ช่วงที่ผมเข้าร่วมโปรเจกต์หนึ่ง ต้องทำความเข้าใจสถาปัตยกรรมปัจจุบัน และทำ PoC (Proof of Concept: การพิสูจน์แนวคิด) ของสถาปัตยกรรมใหม่ ผมพยายามอ่านโค้ดอย่างเต็มที่ และคิดว่า “เข้าใจ” โครงสร้างและวิธีการทำงานส่วนใหญ่แล้ว

แต่พอแลกเปลี่ยนความคิดเห็นเรื่องสถาปัตยกรรมกับวิศวกรเทคนิคคนเก่งที่ไม่เคยอ่านโค้ดนั้นมาก่อน กลับมีคำถามหลายข้อที่ผมตอบไม่ได้ เราจึงตกลงว่าจะมาอ่านโค้ดด้วยกัน



รูปที่ 2 องค์ประกอบ 3 ประการของความเข้าใจ

เขาอ่านและเข้าใจโค้ดได้เร็วกว่าผมมาก **แม้บางจุดที่ดูเหมือนจะอ่านข้ามได้โดยคิดว่า "ตรงนี้น่าจะเป็นประมาณนี้ใช่ไหมนะ" ก็ยังตั้งอกตั้งใจใช้เวลาจดเลขตัวอย่างและพยายามทำความเข้าใจอย่างถ่องแท้** เมื่อเรากลับมาพูดคุยกันในภายหลัง เขาก็เข้าใจสถาปัตยกรรมได้อย่างถูกต้อง รวมถึงเสนอไอเดียต่าง ๆ มากมาย

ตัวผมอ่านโค้ดช้าจึงคิดแต่จะหาวิธีอ่านเร็ว ๆ ในขณะที่เขาไม่ได้อ่านแค่ตรรกะของโค้ด **แต่อ่านเพื่อทำความเข้าใจจุดประสงค์ของโค้ดและสถาปัตยกรรมที่อยู่เบื้องหลัง** เขาไม่เสียดายที่จะใช้เวลาทำความเข้าใจให้เต็มที่

พอกลับมาทบทวนดู หากตัวเองเขียนโปรแกรมด้วยโค้ดง่าย ๆ มักเผลอคิดว่า "ไม่เห็นยาก" จริงอยู่ว่าผมเขียนโปรแกรมได้ แต่เพราะไม่ได้เข้าใจอย่างลึกซึ้งจึงต้องค้นกูเกิลทุกครั้งและเขียนโค้ดด้วยความรู้สึกว่า "ใช่แบบนี้หรือเปล่านะ" ถึงจะเคยทำได้ แต่เพราะจำไม่ได้ ครั้งต่อไปผมเลยต้องค้นหาคำตอบเดิมอีกครั้ง อาจเรียกได้ว่าเป็นโปรแกรมเมอร์กูเกิลไปแล้ว

ในขณะที่เพื่อนร่วมงานคนอื่น ๆ เข้าใจโครงสร้างที่ซับซ้อนได้ทันทีและเขียนโค้ดราวกับเล่นเปียโน เวลาที่ผมรีวิวโค้ดของพวกเขา บางครั้งรู้สึกว่าทำแบบนี้น่าจะดีกว่านะ แต่หากเทียบกับพวกเขาแล้ว "ผมไม่มีพื้นฐานที่ใคร ๆ ก็ทำได้เลย"

ไหน ๆ แล้วขอนอกเรื่องหน่อยครับ ผมตระหนักเรื่องเดียวกันนี้ได้จากการเล่นกีตาร์ที่เป็นงานอดิเรก ทั้งที่ฝึกฝนมาหลายสิบปีด้วยความชื่นชอบ แต่ไม่เคยรู้สึกเลยว่า "เล่นเป็น" มือกีตาร์ระดับท็อปอย่างโทโมะ ฟุจิตะ ที่สอนกีตาร์ให้เหล่านักดนตรีชั้นนำของมหาวิทยาลัยดนตรีเบิร์กลีย์เคยพูดถึงลักษณะนิสัยของคนที่เล่นกีตาร์ไม่คล่องไว้ว่า

