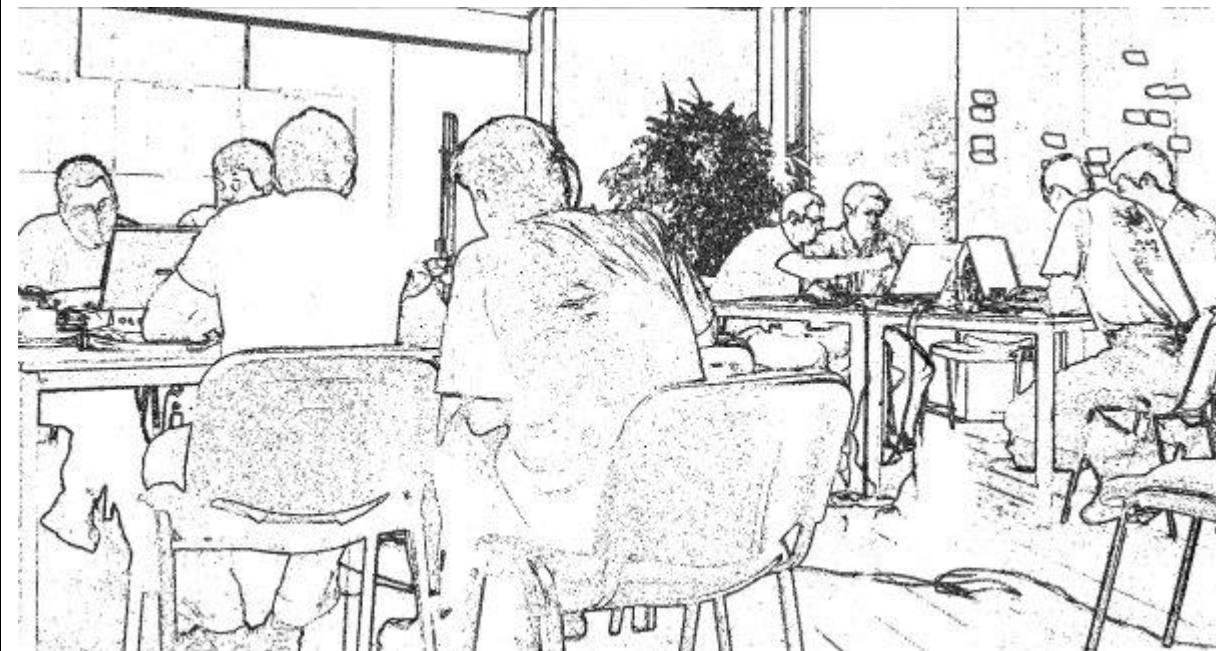


# 2011

เกรียงเพชร

Powered by [www.agile66.com](http://www.agile66.com)



## [ แอจайл์ชามไร ]

หนังสือแอจайл์ภาษาไทยเล่มแรกของโลก เที่รรabeid

## Table of Contents

คำนำ .....	11
ตอนที่ 1 แอจайл์ส์ในเปลือกนักท ..... ส่งงานดีๆทุกอาทิตย์ .....	13
การวางแผนแบบแอจайл์ส์นั้นทำงานอย่างไรกัน? .....	17
เซร์จแปลว่าเซร์จ.....	18
ความจริงทั้งสามประการ .....	19
มาตรฐานการใช้ศัพท์ .....	21
ต่อไปเป็นอะไร .....	21
ตอนที่ 2 พบกับแอจайл์ส์ทีม (ผลกระทบมันชามากหรือนินجا) .....	22
อะไรที่ทำให้แอจайл์ส์โปรดคุณแตกต่าง? .....	23
อะไรที่ทำให้แอจайл์ส์ทีมพริว .....	24
ข่ายที่นั่ง .....	25
วิ่งเสลุกค้า .....	25
จัดระเบียบตัวเองได้ .....	26
ความรับผิดชอบและความมีอำนาจ .....	27
ครอส-ฟังก์ชัน.....	27
บทบาทที่เราเห็น .....	28
คุณลูกค้าแบบแอจайл์ส .....	29
ทีมพัฒนาแบบแอจайл์ส .....	30
แอจайл์ส์อนาคติสต์ .....	31
แอจайл์ส์โปรแกรมเมอร์ .....	32
แอจайл์ส์ทีเชอร์ .....	33
แอจайл์ส์พีเอ็ม .....	34

แอจайл UX ดีไซน์เนอร์ .....	35
คนอื่นๆ .....	35
What's Next? .....	37
ตอนที่ 3 ชวนคนเขียนรถแอจайл .....	38
3.1 อะไรเป็นสิ่งที่妨ไปรบกโส่วนใหญ่ .....	38
3.2 ตามคำถมยากๆ .....	39
3.3 เข้าสู่ขั้นแห่งการเริ่มต้น .....	40
3.4 แล้วมันทำงานยังไง .....	41
3.5 ขั้นแห่งการเริ่มต้นในเปลี่ยนท .....	41
ตอนที่ 4 มองภาพรวม .....	43
4.1 ถาม: ทำไม่เราถึงมาอยู่ตรงนี้ .....	44
4.2 สร้างคำนำเสนอแบบรวมบั้ด .....	46
4.3 ออกแบบกล่องบรรจุผลิตภัณฑ์ .....	49
4.4 สร้างรายการ ไม่ทำ .....	52
4.5 พบประเพื่อนบ้าน .....	54
ตอนที่ 5 Agile ไม่ใช่แค่คิด แต่ต้องทำได้จริง .....	59
5.1 มี solution ดีๆ เอกมั่นของมาการให้ทุกคนดู (ว่าหน้าตามันเป็นยังไง) .....	60
ต้องทำอะไรยังไงล่ะนั้น .....	60
5.2 ตั้งคำถามกับตัวเองว่าอะไรที่ทำให้เราเป็นกังวล .....	61
เปิดเผยและสร้างสรรค์(ติเพื่อก่อ(เรื่องให้เป็นเรื่อง)) .....	62
5.3 ประเมินและประมาณการ .....	65
ย้ำกันอีกทีว่าอะไรบ้างที่ต้องยอมเดีย(เพื่อให้ได้อะไรมา) .....	68
5.5 มีอะไรบ้างที่จำเป็นต้องทำ (เพื่อให้สำเร็จ) .....	74
บทที่ 6 เก็บ User Story .....	81

6.1 ปัญหาของการทำเอกสาร .....	81
หรือจริงๆแล้วเราต้องทำเอกสารให้มากกว่านี้? .....	83
6.2 มาใช้ User Story กันดีกว่า .....	84
6.3 ส่วนประกอบของ User Stories ที่ดี .....	85
Wel come to Big Wave Dave's Surf Shop.....	90
6.4 How to Host a Story-Gathering Works hop.....	93
Step 1: Get a Big Open Room.....	94
Step 2: Draw Lots of Pictures .....	94
Step 3: Write Lots of Stories.....	95
Step 4: Brainstorm Everything Else .....	96
Step 5: Scrub the List and Make It Shine .....	97
What's Next? .....	97
ตอนที่ 7 :ว่าด้วยการประเมิน : คิลปะแห่งการคาดเดา .....	98
7.1 ปัญหาของการประเมินเบื้องต้น .....	99
7.2 เปลี่ยนมะนาวให้เป็นน้ำมะนาว (Turning Lemons into Lemonade) .....	101
ระบบ Point-Based (Point-Based Systems) .....	105
7.3 มันทำงานได้อย่างไร? (How Does It Work?) .....	107
ลำดับต่อไป .....	114
ตอนที่ 8 (DRAMAMAXAPF).....	116
8.1 ปัญหาของแผนแบบ สติติค (โดยไม่ make sense, static) .....	116
8.2 เริ่มวางแผนในแบบแอ็ลไจล์ .....	119
8.3 Scope ของงานสามารถเปลี่ยนแปลงได้เสมอ .....	122
8.4 Your First Plan .....	123
Step 1: สร้าง Master Story List ของคุณขึ้นมา .....	124

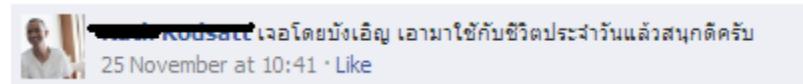
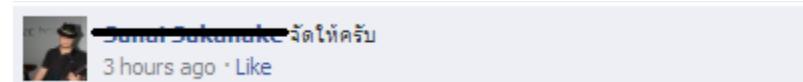
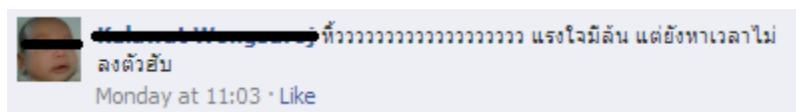
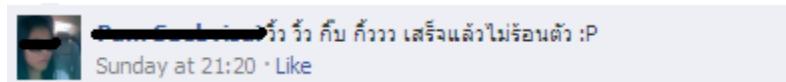
Step 2: กำหนดขนาดของแต่ละสิ่งที่เราจะทำ .....	125
Step 3: ลำดับความสำคัญ(Priority) .....	127
Step 4: ประเมินความเร็วของทีมงาน (Team's Velocity).....	128
Step 5: จัดสรรวัน .....	129
8.5 Burn-Down Chart .....	131
8.6 เปลี่ยนโปรเจคต่างๆให้กลายมาเป็นแอ็ลไจล์ .....	134
8.7 ปฏิบัติจริง .....	135
ตอนที่ 9 การจัดการ iteration ทำให้มันเกิดนะ! .....	143
9.1 ทำอย่างไรจึงจะสามารถส่งงานที่มีคุณค่าแก่ลูกค้าได้ในทุกๆสัปดาห์ ? .....	143
9.2 iteration ในแอ็ลไจล์ .....	144
9.3 ข่าวด้วย yayyy !!! .....	146
9.4 ขั้นที่หนึ่ง: การวิเคราะห์และออกแบบ: ทำให้งานมันพร้อมจะดำเนินการ .....	147
9.5 ขั้นที่สอง: การพัฒนา: การดำเนินงาน .....	152
9.6 ขั้นที่สาม: ทดสอบ: ตรวจสอบงานที่ทำ.....	154
9.7 Kanban.....	155
ตอนที่ 10: การสร้างแผนการสื่อสารแบบแอ็ลไจล์ .....	159
10.1 สื่ออย่างที่จะต้องทำใน Iteration ใดๆ .....	159
10.2 The Story Planning Meeting การประชุมเพื่อวิเคราะห์เนื้องาน .....	160
10.3 โชว์ของ The Showcase .....	161
10.4 Iteration Planning Meeting .....	161
10.5 จัด Mini-Retrospective ยังไงดี .....	163
10.6 อย่าจัด Daily Stand-Up แบบนี้.....	164
10.7 ทำอะไรก็ขอได้ถ้ามัน work สำหรับคุณ.....	165
ตอนที่ 11 สร้างพื้นที่ทำงานที่มองเห็นได้ (visual workspace) .....	168

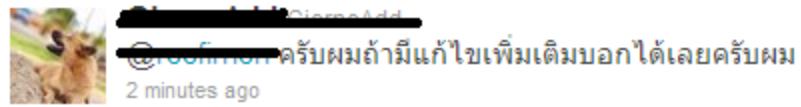
11.1 ໂອີ່ຈົດ ອານເຫັນແລ້ວ .....	169
ຕາມພວກເຮົາໃຫ້ທັນນີ້ ທ່ານຜູ້ປະວິຫາວ .....	169
11.2 ສ່ວນພື້ນທີ່ທຳງານເສີມເອົາໄດ້ຍ່າງໄວ? .....	171
11.3 ແສດງເຈຕາຂອງຄຸນ .....	173
11.4 ສ່ວນແລະແບ່ງບັນກາຫາສ່ວນກລາງ .....	173
11.5 ເຝັ້ນບັກຂອງຄຸນ .....	174
ຕອນທີ 12 ຍຸນິຕເທສ: ຢູ່ເສມວ່າ works .....	177
12.1 ຂອດ້ອນຮັບສູ່ເວັກສຄວບຖຸກທ່ານ .....	178
12.2 ເວີມຕົນທໍາມຢຸນິຕເທສ .....	179
ເຈະໄປສຶກຫາເພີ່ມເຕີມໄດ້ທີ່ແນ? .....	184
ຕອນທີ 13: ວິແຟເຕອຣ: ກລັບມາຈັດຄວາມມ້ວທາງເທກນິກທີ່ທີ່ໄ້ .....	186
13.1 Turn on a Dime (ມອງຜ່ານາຕອນແກກຂ່ານເປັນ Damn) .....	186
13.3 ເຮັມາແກ້ໄຂຄວາມຝຶດພາດດ້ວຍການທຳ Refactoring ກັນ .....	188
ບທີ 14 ທີ່ດີ່ :ເຂີຍຢຸນິຕເທສຕ່າງກ່ອນ .....	195
14.1 ເຂີຍຢຸນິຕເທສຕ່າງກ່ອນ .....	195
14.1 ຈັດກາຮຄວາມຂັບຂ້ອນດ້ວຍຢຸນິຕເທສ .....	199
ຕອນ 15 Continuous Integration: ທຳໄໝມັນພ້ອມຈະຂຶ້ນໂປຣດັກຂັ້ນເສມອ .....	205
15.1 ໄດ້ເລາໂຫຼວງ .....	205
15.2 ວັດນຮຽມການພ້ອມຈະຂຶ້ນໂປຣດັກຂັ້ນອູ່ເສມອ .....	207
15.3 ອະໄໄກສືບ ດັບການ ດັບການ ດັບການ .....	208
15.4 ມັນທໍາຍັງໄ? .....	209
15.5 ສ່ວນກະບວນການເຫັນອົບ .....	210
15.6 ສ່ວນ build ຂັດໂນມືດ .....	211
15.7 ທຳການທີ່ລະນ້ອຍາ .....	213

15.8 จะไปไหนต่อดี?.....	215
Appendix A วิธีสืบการของเจ้า.....	217
มาแปล Agile Samurai กันนะ.....	217
แบ่งงาน.....	217
หลักการ.....	218
Appendix B ทีม.....	219

# ทีมงานเยว:>มากวย่าจก: SNSD

กราบขอบพระคุณทีมงานแพลนนิ่งสีอเดือนเล่นใหญ่ ไม่มีบุคคลเหล่านี้ หนังสือเล่มนี้คงออกมาไม่ได้ครับ \_\_\_.  
ทีมงานเราเยอะมากขอเข้าชื่อและประวัติย่อไว้ที่ Appendix B นะครับ





Collapse Add

@[soi\\_nara](#) ครับผมถ้ามีแก้ไขเพิ่มเติมบอกได้เลยครับผม  
2 minutes ago



# The Agile Samurai

How Agile Masters  
Deliver  
Great Software



Jonathan Rasmusson

Edited by Susannah Davidson Pfeifer

เข้า ช้าบู มา ณ ที่นี่

# ดำเนินการ

ขอขอบคุณทีมงานผู้ร่วมทำหนังสือเล่มนี้มากครับ ที่แสดงพลังอย่างต่อเนื่องนับตั้งแต่หลังจบงาน “BarCamp Bangkok IV” ก็ ปล่อยของกันอย่างเมามันส์ไม่มียั้ง อาจจะมีสะดุดไปบ้าง เพราะมีบางท่านเป็นพ่อลูกอ่อน บางท่านก็หนีไปเมือง บางท่านก็ไปเล่นเกมส์ออนไลน์ บางท่านมาช่วยทำรูปแล้วก็หายไป บางท่านก็ติดงาน บางท่านก็หาย บางท่านก็หล่อ แต่ทุกท่านล้วนแล้วแต่มีความตั้งใจที่จะนำเสนอและนำความรู้ที่มีแบ่งปันสู่คนอื่นอย่างเต็มที่ไม่มีกั๊ก

หนังสือเล่มนี้ไม่สอนให้คุณใช้แอจайл์เป็นดังนั้นอย่าหวังว่าอ่านหนังสือเล่มนี้จะแล้วจะสามารถเดินกร่างไปบอกร้าวบ้านได้ว่า “ผมใช้แอจайл์ ผมรู้จักแอจайл์ดี จากการอ่านหนังสือแอจайл์ชั้นแนวหน้า” เพราะท่านอาจโดนตอบกลับได้เนื่องจากหนังสือเล่มนี้ เปรียบเสมือนหนังสือนำเที่ยวเกาหลีที่จะบอกว่าเกาหลีไปอย่างนั้นอย่างนี้ แอจайл์น่าใช้นะ แต่ละส่วนประกอบของแอจайл์เป็นอย่างไร กระบวนการการทำงานแบบแอจайл์มันเทยังไงแบบคร่าวๆมาก ดังนั้นสิ่งต่อไปที่ควรจะทำหลังจากอ่านหนังสือเล่มนี้جبคือ

## “จ้างทีมงานผู้แปล(บางท่าน)ไปทำ Agile Coaching ครับ”

บรรณา (หนังสือ) + อธิการ (เจ้าก้าว)

