**บทนำ**

ฉันได้ยินเกี่ยวกับ Domain Driven Design และพบกับ Eric Evans ที่การประชุมของนักสถาปัตยกรรมระดับเล็กในยอดเขาที่จัดโดย Bruce Eckel ในฤดูร้อนของปี 2005 การประชุมได้รับการเข้าร่วมโดยบุคคลที่ฉันเคารพอย่างมาก รวมถึง Martin Fowler, Rod Johnson, Cameron Purdy, Randy Stafford, และ Gregor Hohpe

กลุ่มดูเหมือนจะประทับใจกับวิสัยทัศน์ของ Domain Driven Design และกระตือรือร้นที่จะเรียนรู้เพิ่มเติมเกี่ยวกับมัน ฉันยังรู้สึกว่าทุกคนปรารถนาว่าแนวคิดเหล่านี้จะเป็นเช่นนี้กับกลุ่มคนทั่วไปมากขึ้น เมื่อฉันสังเกตเห็นว่า Eric ใช้โมเดลโดเมนเพื่อพูดคุยเกี่ยวกับการแก้ไขบางอย่างจากที่กลุ่มพูดคุยกัน และเขาให้ความสำคัญกับโดเมนธุรกิจมากกว่าการตลาดเทคโนโลยี ฉันรู้ทันทีว่าวิสัยทัศน์นี้เป็นสิ่งที่ชุมชนต้องการอย่างแรงแล้ว

เราในชุมชนการพัฒนาสถาปัตยกรรมธุรกิจ โดยเฉพาะชุมชนการพัฒนาเว็บไซต์ได้รับผลกระทบจากปีหลายๆ ของการตลาดเทคโนโลยีที่เอาเราออกไปจากการพัฒนาซอฟต์แวร์เชิงวัตถุที่เหมาะสม ในชุมชน Java การสร้างโมเดลโดเมนที่ดีหายไปในการตลาดของ EJB และโมเดลคอนเทนเนอร์/คอมโพเนนต์ในปี 1999-2004 โชคดีที่เทคโนโลยีกำลังเปลี่ยนแปลงและประสบการณ์ร่วมกันของชุมชนการพัฒนาซอฟต์แวร์กำลังนำเรากลับสู่แนวคิดเชิงวัตถุเก่าแก่ อย่างไรก็ตาม ชุมชนนั้นยังขาดวิสัยทัศน์ที่ชัดเจนเพื่อนำไปใช้ในการพัฒนาซอฟต์แวร์เชิงวัตถุในองค์กรใหญ่ นี่เป็นเหตุผลที่ฉันคิดว่า DDD มีความสำคัญ

นับเป็นความน่าเสียดายที่นอกจากกลุ่มของนักสถาปัตยกรรมระดับสูงเล็กน้อยแล้ว ฉันรับรู้ว่ามีน้อยมากที่รู้จัก DDD ซึ่งเป็นเหตุผลที่ InfoQ ได้มอบหมายให้เขียนหนังสือเล่มนี้

ฉันหวังว่าโดยการเผยแพร่สรุปและนำเสนอขั้นพื้นฐานของ DDD อย่างสั้น ๆ และอ่านได้ง่าย และทำให้เป็นไปได้ว่าความคิดนี้จะกลายเป็นหลักสูตรหลักในอนาคต โดยที่เราจะเผยแพร่ฉบับพิมพ์ขนาดกระเป๋าราคาไม่แพงพร้อมให้ดาวน์โหลดฟรีบน InfoQ นั่นเอง

หนังสือเล่มนี้ไม่มีการนำเสนอแนวคิดใหม่ๆใดๆ แต่พยายามสรุปสาระสำคัญของ DDD ให้อย่างกระชับ โดยส่วนใหญ่นำเอาหนังสือเล่มเดิมของ Eric Evans เป็นหลัก รวมถึงแหล่งอื่นๆที่เกี่ยวข้องเช่น Applying DDD ของ Jimmy Nilsson และห้องสนทนา DDD ต่างๆ หนังสือเล่มนี้จะช่วยให้คุณเข้าใจเรื่องพื้นฐานของ DDD ได้อย่างรวดเร็ว แต่ไม่สามารถแทนที่ตัวอย่างและกรณีศึกษาจำนวนมากที่ได้รับการจัดเตรียมไว้ในหนังสือของ Eric หรือตัวอย่างที่เสนอในหนังสือของ Jimmy ฉันขอแนะนำให้คุณอ่านทั้งสองหนังสือเหล่านี้อย่างสม่ำเสมอ ในขณะเดียวกัน หากคุณเห็นด้วยว่า DDD เป็นสิ่งที่ชุมชนต้องการจะมีส่วนร่วมและเป็นการรับรู้เฉพาะของกลุ่มของเรา กรุณาแชร์หนังสือนี้และงานของ Eric ต่อคนอื่น ๆ บ้าง

ซอฟต์แวร์เป็นเครื่องมือที่สร้างขึ้นเพื่อช่วยให้เราจัดการกับความซับซ้อนของชีวิตสมัยใหม่ของเราได้ ซอฟต์แวร์เป็นเพียงเครื่องมือที่ช่วยให้เราไปสู่จุดมุ่งหมาย และมักเป็นเรื่องที่เป็นจริงและใกล้ชิดกับชีวิตจริงๆ ตัวอย่างเช่นเราใช้ซอฟต์แวร์สำหรับควบคุมการจราจรทางอากาศ ซึ่งเชื่อมโยงโดยตรงกับโลกที่เราอยู่ ซึ่งเราต้องการบินจากสถานที่หนึ่งไปยังสถานที่อื่น และเราทำเช่นนั้นโดยใช้เครื่องจักรที่ซับซ้อน ดังนั้นเราจึงสร้างซอฟต์แวร์เพื่อประสานงานการบินของพันธมิตรที่อยู่ในอากาศหลายพันเครื่องในเวลาใดก็ตาม

ซอฟต์แวร์ถูกสร้างขึ้นเพื่อช่วยให้เราจัดการกับความซับซ้อนในชีวิตสมัยปัจจุบันของเรา ซอฟต์แวร์เป็นเพียงเครื่องมือเพื่อทำให้เราได้ตามเป้าหมาย และจุดหมายที่สำคัญสำหรับซอฟต์แวร์นั้นเป็นสิ่งที่เป็นจริงและเชื่อมโยงกับความจำเป็นในชีวิตจริง ตัวอย่างเช่น เราใช้ซอฟต์แวร์สำหรับควบคุมการจราจรทางอากาศ ซึ่งเชื่อมโยงโดยตรงกับโลกที่เราดำเนินชีวิตอยู่ เราต้องการบินจากสถานที่หนึ่งไปยังอีกที่หนึ่ง และเราจะใช้เครื่องจักรที่ซับซ้อน เพื่อนำซอฟต์แวร์มาประสานงานในการบินของพันธมิตรที่อยู่ในอากาศพร้อมกันหลายพันเครื่อง

การออกแบบซอฟต์แวร์เป็นศิลปะ และเหมือนกับศิลปะใด ๆ มันไม่สามารถสอนและเรียนรู้เป็นวิทยาศาสตร์ที่แน่นอนได้ โดยใช้ทฤษฎีและสูตรต่าง ๆ ได้ เราสามารถค้นพบหลักการและเทคนิคที่มีประโยชน์ในการสร้างซอฟต์แวร์ตลอดกระบวนการสร้างซอฟต์แวร์ได้ แต่เราอาจไม่สามารถให้เส้นทางที่แน่นอนในการสร้างโมดูลโค้ดที่จะช่วยให้เหตุการณ์จริงสอดคล้องกับโดเมน โดยตรงได้ เหมือนกับรูปภาพหรืออาคาร ผลิตภัณฑ์ซอฟต์แวร์จะรวมความสมบูรณ์ของผู้ออกแบบและพัฒนามัน บางส่วนของเสน่ห์และความมั่นใจ (หรือขาดแคล้ว) ของผู้มีส่วนร่วมในการก่อตั้งและการเติบโตของมัน

มีวิธีการที่แตกต่างกันในการออกแบบซอฟต์แวร์ ตลอด 20 ปีที่ผ่านมา อุตสาหกรรมซอฟต์แวร์ได้ใช้วิธีการหลายวิธีในการสร้างผลิตภัณฑ์ แต่ละวิธีมีข้อดีและข้อเสียของตัวเอง เนื้อหาของหนังสือนี้จะเน้นไปที่วิธีการออกแบบที่เกิดขึ้นและพัฒนามาตลอด 20 ปีที่ผ่านมา แต่รวมตัวมากขึ้นในช่วงไม่กี่ปีที่ผ่านมา นั่นคือการออกแบบตามโดเมน (Domain-Driven Design) Eric Evans ได้มีส่วนร่วมอันมหาศาลในเรื่องนี้โดยเขียนเล่มหนังสือที่รวมความรู้และประสบการณ์ที่สะสมมานานเกี่ยวกับการออกแบบตามโดเมน สำหรับเนื้อหาที่ละเอียดอ่อนขึ้นเกี่ยวกับเรื่องนี้ เราขอแนะนำให้อ่านหนังสือของเขา "Domain-Driven Design: Tackling Complexity in the Heart of Software" ที่จัดพิมพ์โดย Addison-Wesley หมายเลขไอเอสเบิร์น: 0-321-12521-5

