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

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