เกรียงเพรส

สารที่ 1 มกราคม 2554

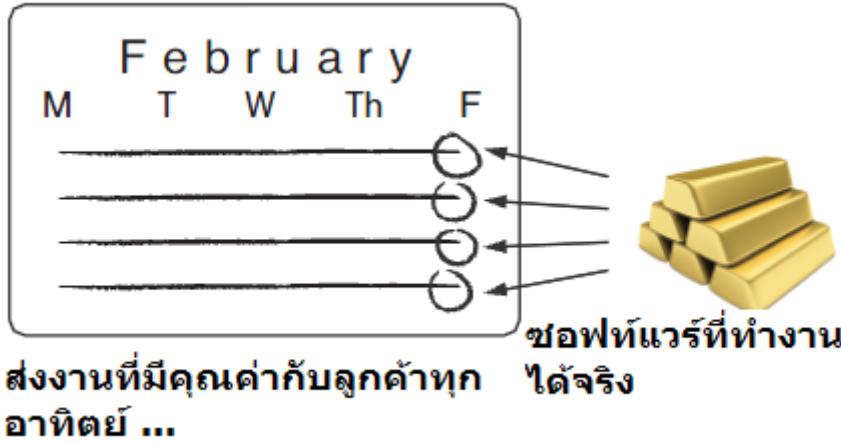
บ้านแม่ คำนาอสวรค์โลก จังหวัดสุโขทัย 64110

อีกนิด นอกจากหนังสือเล่มนี้แล้วท่านที่สนใจไฟแอจайл์ยังสามารถเข้าไปศึกษาเพิ่มเติมได้จากเว็บแอจайл์เท่าๆ อันเป็นจุดเริ่มต้นของภารหลอกท่านสมาชิกหลายท่านมาแปลหนังสือเล่มนี้ รับรองได้ว่าเนื้อหาเข้ม

**<http://www.agile66.com>**



# ตอนที่ 1 แวงในเปลือกน้ำ



เคยนั่งนึกกันเล่นๆ ไหมครับว่าถ้าเราอยากรส่งงานที่ดีและมีคุณค่าให้ลูกค้าทุกๆ อาทิตย์ เราจะต้องทำยังไง? ในบทนี้เราจะมาหาคำตอบของคำถามนี้โดยเราจะลองค้นหาดูว่าแท้ที่จริงแล้วการส่งมอบงานในมุมมองของลูกค้าจะมีลักษณะ เป็นอย่างไร นอกเหนือจากนี้เราจะได้เห็นบางมุมของการทำงานแบบเดิมๆ ที่เราคิดว่าดีแท้ๆ แต่จริงๆ มันเป็นไปไม่ได้ สำหรับเรา ที่จะรับรู้ความจริงนี้ นอกเหนือจากนี้เรายังจะได้เรียนรู้ว่าการรายงานรับทราบจริง ทั้งสามประการ (มันคืออะไร มันคือกฎหมายข้อของ หุ้นยนต์ที่เขียนโดย ไอเซ็ค อชิมอฟ หรือป่าว) จะทำให้เราภาระหนักกับการทำโปรแกรมที่มีเวลาอันแน่นจะจำกัดและยืดหยุ่นสูงสุดกับความล่าช้าอย่างหนักที่จะเกิดขึ้นกับโปรแกรม ได้อย่างไร

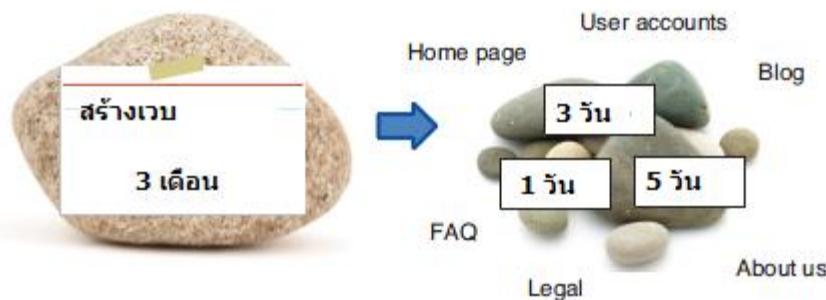
## ส่งงานดีๆ ทุกๆ อาทิตย์

สำหรับหัวข้อนี้ก่อนอื่นขอให้ทุกท่านหลับตาและลืมไปก่อนว่าจริงๆแล้วแอจайлคืออะไร เรามาลองสมมติว่าเราเองว่าเป็นลูกค้าดูกันหน่อยว่าถ้าเรามีโปรแกรมพัฒนาและมีความต้องการอยู่ในมือเราแล้วและแน่นอนเราจ้างทีมพัฒนาที่เราคิดว่าดีที่สุดเท่าที่จะหาได้ในโลกนี้เข้ามาแล้ว แต่! อะไรเป็นสิ่งที่ทำให้เราแน่ใจได้ว่าทีมที่เราจ้างมานั้นสามารถส่งงานได้ตามที่เราต้องการ? สิ่งนั้นคือเอกสารของโดยที่ประกอบไปด้วยแผนงานรายวันรายเดือนรายเดือนรายเดือน ฯลฯ หรือในทางกลับกันถ้าเปลี่ยนจากเอกสารของโดยที่เป็นทุกๆ อาทิตย์ เราจะได้เห็นและลองใช้ซอฟท์แวร์ที่ลูกค้าทดสอบมาแล้วและสามารถทำงานได้จริงโดยเราเองเป็นคนเลือกว่าสัปดาห์ต่อไปเราต้องการเห็นอะไรเพิ่มขึ้น ผนวกคำตอบของทุกคนน่าจะเป็นทางเลือกที่สอง

เราจะเห็นได้ว่าถ้าเราลองมองกระบวนการส่งมอบซอฟท์แวร์ในมุมมองของลูกค้าแล้ว เราจะมองเห็นได้อย่างชัดเจนเลยว่าลูกค้าต้องการอะไรและนั่นถือว่าเป็นการเริ่มต้นที่ดีอย่างยิ่ง คราวนี้เราลองจินตนาการว่าเราจะจัดการกับความต้องการของลูกค้าใน

มุ่งมองของลูกค้า ยังไงถึงจะเหมาะสม เช่น เรามองว่า เรายากสร้างเว็บหนึ่งเว็บ ในระยะเวลาสามเดือน เนื่องจากก้อนใหญ่ๆ นั่ง ก้อนเราระบุจัดการมันยังไงดี

- สิ่งแรกที่เราจะทำคือเปลี่ยนหินก้อนใหญ่ให้เป็นหินก้อนเล็กๆ โดยที่หินก้อนเล็กแต่ละก้อนจะเปรียบเสมือนกับ ความสามารถของเว็บที่เรารออย่างได้และคิดว่า สามารถแต่ละส่วนที่เรารออย่างได้มีความสัมพันธ์และเชื่อมต่อกันได้อย่าง ต่อเนื่องสวยงาม



ดังนั้นหลักการในข้อแรกคือพยายามแตกงานที่ยากมากๆ ออกเป็นส่วนเล็กๆ ที่ดูแล้งง่ายและสามารถจัดการได้ง่ายก่อน

- สองคือพยายามมองหาสิ่งที่ลูกค้าต้องการจริงๆ และล้มลิ่งสิ่งอื่นๆ ไปก่อน นี่เป็นปรัชญาการคิดโดยทั่วไป ให้เนื่องจากการทำงาน หรือการส่งมอบซอฟต์แวร์แบบปกติที่เราใช้กันอยู่ จะไปเน้นเรื่องอะไรก็ไม่ว่าและสิ่งเหล่านั้นเป็นสิ่งที่ลูกค้าไม่ได้ต้องการ ไม่ว่าจะเป็นกองเอกสารขนาดใหญ่ แผนที่ฯ ที่เราให้มองแล้วรู้สึกว่า แต่ถ้าเรามองกันไปที่ก้านบึงของจิตใจกันแล้วสิ่งที่ ลูกค้าต้องการมากที่สุดกลับเป็นซอฟต์แวร์ที่สามารถทำงานได้ต่างหาก ดังนั้นถ้าเราตั้งเป้าหมายในการทำงานว่า เราจะ พยายามส่งซอฟต์แวร์มั่นทุกอาทิตย์ ลูกค้าจะได้เห็นสิ่งเข้าต้องการทุกอาทิตย์ นั่นหมายความว่า เราจะมีสมาร์ทกับสิ่งที่ ต้องทำจริงและตั้งงานน้ำดิบต่างๆ ออกไป ดังนั้นเมื่อเรามีภาระไว้สร้างน้ำดิบอย่างเราสามารถทำงานได้เร็วขึ้น ลูกค้าต้องมาก ขึ้น อย่างแน่นอนและเป็นที่สุด
- แต่เหนือสิ่งอื่นใดเราต้อง มั่นใจว่า ว่า ถ้าเรามีภาระไว้สร้างน้ำดิบต้องการทำงานที่ดีจริงๆ และทุกๆ อาทิตย์ของสิ่งนั้นต้องทำงานได้ดีขึ้นเรื่อยๆ ไม่ใช่ทำงานมั่วๆ สิ่งหนึ่งที่จะทำให้ข้องของเราได้ขึ้นได้คือการทำ Tesst ทำมัน บ่อยๆ ทำแต่เนิ่นๆ ทำเยอะๆ จะไม่มีภาระของงานทั้งงานเกื้อబจัสสแล้วค่อย Tesst (และก็จะเสียเวลา) ทำให้การ Tesst เป็นส่วนหนึ่งของการทำงานรายวัน ทำให้มันเป็นวิถีแห่งชีวิต ไปเลยกว่าได้
- หลังจากมีแผนการส่งงานทุกอาทิตย์ ตอนแม่พิชาก็แล้ว อีกสิ่งที่เราต้องทำเสมอคือ การรับฟังข้อคิดเห็นจากลูกค้า เนื่องโดย ธรรมชาติแล้วเราไม่มีทางรู้เลยว่าสิ่งที่เราทำไปนั้นตรงหรือถูกต้องตามที่ลูกค้าต้องการหรือไม่ ดังนั้นทางออกที่เราต้อง ทำคือ ถามลูกค้าตัวต่อตัวว่า วันนี้เข้าหรือไม่ใช่ ถ้าไม่ใช่แล้วมันต้องเป็นยังไง การรับฟังข้อคิดเห็นเป็นเหมือนแสงส่องทางสำหรับ เรายังขาดมานาคุณภาพทาง เดาไปเองมัวไปหมด ทำให้เราขาดความสามารถในการตัดสินใจ ไม่สามารถตัดสินใจได้
- เราต้องปรับเปลี่ยนกระบวนการท่าเมื่อถึงคราวจำเป็น

## แผนที่วางแผนไว้

### แผนตอนทำจริง



อันเนื่องมาจากการเปลี่ยนแปลงสามารถเกิดขึ้นได้ตลอดไปจากตั้งนั้นวิถีแห่งการทำงานคือเมื่อการเปลี่ยนแปลงที่สำคัญที่เกิดขึ้นในสปดาห์นั้นจะส่งผลให้ขอบเขตการทำงานในสปดาห์หน้าเปลี่ยนไป เพราะโดยทั่วไปแล้วการทำงานแบบเดิมเรามักมีแผนแบบคร่าวๆ ที่เราพยายามทำงานตามแผนแบบคนดูดที่ไม่รู้ว่าปัญหาจะมาเมื่อไหร่ ดังนั้นเราจะมีเวลาอยู่มากในการแก้ปัญหาเมื่อมันวิ่งเข้ามาอยู่ข้างหน้าเราแล้ว ดังนั้นข้อสำคัญอีกข้อหนึ่งคือเมื่อไหร่ที่เกิดปัญหาที่กระทบกับแผนให้เปลี่ยนแผนในทันที ไม่ใช่ปั่นหน้าเหลียนปัญหา

6. ผลของการให้สัญญาว่าเราจะส่งงานให้ลูกค้าทุกอาทิตย์จะทำให้เราดูมีคุณค่าในสายตาลูกค้าขึ้นมาทันใด เนื่องจากเราแสดงให้เห็นว่าเงินของลูกค้าถูกใช้ไปอย่างไร และเมื่อใดที่เราไม่คุณค่าสิ่งที่เราจะได้ตามมาดีอ
  - i. That means owning quality.
  - ii. That means owning the schedule.
  - iii. That means setting expectations.

iv. That means spending the money as if it were your own

### - คำเตือน -



## อย่าเหมาเราว่าทุกคนชอบ ทำงานแบบนี้นะครับ

อย่างไรก็ตามปัญหาคลาสสิกคือไม่ใช่ทุกคนที่จะรักและชอบที่จะทำงานตามแนวคิดนี้ คนบางกลุ่มไม่นิยมบริโภคความจริงและความเชื่อในเรื่องของการส่งงานที่มีคุณค่ากับลูกค้าทุกอาทิตย์ไม่อยู่ในหัวใจของเขาเหล่านั้นเนื่องจาก การทำงานแบบนี้จะทำให้คนเหล่านั้นกลายเป็นคนที่โดยเด่นอย่างที่ไม่เคยเป็นมาก่อน มันทำให้เขาเหล่านั้นไม่มีที่จะลงทุน แต่ในทางกลับกันถ้าคุณเป็นคนที่ชอบความท้าทาย ชอบความมีตัวตน หลงใหลในเรื่องของคุณภาพ การทำงาน กับเจ้าใจที่มีจะเป็นเหมือนแรงวัลแห่งชีวิต คุณจะได้พบกับความสนุกขั้นเทพ อย่าได้กลัวอย่าได้กังวลไม่ว่าจะเป็นเรื่อง อะไรเรา ก็สามารถปรับเปลี่ยนได้ ยกตัวอย่าง เช่น มันไม่จำเป็นเสมอไปที่เราจะต้องส่งงานทุกสิ่งทุกอย่าง ที่มีก็ส่งงานครึ่งละสองสิบดาวน์หรืองานบางอย่างที่ใหญ่มากๆ เรา ก็เลื่อนเป็นสามได้ เนื่องจากแท้ที่จริงแล้วเจ้าใจล้วนเป็นแนวคิดที่ทำให้เราจดจ่ออยู่กับการคิดว่าทำอย่างไรเราถึงจะสามารถส่งงานที่ดีและมีคุณภาพให้ลูกค้าเราได้ และวันขึ้นคิดเห็น กลับมา และเมื่อมีข้อคิดเห็นใดที่ส่งผลกระทบอย่างแรงเราก็สามารถเปลี่ยนแผนได้อย่างทันท่วงที



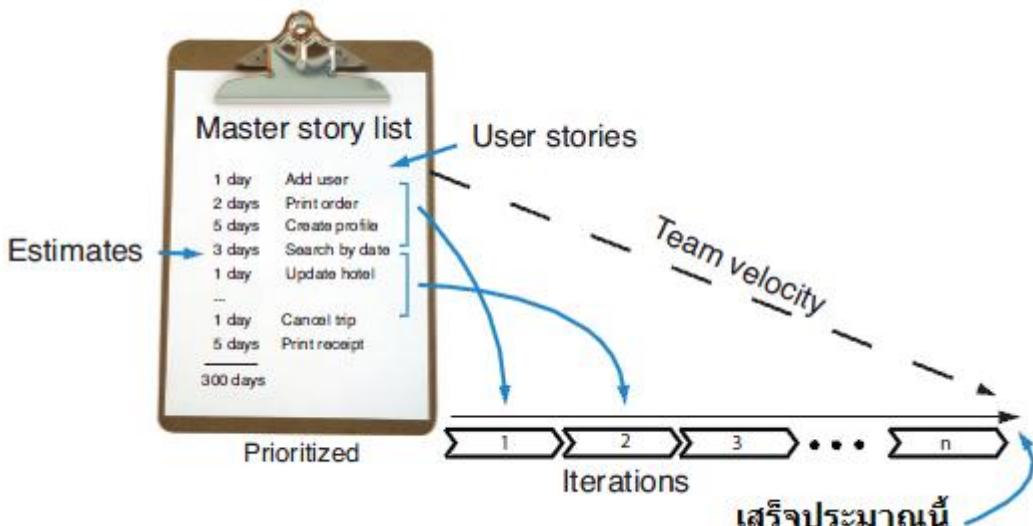
### ธรรมใจล

เห็นอสังยืนได้คือการเติมเต็มความพอด้วยให้ลูกค้าตัวยการส่งงานที่มีคุณภาพเปี่ยมล้นอย่างต่อเนื่องสม่ำเสมอ

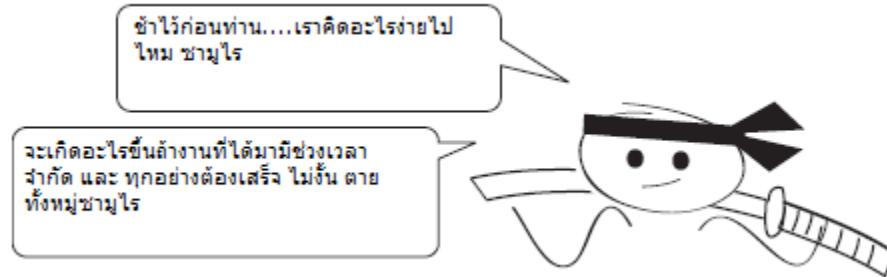
ต่อไปเรามาดูแผนแบบเจ้าใจลกัน

## การวางแผนแบบแอ็คайл์ดันทำงานอย่างไรกัน?

การวางแผนแบบแอ็คайл์ดันไม่มีอะไรพิเศษแต่เรียบง่ายสวยงาม เมื่อนำมาอ่านเข้าใจ ก็จะพบว่าในวันนี้เราสามารถนำ User stories ที่มีอยู่ใน Master story list มาเรียงตามลำดับความสำคัญ ให้เป็นไปตามความต้องการของทีมงานได้โดยง่ายๆ แบบนั้นเพียงแต่เราจะมีชื่อเรียกว่า master plan ที่ประกอบไปด้วยกิจกรรมหลักๆ หลายอันที่เราเรียกว่า user story