หลายความคิดและข้อสังเกตมีให้เรียนรู้ได้จากการติดตามกลุ่ม Diskussion ของ Domain Driven Design ที่ <http://groups.yahoo.com/group/domaindrivendesign> หนังสือเล่มนี้เป็นเพียงการแนะนำพื้นฐานเกี่ยวกับหัวข้อนี้เท่านั้น โดยมีจุดมุ่งหมายเพื่อให้คุณเข้าใจพื้นฐานของ Domain Driven Design แต่ไม่ได้มุ่งเน้นรายละเอียด โดยเราต้องการเรียนรู้แนวคิดและแนวปฏิบัติที่นำไปสู่การออกแบบซอฟต์แวร์ที่ดีในโลกของ Domain Driven Design โดยเฉพาะอย่างยิ่ง

Floyd Marinescu

Co-founder & Chief Editor of InfoQ.com

**Domain-Driven Design คืออะไร**

การพัฒนาซอฟต์แวร์มักถูกนำไปใช้ในการออโตเมติกขบวนการต่างๆ ที่มีอยู่ในโลกจริงหรือให้การแก้ไขปัญหาทางธุรกิจต่างๆ โดยกระบวนการธุรกิจที่กำลังถูกออโตเมติกส่วนหนึ่งหรือปัญหาทางธุรกิจที่ต้องการใช้ซอฟต์แวร์จึงเป็นโดเมนของซอฟต์แวร์เหล่านั้น ดังนั้นเราต้องเข้าใจตั้งแต่ต้นว่าซอฟต์แวร์เกิดขึ้นมาจากโดเมนนี้และสัมพันธ์กับโดเมนนี้อย่างลึกซึ้ง

ซอฟต์แวร์ประกอบด้วยโค้ด บางครั้งเราอาจจะต้องการใช้เวลามากเกินไปกับโค้ด และมองซอฟต์แวร์เป็นเพียงวัตถุและเมธอดเท่านั้น

ในการผลิตรถยนต์ เป็นการใช้สมมติฐาน เพื่อให้เห็นภาพการผลิตซอฟต์แวร์ แรงงานที่มีส่วนร่วมในการผลิตรถยนต์ อาจเชี่ยวชาญในการผลิตชิ้นส่วนของรถยนต์ แต่ด้วยการเชื่อมโยงของชิ้นส่วนต่างๆ สามารถมองรถยนต์เป็นชุดของชิ้นส่วน แต่รถยนต์ก็ไม่ได้เป็นแค่นั้นเท่านั้น การผลิตรถยนต์ที่ดีเริ่มต้นจากวิสัยทัศน์ การเขียนสเปคอย่างละเอียดถูกต้องและตระหนักถึงการออกแบบ มีการออกแบบมากมาย เสริมเติม หลายเดือนหรือบางครั้งก็อาจใช้เวลาหลายปีในการออกแบบ โดยการเปลี่ยนแปลงและปรับปรุงจนถึงความสมบูรณ์ จนกระทั่งตระหนักถึงวิสัยทัศน์เดิมได้ การออกแบบไม่จำเป็นต้องอยู่บนกระดาษเท่านั้น มีการออกแบบโมเดลรถยนต์และทดสอบในสถานการณ์ต่างๆ เพื่อดูว่ามันจะทำงานได้หรือไม่ การออกแบบจึงได้รับการปรับปรุงตามผลการทดสอบ รถยนต์จึงถูกส่งเข้ากระบวนการผลิตและชิ้นส่วนถูกสร้างและประกอบเข้าด้วยกัน

การพัฒนาซอฟต์แวร์คล้ายกับนี้ ไม่สามารถเริ่มต้นโดยการนั่งพิมพ์โค้ดได้เลย แม้ว่าวิธีนี้จะเหมาะสำหรับกรณีที่ง่าย ๆ แต่สำหรับซอฟต์แวร์ที่ซับซ้อน เราไม่สามารถสร้างได้แบบนั้น

เพื่อสร้างซอฟต์แวร์ที่ดี คุณต้องรู้ว่าซอฟต์แวร์นั้นเกี่ยวข้องกับอะไร คุณไม่สามารถสร้างระบบธนาคารได้เลย หากคุณไม่มีความเข้าใจที่ดีเกี่ยวกับบริบทของการเงิน คุณจึงต้องเข้าใจดีว่าสิ่งที่เรียกว่า Domain คืออะไร ด้วยความเข้าใจเหล่านี้ จึงจะช่วยให้คุณสร้างซอฟต์แวร์ได้ดียิ่งขึ้น

เป็นไปได้หรือไม่ที่จะสร้างซอฟต์แวร์ธนาคารที่ซับซ้อนโดยไม่มีความรู้ด้านโดเมนที่ดี? ไม่มีทางเลย ไม่เคยเป็นเช่นนั้น เท่าที่รู้ ใครรู้จักธนาคารบ้าง? นักสถาปัตยกรรมซอฟต์แวร์หรือเปล่า? ไม่ใช่ พวกเขาเพียงแค่ใช้ธนาคารเพื่อเก็บเงินไว้เมื่อต้องการใช้ นักวิเคราะห์ซอฟต์แวร์? ไม่ได้จริงๆ พวกเขารู้จักการวิเคราะห์เรื่องที่กำหนดให้ เมื่อมีอุปสรรคทั้งหมดที่จำเป็น นักพัฒนา? ลืมมันไปเลย ใครแทนบ้าง? คนงานธนาคารแน่นอน ระบบธนาคารนั้นเข้าใจอย่างดีโดยคนภายในสถาบัน โดยผู้เชี่ยวชาญของพวกเขา พวกเขารู้จักรายละเอียดทั้งหมด ทุกๆ อย่าง กฎเกณฑ์ที่เป็นไปได้ทั้งหมด ปัญหาที่เป็นไปได้ทั้งหมด กฎข้อบังคับทั้งหมด ตรงนี้เป็นจุดเริ่มต้นที่เราควรจะเริ่มต้นเสมอ: ด้านโดเมน

เมื่อเราเริ่มโครงการพัฒนาซอฟต์แวร์ เราควรให้การสำคัญกับโดเมนที่ซอฟต์แวร์จะถูกใช้งานอยู่ จุดประสงค์ของซอฟต์แวร์ทั้งหมดคือการเพิ่มประสิทธิภาพให้กับโดเมนเฉพาะนั้น โดยเพื่อที่จะสามารถทำได้นั้น ซอฟต์แวร์ต้องเหมาะสมกับโดเมนที่มันถูกสร้างขึ้นมาเพื่อใช้งาน มิฉะนั้น ซอฟต์แวร์ก็จะนำความตึงเครียดเข้าสู่โดเมน ทำให้เกิดข้อผิดพลาด ความเสียหายและอาจทำให้เกิดความสับสนและความขัดแย้งในโดเมนด้วย

การออกแบบโดยใช้โดเมนมอเดลจะช่วยให้ผู้ที่ไม่มีความรู้เกี่ยวกับธนาคารสามารถเรียนรู้ได้มากมายโดยการอ่านโค้ดในโมเดลโดเมน นี่เป็นสิ่งที่สำคัญ โปรแกรมที่ไม่มีรากฐานที่ลึกซึ้งในโดเมนจะไม่สามารถรองรับการเปลี่ยนแปลงได้ดีในระยะยาว

การเริ่มต้นด้วยโดเมน จากนั้นจะทำอย่างไรต่อไป? โดเมนเป็นสิ่งหนึ่งในโลกนี้ ไม่สามารถเอามากรอกลงบนแป้นพิมพ์เพื่อกลายเป็นโค้ดได้อย่างง่ายดาย เราต้องสร้างการสรรหาของโดเมน และเราจะได้เรียนรู้จากผู้เชี่ยวชาญด้านโดเมนได้มากมาย แต่ความรู้เบื้องต้นนี้จะไม่สามารถแปลงเป็นโครงสร้างซอฟต์แวร์ได้อย่างง่ายดาย เราจึงต้องสร้างนวัตกรรมของโดเมน ซึ่งเป็นแบบจำลองของโดเมน เรียกได้ว่าโดเมนโมเดล ตาม Eric Evans โดเมนโมเดลไม่ใช่แผนภูมิเพียงแค่อย่างเดียว แต่เป็นความคิดที่แผนภูมินั้นจะต้องสื่อความหมาย โดเมนโมเดลไม่ได้แค่เป็นความรู้ที่อยู่ในหัวของผู้เชี่ยวชาญด้านโดเมน แต่เป็นแบบจำลองที่เราสร้างขึ้นจากความรู้ที่ได้จากผู้เชี่ยวชาญด้านโดเมน แผนภูมิสามารถแสดงและสื่อสารโดเมนโมเดลได้ เช่นโค้ดที่เขียนอย่างรอบคอบและรัดกุม หรือประโยคภาษาอังกฤษที่เขียนอย่างถูกต้อง โดยสามารถเป็นแบบจำลองของโดเมนได้อย่างมีประสิทธิภาพ