"คุณไม่เข้าใจจังหวะอย่างละเอียด เลยแค่พอเล่นได้"

การฝึกฝนที่เขาแนะนำเป็นวิธีมาตรฐานที่อยู่ในหนังสือสอนเล่นกีตาร์มากมาย นั่นคือ วางกีตาร์ลง ปรบมือทั้งสองข้างพร้อมกับใส่จังหวะเน้น โดยเริ่มจากจังหวะที่ช้ามาก ๆ แล้วปรบมือเน้นจังหวะที่เสียงแรก กลาง และสุดท้ายของโน้ตสามพยางค์ตามลำดับ

ผมจึงตกใจว่า "อ๋อ! นี่ไงล่ะ"

นี่เป็นวิธีฝึกกีตาร์ที่คนเล่นกีตาร์ทุกคนรู้จักดี ผมเองก็เคยลองอยู่นิดหน่อยแล้วคิดว่า "เอ้อ ก็ทำได้นี่นา" แต่ไม่เคยฝึกจริง ๆ จัง ๆ ทว่าพอได้ลองทำอีกครั้ง กลับพบว่าแม้เป็นจังหวะที่ง่าย ๆ ก็เล่นให้ตรงได้ยากกว่าที่คิด

ผมเล่นกีตาร์ได้เร็วก็จริง แต่จับจังหวะไม่แม่น แม้แต่การดีดง่าย ๆ ก็เล่นสบาย ๆ ไม่ได้ สรุปคือ ผมไม่ได้เรียนรู้ "พื้นฐาน" ที่ใคร ๆ ก็รู้จัก **และการฝึก "พื้นฐาน" เป็นสิ่งที่ "ใคร ๆ ก็ทำได้" แต่การจะทำได้ดีต้อง "ใช้เวลานาน"**

เพราะโลกเปลี่ยนเร็ว อยากเก่งต้องคิดให้ไวและให้ทัน!

เคล็ดลับไม่ได้อยู่ที่คิดอย่างไรจึงจะเร็ว แต่คิดอย่างไรให้เป็นระบบ
เรียนรู้แนวทางการคิดแบบ **"วิศวกรระดับโลก"** ที่ไม่ได้เชี่ยวชาญแค่เรื่องเทคนิค
แต่หัวใจในการทำงานของพวกเขาคือ **"วิธีคิดอย่างเป็นระบบ"**
ระบบที่ทำให้การแก้ปัญหาเป็นเรื่องง่าย ลงแรงทำน้อย แต่ได้ผลลัพธ์มาก

- รีบคิด รีบล้มเหลว และเรียนรู้ให้เร็วขึ้น
- ฝึกคิดแบบ "คนขี้เกียจ" ทำงานเท่าที่จำเป็น แต่ได้ผลลัพธ์มากกว่าที่คาด
- ปลดล็อกวิธีสื่อสารกับทีมให้ไหลลื่น ลดความวุ่นวาย เพิ่มประสิทธิภาพ
- เปลี่ยนความเชี่ยวชาญเฉพาะทางให้กลายเป็นแต้มต่อที่ไม่มีใครเทียบได้

และสุดยอดกระบวนการอื่นๆ อีกมากมาย
ที่จะช่วยยกระดับวิธีคิดของคุณให้กลายเป็นอาวุธที่ทรงพลัง
พร้อมเผชิญทุกอุปสรรคในโลกของการทำงานอย่างรวดเร็วและเปี่ยมประสิทธิภาพ

หมวดจิตวิทยาพัฒนาตนเอง
ISBN 978-616-18-8398-0
9 786161 883980
Cover Design: FIELD
285 บาท