เราสามารถอธิบายกระบวนการด้านบนหน่อย จะเห็นว่า master story ของเราก็คือรายการงานที่ต้องทำ(to-do list) นั่นเอง โดยที่เราจะใส่ฟีเจอร์ที่ลูกค้าอยากได้แบบคร่าวๆ ไว้ใน master plan ของเราก่อนและลูกค้าเองก็จะเป็นคนเลือกหรือจัดลำดับความสำคัญของฟีเจอร์เหล่านี้เองว่าฟีเจอร์ไหนสำคัญกว่ากัน ฟีเจอร์ไหนที่ลูกค้าอยากให้ทำก่อน โดยการแบ่งนี้จะถูกใส่ไว้ในสิ่งที่เรียกว่า ไอทีโอลิจิเตอร์(Iteration) นอกจากนี้ที่มีงานของเราก็จะประเมินเวลาในการพัฒนาฟีเจอร์เหล่านั้นที่ลักษณะ(Estimation) ซึ่งเมื่อเราสามารถนิ่งๆ กันเราระบุได้แผนการทำงานแบบคร่าวๆ ของเราแล้วก่อนจะไปไกลกวนว่ามีเวลา多少 ความพยายาม เวลาเท่าไหร่ที่จะใช้กันหน่อย จริงๆ แล้วไอทีโอลิจิเตอร์คือกลัจจารสำคัญที่ขับเคลื่อนแอ็คายล์ดันโปรเจกต์ของเราโดยที่เรามักจะแบ่งช่วงการทำงานแต่ละไอทีโอลิจิเตอร์ให้ยาวประมาณหนึ่งถึงสองอาทิตย์ ส่วนปริมาณงานที่จะสามารถใส่ลงมาได้ในแต่ละไอทีโอลิจิเตอร์นั้นจะถูกประมาณได้จากความเร็ว (velocity) ในการทำงานของทีมขึ้นอยู่กับความสามารถของงานต่อไอทีโอลิจิเตอร์แบบนี้คือจะไม่มีการสูญเสียกับลูกค้าแบบนัวๆ นี่จะส่งผลให้แผนการทำงานของเรารู้สึกว่ามีความเป็นจริงเสมอและเนื่องจากความต้องการที่สูงของลูกค้า ไอทีโอลิจิเตอร์จะต้องแบ่งเป็นช่วงๆ ที่มีความสำคัญต่อการดำเนินการ เช่น ช่วงของการออกแบบ UI, การพัฒนาโค้ด, การทดสอบ, และการนำร่อง ฯลฯ ไอทีโอลิจิเตอร์ที่สั้นๆ อาจไม่สามารถสนับสนุนให้ลูกค้าได้รับประสบการณ์ที่ดีที่สุด แต่หากไอทีโอลิจิเตอร์ที่ยาวนานกว่า 4-6 อาทิตย์ อาจทำให้ลูกค้ารู้สึกว่าการดำเนินการลากยาวไปโดยไม่มีความคืบหน้า ดังนั้นจึงต้องหาจังหวะที่เหมาะสมที่สุดสำหรับแต่ละโปรเจกต์



จากคำถามชานูไรท่านข้างบน คำตอบคือถ้าคิดพอดี เรียกว่า “การวางแผน” ตามเราต้องแน่ใจว่าเรา กำลังพลิก้ายสละชีพในสิ่งที่สมเหตุสมผลไม่ใช่สิ่งที่ทำไปโดยไม่ที่มีนาหาความจริงไม่ได้ อย่างไรก็ตาม เรายังต้องยอมรับโดยสุดดีว่า แผนแบบไม่สมเหตุสมผลมักถูกสร้างขึ้นมาเสมอและเราเองก็มักจะถูกหลอกให้ทำตามแผนแบบนี้ซึ่งเป็นเรื่องที่ไม่ถูกต้อง การทำงาน ตามแผนซึ่งการทำแบบนี้เรามักเรียกว่า “ปฏิบัติการในเชิงกลยุทธ์”



แต่สำหรับแอจайл์แล็บเราไม่ต้องการปฏิบัติการเหล่านั้น เพราะเราทำงานกันแบบเปิดใจไม่晦涩难懂 และซื่อสัตย์กับลูกค้าของเราร่วมกัน ตั้งแต่วันที่เริ่มงาน เราบอกและทำในสิ่งที่มั่นใจว่าเป็นไปตามจริงและให้ลูกค้ามีส่วนร่วมในการตัดสินใจในเรื่องของบทบาท, งบประมาณ และ ระยะเวลา เราเลือกที่จะทำงานร่วมกับลูกค้าเพื่อให้ได้มารูปแบบที่ทั้งเราและลูกค้าต่างเชื่อว่ามันเป็นไปได้ ดังนั้นส่วนต่อไปคือเราต้องรู้ว่าแอจайл์กำหนดสิ่งที่จะต้องทำได้อย่างไร

## เสรีจแปลว่าเสรีจ

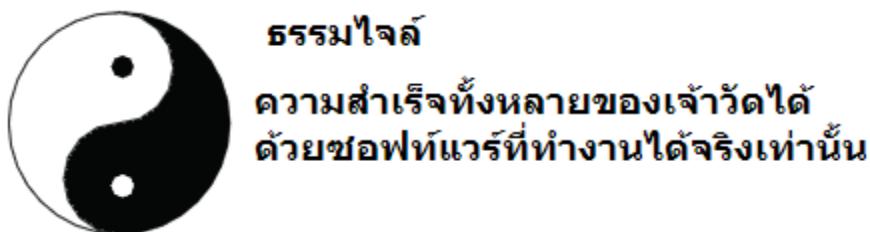
อ่านหัวข้อแล้วคงมาฟังนิทานกันดีกว่า “ถ้าเราเป็นภารกิจใหญ่” ที่สนับสนุนน้ำหน้าบ้านหลังจากนั้นคุณย่ากิจออกไปสปา เรายังคงนึกภาพดูว่าคุณย่าจะตรวจงานเด็กเกรียนยังไง คุณย่าจะพิจรณามาสิ่งเหล่านี้ไหม

- พิจรณามากแผนการตัดหญ้า การตัดหญ้าของเกรียนว่ามันจะวางแผน
- พิจรณามากเอกสารขอแบบขั้นตอนการตัดหญ้าขั้นตอนการของเกรียนเรื่องตัดหญ้า
- พิจรณามากเอกสารแผนการทดสอบความเรียบร้อยของการตัดหญ้าขั้นตอน

ถ้าย่าเราอ่านเอกสารเหล่านี้แล้วจ่ายเงินเด็กเกรียนโดยไม่ดูเลยว่า “ถ้าเราเป็นภารกิจใหญ่” เพาะในความเป็นจริงแล้วเด็กเกรียนจะได้เงินก็ต่อเมื่อ “ถ้าเราเป็นภารกิจใหญ่” ได้เรียบร้อยสะอาดตาและรวดเร็ว เนื่องจากเด็กเกรินก็ได้รับเงินแล้ว ไม่ต้องเสียเวลาตรวจสอบความต้องการของคุณย่า ดังนั้นสำหรับแอจайл์เราใช้แนวคิดเดียวกันว่า การส่งงานหมายความว่างานที่นั้นจะต้องทำได้ตามที่ลูกค้าต้องการและการและถูกทดสอบมาเป็นอย่างดีแล้วนั้นแปลว่ามันพร้อมส่งแล้ว

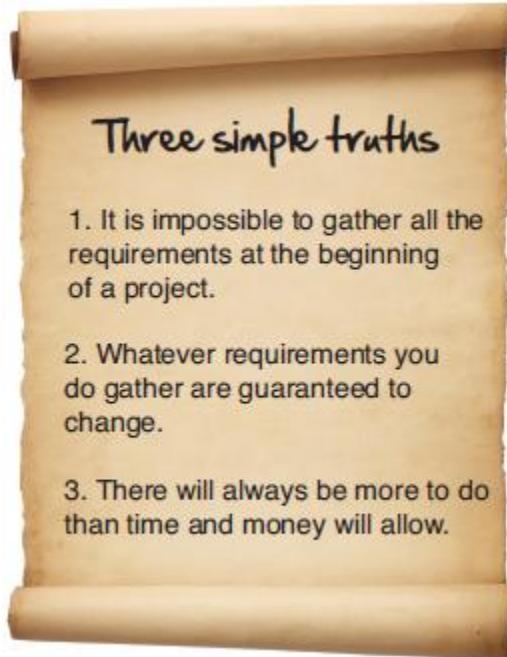


ดังนั้นถ้าเราดูจากภาพการสร้างหนึ่งฟีเจอร์จะประกอบไปด้วยกระบวนการ analysis, design, coding, testing, usability experience ทั้งหลายทั้งมวลจะถูกอัดไว้พร้อมกันเลยดังนั้น attitude ของแอจайл์คือความสามารถทำให้ถ้าเราจำเป็น จะต้องทำ และถ้าผลงานที่ได้ออกมาถูกวัดแล้วว่าไม่พร้อมแปลงว่างานยังไม่เสร็จ ดังนั้นการทำงานในรูปแบบของแอจайл์จะต้องเคร่งครัดในหลักการและมีใจน้อมรับหลักแห่งความจริงสามประการ



## ความจริงทั้งสามประการ

ต่อไปนี้คือเรื่องราวของความจริงทั้งสามประการของการสร้างซอฟท์แวร์ซึ่งเมื่อครก์ตามที่ทำใจยอมรับทั้งสามข้อนี้ได้ขาดนั้น จะเป็นอิสระพบแล้วซึ่งสჯจะรวมต่อความม่าทั้งหลายที่จะเกิดขึ้นกับการทำงานโครงการสร้างซอฟท์แวร์

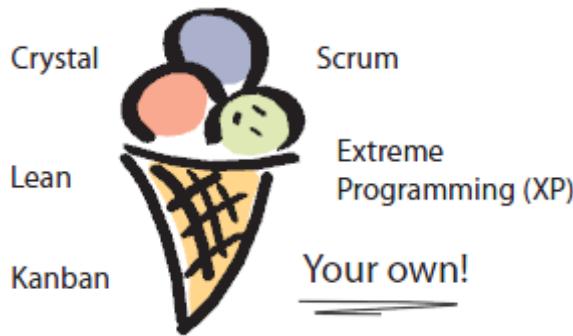


การรับความจริงข้อแรกได้นั่นหมายความว่าเราพร้อมแล้วที่จะเริ่มการเดินทางอันแสนสนุก(มั้ง) โดยที่เราไม่จำเป็นจะต้องรู้รายละเอียดเกี่ยวกับเส้นทางที่เราจะเดินทางทั้งหมด เราไปหารายละเอียดต่อเวลาหน้า และเราจะไม่ห่วงใจหากบุปผาที่มีมวลที่จะเกิด เพราะถ้าเรามัวแต่กลัวว่าจะมีข้อมูลไม่ครบเราก็จะไม่ได้เริ่มสักที

การยอมรับความจริงข้อที่สองคือเราจะไม่คาดหวังต่อ “changeeeee” เรายอมรับมันได้อย่างที่มั่นคงจะเป็น เรายสามารถปรับแผนได้เมื่อยามจำเป็นเพื่อให้เหมาะสมกับการเดินหน้าจากเดียวของเรา

การยอมรับความจริงข้อที่สามหมายความว่าเราจะไม่เครียดมากมายเมื่อเรากำลังเผชิญหน้ากับเส้นตายของทั้งเรื่องเวลาและการขาดกำลังคนที่เป็นกองกำลังเสริม เพราะเรื่องเหล่านี้ล้วนเกิดกับทุกๆ โปรเจค สิ่งที่เราทำได้เพื่อแก้ปัญหาเหล่านี้คือ จัดลำดับความสำคัญใหม่ จัดการสิ่งที่สำคัญที่สุดก่อน อันไหนไม่สำคัญมากปล่อยไว้ก่อนเดี๋ยวค่อยกลับมาทำมัน

และเมื่อเรายอมรับแล้วว่าซึ่งความจริงทั้งสามประการนี้เราจะรู้สึกปล่อยวาง ไม่เกรงกลัวต่อความกดดัน และปัญหาทั้งปวงที่เกิดกับการพัฒนาซอฟต์แวร์ “พยายามไม่มันก็ต้องเกิด” อีก เมื่อเราปล่อยวางต่อสิ่งเหล่านี้เราจะมีสมาธิกับการคิดอะไรใหม่ๆ พลิกแพลงกระบวนการท่าเพื่อจัดการกับปัญหามาใหม่ๆ เนื่องจากยังมีอีกสิ่งเล็กที่เราต้องระลึกไว้เสมอคือการใช้แอจайл์ไม่ได้มีเพียงแค่วิธีเดียว เรายังสามารถพลิกแพลงตามสถานการณ์ได้เสมอจับโน่นผอนนนี่ ให้อกมาเป็นท่าใหม่ได้เสมอเหมือนกับเวลาเราซื้อไอติม เราสามารถเลือกเอาไอติมหลายๆ รสมาร์กันให้เป็นไอติมที่เราในวันนั้นได้เสมอ



แล้วในแอจайл์มีอยู่มารสอั่วมั่ง นี่คือรายการไอดีติมที่เราสามารถเลือกได้แต่ชื่ออาจจะประหลาดหน่อย

- Scrum—แนวทางการจัดการบริหารโปรเจคที่เป็นแอจайл์
- XP—แก่นวิชาอันศักดิ์สิทธิ์ที่ถือเป็นหัวใจของกระบวนการทางวิศวกรรมซอฟต์แวร์ที่เหมาะสมมากกับทุกโปรเจคที่เป็นแอจайл์
- Lean—มหัศจรรย์สุดยอดประสิทธิภาพ, ปรัชญาการผลิตแบบโตโยต้า เป็นสุดยอดแนวทางที่ทำให้เกิดในเรื่องของการปรับปรุงองค์กร

และเนื่องจากนี่ได้คือวิธีของเรางานที่เราสร้างขึ้นมาเองและเราคิดว่าเหมาะสมกับเราเป็นที่สุด

ดังนั้นเนื้อหาหลักๆ ของหนังสือเล่มนี้จะเป็นการแบ่งปันความรู้ของผู้เขียนที่ได้ใช้งานมาไม่ว่าจะเป็นกระบวนการมาตรฐานของแอจайл์หลายแบบหรือแม้จะเป็นการที่ได้คิดค้นขึ้นมาเอง แต่อย่างไรก็ตามขอให้เราอย่าหยุดคิดค้นหากกระบวนการที่เหมาะสมกับเราเนื่องจากอย่างที่บอก ไม่มีสิ่งใดที่สามารถหยิบยกมาใช้งานได้เสมอเนื่องจากทุกโปรเจค มีความแตกต่าง ดังนั้นเราต้องรู้จักปรับกระบวนการมาตรฐานให้เข้ากับปัญหาใหม่ๆ ได้ตลอดเวลา

## มาตรฐานการใช้ศัพท์

คำศัพท์หลายคำในแอจайл์ที่ถูกใช้ในหลาย methodology ค่อนข้างจะมีความหมายที่ต่างกันแต่ก็จะมีบางคำที่มีความหมายแตกต่างกันเมื่ออยู่กับคนละที่ เช่นระหว่าง XP กับ Scrum ดังนั้นเพื่อความเป็นหนึ่งเดียวของเนื้อหาและความหมาย จะขออ้างอิงกับ XP เป็นหลัก

- ใช้ Iteration แทนคำว่า sprint
- ใช้ Master story list แทนคำว่า product back log
- ใช้คำว่า Customer แทน product owner

## ต่อไปเป็นอะไร

Project ตอนนี้เรามีวิชาพื้นฐานแล้วต่อไปเราจะเดินหน้าศึกษาเรื่องทีม บทต่อไปคือเรื่องทีมล้วนๆ เราจะได้ดูว่าทีมของแอจайл์หน้าตาเป็นไง และทำงานกันยังไง และ รายละเอียดปลีกย่อยที่ทีมต้องรู้ก่อนเริ่มการทำงาน เช่น แอจайл์ครั้งแรก

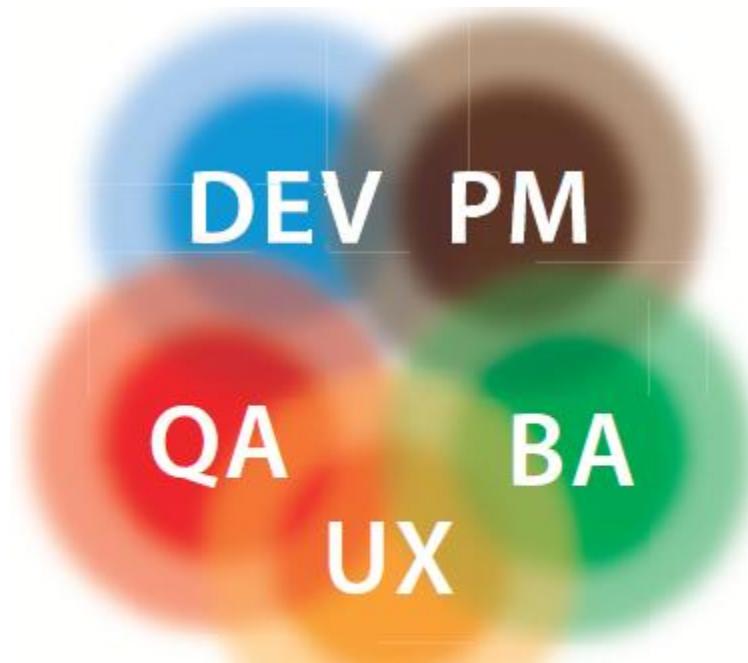
# ตอนที่ 2 พบกับเวจใจลึก (ตกลง มันชาญในเรื่องนิรนาม)



