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