โมเดลเป็นการแสดงภายในของโดเมนเป้าหมายของเรา และมันเป็นสิ่งที่จำเป็นมากในขั้นตอนการออกแบบและพัฒนา ระหว่างการออกแบบเราจะจำได้และอ้างอิงมากมายต่อโมเดล เพราะโลกรอบตัวเรามีมากเกินไปสำหรับการจัดการด้วยความจำของมนุษย์ แม้ว่าโดเมนที่เฉพาะเจาะจงจะเป็นไปได้ว่ามันจะมากกว่าที่จะจัดการได้ในเวลาเดียวกัน ดังนั้นเราจำเป็นต้องจัดระเบียบข้อมูล ทำระบบ แบ่งแยกให้เป็นชิ้นเล็ก ๆ จัดกลุ่มชิ้นเหล่านั้นเป็นโมดูลที่มีเหตุผล และจะต้องดูแลเรื่องละเอียดของชิ้นงานแต่ละชิ้น และแยกส่วนอื่น ๆ ของโดเมนออกไปบ้าง โดเมนมีข้อมูลมากเกินไปที่จะรวมทั้งหมดลงในโมเดล และส่วนใหญ่จะไม่จำเป็นต้องพิจารณาเพิ่มเติม นี่เป็นภาระที่ต้องพิจารณาด้วยตัวเองว่าจะเก็บอะไรไว้ และจะทิ้งอะไรไป มันเป็นส่วนหนึ่งของการออกแบบและการสร้างซอฟต์แวร์ โปรแกรมธนาคารเช่นเดียวกันจะเก็บที่อยู่ของลูกค้า แต่ไม่จำเป็นต้องสนใจเรื่องสีตาของลูกค้า เป็นต้น นั่นเป็นกรณีที่ชัดเจน แต่กรณีอื่น ๆ อาจไม่ชัดเจนเท่านั้น

แบบจำลอง (Model) เป็นส่วนสำคัญของการออกแบบซอฟต์แวร์ ที่จำเป็นต้องมีเพื่อจัดการกับความซับซ้อน ทั้งหมดที่เราคิดเกี่ยวกับโดเมนจะถูกสังเคราะห์เข้าด้วยกันเป็นแบบจำลองเดียวกัน โดยแบบจำลองนี้จะเกี่ยวข้องกับข้อมูล โครงสร้าง และกระบวนการต่าง ๆ ของโดเมนนั้น ๆ ซึ่งจะช่วยให้เราเข้าใจและพัฒนาซอฟต์แวร์ได้ง่ายขึ้น โดยขั้นตอนการสร้างแบบจำลองจะขึ้นอยู่กับประเภทของโดเมนและขอบเขตของโครงการซอฟต์แวร์นั้น ๆ ด้วย นั่นเป็นสิ่งที่ดี แต่มันต้องออกจากสมองของเราด้วย มันไม่ได้มีประโยชน์มากนักถ้ามันยังอยู่ในนั้น ใช่ไหม? เราต้องสื่อสารโมเดลนี้กับผู้เชี่ยวชาญด้านโดเมน นักออกแบบเหล่านี้และนักพัฒนาด้วย ตัวแบบ (model) เป็นสารอาหารสำคัญของซอฟต์แวร์ แต่เราต้องสร้างวิธีการแสดงออกและสื่อสารกับผู้เชี่ยวชาญด้านโดเมน (domain experts) ผู้ออกแบบและนักพัฒนาอื่นๆ เพื่อให้เข้าใจและเห็นภาพของตัวแบบนั้น ได้อย่างชัดเจน เราไม่ได้อยู่คนเดียวในกระบวนการนี้ดังนั้นเราต้องแบ่งปันความรู้และข้อมูล และเราต้องทำมันอย่างดี แม่นยำ สมบูรณ์และไม่มีความกำกวม มีวิธีที่แตกต่างกันเพื่อทำให้เกิดการสื่อสารได้ ได้แก่การวาดแผนภาพ ใช้กรณีการใช้งาน วาดภาพ รูปภาพ เป็นต้น อีกวิธีหนึ่งคือการเขียน ทำให้เราสามารถเขียนเกี่ยวกับวิสัยทัศน์ของเราเกี่ยวกับโดเมนได้ วิธีสุดท้ายคือภาษา สามารถสร้างภาษาเพื่อสื่อสารเกี่ยวกับปัญหาที่เกี่ยวกับโดเมนได้ แต่เราจะรายละเอียดเพิ่มเติมทั้งหมดในภายหลัง แต่จุดสำคัญคือเราต้องสื่อสารแบบชัดเจนเกี่ยวกับโมเดล

การออกแบบโค้ด (code design) นั้นต่างจากการออกแบบซอฟต์แวร์ (software design) โดยการออกแบบซอฟต์แวร์คือการสร้างโครงสร้างของบ้านที่เหมือนกับมองภาพทั้งหมด ส่วนการออกแบบโค้ดจะเน้นที่รายละเอียด เช่นตำแหน่งของรูปภาพบนผนังบางพื้นที่ การออกแบบโค้ดเป็นสิ่งสำคัญอย่างมาก แต่ไม่มีความสำคัญเท่ากับการออกแบบซอฟต์แวร์ การออกแบบโค้ดที่ผิดพลาดส่วนใหญ่สามารถแก้ไขได้อย่างง่ายดาย ในขณะที่การออกแบบซอฟต์แวร์ที่ผิดพลาดจะต้องใช้ค่าใช้จ่ายสูงมากในการซ่อมแซม การย้ายรูปภาพไปทางซ้ายอาจจะเป็นเรื่องง่าย แต่การแตกไปยังฝั่งหนึ่งของบ้านเพื่อทำใหม่อีกครั้งนั้นเป็นเรื่องต่างหาก แม้ว่าผลงานสุดท้ายจะดี ก็ยังต้องมีการออกแบบโค้ดอย่างดี เพื่อให้ได้รหัสที่สะอาด และสามารถบำรุงรักษาได้ดี ในการออกแบบโค้ดนั้น รูปแบบการเขียนโค้ดจะช่วยให้การออกแบบโค้ดดียิ่งขึ้น และควรนำเอารูปแบบการเขียนโค้ดที่ดีมาใช้เมื่อจำเป็น

แนวทางในการออกแบบซอฟต์แวร์มีหลายแนวทาง แบบหนึ่งคือวิธีการออกแบบแบบน้ำตก (waterfall design method) วิธีนี้ประกอบด้วยขั้นตอนหลายขั้นตอน โดยผู้เชี่ยวชาญด้านธุรกิจจะเสนอเกณฑ์ที่ต้องการ ซึ่งจะถูกสื่อสารไปยังนักวิเคราะห์ธุรกิจ นักวิเคราะห์ธุรกิจจะสร้างโมเดลโดยพึ่งข้อมูลเหล่านั้น แล้วส่งผลการวิเคราะห์ให้กับนักพัฒนา ซึ่งจะเริ่มต้นเขียนโค้ดตามที่ได้รับมา วิธีนี้เป็นการไหลเดียวของความรู้ แม้ว่าวิธีนี้จะเป็นแนวทางที่เก่าแก่ในการออกแบบซอฟต์แวร์ และใช้ได้บางครั้งโดยมีความสำเร็จในหลายปี แต่มีข้อจำกัดและข้อบกพร่อง ปัญหาหลักคือไม่มีการตอบกลับจากนักวิเคราะห์ธุรกิจถึงผู้เชี่ยวชาญด้านธุรกิจหรือจากนักพัฒนาถึงนักวิเคราะห์ธุรกิจ

วิธีการที่มีอีกแบบคือ Agile methodologies เช่น Extreme Programming (XP) วิธีการเหล่านี้เป็นการรวมตัวกันเพื่อต่อต้านการใช้วิธีการ Waterfall ที่มีข้อจำกัดและข้อบกพร่อง โดยเป็นผลมาจากความยากที่จะคาดเดาทุกข้อกำหนดได้แบบสมบูรณ์ไว้แต่แรก โดยเฉพาะเมื่อมีการเปลี่ยนแปลงกฎเกณฑ์ การสร้างโมเดลที่ครอบคลุมทุกภาคส่วนของโดเมนแบบสมบูรณ์ตั้งแต่แรกยิ่งไปกว่านั้นยิ่งเป็นเรื่องยาก การออกแบบซอฟต์แวร์ต้องใช้เวลาในการคิดเยอะแยะและบางครั้งคุณไม่สามารถมองเห็นปัญหาทั้งหมดตั้งแต่เริ่มต้นได้ นอกจากนี้ยังไม่สามารถคาดการณ์ผลข้างเคียงหรือข้อผิดพลาดของการออกแบบได้เสมอไปเช่นกัน อีกปัญหาที่ Agile พยายามแก้ไขคือการติดตั้ง “analysis paralysis” หรือความกลัวที่สมาชิกในทีมจะตัดสินใจออกแบบและพัฒนาไม่ได้เพราะกลัวที่จะเกิดข้อผิดพลาด จนทำให้ไม่มีความคืบหน้าเลยโดยสมบูรณ์ Agile นำเสนอการตัดสินใจในการออกแบบแต่ไม่ได้ต้องการการออกแบบล่วงหน้า แทนที่จะใช้ความยืดหยุ่นในการดำเนินการ และผ่านการพัฒนาต่อเนื่องด้วยการมีผู้เข้าร่วมธุรกิจอยู่เสมอ รวมถึงการทำการปรับปรุงโค้ดอย่างสม่ำเสมอ เที้ยงซ้ำเติม ทีมพัฒนาจะได้เรียนรู้เพิ่มเติมเกี่ยวกับโดเมนลูกค้าและสามารถผลิตซอฟต์แวร์ที่ตอบสนองความต้องการของลูกค้าได้ดีขึ้น

วิธีการ Agile มีปัญหาและข้อจำกัดของตัวเอง วิธีนี้สนับสนุนความเรียบง่าย แต่ทุกคนมีมุมมองของตัวเองว่ามันหมายถึงอะไร นอกจากนี้ การ Refactoring ต่อเนื่องที่ผู้พัฒนาทำโดยไม่มีหลักการออกแบบที่มั่นคงจะสร้างโค้ดที่ยากต่อการเข้าใจหรือเปลี่ยนแปลง และในขณะที่วิธีวางแผนแบบ waterfall อาจนำไปสู่การออกแบบโดยมีการเคลื่อนไหวมากเกินไป (over-engineering) ความกลัวที่จะเกิดการออกแบบโดยมีการเคลื่อนไหวมากเกินไปอาจเป็นเหตุผลหนึ่งที่ส่งผลต่อความกลัวของการออกแบบที่ลึกซึ้งและคิดอย่างเต็มความสามารถ