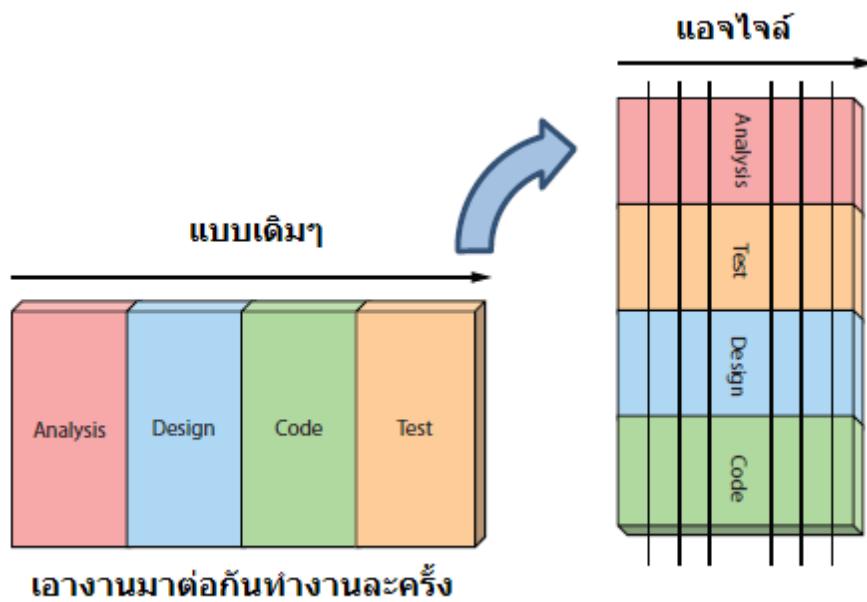
แอจไจล์ทีมเป็นสุดยอดของแปลงถ้าเราเบรียบเที่ยบกับสไตร์การเล่นฟุตบอลแล้ว มันคือให้ท่อฟุตบอล(สาวยังอ่าดิ) เราคืออย่างมาดูลักษณะพิเศษของแอจไจล์ทีมกันทีละข้อกันดีกว่า ข้อแรกแอจไจล์ทีมไม่มีการแบ่งบทบาทที่ชัดเจน ทุกคนเล่นได้ทุกตำแหน่ง แล้วแต่โอกาส เดียวากองหน้า เดียวากองหลัง เดียวากองกลาง เดียวปีก นี่คือความสับสน บุนบน เบลอๆ ไม่มีโครงสร้างทีมที่แน่นอน แต่เชื่อหรือไม่ว่าทีมแบบนี้ผลิตงานได้ขั้นสุดยอด? ในบทนี้เราจะได้เรียนรู้ถึงลักษณะอันเป็นสุดยอดของแอจไจล์ทีม อะไรทำให้แอจไจล์ทีมแตกต่างจากทีมเดอๆ กัน เทคนิคต่างๆ ที่แอจไจล์ทีมใช้ ทำให้เรารู้ว่าเราจะสร้างโคตรทีมแบบนี้เองได้อย่างไรเพื่อให้เราภีทีมที่พร้อมก่อนออกศึก

## อะไรที่ทำให้แอปพลิเคชันแตกต่าง?

ก่อนที่เราจะได้รู้ว่าอะไรคือปัจจัยที่ทำให้แอปพลิเคชันนี้มีความแตกต่างเราต้องเข้าใจเกี่ยวกับแอปพลิเคชันก่อน อย่างแรกเลยสำหรับแอปพลิเคชันน้ำที่ความรับผิดชอบของแต่ละคนจะเป็นอย่างไร ดังนั้นการทำงานกับแอปพลิเคชันจะให้อารมณ์เหมือนกับการเข้าทำงานกับบริษัทขนาดเล็กที่เพียงตัวเดียว อาจจะออกแนวทำมันๆ กันอย่างที่จะทำให้โปรเจคสำเร็จโดยไม่ได้สนใจว่าตัวแทนนั้น หรือหน้าที่ของตัวเองคืออะไร (ตัวอย่างนี้อาจจะดูสุดขั้วไปนิด)



แต่อย่างไรก็ตามแต่ละคนเองก็จะมีความถนัดไม่เหมือนกันดังนั้นแต่ละคนก็จะวนเวียนอยู่กับสิ่งที่ตัวเองถนัดจะเป็นส่วนใหญ่แต่สิ่งพิเศษสำหรับแอปพลิเคชันคือจะไม่มีการระบุตำแหน่งในระดับลึกๆ เช่นตำแหน่ง analyst, programmer หรือ tester เราหมายังคงให้ไม่เจอหรือถ้าหากเจอก็ความหมายของมันก็ต่างจากความหมายทั่วไปที่เราคุ้นเคยและอีกสิ่งที่ทำให้แอปพลิเคชันแตกต่างคือการทำ analyst, coding, design และ testing จะเกิดขึ้นตลอดเวลา มันจะวนไปเรื่อยๆ จนกระทั่งโปรเจคจบ



### พลิกมิติท่าครນทุกงานหลายครั้ง

การวนแบบนี้หมายความว่าเราไม่สามารถควบคุมที่แน่นอนของกิจกรรมเหล่านั้นได้ คนที่อยู่ในทีมต้องมีส่วนร่วมกับกิจกรรมมาก สนุกนี้ตลอดเวลาวนไปเรื่อยๆ ดังนั้นส่วนที่เราจะต้องให้ความสำคัญมากคือการทำงานเป็นทีม การมองภาพทุกอย่างจะมองผ่านทีมทั้งหมด ความรับผิดชอบทั้งหมดต้องตกอยู่กับทีมมิใช่คนใดคนหนึ่ง



และในเมื่อเรารู้ว่ากิจกรรม analyst, design, coding และ testing ไฉไลกว่ากันแล้วอีกสิ่งที่เราต้องยอมรับคือจะไม่มีส่วนที่ทำหน้าที่ QA เพราะคุณภาพคือความรับผิดชอบหลักของทีม การทำ QA เกิดขึ้นตลอดเวลาดังนั้นเราจะไม่พบคำอุทานหรือคำบ่นว่า “ทำไม QA ทีมถึงได้พลาดข้อนี้ไปได้?” ในแอจайл์プロジェクトดังนั้นสรุปโดยรวมเราจะเห็นคุณลักษณะสามสิ่งในแอจайл์ทีมคือ หน้าที่แต่ละคนจะเบ็ดเตล็ด, ทำ analyst, design, coding, testing อย่างต่อเนื่องตลอดเวลาและต้องทำงานเป็นทีมรับผิดชอบเป็นทีมเสมอ ดังนั้นต่อไปเราจะมาดูกันว่าแอจайл์ทีมทำอะไรที่ทำให้ทีมของเขางดงามนั้นประสมความสำเร็จ

## อะไรที่ทำให้แอจайл์ทีมพริ้ว

ก่อนที่ทีมของเราจะเดินหน้าดูแลก มีบางสิ่งบางอย่างที่เราต้องเรียกร้องให้ได้มาเพื่อเป็นส่วนเสริมที่ช่วยให้เราประสบความสำเร็จได้

## ย้ายที่นั่ง

การย้ายที่นั่งให้ทีมเข้ามานั่งใกล้กันเป็นหนึ่งในปัจจัยแห่งความสำเร็จ เพราะเมื่อเราอยู่ใกล้กันในイヤมที่ มีปัญหาเราจะได้รับการช่วยเหลืออย่างรวดเร็ว นั่นเป็นผลให้ความเชื่อใจกันก่อเกิดอย่างรวดเร็ว อย่างไรก็ตาม เมื่อเราเห็นข้อดีของการนั่งทำงานใกล้กัน แล้วคำถามก็จะตามมาว่า อย่างนี้เปลี่ยนว่าทีมถูกแยกออกจากกันไม่ได้หรือ? คำตอบคือไม่จริง ในบางกรณีที่ทีมไม่สามารถย้ายมาทำงานด้วยกันได้ก็ไม่ได้หมายความว่าเราจะทำ霞ใจส์ไม่ได้ เรายังมีทางออกอุดหนั่นคือ เราต้องหาวิธีเดิมเต็มซึ่งว่างของระยะเวลาด้วยการหางบประมาณมาสักก้อนหนึ่งเพื่อ จัดกิจกรรมที่ทีมจะได้มาเจอกันในช่วงก่อนเริ่มโครงการสักสี่หัวนั่น เนื่องจากนั้นมาทำอะไรก็ได้ เช่น การจัดงานและหลังจากนั้นเมื่อเริ่มโครงการเราต้อง หาเครื่องมือสื่อสารที่ง่ายและสะดวกมาให้ทีมที่อยู่แยกกันสื่อสารกันได้ง่ายๆ เช่นใช้ skype, vdo conference หรือ social media เพื่อให้เข้าเหล่านั้นรู้สึกใกล้กันแม้ว่าจะวิ่งฯแล้วไม่

## ร่วงใส่ลูกค้า

มีความจริงที่น่าเครียดอยู่หนึ่งข้อในเรื่องของการพัฒนาซอฟท์แวร์คือ มีซอฟท์แวร์หลายตัวที่ถูกสร้างขึ้นโดยทีมไม่เคยได้เจอกับลูกค้าเลยนั่นหมายความว่าซอฟท์แวร์ที่ได้ขอมาจากแนวโน้มที่ลูกค้าต้องการมากเลยที่เดียว และปัจจัยอีกข้อคือคนสร้างจะสร้างงานออกแบบได้ดีได้อย่างไร

ตัวไม่เคย เห็นหรือ เป็นส่วนหนึ่งของการบวนการทำงานเลย ดังนั้นเราต้องร่วงใส่ลูกค้าครับโดย เฉพาะคนที่เราไปแสดงเดโม คนที่ตอบคำถามเรา คนที่ให้ข้อเสนอแนะเรา ให้คำแนะนำ намายามเรา มีปัญหา จนนำคนเหล่านั้นเข้ามาร่วมส่วนหนึ่งของทีมเรา ทำให้เดาอยู่กับเราตั้งแต่เริ่มจะจบงานให้ได้

### Encourage Unplanned Collaborations



จากสารคดีเกี่ยวกับบริษัท Pixar นั้น Steve Jobs ให้ข้อคิดไว้ในสารคดีนี้ว่า Pixar มีคณะกรรมการที่ผูกติดอยู่กับสิ่งที่เรียกว่า Unplanned Collaboration มากขนาดไหน ในการสร้างภาพยนตร์ที่ประสบความสำเร็จ ขนาดนั้น โดยที่เหตุการณ์สำคัญที่เป็นตัวอย่างที่ดีคือหลังจาก Pixar ปล่อยภาพยนตร์เรื่อง Toy Story 2 ออกมานั้น (ค่อนข้างล้มเหลว) Steve รู้สึกในทันทีว่า บริษัทกำลังมีปัญหาเนื่องจากบริษัทนั้น นอยู่แยกกัน มากจนเกินไป ทำให้เหมือนทีมทำงานแบบแยกกัน ซึ่งถ้าปล่อยให้เป็นแบบนี้อีกต่อไปปัญหาใหญ่ต้องเกิดกับบริษัทนั่น นี่เป็นสาเหตุหลักที่ทำให้ Pixar ซื้อที่นา 20 Acres ที่ Emeryville, California เพื่อนำทีมที่แยกกันอยู่กลับมาร่วมกันอย่างอบอุ่น ลดลงของการย้ายครั้งนี้อกรามาในแบบอย่างมาก เนื่องจากการสื่อสารระหว่างทีมดีขึ้นส่งผลให้บริษัทสามารถสร้างภาพยนตร์ได้มากขึ้น

การทำงานด้วยกันอย่างใกล้ชิดถือ เป็นหัวใจหลักของแอลจีล์โดยก้าวได้ เราจะเป็นว่าทั้ง extreme programming และ scrum ต่างต่อสู้อย่าง แข็งขันที่จะมีทีมที่ทำงานใกล้กัน เช่นต้องยกทีมพัฒนา

ไปไว้ที่บริษัทลูกค้า และสำหรับ scrum มีการตั้ง ตำแหน่งที่เรียกว่า product owner ขึ้นมาโดยที่ตำแหน่งนี้เป็นตำแหน่งที่สำคัญมาก แต่มากเพียงใดเราจะได้ ภูมิหลังจากนี้

เกิดอะไรขึ้นถ้าเราหา “ลูกค้าคนสำคัญของเรามีได้ หรือ ไม่มีเลย?” เพราะในบางครั้งงาน ที่เรากำลังทำอาจเป็นงานที่ไม่มีใครอยากได้ หรือ ลูกค้าทุกคนคิดว่างานที่เรา กำลังจะส่งเป็นต้นเหตุแห่งความไม่สงบในชีวิตของเขาเหล่านั้นแน่ๆ ดังนั้นทางออกของปัญหานี้ ต้องแก้ด้วยการสร้างความไว้เนื้อเชื่อใจ ให้กับลูกค้าให้ได้ จริง!!!

กระบวนการสร้างความเชื่อมั่นคือเดินไปหาลูกค้าคนที่มีส่วนเกี่ยวข้องกับงาน จากนั้นบอกเขาว่า “ผมกำลังจะสร้างปัญหาให้คุณหลังจากนี้สองอาทิตย์” แล้วเดินจากมาแบบเท่า ไม่ต้องขออนุญาต ไม่ต้องฉลอง หน้าที่คือสร้างปัญหาให้เขานิดหน่อย ทำให้เขาคันและหลังจากนั้นอีกสองอาทิตย์ ให้กลับไปหาเขาแล้วแสดงให้เห็นว่าปัญหาที่เราบอกก่อนหน้านี้แก้ไปแล้วเรียบร้อย “หล่อ” แต่อย่าหยุดแค่นั้นให้หาปัญหาอื่นแล้วแก้ให้ได้อีกทำอย่างนี้สักสองสามรอบ ในไม่ช้าลูกค้าจะเริ่มมอง เราเปลี่ยนไป เปลี่ยนไปในมุมมองที่ดีขึ้น มองเห็นในลิสต์ที่เราเป็น (เหมือนจีบหญิงเลยก็ว่า ต้องอดทน) แล้วเราจะได้ใจของลูกค้าคนนั้นดังนั้นอย่ายอมแพ้ที่จะสร้างความไว้เนื้อเชื่อใจโดยให้เริ่มจากจุดเล็กๆ ก่อนแล้วจะพอกับ “ความรัก” ไม่ใช่ “ลูกค้าคนสำคัญของเรา”

## จัดระเบียบตัวเองได้

แล้วเจ้าทีมชุดที่จะทำงานในรูปแบบพิศดารเด็กน้อยคือโอนิจทัย ให้หนึ่งปีกแล้วเดียวที่มีจะสูญเสีย กว่าจะแก้ปัญหาเหล่านี้ ยังไงซึ่งการทำงานในลักษณะนี้ทีมจะต้องมีคุณสมบัติพิเศษคือจัดระเบียบตัวเองให้ซึ่งความหมายหลักของมันคือทุกคนต้องลงทะเบียนให้กับของตัวเอง ให้ที่ประดุจและเมื่อก้าวผ่าน ประตูเข้ามาแล้วต้องทำงานเพื่อทีมให้ดีที่สุด (จัดทุกอย่างของมาให้หมดความสามารถ ความคิดสร้างสรรค์ และ พรสวารค์) เพื่อให้งานที่ได้อกมาดีที่สุด

“สมชาย อาจจะเก่งเรื่องเขียนโค้ดแต่สมชายยังทำงานเกี่ยวกับการออกแบบได้ดี ดังนั้นสมชายควร จะไปช่วยทำงานต้นแบบ”  
“วนิดาเป็นเทสท์เตอร์ขั้นเทพแต่ในอีกฝั่งมุ่นใจวนิดาชายแวร์เรื่องการกำหนดความต้องการของลูก ค้าได้ ดังนั้นวนิดาต้องไปช่วยงานด้านนั้น”

แต่ทั้งนี้ทั้งนั้นเจ้าทีมสร้างไม่สามารถเหมารวมๆ เอาได้ว่า โปรแกรมเมอร์ต้องไปยุ่งเรื่องการออกแบบหน้าจอ หรือ เทสท์เตอร์ต้องไปช่วยงานผู้จัดการโครงการสมมอไปแต่เราต้องยึดหลักการที่ว่า “การสร้างทีม ที่ดีที่สุดคือการทำให้ตำแหน่งเหมาะสมกับคน ไม่ใช่ พยายามบีบคนให้ยัดลงไปในตำแหน่ง” นอกจากนี้ เรายังสามารถทำให้ทีมจัดการตัวเองได้ด้วยการทำ

- ให้ทีมสร้างแผนการทำโปรดัก ประเมินทุกอย่างของ และ เลือกที่จะรับผิดชอบงานเอง
- อย่าไปสนใจและใส่ใจเรื่องตำแหน่งและหน้าที่มากจนเกินไปทางกลับกันให้เรา หันมาใส่ใจเรื่องการส่งงานอย่างต่อเนื่องดีกว่า
- มองหาคนที่สามารถทำอะไรด้วยตัวเอง ได้คิดเองได้ว่าต้องทำอะไร ไม่ใช่ร้องขออย่างเดียวเพื่อรอคำสั่ง

ดังนั้นบทสรุปสั้นๆ สำหรับแอจайлทีมคือปล่อยให้ทีมผ่อนผันตัวเองตามธรรมชาติ จงเชื่อมั่นและ ผลักดันทีมให้ทำงานดุล่วงตามกำหนด

## Agile principle

ดังนั้นเมื่อเรามีทีมที่จัดการตัวเองได้ เราจะได้ผลงานที่ดี ออกแบบด้วยไม่ว่าจะเป็นสถาปัตยกรรมของระบบ ดีไซน์ รีโควเม้นท์ ซึ่งผลที่ออกแบบดีก็ เพราะมีบางสิ่ง บางอย่างผลักดันมันนั่นก็คือ “ความรับผิดชอบและอำนาจ” นั่นเอง

## ความรับผิดชอบและความมีอำนาจ

เจ้าใจลทีมที่ดีจะต้องรับผิดชอบต่อสิ่งที่เข้าเหล่านั้นให้สร้างออกแบบเพราะเข้ารู้ว่าลูกค้าจับตา มองเขาอยู่อย่างใกล้ชิดตั้งแต่วันแรกที่เริ่มงาน และแน่นอนความรับผิดชอบต่องานย่อมมากับทีมที่ มีอำนาจมากจากเราว่าจะปล่อยให้ทีมเป็นคนตัดสินใจ แก้ปัญหากันเอง ทีมไม่ต้องขอฟัง คำสั่งจากใคร นั่นหมายความว่าอาจเกิดข้อผิดพลาดบ้างในบางโอกาสแต่เมื่อเราเทียบกับผลงานอัน มหัศจรรย์ที่ต้องออกแบบ มันก็คุ้มมากพอก็จะลอง

อย่างไรก็ตามการสร้างทีมที่มีทั้งอำนาจและความรับผิดชอบนั้นเป็นเรื่องง่ายที่จะพูดแต่ยากที่จะลงมือสร้างเพราะไม่ใช่ทุกคนที่อยากรับมีอำนาจ ซึ่งจริงแล้วก็ไม่มีอะไรยากเลยแค่ทำให้ได้อย่างที่พูด หรือรักษาคำพูดเท่านั้นเอง

วิธีง่ายๆที่เราจะเริ่มปลูกฝังความรับผิดชอบต่องานคือเริ่มจากหยิบปัญหาง่ายๆที่แก้ได้แน่นอนออก มาแล้วให้ทีมทำเดิมสำหรับปัญหานั้นไปเสนอสู่ลูกค้า การนำทีมไปเสนองานให้ลูกค้าต่อหน้าลูกค้า เป็นการส่งเสริมให้ทีมมีอำนาจความรับผิดชอบขึ้นที่หนึ่ง เพราะเมื่อทีมได้สัมผัสถึงการรอคอยหรือ ความหวังที่คุณอื่นเมื่อต่อทีมจะทำให้ทีมมีความรู้สึกที่อยากรับมีอำนาจแก้ไขหรือช่วยเหลือคนเหล่านั้น

Second, it will take only one bad demo for your team to take a sudden interest in making sure the software is ready for feedback and every-thing works. They will insist on becoming empowered to make this happen. If they don't, you have a bigger problem.

## ครอส-ฟังก์ชัน

การที่เรามีทีมที่ทำงานแบบครอบครอสฟังก์ชันเป็นสิ่งจำเป็นมาสำหรับการรับใช้ลูกค้าสุดที่รักของเราได้ตั้งแต่ต้น จนจบ นั่นหมายความว่าเราจำเป็นต้องมีทีมที่มีความสามารถและเชี่ยวชาญในงานที่ทำเพื่อตอบสนองสิ่งที่ลูกค้าต้องการได้อย่างรวดเร็วทันท่วงที ดังนั้นที่สำคัญเริ่มต้นเมื่อเราต้องรับคนเข้ามายังทีมเรารายจะต้องรับ คนที่ไม่ได้ เชี่ยวชาญงานอะไรมากเป็นพิเศษมากๆแต่เราต้องการคนที่สามารถเรียนรู้ในงานที่ตัวเองต้อง ทำได้อย่างรวดเร็ว สามารถเรียนรู้อะไรได้หลายอย่าง เนื่องจากเมื่อคนคนนั้นเข้ามาอยู่ในทีมแล้ว เขาต้องทำงานหลายอย่าง เพื่อให้ไปถึงเป้าหมายสุดท้ายคือส่งงานที่ดีและมีคุณภาพให้กับลูกค้า และแน่นอนว่าความสวยงามของทีมแบบครอบครอสฟังก์ชันคือความเร็วระดับเทพเนื่องจากไม่ต้องมีการรอ กัน ไม่ต้องขออนุญาติใคร บางคนเพื่อทำงาน ไม่ต้องรอคนจากทีมอื่น ทีมสามารถเริ่มทำงานและส่งงาน ได้ตั้งแต่วันแรกของโปรเจคและที่สำคัญไม่มีใครสามารถหยุดคนเหล่านี้ได้

หลังจากที่เราได้เรียนรู้ว่าคุณลักษณะของแอจไจล์ทีมที่ดีเป็นอย่างไรแล้วต่อไปเรายังเริ่มมองลงไปในรายละเอียดของแต่ละคนที่มีส่วนก่อขึ้นกับโปรเจคในมุมมองของแอจไจล์

## Who Moved My Cheese?

Who Moved My Cheese? [Joh98]

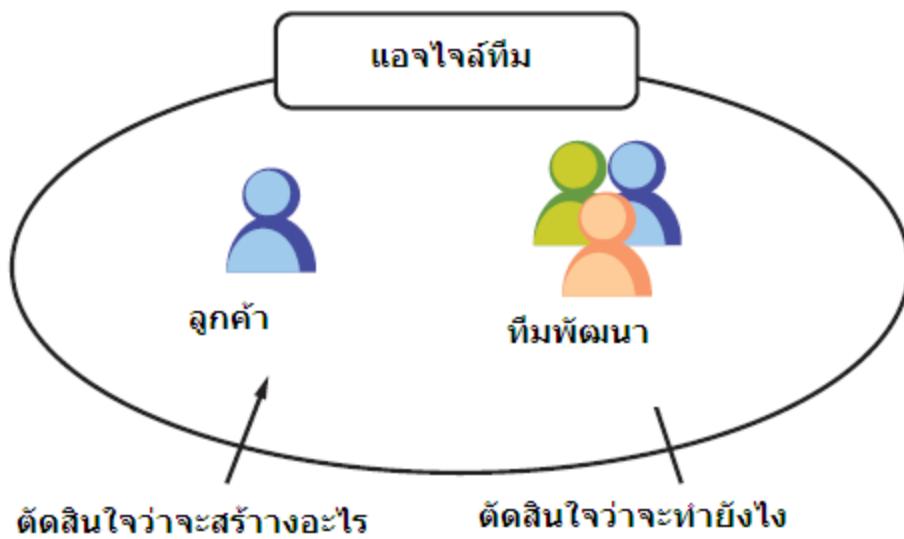
เรื่องราวของหนูที่แสนสุขสบายกับชีสบริบามนหาศากที่ตัวเองบังเอิญไปพบแต่แล้ววันหนึ่งมันก็ตื่น ขึ้นมาแล้วพบว่าชีสเหล่านั้นได้หมดไป ใครมาแอบย้ายชีสของมันไป? หนูตัวนั้นเลยตกที่นั่งลำบาก เพราะมันไม่รู้ว่าต้องทำตัวยังไงเพราะสภาพดีความสบายนี้แล้ว ดังนั้นสำหรับคนที่ถูกจับให้เปลี่ยน สภาพให้เปลี่ยนเป็นแอใจล์จะเริ่มรู้สึกว่าชีสของเขากำลังหายไป!!!!

- สำหรับโปรเจคเมเนเจอร์คนเหล่านั้นจะรู้สึกว่า ทำไม่กัน ไม่ว่าจะพยายามแค่ไหนรีควร เมนท์ก็เปลี่ยน เสมอ เช่นๆ
- สำหรับอนัลิสท์จะสงสัยว่าทำไมต้องวิเคราะห์ระบบอยู่นั้นแหล่งไม่จบสักที
- สำหรับเดเวลอปเปอร์ คนเหล่านี้จะถูกคาดหวังว่าจะต้องเขียนเทสท์(酵母)ด้วย

จะเห็นว่าการที่เราเข้าไปเปลี่ยนชีวิตคนที่ปกติเขาเหล่านั้นเคยชินกับการทำงานแบบเดิมๆ ให้เข้า เหล่านั้นทำงานกันแบบใหม่มันก็เหมือนหนูที่ถูกคนอื่นย้ายชีสไปที่อื่น ดังนั้นถึงที่เราจะช่วยคนเหล่านี้ ได้คือการสอนให้อกเดินทางไปหาชีสใหม่ นั้นเอง

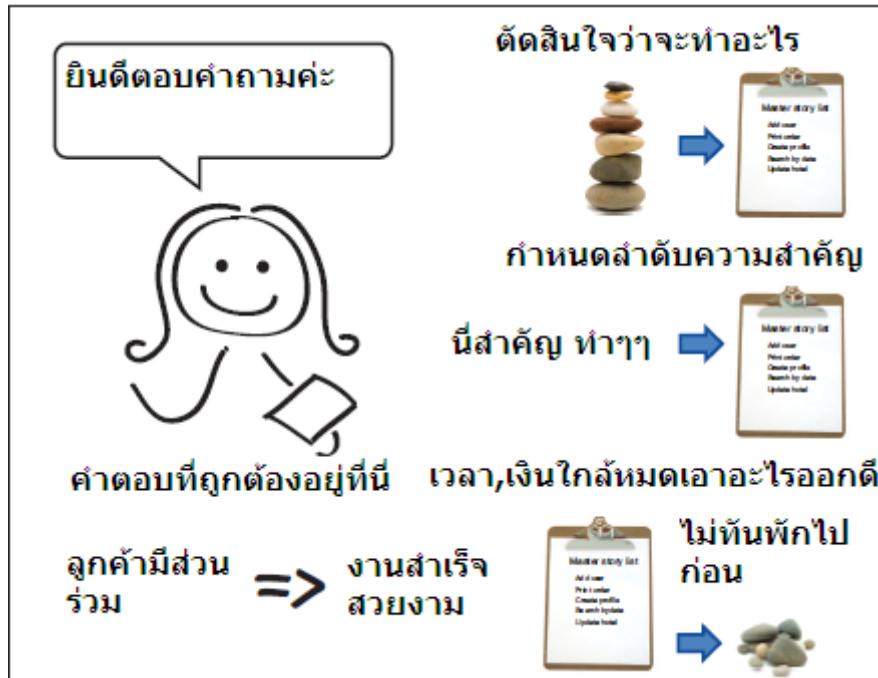
## บทบาทที่เราเห็น

สำหรับแอใจล์แล้วไม่ว่าจะเป็น XP หรือ Scrum เราจะไม่ค่อยเห็นบทบาท ที่เป็นทางการ เมื่อเราทำงานในโปรเจคแล้ว บทบาทหลักๆจะมีอยู่แค่ “คนที่รู้ว่าตัวเองอยากรีดอะไรหรือ ลูกค้า” และ “คนที่รู้ว่าไฝ่ที่คนตะกี้ต้องการมันสร้างยังไงหรือ ที่มันก้าพฒนาตนเอง”



อ่าวแย่แล้วแล้วไอล์พากไปrogramเมอร์ เทสท์เตอร์ อนัลิสท์ มันหายไปไหนหมด? อย่าได้กังวลไปครับเขามาเหล่านั้นยังอยู่แต่เราจังไม่สนใจเนื่องจากปรัชญาของแอใจล์เราสนใจเรื่องของโครงสร้างและแนวทางที่ให้มากกว่าจึงไม่สนใจว่าใช้ทบทาที่จะเล่นมัน เล่นยังไง ดังนั้นก่อนอื่นเรามาดูบทบาทแรกก่อนนั่นคือ “ลูกค้า” ในมุมมองของแอใจล์

## คุณลูกค้าแบบแอโจ้ใจล์

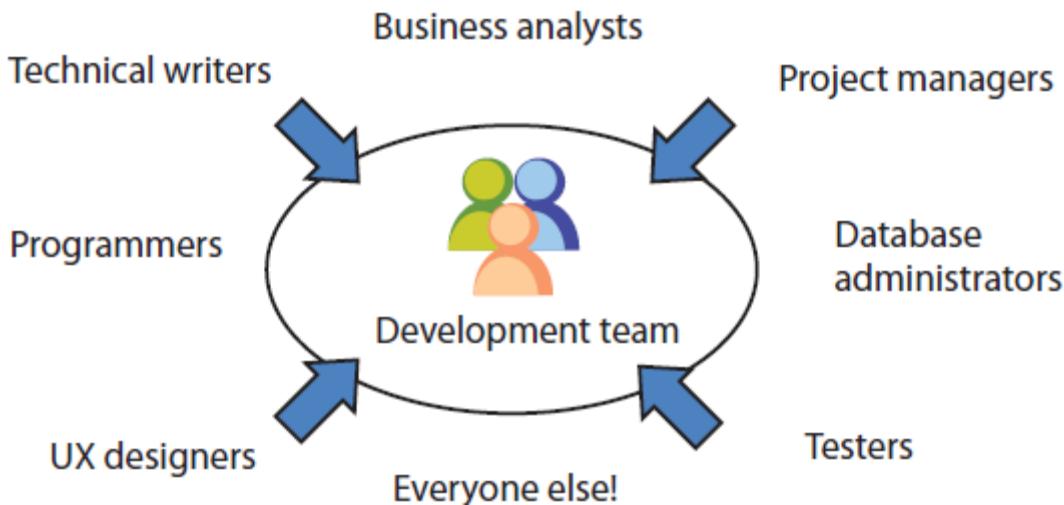


คุณลูกค้าคนนี้ภาษาฝรั่งจะเรียกว่า "Subject Matter Expert" เป็นคนที่มีความรู้เกี่ยวกับธุรกิจ รู้ว่าซื้อฟ์แวร์ควรจะทำงานอย่างไรหน้าตาเป็นยังไง นอกจากนี้ยังเป็นคนที่ช่วยเหลือในเรื่องของ การตอบคำถาม ให้ข้อคิดเห็นตลอดระยะเวลาในการพัฒนาโปรแกรมและที่สูงขึ้นมาอีกนิดคุณลูกค้าของเราต้องเป็นคนกำหนดความสำคัญของฟีเจอร์ต่างๆว่าอะไรควรจะทำก่อนอะไรควรจะทำหลัง โดยคำนึงถึงความต้องการของผู้ใช้งาน หลักก่อนแล้วนำมารีกิษากับทีมพัฒนาเพื่อแก้ไขในบางกรณีการเรียงลำดับ อาจจะต้องใช้เหตุผลทางเทคนิคเข้าไปประกอบด้วย (เพื่อลดความเสี่ยงทางด้านเทคโนโลยีไป) หลังนั้นทั้งลูกค้าและทีมพัฒนาจะมาว่ามกันทำแผนคร่าวๆโดยภาพรวมแล้วการจะทำให้สำเร็จต้องทำอะไรก่อนอะไรหลัง

สุดท้ายอันเนื่องจากสุดๆคือคุณลูกค้าจะต้องเป็นคนตัดสินใจว่าจะ "ไป" ทำอะไรในกรณีที่เราพบว่าเวลาหรืองบประมาณจำกัดแล้ว เราจะเห็นว่าคุณลูกค้ามีความสำคัญกับเรามากแค่ไหนดังนั้นทางที่ดีที่สุดคือคุณลูกค้าแบบจะต้องมาทำงานเต็มเวลา กับทีมพัฒนาเลยก็ว่าได้ เพื่อความถูกต้อง ดังนั้นถ้าเจ้าย้อนกลับไปดูกระบวนการการทำงานของ XP ในสมัยก่อนจะมีการอ้างถึงคำว่า "ไปทำงานที่บริษัทลูกค้า" เช่นหรือแม้กระทั่ง Scrum เองก็มีการกำหนดบทบาทนี้ให้เป็นบทบาทที่ต้องทำงานเต็มเวลาหรือเรียกว่า มีบทบาทที่เรียกว่าเจ้าของโปรดักส์ ผลลัพธ์มาเลยก็ว่าได้ แต่สำหรับมือใหม่หัดแอปพลิเคชันอย่าเพิ่งไปตอกอกติกามากกับเรื่องนี้ เพราะจริงๆแล้วคุณลูกค้าที่สามารถมาทำงานให้เราแบบเต็มเวลาได้ยากเต็มที่ แต่เราถึงมีหนทางในการทำงานที่ยังช่วยให้เราส่งงานที่ดีได้เหมือนกัน เพราะหนึ่งสิ่งที่สำคัญที่สุดคือความสามารถในการทำงานเข้าใจถึงกระบวนการทำงานแบบแอปพลิเคชันต่างหากเป็นปัจจัยหลักในการส่งเสริมให้งานสำเร็จ เพราะเมื่อเราเข้าใจเราจะพยายามทำงานโดยตรงกับคุณลูกค้ามากขึ้น พยายามแสดงให้เห็นว่า เข้าเล่นนั้นมีความสำคัญต่อโปรดักส์มากขนาดไหน ความสำเร็จก็จะตามมาอย่างแน่นอน