หนังสือเล่มนี้นำเสนอหลักการของการออกแบบด้วยการตระหนักถึงโดเมน (Domain Driven Design) ซึ่งเมื่อนำมาใช้ จะช่วยเพิ่มประสิทธิภาพของกระบวนการพัฒนาในการจำลองและดำเนินการแก้ไขปัญหาที่ซับซ้อนในโดเมน ในทางกลับกัน การออกแบบและการพัฒนาสามารถทำงานร่วมกันเพื่อสร้างตัวแก้ไขที่ดีกว่าได้ การออกแบบที่ดีจะเร่งความเร็วในการพัฒนา ในขณะที่ข้อมูลตอบรับจากกระบวนการพัฒนาจะเสริมสร้างการออกแบบให้ดียิ่งขึ้น

**การสร้างความรู้ในเชิงพื้นฐาน**

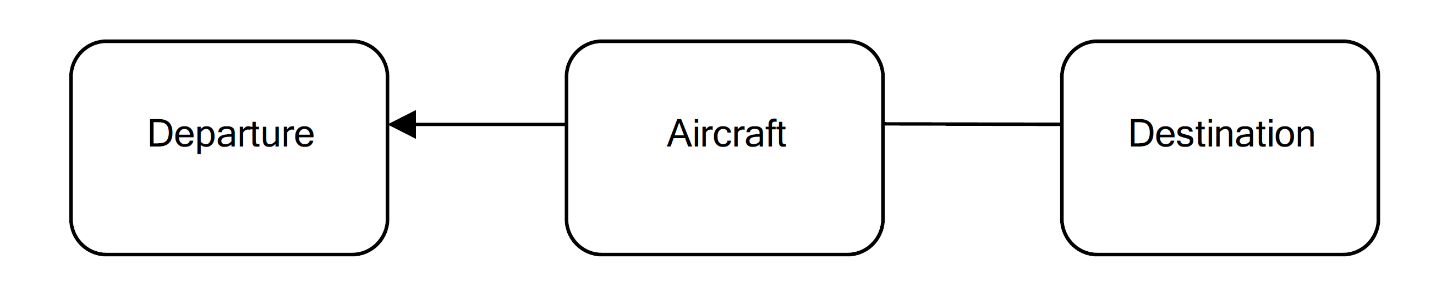
เรามาพิจารณาตัวอย่างของโครงการระบบควบคุมเครื่องบินและวิธีการสร้างความรู้ในโดเมนนี้กันดีกว่า

มีเครื่องบินหลายพันเครื่องบินอยู่ในอากาศในขณะใดขณะหนึ่งทั่วโลก เราจะไม่พยายามอธิบายระบบควบคุมการจราจรทั้งหมด แต่จะให้ความสำคัญกับส่วนย่อยที่เล็กลงซึ่งคือระบบติดตามการบิน โครงการที่เสนอคือระบบติดตามการบินที่ติดตามทุกการบินในพื้นที่ที่กำหนด กำหนดว่าการบินนั้นเป็นตามเส้นทางที่กำหนดหรือไม่ และหากมีโอกาสชนกับเครื่องบินอื่นในท้องฟ้าไหม

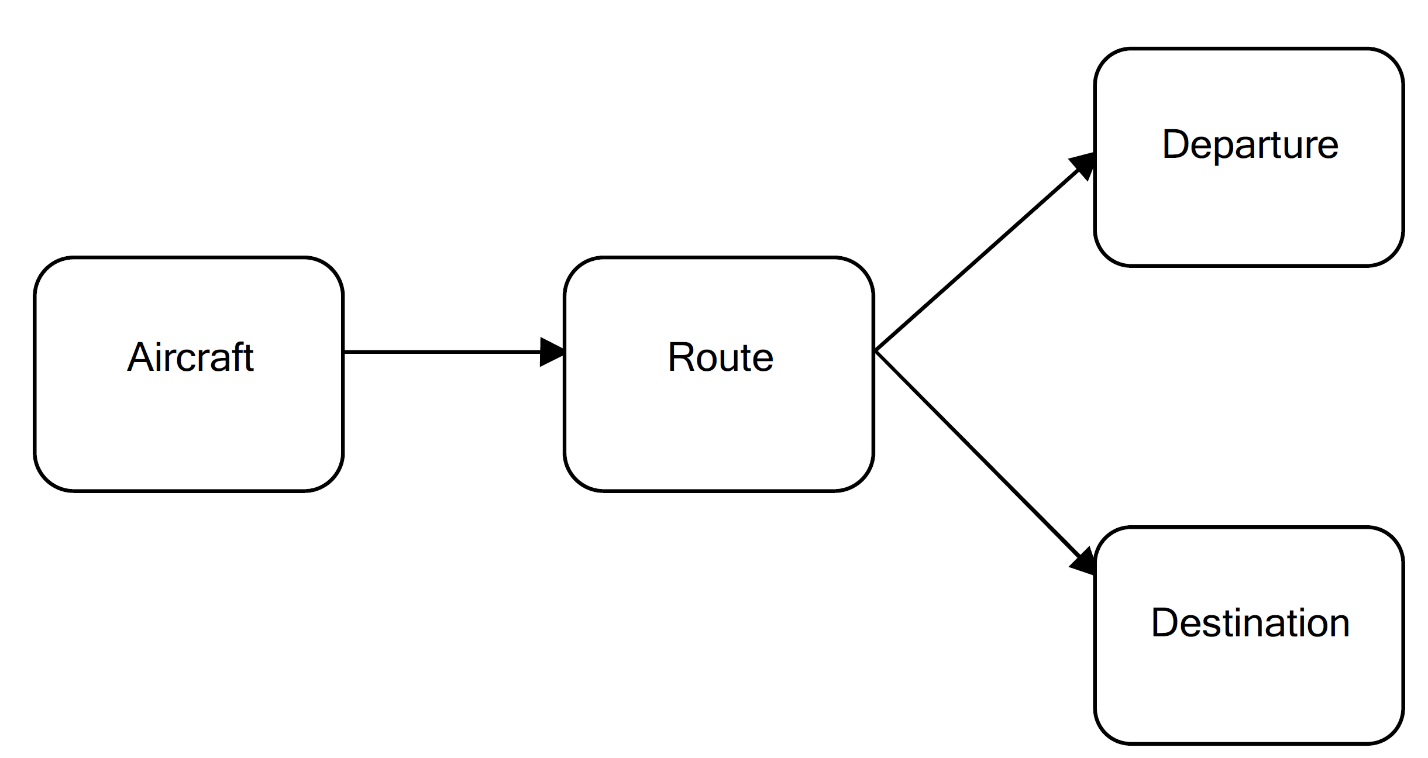
จากมุมมองการพัฒนาซอฟต์แวร์ เราจะเริ่มต้นจากไหนดี? ในส่วนก่อนหน้านี้เรากล่าวไว้ว่าเราควรเริ่มต้นด้วยการเข้าใจเรื่องโดเมน (domain) ซึ่งในกรณีนี้คือการติดตามการบินในอากาศ ผู้ควบคุมการจราจรทางอากาศเป็นผู้เชี่ยวชาญในสาขานี้ แต่ผู้ควบคุมไม่ใช่นักออกแบบระบบหรือผู้เชี่ยวชาญด้านซอฟต์แวร์ คุณไม่สามารถคาดหวังให้พวกเขามอบคำอธิบายที่สมบูรณ์ของโดเมนปัญหาให้คุณได้โดยตรง

ผู้ควบคุมการจราจรทางอากาศมีความรู้ที่มหาศาลเกี่ยวกับสาขางานของพวกเขา แต่ในการสร้างโมเดลจำเป็นต้องสกัดข้อมูลสำคัญและทำให้เป็นทั่วไป ผู้ควบคุมการจราจรทางอากาศมีความรู้ที่มหาศาลเกี่ยวกับสาขางานของพวกเขา แต่ในการสร้างโมเดลจำเป็นต้องสกัดข้อมูลสำคัญและทำให้เป็นทั่วไป เพื่อหาความเป็นระเบียบในปริมาณข้อมูลที่ดูจะไม่มีลำดับเราจำเป็นต้องเริ่มต้นที่ไหนบางที

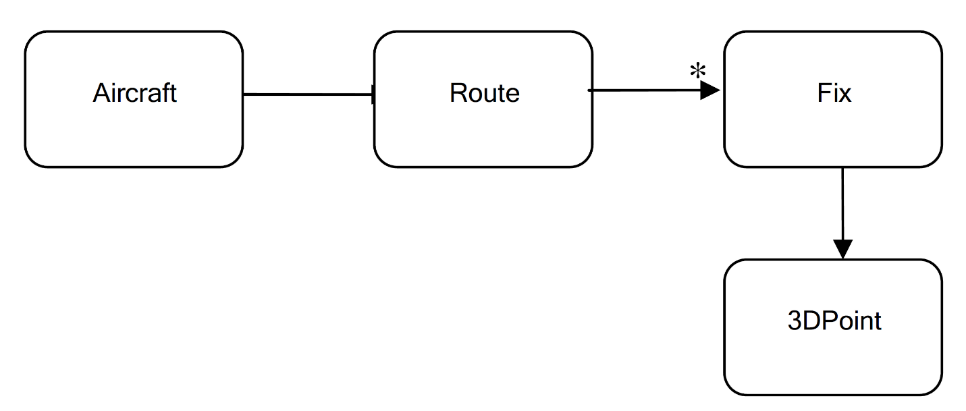
ผู้ควบคุมและคุณตกลงว่าแต่ละเครื่องบินมีสนามบินขึ้นลงต้นทางและปลายทาง ดังนั้นเรามีเครื่องบิน สนามบินขึ้นลงต้นทางและสนามบินปลายทางตามที่แสดงในรูปด้านล่าง