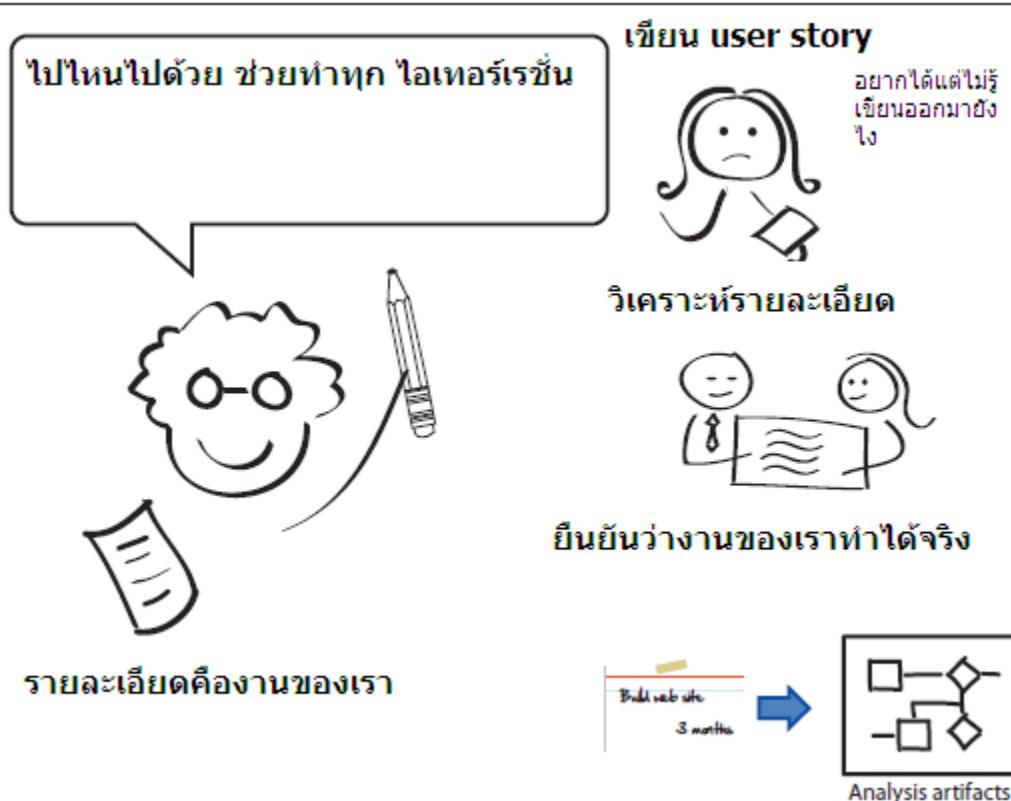
ต่อไปเรามาดูกันสิว่า ทีมพัฒนาในแบบของแอจайлเป็นอย่างไร

## ทีมพัฒนาแบบแอจайл



แอจайлทีมเป็นทีมเมมพล์ที่หนึ่งคนสามารถทำงานได้หลากหลายหน้าที่มีพลังมากมายที่จะเปลี่ยนจากสิ่งที่ลูกค้าต้องการให้ออกมาระบบงานที่สวยงามและใช้งานได้ดี แต่ก็ไม่ใช่ว่าแอจайлทีมจะไม่มีการแบ่งแยกหน้าที่เลยนะครับเราสามารถแบ่งหน้าที่ในทีมได้คร่าวๆดังนี้ analysts, developers, testers, database administrators (DBAs), และคนอื่นๆ อย่างไรก็ตามการที่เราจะเดิน โงงโงงไปปะกอกนี่คือทำงานแบบเดิมๆมานานกว่าต่อจากนี้ไปขอให้ทุกท่าน “จัดระเบียบ หน้าที่ ความรับผิดชอบกันเองนะครับ” รับรองได้เลยว่า “ไม่รอด” ดังนั้นเพื่อให้การเปลี่ยนแปลงทีมทำได้ง่ายขึ้นเราสามารถอธิบายหน้าที่ความรับผิดชอบต่างๆในแอจайлทีมด้วยศัพท์แสงที่คนเข้าใจง่ายๆและคุ้นเคย ดังที่จะกล่าวกันต่อไปหลังจากนี้

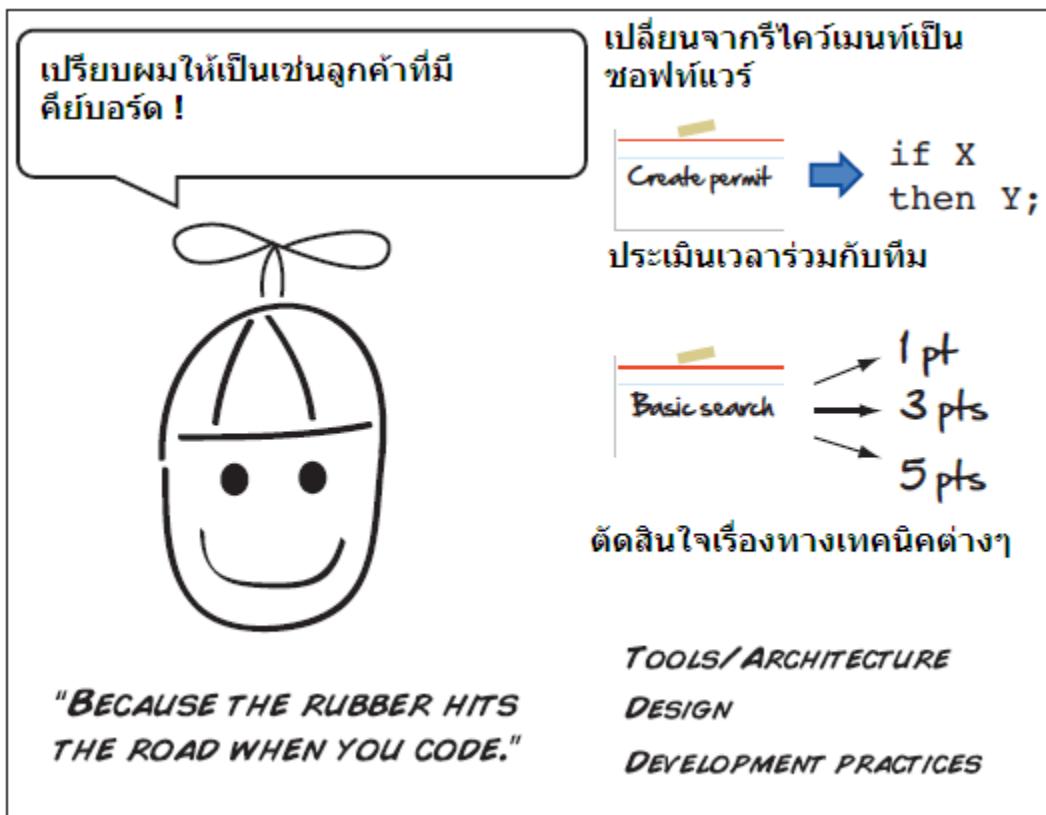
## แอจайл์อนาคตส์



หลังจากที่เราได้ฟีเจอร์ที่ลูกค้าต้องการของมาแล้วแต่สิ่งที่ได้มายังไม่พร้อมที่เราจะลงมือพัฒนามันขยายจนเกินไปเราต้องให้คนๆ หนึ่งไปจัดการหารายละเอียดของแต่ละฟีเจอร์มาให้เรา ก่อน คนที่เราต้องการคือ อนาคตส์

อนาคตส์เป็นคนที่ทำหน้าที่เก็บรายละเอียดต่างๆ ที่ระบบควรจะทำได้โดยวิธีการก็คือ "นั่งทำงานใกล้ชิดลูกค้า" เพราะเราจะได้รู้ จริงๆ ว่าลูกค้าอยากรู้อะไรกันแน่ ดังนั้นเราจะเห็นว่าอนาคตส์ในมุมมองของแข偶จайл์ต้องทำงานเยอะมาก เช่น ช่วยลูกค้าเขียน user story (รายละเอียดอยู่ที่ 6 Gathering User Stories) นอกจากนี้ยังต้องช่วยทำ mock-up, prototype ตลอดจนใช้เครื่องไม้เครื่องมือ กระบวนการท่าทั้งหมดที่มีเพื่อให้การสื่อสาร user story สามารถทำออกมาได้อย่างไม่ผิดเพี้ยน ซึ่งเราจะดูกันในรายละเอียดในส่วนที่ 9.4 Step 1: Analysis and Design: Making the Work Ready

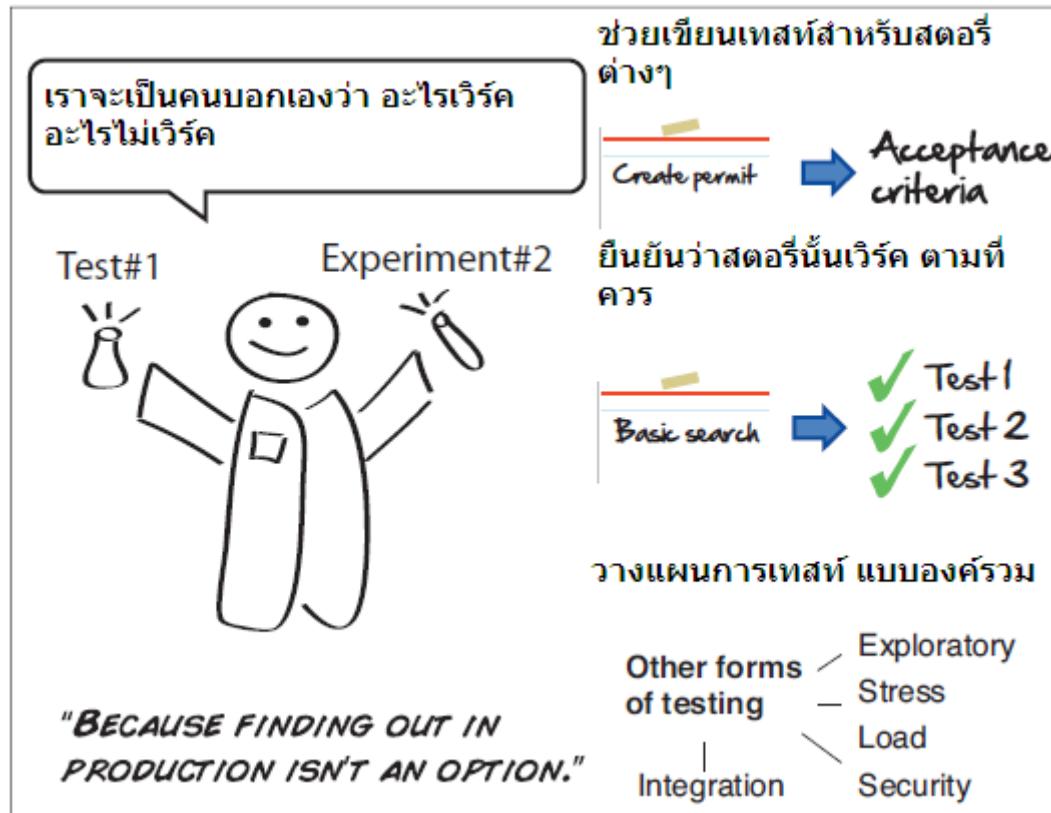
## แอจайлโปรแกรมเมอร์



แอจайлโปรแกรมเมอร์จะทำงานขั้นเทพ เพราะเขาหล่านั้นนิยมคุณภาพ งานดี เนื่องจากโปรแกรมเมอร์จะทำงานใกล้ชิดกับทดสอบที่จะช่วยคิดเรื่องมุมของการทดสอบว่าออกแบบมาทำไหนจะดี ได้ดีออกแบบมีคุณภาพ

\* โปรแกรมเมอร์ของเราจะเรียนทดสอบอย่างบ้าคลั่ง เพื่อให้งานออกแบบดี เพื่อให้มันเป็นตัวสะท้อนการออกแบบ (บทที่ 12 เรื่องยุนิตทดสอบ และบทที่ 14 เรื่องขับเคลื่อนการพัฒนาด้วยการทดสอบ)\* วงจรของการออกแบบและปรับปรุงเกิดขึ้นตลอดเวลา นอกจากรันแล้วตรวจสอบให้ถูกต้องไม่แล้วก็การปรับปรุงเพื่อให้เกิดสิ่งที่ดีกว่าจะเกิดขึ้นอยู่เสมอ (บทที่ 13 การทำรีเฟคเตอร์ริ่ง) บรรดาโปรแกรมเมอร์ต้องมั่นใจเสมอว่าโค้ดของพวกเขายังคงด้อยในสถานะโปรดักชันและสามารถนำไปติดตั้งได้ทันทีที่ได้รับสัญญาณ (บทที่ 15 เราจะพูดถึงเรื่องคอนฟิวอัลส์อินทิเกรชัน) และแน่นอนว่าแม้กระทั่งโปรแกรมเมอร์ของเรายังต้องทำงานใกล้ชิดกับลูกค้าและผู้ร่วมทีมคนอื่น เพราะโปรแกรมเมอร์ก็ต้องแนะนำสิ่งที่ทำออกแบบนั้นคุณตรงกับที่ทุกคนเข้าใจ นี่คือหน้าที่อันแสนเรียบง่ายที่ช่วยส่งเสริมให้เราส่งงานได้ไปมีที่ติ

## แอจайл์ทีสท์เตอร์



แอจайл์ทีสท์เตอร์คือคนที่รู้ว่าจะมีการสร้างอะไรบางอย่างและให้อะไรบางอย่างนั้นมันต้องทำงานยังไงดึงจะถูกต้อง ดังนั้นเราจะเห็นหน้าแอจайл์ทีสท์เตอร์ตั้งแต่เริ่มโปรเจค เพื่อเข้าไปเก็บรายละเอียดว่าแต่ละสตอรี่นั้นผลที่ได้ออกมาตามที่ลูกค้าต้องการเป็นแบบไหนดังนั้นเมื่อเราสร้างมันออกมามาแล้วมันจะ "ถูกต้อง" และอย่างที่เรารู้กันແນ່ງๆแล้วว่าทุกๆอย่างในแอจайл์โนโปรเจคต้องถูกทดสอบบังนั้นเราจะเห็นทีสท์เตอร์อยู่ทุกที่ เราจะเห็นทีสท์เตอร์นั้นหลอกอนเก็บวิเคราะห์เมนท์กับลูกค้า ช่วยลูกค้าหาสิ่งที่ต้องการโดยทั้งหมดจะແຜงอยู่ในแบบของการทำทีสท์

ลองจินตนาการดูว่าถ้าเราเริ่มต้องไปเจอด้วยการซ้ายกันหากำตอบของคำนามธรรมดาสีข้อเหล่านี้เกี่ยวกับตัวเราให้heim

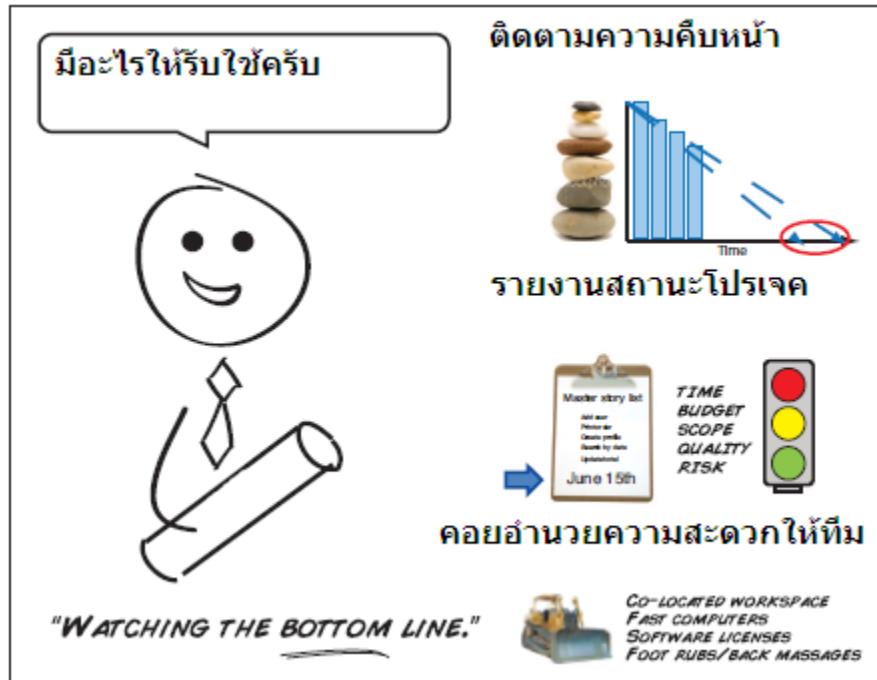
- ผอมทำอะไรได้ดีที่สุด
- ผอมทำงานยังไง
- ผอมมีค่ายังไงกับทีม
- อะไรคือสิ่งที่คุณสามารถคาดหวังได้จากผอม

การตั้งคำถามเหล่านี้เป็นการปลดปล่อยพื้นที่ในการแสดงความคิดเห็นและทำลายกำแพงของปัจจุบัน วิธีการนี้เราเรียกว่า แบบทดสอบของ Drucker เป็นวิธีการที่เรียบง่ายและสามารถมากในการสร้างทีมเพื่อรวมมือเราเข้า一起去ศักยภาพพื้นฐานของแต่ละคนแล้วเราจะได้มามีความไว้วางใจและสิ่งนี้จะนำไปสู่การสื่อสารระหว่าง ผู้ร่วมทีมที่มีประสิทธิภาพสูงซึ่งเป็นสิ่งสำคัญมากสำหรับทีมที่สร้างงานได้อย่างมีประสิทธิภาพ รายละเอียดเพิ่มเติมสำหรับ Drucker สามารถหาได้ที่ url ด้านล่าง

<http://agilewarrior.wordpress.com/2009/11/27/the-drucker-exercise>

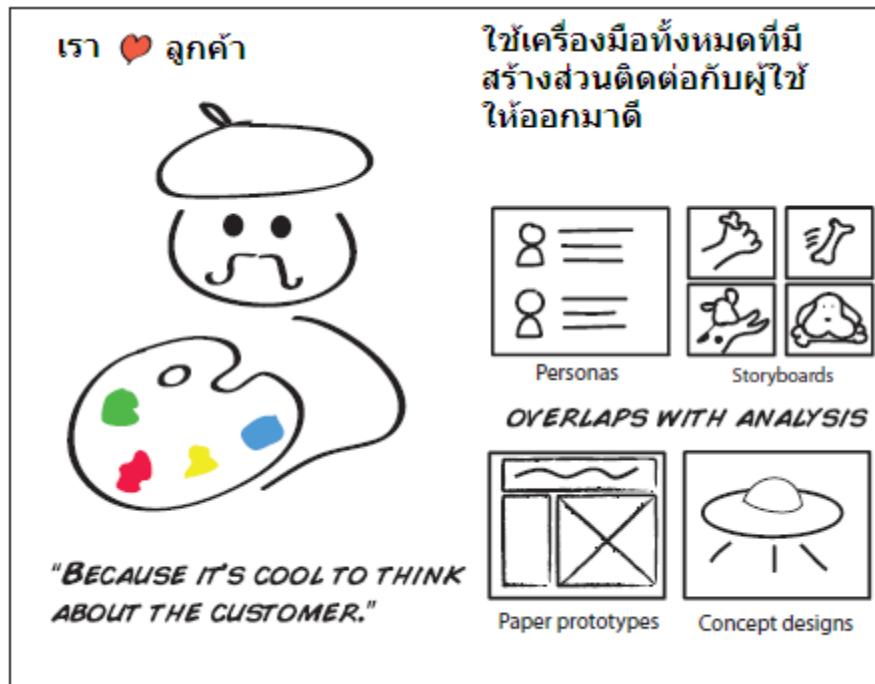
ดังนั้นเราจะเห็นแล้วเจล์ท์เตอร์นั่งทำงานติดกับเดเวลาอีกอย่างหนึ่งจากหน้าที่หลักของ เทสท์เตอร์คือทำ test automation, หาก เขียน หาช่องโหว่หรือจุดผิดพลาดของโปรแกรมจากทุกแง่มุม นอกจากนั้นเขายังต้องมีภาพของแผนการทดสอบในส่วนอื่นๆอยู่อีกไม่เว่าจะเป็น performance test, scalability และสิ่งอื่นๆที่จะทำให้งานที่จะสร้างออกมา มีประสิทธิภาพสูงสุด รายละเอียดเรื่องราวเจล์ท์เตอร์ที่ดึงความสามารถศึกษาเพิ่มเติมได้จากหนังสือ A Practical Guide for Testers and Agile Teams ของ Janet Gregory และ Lisa Crispin

## เจล์ท์พีเอ็ม



เจล์ท์พีเอ็มโดยรวมชาติเหล่าจะรู้ตัวเองว่าความสำเร็จของตัวเองด้วยจากความสำเร็จของทีม ดังนั้นเราจะไม่เห็นแล้วเจล์ท์พีเอ็มทำอะไรสุดขั้ว บ้าบัน ยกตัวอย่างเช่นเอาอยู่ทุกข้อที่ขวางทาง ออกแบบงานของโปรเจคเพื่อให้ได้พบกับคำว่า “เสร็จ” ดังนั้นเราจะเห็นการทำแผนของโปรเจค การแก้ไขแผน การปรับแผนอย่างต่อเนื่องตลอดเวลาเนื่องจาก เรื่องดีนั้นเต้นมักเกิดขึ้นในการทำโปรเจคเสมอ เราจะเห็นว่าเจล์ท์พีเอ็มจะไม่มาสนใจสิ่งที่มีว่าต้องทำอะไร ต้องทำอย่างโน้น ต้องทำอย่างนั้น แต่ในทางกลับ กันแล้วเจล์ท์พีเอ็มจะมีหน้าที่สร้างบรรยากาศการทำงานที่ทีมสามารถทำงานต่อ กันไปได้เองแม้กระทั่ง ตัวพีเอ็มเองต้องหยุดงานหรือหายตัวไป ดังนั้นคุณสมบัติของพีเอ็มที่ดีคือสามารถหายตัวไปทำอย่างอื่นได้สัก ห้ากวันได้โดยที่โปรเจคไม่เพี้ยนออกจากทาง ซึ่งเราจะศึกษารายละเอียดเรื่องราวเจล์ท์พีเอ็มในบทที่ 8 เรื่องการวางแผนแบบเจล์ท์ การรับมือกับโลกอันโหดร้าย

## แอจайл์ UX ดีไซน์เนอร์



แอจайл์ยูเอ็กซ์คือคนที่มีสมารธิมุ่งมั่นอยู่กับการ ออกแบบ สร้าง ส่วนติดต่อกับผู้ใช้ให้ตรงตาม ประสบการณ์ของผู้ใช้แต่ละคน เพื่อให้ผู้ใช้สึกได้รู้สึกใช้งานสนั่น กับแอพพลิเคชันของเรา และเมื่อนำเข้าแนวคิดเรื่องการออกแบบ ยูไอ ที่มีการแสดงร่างกับแอจайл์แล้วเราจะได้มาซึ่งซอฟต์แวร์ที่ มีคุณภาพครอบด้าน ส่วนการทำงานของแอจайл์ยูเอ็กซ์ก็จะมาแนะนำเม็ดคือสามารถแก้ไขแบบได้เสมอโดยสามารถทำไปพร้อมกับการเขียนโค้ดได้ (แทนที่จะซิงออกแบบก่อนที่ทุกคนจะเริ่มทำงาน) ดังนั้นในกรณีที่แอจайл์ที่มีของท่านมีคุณที่มีความสามารถเรื่อง usability นั้นถือเป็นโชค เพราะ เข้าคนนั้นจะช่วยทีมได้เป็นอย่างมากในเรื่องของการสร้างงานที่เป็นมิตรกับคนใช้

## คนอื่นๆ

ที่กล่าวมาทั้งหมดคือบทบาทที่สำคัญสำคัญในแอจайл์ทีมแต่อย่างไรก็ตามยังมีบทบาทอื่นๆที่ไม่ได้กล่าวถึงเช่น database administrators (DBAs), system administrators (SAs), technical writers, trainers, business improvement, infrastructure, และ networking บทบาทเหล่านี้ก็สำคัญมากเช่นกัน สำหรับแอจайл์ทีม

สำหรับแนวคิดแบบ Scrum นั้นจะมีบทบาทที่สำคัญอีกอย่างที่เรียกว่า scrum master หรือเราสามารถเปลี่ยน เทียบได้กับการ เอาแอจайл์ให้ชุมารวมร่วงกับร็อกสตาร์ในวงเจอร์ ซึ่งเราจะเห็นว่าแอจайл์ได้ชัดเจน ประโยชน์กับทีมที่ยังไม่เคยได้แอจайл์มาก่อนและต้องการคนคอยแนะนำแนวทางที่ถูกต้องให้ ไม่ให้เข้าใจ ผิดเกี่ยวกับแอจайл์ได้สำหรับหนังสือที่ได้เกี่ยวกับเรื่องนี้คือ Agile Coaching แต่สำหรับทีมที่มีพลังแอจайл์ เข้มแข็งแล้วต้องการให้หนังสือที่มีคุณภาพมากที่สุด เช่น Agile

แต่อย่างไรก็ตามเมื่อเราต้องการบทบาทนี้ในทีมเราต้องมั่นใจว่าทุกคนในทีมเข้าใจว่าสำหรับแอจайл์ทีมมันการสัมบทบาทหลายบทบาทเป็นเรื่องปกติ ยกตัวอย่างเช่นเราอาจจะเห็นเดเวลาอปเปอร์เดินไปคุยกับลูกค้าหรืออาจารย์จะเห็นทดสอบเว็บ หรือว่าเดเวลาอปเปอร์จะต้องทำ automate test เยอะๆและเนื่องจากในทีมอาจจะไม่มีคนทำหน้าที่ UX จริงๆแต่ก็ไม่ได้หมายความว่าเรื่องนี้จะถูกละเลยไป เรื่องนี้ยังถูกทำอยู่แต่เป็นครือขึ้นที่ทำหน้าที่นี้อยู่ เดียวเราลองมาดูกันว่าจะเกิดขึ้นเมื่อเราต้องการหาคนมาทำงานในทีม

### Tips for Forming Your Agile Team เคล็ดการรวมแอจайл์

การทำงานกับทีมที่มีประสิทธิภาพสุดยอดอย่างแอจайл์ทีมเป็นเรื่องที่ทุกคนไฟฝันแต่อย่างไรก็ตามการคัดเลือกคนเข้ามาทำงานในทีมเราต้อง มีวิธีการคัดเลือกคนนิดหน่อย

#### Look for Generalists

คนที่มีความรู้หลากหลายสาขา และแอจайл์ทีมต้องการคนแบบนี้เนื่องจากสามารถตอบสนองต่อความต้องรับผิดชอบ การทำงานของตัวเองแบบหน้าถึงหลังซึ่งสำหรับโปรแกรมเมอร์จะหมายความว่าขาดคนหนึ่งจะต้องรับผิดชอบทำงานตั้งแต่ส่วนติดต่อฐานข้อมูลจนถึงหน้าเว็บ นอกจากนี้คนประเภทรอบรู้แบบนี้จะรู้สึกไม่เปลกถ้าต้องทำหน้าที่รับผิดชอบหลายบทบาทในทีม เช่นวันนี้ต้อง พิสูจน์เว็บไซต์ พรุ่งนี้ไปทดสอบ

#### People Who Are Comfortable with Ambiguity

อย่าเพ้อฝันคำว่า “พร้อม” แล้วค่อยลงมือทำงาน เพราะสิ่งนั้นไม่เคยมีจริง อย่ารอ requirement ต้องออกไปดำเนินงาน มันเอง แทนไม่เคยนิ่งมันเปลี่ยนตลอด เราต้องพร้อมรับมือกับความเปลี่ยนแปลง จึงมองหาคนที่ไม่หวั่นไหวต่อ change request มาหาก

#### Team Players Who Can Check Their Egos at the Door

จึงมองหาคนที่ปล่อยวางไม่ยึดติด เมื่อมาก็งวนประดุจที่ทำงานความเป็นตัวตนทั้งหมดต้องถูกทิ้งไว้ที่นั่น จึงมองหาคนที่พร้อมจะเรียนรู้อยู่เสมอ ไม่กลัวที่จะแบ่งปัน และพร้อมที่จะก้าวหน้าไปกับทีม

**ศิษย์:** ท่านอาจารย์เข้ารู้สึกสับสน ในเมื่อไม่มีการแบ่งหน้าที่ชัดเจนสำหรับการทำแอจайл์ไปเจอกัน จะทำงานให้เสร็จกันได้อย่างไร

**อาจารย์:** ทุกอย่างต้องถูกทำให้เสร็จศิษย์เอ่ย ทีมจะเป็นคนทำมัน

**ศิษย์:** ครับท่านอาจารย์ แต่ข้าน้อยยังสงสัยว่าถ้าไม่มีการแยกหน้าที่แบ่งคนมาทำงานแบบงานโครงการ นั้นเรื่องถ้าไม่มีการสร้างทีมเทสท์ขึ้นมาโดยเฉพาะเราจะแน่ใจได้อย่างไรว่า งานของเราถูกทดสอบมาแล้วเป็นอย่างดี

**อาจารย์:** เทสท์ก็เป็นสิ่งที่ต้องทำเช่นกัน และแน่นอนสำหรับการทำเทสท์นั้นทีมก็ต้องทำอีกเช่นกัน การทำงานทำน้อยเท่าไหร่ทีมต้องตัดสินใจ

**STUDENT:** What if no one wants to test? What if everyone just wants to sit around and write code?

**ศิษย์:** อ่าวท่านอาจารย์จะเกิดอะไรขึ้นถ้าทุกคนไม่อยากเทสท์เลย ทุกคนต้องการเป็นเพphenร์เขียนโค้ด อย่างเดียวครับ ทำเทสท์

**อาจารย์:** ถ้าเข่นนั้นมันก็ต้องตั้งแต่หานามาร่วมทีมแล้ว เพื่อหลักเลี่ยงปัญหานี้เจ้าต้องเสาะหาคนที่ยินยอมพร้อมทำเทสท์เข้ามาเป็นส่วนหนึ่งของทีมอันทรงคุณค่าของเจ้า

គិម្យៈ ខុបគុណទាំងអាជរយៈ ត្រូវឱ្យកិច្ចការរៀបចំនៅក្នុងពេលវេលាដើម្បី ដើម្បីក្រោង

## What's Next?

You now see how roles blur on agile projects, why we would ideally like our teams to be co-located, and how, when finding people for your team, you are going to want generalists and people who are cool with dealing with ambiguity. You are now ready for what is perhaps one of the most important steps in kick-starting your agile project (and an area that most agile methods are completely silent on)—agile project inception.

Turn the page, to Part II of the book, and find out how to set your project up for success from the start and make sure you have the right people.

# ตอนที่ 3 ชวนดูบีนรถเมล์



หมายฯ โปรเจคถูกฟ่าตายก่อนจะได้เริ่มต้นด้วยช้า ส่วนใหญ่ก็เป็นเพราะเหตุผลต่อไปนี้:

- คนในโปรเจคไม่ได้ตั้งคำถามที่เหมาะสม
- คนในโปรเจคไม่กล้าพูดที่จะตั้งคำถามที่ตอบยากๆ

ในส่วนนี้ เราจะมาชี้จุดกับเครื่องมือตั้งความคาดหวังประสพธิกภาพสูง ชื่อ ชั้นแห่งการเริ่มต้น (Inception Deck) — สิบคำถาม ที่ ถ้าไม่ถามก่อนเริ่มโปรเจคซอฟต์แวร์ได้ก็บ้าแล้ว ด้วยการใช้ประโยชน์จากชั้นแห่งการเริ่มต้น เราจะมั่นใจได้ว่า เราได้คนที่ใช่ มาร่วมการเดินทาง และเรากำลังเดินทางไปในทิศทางที่ถูกต้อง ก่อนที่เราจะเริ่มเขียนโค้ดบรรทัดแรก

## 3.1 อะไรเป็นสิ่งที่ผ้าโปรเจคส่วนใหญ่

ณ จุดเริ่มต้นของโปรเจคใหม่ คนในทีมมักจะมีแนวคิดที่ต่างกันโดยสิ้นเชิง ว่าความสำเร็จน่าตาเป็นอย่างไร



นี่เป็นสิ่งที่ร้ายแรง ที่ทำให้โปรเจคของคุณคาดถึงตายได้ เพราะถึงแม้ว่าเจ้าทุกคนจะใช้คำศัพท์และประยุนแบบเดียวกันอย่างเดียวกัน แต่ความคิดของแต่ละคนไม่ได้คิดเหมือนกันเลยแม้แต่นิดเดียว ก็เมื่อตอนที่เราเริ่มนำเสนอองานออกไปแล้วเท่านั้น แหล่ง

และปัญหามันก็ไม่ได้อยู่ที่ว่า เราไม่ได้คิดเห็นไปในทางเดียวกันตอนที่เราร่วมต้น (คิดเห็นไม่ตรงกัน มันเป็นเรื่องธรรมชาติ) แต่มันอยู่ที่ว่าเราเริ่มโปรเจคก่อนที่ทุกคนจะเข้าร่วมมาพร้อมเพียงกัน

การตั้งสมมติฐานว่า ทุกคนมีความคิดเห็นตรงกัน ทั้งที่จริงไม่มี นั่นแหล่ะ คือสิ่งที่妨礙ไปร่องส่วนใหญ่ สิ่งที่เราต้องการก็คือ อะไรมากอย่างที่จะทำให้เกิดสิ่งต่อไปนี้:

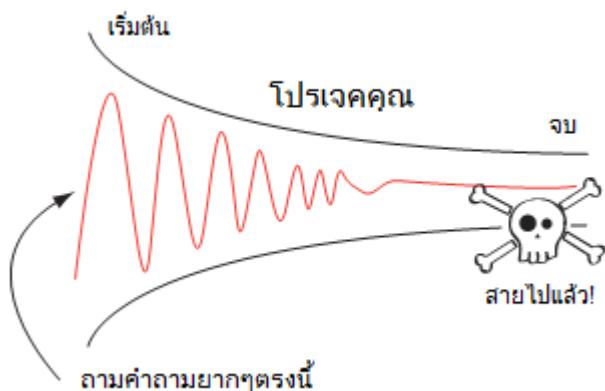
- การสื่อสารเป้าหมาย วิสัยทัศน์ และสภาพแวดล้อม ของโปรเจค ไปยังผู้ร่วมทีม เพื่อที่ผู้ร่วมทีมจะได้ตัดสินใจอย่างช้าๆ ฉลาด เมื่อทำงานจริง
- ให้ข้อมูลกับผู้ที่มีส่วนเกี่ยวข้องตามที่พากษาต้องการ เพื่อเขาไปใช้ในการตัดสินใจว่าจะไปหรือไม่ไปต่อกับโปรเจคนี้ดี



แล้ววิธีเดียวที่จะทำให้เกิดสิ่งเหล่านี้ได้ก็คือ การถ่ายทอดตามยากๆนั่นเอง

### 3.2 ถ่ายทอดตามยากๆ

ตอนไปทำงานที่แคนจิง โจนได้มีโอกาสได้ร่วมทางไปกับพนักงานขายเมื่ออาชีพระดับต้นๆ ของบริษัท ThoughtWorks — สุภาพบุรุษที่มีชื่อว่า Keith Dodds อย่างนึงที่ Keith สอนผมก็คือ ความสำคัญของการถ่ายทอดตามยากๆ ในช่วงที่เริ่มเข้าไปมีส่วนร่วม หรือ ในช่วงเริ่มต้นของการขาย



คุณเห็นมั้ยว่า ในตอนเริ่มต้นของการมีส่วนร่วมใดๆ หรือโปรเจคใดๆ เราเมื่อทางในการถ่ายทอดตามยากๆ ที่เปิดกว้างมาก โดยที่เราจะสูญเสียน้อยมาก ดังนั้นเราสามารถถ่ายทอดตามก้าวๆ อย่างเช่นถ่ายทอดเหล่านี้:

- ทีมคุณมีประสบการณ์การทำงานมากเท่าไร?
- คุณเคยทำอะไรคล้ายๆแบบนี้มา ก่อนรึเปล่า?