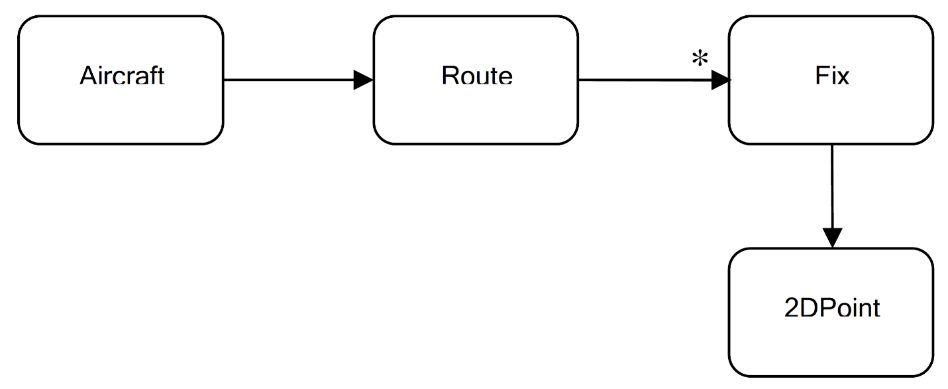
โอเค โดยอากาศยานจะบินขึ้นจากสถานที่หนึ่งและลงจอดในสถานที่อื่น แต่เกิดอะไรขึ้นในอากาศ? เส้นทางการบินที่มันผ่านไปเป็นอย่างไร? จริงๆ แล้วเราสนใจมากกว่าเกี่ยวกับสิ่งที่เกิดขึ้นขณะที่มันอยู่ในอากาศ ตัวควบคุมบอกว่าแต่ละเครื่องบินได้รับแผนการบินที่ควรอธิบายการเดินทางทางอากาศทั้งหมด ฟังเกี่ยวกับแผนการบิน คุณอาจคิดในใจว่าเป็นเกี่ยวกับเส้นทางที่เครื่องบินตามขณะที่อยู่ในอากาศ หลังจากพูดคุยกันต่อไป คุณได้ยินคำว่า "route" ที่น่าสนใจมาก และคุณตอบสนองอย่างทันที เพราะมีเหตุผลที่ดีอยู่ด้วย เส้นทางนั้นมีแนวความคิดสำคัญในการเดินทางโดยเครื่องบิน นั่นคือเครื่องบินทำอะไรขณะบินอยู่ พวกมันตามเส้นทาง มันเป็นเรื่องชัดเจนว่าจุดออกและปลายทางของเครื่องบินก็เป็นจุดเริ่มต้นและจุดสิ้นสุดของเส้นทางด้วย ดังนั้น ไม่ใช่การเชื่อมโยงเครื่องบินกับจุดต้นทางและปลายทาง แต่ดูเหมือนจะเป็นธรรมชาติมากกว่าที่จะเชื่อมโยงกับเส้นทาง ซึ่งในเทิร์นนั้นจะเชื่อมโยงกับจุดต้นทางและปลายทางที่เกี่ยวข้อง



พูดคุยกับผู้ควบคุมเครื่องบินเกี่ยวกับเส้นทางที่เครื่องบินตามไป คุณพบว่าจริงๆแล้วเส้นทางประกอบไปด้วยส่วนย่อยๆ ซึ่งรวมกันเป็นเส้นทางที่เบี่ยงเบนจากจุดออกเดินทางไปยังจุดหมายของเรา การวาดเส้นตรงควรผ่านไปที่จุดที่กำหนดไว้ล่วงหน้า ดังนั้นเส้นทางสามารถพิจารณาได้เป็นชุดของจุดที่ต่อเนื่องกัน โดยที่จุดเหล่านั้นเป็นจุดที่กำหนดไว้ล่วงหน้าแล้วว่าต้องผ่านไปในเส้นทางนั้นๆ ในจุดนี้ คุณไม่ได้มองการออกเดินทางและปลายทางเป็นจุดปลายทางของเส้นทางแล้ว แต่เพียงแค่เป็นสองจุดจากหนึ่งในจุดประสงค์นั่นเอง น่าจะแตกต่างออกไปจากวิธีการควบคุมที่ผู้ควบคุมมองเห็น แต่เป็นการแยกสิ่งที่จำเป็นที่ช่วยให้ง่ายต่อการใช้งานต่อไป การเปลี่ยนแปลงที่เกิดขึ้นจากการค้นพบเหล่านี้ มีดังนี้:



แผนภาพแสดงส่วนประกอบอีกองค์ประกอบหนึ่ง คือ ข้อความที่ว่าแต่ละจุดซ่อมเป็นจุดในพื้นที่ที่ตามมาด้วยเส้นทาง และมันถูกแสดงเป็นจุดสามมิติ แต่เมื่อคุณพูดคุยกับผู้ควบคุม คุณจะค้นพบว่าเขาไม่เห็นเหมือนกับนั้น ในความเป็นจริงเขาเห็นเส้นทางเป็นการโปรเจกชันของการบินของเครื่องบินบนโลก การแก้ไข (The fixes) คือจุดบนพื้นผิวโลกที่กำหนดโดยพิกัดละติจูดและลองจิจูดอย่างไม่เหมือนกัน เพื่อนำมาใช้ในการปรับปรุงหรือแก้ไขปัญหาที่เกิดขึ้น ดังนั้นแผนภาพที่ถูกต้องคือ:



คุณกำลังสนทนากับผู้เชี่ยวชาญด้านหนึ่ง เพื่อแลกเปลี่ยนความรู้ สิ่งที่เกิดขึ้นที่นี่คืออะไรจริงๆ? คุณเริ่มถามคำถาม และพวกเขาตอบ ในขณะที่พวกเขาทำเช่นนั้น พวกเขาขุดค้นแนวคิดสำคัญจากโดเมนการจราจรทางอากาศออกมา แนวความคิดเหล่านั้นอาจออกมาไม่สมบูรณ์และไม่เป็นระบบ แต่ยังคงเป็นสิ่งที่สำคัญต่อการเข้าใจด้านที่เกี่ยวข้อง คุณต้องเรียนรู้เกี่ยวกับโดเมนอย่างมากที่สุดจากผู้เชี่ยวชาญ และโดยการถามคำถามที่ถูกต้องและประมวลผลข้อมูลในทางที่ถูกต้อง คุณและผู้เชี่ยวชาญจะเริ่มสร้างภาพรวมของโดเมน นั่นคือ โมเดลโดเมน มุมมองนี้ไม่สมบูรณ์และไม่ถูกต้อง แต่มันเป็นจุดเริ่มต้นที่คุณต้องการ พยายามค้นหาแนวคิดสำคัญของโดเมนได้เลย

นี่เป็นส่วนสำคัญของการออกแบบ โดยทั่วไปจะมีการอภิปรายในระยะยาวระหว่างนักออกแบบซอฟต์แวร์หรือนักพัฒนากับผู้เชี่ยวชาญในด้านนี้ ผู้เชี่ยวชาญซอฟต์แวร์ต้องการแยกความรู้จากผู้เชี่ยวชาญด้านสาขาและพวกเขายังต้องแปลงมันให้เป็นรูปแบบที่มีประโยชน์ ในบางช่วง, พวกเขาอาจต้องการสร้างโปรโตไทป์เร็วๆนี้เพื่อดูว่ามันทำงานได้ดีแค่ไหนแล้ว ในขณะที่ทำเช่นนั้นพวกเขาอาจพบปัญหาบางอย่างกับโมเดลหรือวิธีการของพวกเขา และอาจต้องการเปลี่ยนโมเดล การสื่อสารไม่ได้เป็นแค่ทางเดียว จากผู้เชี่ยวชาญในโดเมนไปสู่สถาปนิกซอฟต์แวร์และต่อมาถึงนักพัฒนาโปรแกรมเท่านั้น มีการตอบกลับ (feedback) อีกด้วย ซึ่งช่วยให้มีโมเดลที่ดีขึ้นและเข้าใจโดเมนได้ชัดเจนและถูกต้องมากขึ้นด้วย ผู้เชี่ยวชาญโดเมนรู้ความเชี่ยวชาญของพวกเขาอย่างดี แต่พวกเขาจัดระบบและใช้ความรู้ของพวกเขาในวิธีที่เฉพาะเจาะจง ซึ่งไม่ได้เป็นวิธีการที่ดีที่สุดสำหรับการนำไปใช้ในระบบซอฟต์แวร์เสมอไปทุกครั้ง ความคิดวิเคราะห์ของนักออกแบบซอฟต์แวร์ช่วยหาคำศัพท์สำคัญในโดเมนขณะที่พูดคุยกับผู้เชี่ยวชาญในโดเมนและยังช่วยสร้างโครงสร้างสำหรับการพูดคุยในอนาคตตามที่จะเห็นในบทถัดไป เราผู้เชี่ยวชาญด้านซอฟต์แวร์ (สถาปนิกและนักพัฒนาซอฟต์แวร์) และผู้เชี่ยวชาญด้านโดเมนกำลังสร้างแบบจำลองของโดเมนร่วมกัน และแบบจำลองนี้คือสถานที่ที่สองพื้นที่เชี่ยวชาญเหล่านั้นพบกัน การดูเหมือนจะเป็นกระบวนการที่ใช้เวลามากเป็นอย่างมาก และจริงๆ แล้วเป็นเช่นนั้น เพราะในที่สุดจุดมุ่งหมายของซอฟต์แวร์คือการแก้ไขปัญหาธุรกิจในโดเมนชีวิตจริง ดังนั้นจึงต้องผสมผสานอย่างลงตัวกับโดเมนอย่างสมบูรณ์แบบ

**ภาษาที่มีอยู่ทั่วไป (The Ubiquitous Language)**

**ความจำเป็นของภาษาที่เหมือนกัน**

บทก่อนหน้านี้ได้ยกตัวอย่างว่าจำเป็นต้องพัฒนาโมเดลของโดเมนโดยการทำงานร่วมกับผู้เชี่ยวชาญด้านโดเมน อย่างไรก็ตามวิธีการนี้มักมีความยากลำบากในตอนแรกเนื่องจากมีการขัดข้องในการสื่อสารพื้นฐาน นักพัฒนามีความคิดเต็มไปด้วยคลาส เมธอด อัลกอริทึม และแพทเทิร์น และมักจะเชื่อมโยงระหว่างแนวความคิดในชีวิตจริงกับศิลปะการเขียนโปรแกรม พวกเขาต้องการดูว่าคลาสอ็อบเจ็กต์จะถูกสร้างขึ้นอย่างไรและมีความสัมพันธ์กันอย่างไร พวกเขาคิดเช่นเดียวกับ inheritance, polymorphism, OOP เป็นต้น และพูดคุยกันเช่นนั้นตลอดเวลา และมันเป็นสิ่งปกติสำหรับพวกเขาที่จะทำเช่นนั้น เพราะนักพัฒนาจะยังคงเป็นนักพัฒนาอยู่เสมอ แต่ผู้เชี่ยวชาญด้านของโดเมนที่มักจะไม่รู้อะไรเกี่ยวกับสิ่งเหล่านี้ เขาไม่มีความรู้เกี่ยวกับ software libraries, frameworks, persistence และในบางกรณีอาจไม่รู้อะไรเกี่ยวกับฐานข้อมูลเลย พวกเขารู้เกี่ยวกับพื้นที่ความเชี่ยวชาญของตนเองเท่านั้น

ในตัวอย่างการตรวจสอบการจราจรทางอากาศ ผู้เชี่ยวชาญด้านโดเมนรู้เกี่ยวกับเครื่องบิน เส้นทาง ความสูงของเครื่องบิน ลองจิจูดและละติจูด พวกเขารู้เกี่ยวกับการเบนทางจากเส้นทางปกติและเกี่ยวกับแนวโน้มของเครื่องบิน และพวกเขาพูดถึงเรื่องเหล่านี้โดยใช้ภาษาคำศัพท์ของตนเองซึ่งไม่ใช่สิ่งที่เข้าใจได้ง่ายโดยผู้ที่ไม่ได้อยู่ในวงการนั้น

เพื่อเอาชนะความแตกต่างในสไตล์การสื่อสาร ในขณะที่เรากำลังสร้างโมเดล เราต้องสื่อสารเพื่อแลกเปลี่ยนความคิดเห็นเกี่ยวกับโมเดล การสื่อสารเกี่ยวกับองค์ประกอบที่เกี่ยวข้องในโมเดล ว่าเราเชื่อมต่ออย่างไร สิ่งที่เป็นประโยชน์และสิ่งที่ไม่เกี่ยวข้องกับโมเดล การสื่อสารในระดับนี้เป็นสิ่งที่สำคัญอย่างยิ่งสำหรับความสำเร็จของโครงการ หากมีคนพูดอะไรไปแล้วอีกคนไม่เข้าใจ หรืออย่างแย่ที่สุดเข้าใจผิด โอกาสในการประสบความสำเร็จของโครงการจะเป็นอย่างไรบ้าง?

โปรเจกต์จะเผชิญกับปัญหาหนัก เมื่อสมาชิกในทีมไม่มีภาษาที่ร่วมกันในการพูดคุยเกี่ยวกับโดเมน ผู้เชี่ยวชาญด้านโดเมนใช้ภาษาเฉพาะของพวกเขา ในขณะที่สมาชิกในทีมทางเทคนิคมีภาษาของตนเองที่ปรับให้เหมาะกับการพูดคุยเกี่ยวกับโดเมนจากมุมมองด้านการออกแบบ

คำศัพท์ที่ใช้ในการพูดประจำวันไม่สอดคล้องกับคำศัพท์ที่ฝังอยู่ในโค้ด (ซึ่งเป็นผลิตภัณฑ์ที่สำคัญที่สุดของโครงการซอฟต์แวร์) และแม้ว่าคนเดียวกันจะใช้ภาษาที่แตกต่างกันในการพูดและเขียน ดังนั้น แสดงให้เห็นว่าประโยคที่เจาะจงที่สุดของโดเมนบ่งชี้ในรูปแบบชั่วคราวที่ไม่เคยถูกบันทึกไว้ในโค้ดหรือแม้แต่เขียนลงกระดาษ

ในช่วงการสื่อสารเหล่านี้ การแปลถูกนำมาใช้บ่อยครั้งเพื่อให้ผู้อื่นเข้าใจเกี่ยวกับแนวคิดบางอย่าง นักพัฒนาอาจพยายามอธิบายแนวคิดอาจจะใช้ภาษาที่ไม่เชี่ยวชาญและบางครั้งก็ไม่ประสบความสำเร็จ ผู้เชี่ยวชาญด้านสาขาอาจพยายามอธิบายแนวคิดของพวกเขาโดยสร้างภาษาวิชาการใหม่ ในขั้นตอนนี้ การสื่อสารมีปัญหา และการแปลประเภทนี้ไม่ช่วยในการสร้างความรู้

เรามักใช้ภาษาโดยเฉพาะของเราในช่วงการออกแบบเหล่านี้ แต่ไม่มีภาษาโดยเฉพาะใดที่เป็นภาษาทั่วไปเนื่องจากไม่มีภาษาใดที่ตอบโจทย์ของทุกคน

เราจำเป็นต้องพูดภาษาเดียวกันเมื่อเราพบกันเพื่อพูดคุยเกี่ยวกับโมเดลและกำหนดกฎเกณฑ์ ภาษาที่จะเป็นอย่างไร? ภาษาของนักพัฒนา? ภาษาของผู้เชี่ยวชาญในด้าน? หรือบางอย่างอยู่ระหว่างทั้งสอง?

หลักการหลักของการออกแบบด้วยการใช้งาน Domain-Driven Design คือการใช้ภาษาที่เชื่อมโยงกับโมเดล โดยเนื่องจากโมเดลเป็นจุดรวมกลางที่ซอฟต์แวร์และโดเมนพบกัน ดังนั้น เป็นเหมาะที่จะใช้โมเดลนี้เป็นพื้นฐานสำหรับการสร้างภาษาในการออกแบบโปรแกรม โดยใช้โมเดลเป็นฐานในการสร้างภาษานี้

ใช้โมเดลเป็นตัวกระดูกหลักของภาษา ขอให้ทีมใช้ภาษาโดยสม่ำเสมอในการสื่อสารทั้งหมด รวมถึงในรหัสโปรแกรมด้วย ในขณะที่แบ่งปันความรู้และพัฒนาโมเดล ทีมใช้สื่อต่างๆ เช่น การพูด การเขียน และแผนภูมิ ให้แน่ใจว่าภาษานี้ปรากฏอย่างสม่ำเสมอในรูปแบบการสื่อสารที่ทีมใช้ทั้งหมด เพื่อเหตุนี้ ภาษาเรียกว่า Ubiquitous Language

ภาษาสากลเชิงกว้างเชื่อมโยงทุกส่วนของการออกแบบ และสร้างพื้นฐานให้ทีมออกแบบทำงานได้อย่างดี ในโครงการออกแบบขนาดใหญ่ อาจใช้เวลาหลายสัปดาห์หรือแม้แต่หลายเดือนในการสร้างแบบจากต้นแบบ สมาชิกในทีมค้นพบว่าบางแนวคิดเริ่มต้นไม่ถูกต้องหรือถูกใช้ไม่เหมาะสม หรือพวกเขาค้นพบองค์ประกอบใหม่ของการออกแบบที่ต้องพิจารณาและเหมาะสมกับการออกแบบโดยรวม นี้ทั้งหมดไม่สามารถทำได้โดยไม่มีภาษาที่เป็นร่วมกัน

ภาษาไม่ปรากฏขึ้นแบบครั้งใดครั้งหนึ่ง มันต้องใช้ความพยายามและการมุ่งมั่นอย่างมากเพื่อให้แน่ใจว่าองค์ประกอบสำคัญของภาษาถูกเปิดเผยขึ้นมา เราต้องหาแนวคิดสำคัญเหล่านั้นที่กำหนดโดเมนและการออกแบบ และหาคำที่เกี่ยวข้องกับพวกเขา และเริ่มใช้พวกเขา บางอย่างจะมองเห็นได้ง่าย แต่บางอย่างยากขึ้น

ปรับปรุงปัญหาโดยการทดลองใช้ส่วนประกอบทดแทน ที่สะท้อนรูปแบบทดแทนต่าง ๆ จากนั้นทำการปรับโค้ดโดยการเปลี่ยนชื่อคลาส และเมท็อด และโมดูลเพื่อให้เข้ากับรูปแบบใหม่ เรียกใช้คำศัพท์ที่สับสนในการสนทนาในลักษณะเดียวกับเราเข้าใจความหมายของคำธรรมดา

การสร้างภาษาเช่นนั้นมีผลลัพธ์ที่ชัดเจน: โมเดลและภาษาที่สร้างขึ้นมีความเชื่อมโยงกันอย่างแน่นหนา การเปลี่ยนแปลงในภาษาจะต้องเป็นการเปลี่ยนแปลงในโมเดลด้วย

**การสร้างภาษาแบบทั่วไป**

วิธีที่จะเริ่มต้นสร้างภาษาคืออะไร? นี่คือสมมติฐานการสนทนาระหว่างนักพัฒนาซอฟต์แวร์และผู้เชี่ยวชาญด้านการติดตามการจราจรทางอากาศ โปรดสังเกตคำที่ปรากฏเป็นตัวหนา

**นักพัฒนา:** พวกเราควรเริ่มต้นจากอะไรกันดีเพื่อสร้างภาษาในการติดตามการจราจรทางอากาศ?

**ผู้เชี่ยวชาญด้านสนามบิน:** เริ่มต้นกับพื้นฐานกันก่อนนะคะ การจราจรทั้งหมดนี้ประกอบด้วย**เครื่องบิน(planes)** แต่ละเครื่องบินจะขึ้นจาก**สนามบินต้นทาง(departure)**และลงที่**สนามบินปลายทาง(destination)**

**นักพัฒนา:** ง่ายมากครับ ตอนที่เครื่องบินบินได้แล้ว นักบินสามารถเลือกเส้นทางอากาศที่ต้องการได้เองหรือไม่? สามารถเป็นไปตามต้องการของพวกเขาในการตัดสินใจเส้นทางที่จะบินไปได้ เพียงแค่พวกเขาเดินทางไปถึงจุดหมายได้เท่านั้นใช่ไหมครับ?

**ผู้เชี่ยวชาญด้านสนามบิน:** โอ้ไม่ นักบินได้รับเส้นทางที่ต้องเดินทางตาม และพวกเขาควรที่จะติดตามเส้นทางนั้นให้ใกล้เคียงที่สุดเท่าที่จะเป็นไปได้

**นักพัฒนา:** ฉันกำลังคิดเส้นทางนี้เป็นเส้นทางสามมิติในอากาศ หากเราใช้ระบบพิกัดคาทีเซียน แล้วเส้นทางก็จะเป็นเพียงชุดของจุดสามมิติ

**ผู้เชี่ยวชาญด้านสนามบิน:** ฉันไม่คิดว่าเป็นเช่นนั้น เพราะเส้นทางจราจรอากาศจะเป็นการสร้างภาพลักษณ์ของเส้นทางที่คาดหวังของเครื่องบินบนพื้นดิน ซึ่งเส้นทางจะผ่านไปที่จุดต่างๆบนพื้นดินที่กำหนดโดยพิกัดละติจูดและลองจิจูดของมัน

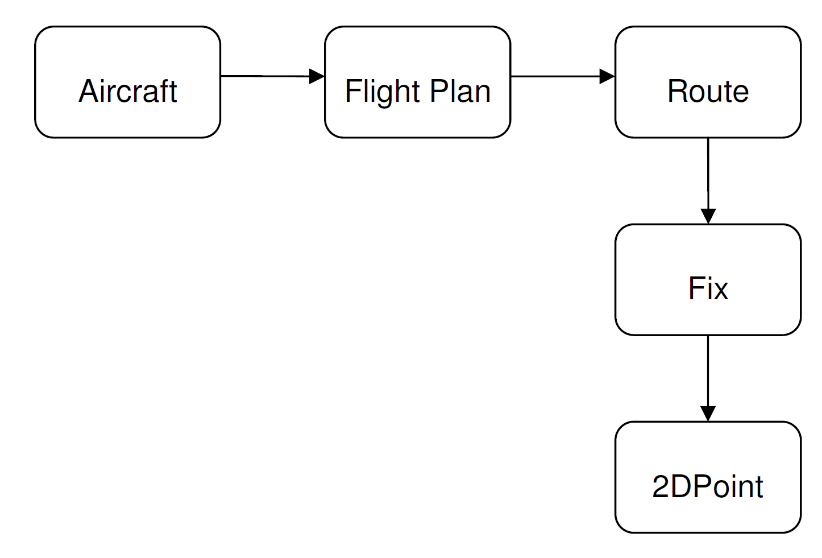
**นักพัฒนา:** โอเค แล้วเราจะเรียกจุดเหล่านั้นว่า "fix" เพราะว่าเป็นจุดที่ติดตั้งบนพื้นผิวของโลก และเราจะใช้จุด 2 มิติเพื่ออธิบายเส้นทาง นอกจากนี้ เราไม่ควรพิจารณาจุดเริ่มต้นและปลายทางเป็นแนวคิดที่แตกต่างกัน เส้นทางจะไปถึงปลายทางเหมือนกับการไปถึง fix ใด ๆ อื่น ๆ ของเส้นทาง อากาศยานต้องปฏิบัติตามเส้นทาง แต่นั่นหมายความว่ามันสามารถบินสูงหรือต่ำได้ตามที่ต้องการหรือไม่?

**ผู้เชี่ยวชาญด้านสนามบิน:** ไม่ใช่ **ความสูง(altitude)**ที่เครื่องบินจะต้องอยู่ในขณะใดขณะหนึ่งก็ถูกกำหนดไว้ในแผนการบิน**(flight plan)**

**นักพัฒนา**: **แผนการบิน(flight plan)**? มันคืออะไร?

**ผู้เชี่ยวชาญด้านสนามบิน:** ก่อนออกจากสนามบิน นักบินจะได้รับแผนการบินที่ละเอียดอ่อนซึ่งรวมถึงข้อมูลทั้งหมดเกี่ยวกับ**การบิน(flight):** **เส้นทางเที่ยวบิน(route)** **ความสูง(altitude)**ที่จะบิน **ความเร็ว(speed)** ในการบิน ประเภทของเครื่องบิน และข้อมูลเกี่ยวกับสมาชิกของลูกอากาศยาน

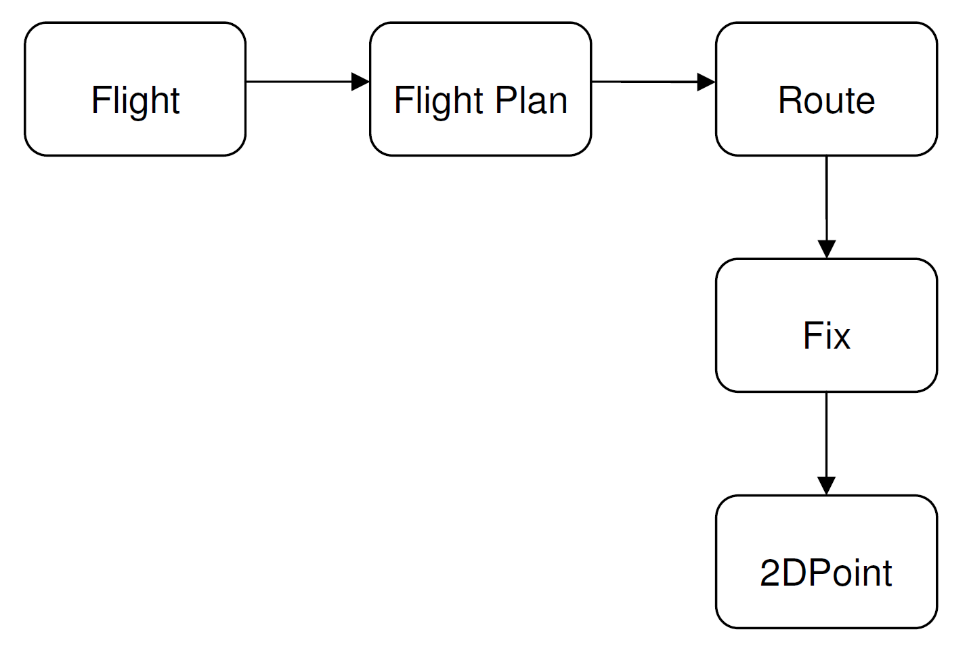
**นักพัฒนา:** ฮึ่ม **แผนการบิน(flight plan)** ดูเป็นสิ่งสำคัญอย่างมากสำหรับผม เรามาเพิ่มมันเข้าไปในโมเดลกันเถอะ



**นักพัฒนา:** ดีขึ้นแล้ว ตอนนี้ที่ฉันมองอยู่ฉันเข้าใจว่ามีสิ่งหนึ่งที่สำคัญ คือเมื่อเราตรวจสอบการจราจรทางอากาศ เราไม่ได้สนใจเครื่องบินเป็นอย่างแท้จริง ว่ามันสีขาวหรือสีน้ำเงินหรือเป็น Boeing หรือ Airbus อย่างไร แต่เราสนใจ**การบิน**ของมัน เป็นสิ่งที่เราติดตามและวัดได้อย่างแท้จริง ฉันคิดว่าเราควรเปลี่ยนโมเดลนิดหน่อยเพื่อให้มีความแม่นยำขึ้น

โปรดสังเกตว่าทีมนี้กำลังพูดถึงด้านการตรวจสอบการจราจรทางอากาศและโมเดลเริ่มต้นของพวกเขา พวกเขากำลังสร้างภาษาที่ประกอบด้วยคำที่เป็นตัวหนาอย่างช้าๆ โปรดสังเกตว่าภาษาดังกล่าวสามารถเปลี่ยนแปลงโมเดลได้!

อย่างไรก็ตามในชีวิตจริง การสนทนาแบบนี้มักจะมีคำพูดที่ยืดหยุ่นมากขึ้นและบุคคลมักพูดเกี่ยวกับเรื่องที่ไม่ตรงตามประเด็น หรือเข้ารายละเอียดมากเกินไป หรือเลือกใช้คำศัพท์ที่ไม่ถูกต้อง ซึ่งสามารถทำให้การคิดค้นภาษาเป็นเรื่องยากมากๆ อย่างไรก็ตาม เพื่อเริ่มต้นการแก้ไขปัญหานี้ สมาชิกทุกคนควรมีความตระหนักถึงความจำเป็นในการสร้างภาษาที่เหมือนกัน และควรได้รับการเตือนให้มุ่งเน้นสิ่งที่สำคัญและใช้ภาษาตลอดเวลาที่จำเป็น ในการประชุมดังกล่าว เราควรใช้ภาษาเฉพาะทางของเราให้น้อยที่สุดและควรใช้ภาษาแบบ Ubiquitous เพราะสิ่งนี้ช่วยให้เราสื่อสารได้อย่างชัดเจนและแม่นยำ



สามารถเขียนคลาสสำหรับ Route และอีกคลาสสำหรับ Fix ได้ คลาส Fix อาจสืบทอดจากคลาส 2DPoint หรือสามารถมี 2DPoint เป็นแอตทริบิวต์หลักได้ ซึ่งขึ้นอยู่กับปัจจัยอื่น ๆ ที่จะถูกพูดถึงในภายหลัง การสร้างคลาสสำหรับแนวคิดแบบนี้จะช่วยให้เราแมประหว่างแบบจำลองและโค้ด และแมพระหว่างภาษาและโค้ดได้ ซึ่งมีประโยชน์อย่างมาก เนื่องจากทำให้โค้ดอ่านง่ายขึ้น และทำให้โค้ดสามารถสะท้อนแบบจำลองได้ เมื่อโค้ดของแบบจำลองมีขนาดใหญ่ขึ้น และเมื่อมีการเปลี่ยนแปลงในโค้ดที่ออกแบบไม่เหมาะสมอาจทำให้เกิดผลกระทบที่ไม่ต้องการในโปรเจกต์ในภายหลัง

เราได้เห็นว่าภาษาถูกแชร์โดยทีมทั้งหมดแล้ว และยังช่วยสร้างความรู้และสร้างโมเดลได้อย่างไร้ข้อจำกัด แต่เราควรใช้อะไรสำหรับภาษา? เพียงแค่พูดหรือเท่านั้นหรือเราจะใช้ภาษาเขียนด้วย?

บางคนอาจพูดว่า UML เพียงพอต่อการสร้างโมเดล และในความจริงเป็นเครื่องมือที่ยอดเยี่ยมในการเขียนแนวคิดหลักเช่นคลาส และในการแสดงความสัมพันธ์ระหว่างพวกเขา คุณสามารถวาดคลาสสี่หรือห้าอย่างบนสก็อตแบบดินสอ และเขียนชื่อของพวกเขา และแสดงความสัมพันธ์ระหว่างพวกเขาได้อย่างง่ายดาย ทุกคนสามารถเข้าใจว่าคุณคิดอย่างไร และการแสดงกราฟิกของความคิดเป็นเรื่องง่ายต่อการเข้าใจ ทุกคนมีวิสัยทัศน์เดียวกันต่อเรื่องที่กำหนดไว้ และการสื่อสารกันก็ง่ายขึ้นตามนั้น โดยเมื่อมีไอเดียใหม่ๆเกิดขึ้นแล้ว แผนภาพก็จะถูกปรับเปลี่ยนให้ตรงกับการเปลี่ยนแปลงทางคอนเซปชั่นนั้นๆ

แผนผัง UML มีประโยชน์มากเมื่อจำนวนองค์ประกอบมีน้อย แต่ UML สามารถเติบโตได้เหมือนเห็ดหลังฝนฤดูร้อนที่สวยงาม คุณจะทำอย่างไรเมื่อมีร้อยๆ คลาสเต็มกระดาษเหมือนแม่น้ำมิสซิซิปี? มันยากในการอ่านแม้แต่สำหรับผู้เชี่ยวชาญซอฟต์แวร์ ไม่ต้องกล่าวถึงผู้เชี่ยวชาญด้านสาขาอื่นๆ พวกเขาจะไม่เข้าใจมันมากนักเมื่อมันใหญ่ขึ้น และมันก็จะเป็นเช่นนั้นแม้จะเป็นโปรเจกต์ขนาดกลาง

นอกจากนี้ UML ยังเหมาะสมในการแสดงคลาส แอตทริบิวต์ และความสัมพันธ์ระหว่างพวกเขา แต่พฤติกรรมของคลาสและข้อจำกัดก็ไม่ง่ายต่อการแสดงออกมา ดังนั้น UML ใช้เนื้อหาเป็นหมายเหตุที่วางไว้ในแผนภูมิเพื่อแสดงออกถึงสิ่งที่เราไม่สามารถแสดงได้ด้วย UML สองสิ่งสำคัญของโมเดล: ความหมายของแนวความคิดที่มันแทนและวัตถุที่ต้องการทำอะไร แต่นั่นไม่เป็นไร เนื่องจากเราสามารถเพิ่มเครื่องมือสื่อสารอื่น ๆ เข้าไปเพื่อทำเช่นนี้ได้

คือการสร้างภาพแผนผังเล็กๆ โดยแต่ละภาพจะประกอบด้วยส่วนย่อยของโมเดล แต่ละส่วนย่อยนั้นจะประกอบด้วยหลายคลาสและความสัมพันธ์ระหว่างพวกเขา ซึ่งรวมถึงส่วนใหญ่ของแนวคิดที่เกี่ยวข้องอยู่แล้ว จากนั้นเราสามารถเพิ่มข้อความในแผนผัง ข้อความนั้นจะอธิบายพฤติกรรมและผลของข้อจำกัดที่แผนผังไม่สามารถอธิบายได้ แต่ละส่วนย่อยพยายามอธิบายหนึ่งแง่มุมสำคัญของโดเมน โดยจุดเด่นที่เลือกจะช่วยให้เข้าใจส่วนที่สำคัญของโดเมนได้ดียิ่งขึ้น

เอกสารเหล่านั้นสามารถวาดเทียนมือได้เลย เพราะว่าการวาดนั้นสื่อถึงความรู้สึกว่าเป็นชั่วคราวและอาจเปลี่ยนแปลงได้ในอนาคตไม่กี่วันข้างหน้า ซึ่งเป็นจริง เนื่องจากโมเดลจะถูกเปลี่ยนแปลงหลายครั้งในช่วงเริ่มต้นก่อนที่จะเข้าสู่สถานะที่เสถียรมากขึ้น

อาจจะมีความต้องการที่จะพยายามสร้างแผนผังขนาดใหญ่ทั้งโมเดล อย่างไรก็ตาม ในส่วนใหญ่การสร้างแผนผังเช่นนั้นเป็นเรื่องยากมากแทบจะเป็นไปไม่ได้ และนอกจากนี้ แม้ว่าคุณจะสำเร็จในการสร้างแผนผังรวมนั้น แต่มันก็จะทับซ้อนกันมากเกินไปจนทำให้เข้าใจได้ยากและไม่ดีกว่าการใช้แผนผังขนาดเล็กหลายภาพแทนกัน

ควรระวังเอกสารที่ยาวเนื่องจากจะใช้เวลาเขียนมากและอาจกลายเป็นเอกสารที่ล้าสมัยก่อนที่จะเสร็จสมบูรณ์ ควรให้เอกสารตรงกับแบบจำลอง หากเอกสารเก่าไม่ใช้ภาษาที่ถูกต้องและไม่สอดคล้องกับแบบจำลองจะไม่เป็นประโยชน์มากนัก ควรพยายามหลีกเลี่ยงเอกสารเหล่านี้ในกรณีที่เป็นไปได้

การสื่อสารโดยใช้รหัสก็เป็นไปได้ แนวทางนี้ถูกสนับสนุนอย่างกว้างขวางโดยชุมชน XP รหัสที่เขียนได้ดีสามารถสื่อได้อย่างมากมาย แม้ว่าพฤติกรรมที่แสดงโดยวิธีการจะชัดเจน แต่ชื่อเมธอดชัดเจนเหมือนกับร่างกายของมันหรือไม่? การยืนยันของการทดสอบพูดเองแต่ตัวแปรและโครงสร้างรหัสโดยรวมเป็นอย่างไรบ้าง? พวกเขากำลังพูดเรื่องเต็มรูปแบบและชัดเจนหรือไม่? รหัสที่ทำงานตามประสิทธิภาพไม่จำเป็นต้องแสดงออกมาอย่างชัดเจนเสมอไป การเขียนโมเดลในรหัสเป็นเรื่องยากมาก

มีวิธีการสื่อสารอื่นๆ ในระหว่างการออกแบบ แต่เป็นเรื่องไม่ใช่วัตถุประสงค์ของหนังสือเล่มนี้ที่จะนำเสนอทั้งหมด อย่างไรก็ตาม สิ่งหนึ่งที่ชัดเจนอยู่แล้วคือ ทีมออกแบบที่ประกอบด้วยนักสถาปัตยกรรมซอฟต์แวร์ นักพัฒนา และผู้เชี่ยวชาญในสาขาความรู้ต่างๆ ต้องการภาษาที่รวมกันสำหรับการดำเนินการของพวกเขา และช่วยให้พวกเขาสร้างโมเดลและแสดงโมเดลนั้นด้วยโค้ดได้อย่างชัดเจน