- คุณมีเงินอยู่เท่าไร?
- ใครเป็นคนตัดสินใจสูงสุดในโปรเจคนี้?
- คุณมองเห็นความท้าทายของการมีนักวิเคราะห์ซอฟต์แวร์สองคนกับผู้พัฒนาซอฟต์แวร์สามสิบคนรึเปล่า?
- มีโปรเจคอื่นๆ อันไหนที่คุณเคยทำมา ที่คุณสามารถนำเอาทีมของผู้พัฒนาซอฟต์แวร์ตัวจ้ำย ที่มีประสบการณ์การทำโปรแกรมเชิงวัตถุ (object-oriented) เพียงเล็กน้อย หรือไม่มีประสบการณ์เลย ไปใช้ในโค้ดเมนเฟรมเวิร์กใหม่เป็น Ruby on Rails โดยใช้แอ็อกาจีล์ แล้วประสบความสำเร็จได้บ้าง

คุณควรต้องการที่จะใช้วิธีการในแบบเดียวกันนี้เพื่อเริ่มต้นโปรเจคแอ็อกาจีล์ของคุณ คุณควรต้องการที่จะถามคำถานที่มันน่ากลัว ตั้งแต่เนื่องๆ และเครื่องมือนึงที่จะช่วยคุณได้ก็คือ ขั้นแห่งการเริ่มต้น

### 3.3 เข้าสู่ขั้นแห่งการเริ่มต้น



ขั้นแห่งการเริ่มต้น เป็นเหมือนกับไฟฉายส่องทาง ที่ช่วยกำจัดหมอกบังตาและสิ่งที่ไม่สามารถอธิบายได้ในโปรเจคแอ็อกาจีล์ของคุณ มันคือการรวมถามความยากๆ สิบถาน และแบบฝึกหัด ที่ถ้าคุณไม่ทำแล้วไม่ถานก่อนจะเริ่มโปรเจค คุณก็ไม่ไหวแล้ว เจ้า ให้สิ่งนี้ปอยมากในบริษัท ThoughtWorks เพื่อจะครอบคลุมในมุ่งของการเริ่มต้นโปรเจค ซึ่งวิถีทางแบบแอ็อกาจีล์อย่าง Extreme

Programming หรือ Scrum ไม่ได้พูดถึงไว้ — ในอนุญาตทำโปรเจค เราอู้กันดีว่า การจะไปปั้งวิเคราะห์กันอย่างหนักเป็นเวลาหากเดือน หรือการทำแบบฝึกหัดรวมความต้องการของลูกค้า ไม่ใช่แนวทางแนวๆ แต่เราเก็บไม่วิธีแบบสบายๆ ที่จะเขามาทดสอบให้ด้วยเหตุนี้เอง Robin Gibbons จึงได้สร้างต้นฉบับ ขั้นแห่งการเริ่มต้น ขึ้นมา วิธีง่ายๆ เร็วๆ ที่จะกลั่นกรองโปรเจคจนถึงแก่น และสื่อสารความเข้าใจร่วมกันนั้นไปยังทีมและมวลชนที่เกี่ยวข้องทั้งหมด

### 3.4 แล้วมันทำงานยังไง

แนวคิดที่อยู่เบื้องหลัง ขั้นแห่งการเริ่มต้น ก็คือ ถ้าเราสามารถนำคนที่เหมาะสมเข้ามาร่วมกันและสามารถคำนวณที่เหมาะสมได้ เราจะสามารถตั้งความคาดหวังของโปรเจคในภาพรวมได้อย่างน่าอัศจรรย์ โดยการที่เราให้ทีมได้ทำแบบฝึกหัดต่างๆ และเก็บสะสมผลลัพธ์ที่ได้มาทำเป็นสไลด์ (ส่วนใหญ่จะเป็น PowerPoint) เรายังสามารถเห็นภาพรวมว่า โปรเจคนี้คืออะไร ไม่ใช่อะไร และจะต้องใช้อะไรบ้างที่จะทำให้โปรเจคสำเร็จลุล่วงไปได้

คนที่เหมาะสมสำหรับ ขั้นแห่งการเริ่มต้น ก็คือ ครอกร์ตามที่มีส่วนร่วมกับโปรเจคโดยตรง ซึ่งนี่หมายถึง ลูกค้า ผู้มีส่วนได้ส่วนเสีย คนในทีม คนพัฒนา คนทดสอบ คนวิเคราะห์ — ครอกร์ตามที่จะทำให้การทำโปรเจคนี้ดำเนินไปอย่างมีประสิทธิภาพ สิ่งสำคัญอย่างนึงก็คือ การให้ผู้ที่มีส่วนได้ส่วนเสีย มา มีส่วนร่วม เพราะ ขั้นแห่งการเริ่มต้น ไม่ได้เป็นแค่เครื่องมือแค่สำหรับคนในทีม แต่กับคนเหล่านั้นด้วย เพื่อให้เค้าได้สามารถตัดสินใจว่า จะไปหรือไม่ไปต่อ กับโปรเจคนี้ได้

โดยทั่วไป ขั้นแห่งการเริ่มต้น จะใช้เวลาประมาณสองสามวัน หรือ ประมาณสองอาทิตย์ ในการทำ ซึ่งก็จะใช้ได้กับโปรเจคที่มีระยะเวลาประมาณหนึ่งเดือน และควรจะมีการกลับมาทำใหม่ ถ้ามีการเปลี่ยนแปลงเยอะๆ ของจิตวิญญาณและแนวทางของโปรเจค นั่นก็เพราะ ขั้นแห่งการเริ่มต้น เป็นวัตถุที่จะมีวิวัฒนาการ เปลี่ยนแปลงได้เรื่อยๆ ไม่ใช่อะไรที่เราทำที่ได้ยกเว้นก็จะกับกัน ตอนที่ทำเสร็จแล้ว คนในทีมมักจะเอาไปประวัติบันทึกห้อง เพื่อเป็นการย้ำเตือนจิตใจ ว่าตอนนี้เราทำสิ่งที่เราต้องทำอยู่ และทำไปทำไม่แล้วน่อน คำนวณและแบบฝึกหัดที่เราจะนำเสนอในที่นี้ แค่เริ่มต้นให้เท่านั้น คุณจะต้องคิดต่อไปอีก ถึงคำนวณและแบบฝึกหัด และสิ่งที่คุณอยากรู้จะทำให้กระฉ่าง ก่อนที่คุณจะเริ่มโปรเจค ดังนั้น ขอให้คุณใช้สิ่งนี้เป็นเสมือนจุดเริ่มต้น และอย่าไปตามมันอย่างเมามาย หรือกลัวที่จะเปลี่ยนแปลงมันเพื่อให้เหมาะสมกับงานของคุณ

### 3.5 ขั้นแห่งการเริ่มต้นในเปลือกน้ำ

ต่อไปนี้จะเป็น การอธิบายคร่าวๆ ถึงคำนวณและแบบฝึกหัดใน ขั้นแห่งการเริ่มต้น

#### 1. ถามว่าทำไม่เราถึงมากอยู่ตรงนี้

- ขันนี้เป็นสิ่งเดือนใจง่ายๆ ว่าเรามากอยู่ตรงนี้ทำไม ลูกค้าของเรานี่เป็นใคร และทำไม่เราถึงตัดสินใจที่จะทำโปรเจคนี้ในตอนแรก

#### 2. สร้างคำนวณแบบรวมรัด

- ถ้าเรามีสามสิบวิ และใช้ประโยชน์ได้แค่สองประโยชน์ เราจะอธิบายถึงโปรเจคเราไว้ยังไง

#### 3. ออกแบบกล่องบรรจุผลิตภัณฑ์

- ถ้าเราทำสิ่งใดสิ่งหนึ่ง สื่อแมกกาซีนเล่มนึง แล้วเห็นโฆษณาของผลิตภัณฑ์หรือบริการของเรา เราอยากรู้จะเห็นมันเขียนว่ายังไง หรือที่สำคัญกว่านั้น เราจะซื้อมันมั้ย?

#### 4. สร้างรายการ ไม่ทำ

- สิ่งที่เราอยากรู้ทำในโปรเจคอาจจะซัดเจน แต่สิ่งที่ไม่ควรทำในโปรเจคควรที่จะยิ่งซัดเจนยิ่งกว่า

## 5. พบປະເພື່ອນບ້ານ

- ທຸມໝານຂອງໂປຣເຈກເຮົາ ມັກຈະໃຫຍ່ກວ່າທີ່ໄຮຄິດເສມອ ທຳໄມ່ໄມ່ຈວນເຄົ້າມານັ່ງດີມກາແພດ້ວຍກັນ ແລະແນະນຳ ຕັວເອງຂະໜໍ່ອຍລະ

## 6. ແສດງແນວທາງຂອງສິ່ງທີ່ຈະທຳ

- ມາວັດແບບຮ່າງໂຄຮງສ້າງທາງເຖິງນິກິນແບບກວ້າງາຂອງສິ່ງທີ່ຈະທຳກັນເຕອະ ແລະຕຽບສອບໃຫ້ແນ່ໃຈວ່າເຈົ້າທຸກ ດັນຄິດຕຽບກັນຈົງຈາ

## 7. ຄາມຕັວເອງ ວ່າຂະໜໍ່ໄວ້ທີ່ທຳໃຫ້ເຮັນອນໄມ່ໜັບ

- ບາງສິ່ງທີ່ເກີດໃນໂປຣເຈຂອງເຈົ້າ ຂ່າງນ່າກລ້ວເລີຍຈົງຈາ ແຕ່ກາຣທີ່ເຈົ້າໄດ້ພູດລຶ່ມມັນ ກັບສິ່ງທີ່ເຈົ້າຈະທຳເພື່ອ ປັ້ນກັນໄມ້ໃຫ້ເກີດສິ່ງເໝລ່ານັ້ນ ຈະທຳໃຫ້ມັນນ່າກລ້ວນ້ອຍລົງ

## 8. ກະເກມທົ່ວໜາດ

- ໂປຣເຈນີ້ນີ້ເປັນໂປຣເຈ ສາມເດືອນ ຮົກເດືອນ ອົງກໍາເດືອນ ກັນນະ

## 9. ບາກໃຫ້ຊັດເຈນວ່າຂະໜໍ່ໄວ້ທີ່ຢູ່ອມເສີຍໄດ້ຂະໜໍ່ໄວ້ທີ່ເສີຍໄມ້ໄດ້

- ຖຸກໂປຣເຈມັກຈະມີສິ່ງທີ່ຄານໄວ້ອູ່ ເຊັ່ນ ເວລາ ຂອບເຂດ ກົບປະມານ ແລະຄຸນກາພ ອະໄວຄື່ອສິ່ງທີ່ສຳຄັນມາກ ທີ່ສຸດແລະນ້ອຍທີ່ສຸດໃນໂປຣເຈນີ້

## 10. ແສດງໃຫ້ເຫັນວ່າຈະຕ້ອງໃຫ້ຂະໜໍ່ໄວ້ບ້າງ

- ໂປຣເຈນີ້ຈະໃຫ້ເລານານເທົ່າໄວ ຈະຕ້ອງໃໝ່ບເທົ່າໄວ ແລະຕ້ອງໃຫ້ທົມແບບໄຫນເພື່ອໃຫ້ກັນເສົ້າຈຳໄດ້

ເຈົ້າຈະພູດລຶ່ມ ຊັ້ນແໜ່ງກາຣເວີມຕົ້ນ ໂດຍແປ່ງອອກເປັນສອງສ່ວນ ໃນຕອນທີ່ 4 ພົບ ມອງກາພວມ ໃນໜ້າລັດໄປ ເຈົ້າຈະພູດລຶ່ມຄໍາດາມວ່າ ທຳໄມ່ ເພື່ອໃຫ້ເຂົ້າສົ່ງເຫດຜລທີ່ເຈົ້າທຳໂປຣເຈ ສ່ວນຕອນທີ່ 5 ພົບ ທຳໃຫ້ມັນເປັນຈົງຈາ ໃນໜ້າ 72 ເຈົ້າຈະພູດລຶ່ມ ຄໍາດາມວ່າ ທຳຍ່າງໄວ ມາເຣີມຄາມວ່າ ທຳໄມ່ ກັນ

# ตอนที่ 4 มองการรวม



การทำซอฟต์แวร์เป็นกิจกรรมที่มีเอกลักษณ์ ที่รวมเอาทั้ง การออกแบบ การสร้าง ศิลปะ และวิทยาศาสตร์ รวมเข้าอยู่ด้วยกัน ในทุกวัน คนในทีมจะต้องเจอกับการตัดสินใจเป็นพันๆครั้ง และการได้อ่านต้องเสียอีกอย่าง และถ้าไม่เข้าใจในสภาพแวดล้อม และภาพรวมแล้ว ก็คงจะเป็นไปไม่ได้ ที่จะตัดสินใจเลือกในสิ่งที่ถูกต้องด้วยข้อมูลที่มีอยู่ และทำให้สิ่งที่ได้กับสิ่งที่เสียมีสมดุลที่ดีได้ ในครึ่งแรกของ ชั้นเร่งการเริ่มต้น เราจะมาทำให้เราเข้าใจกันอย่างชัดเจนว่า ทำไม เรายังได้มาทำโปรเจค โดยการทำความคิดเห็น ต่อไปนี้:

- ทำไม่เราถึงมาอยู่ตรงนี้
- อะไรคือคำนำเสนอบรรบัดของเรา
- โฆษณาของผลิตภัณฑ์เรา ควรจะออกแบบหน้ายังไง
- เราจะต้องไม่ทำอะไรบ้าง
- ใครเป็นเพื่อนบ้านของเราบ้าง

เมื่อคนอ่านตอนนี้จบ คุณและคนในทีม จะมีความเข้าใจที่ชัดเจนว่าเป้าหมายของโครงการคืออะไร ทำไม่คุณกำลังจะทำมัน และสามารถที่จะสื่อสารให้คนอื่นฟังได้เป็นจากอย่างรวดเร็ว แต่ตอนนี้ เราจะเริ่มจากการถามผู้สนับสนุนของเรา ว่าทำไม่เราถึงนาอยู่ตรงนี้

## 4.1 ถาม: ทำไมเราถึงมาอยู่ตรงนี้

### ทำไมเราถึงมาอยู่ตรงนี้?

- เพื่อติดตามและเฝ้าสังเกตกิจกรรมการทำงานในไซต์ก่อสร้าง



ก่อนที่ทีมโครงการเดาจะประสบความสำเร็จได้ เค้าจะต้องเข้าใจว่า ทำไม เค้าถึงกำลังจะทำในสิ่งที่เค้ากำลังจะทำ เมื่อไรที่เค้าเข้าใจว่า ทำไม แล้ว เค้าจะสามารถทำในสิ่งต่อไปนี้ได้:

- ตัดสินใจได้ดีขึ้น จากข้อมูลที่ได้มากขึ้น
- สร้างความสมดุลกับสิ่งที่ขัดแย้งกัน และสิ่งที่จะได้มาและเสียไป ได้ดีขึ้น
- สามารถคิดค้นวิธีการและแนวทางที่สร้างสรรค์ เพราะเค้าได้รับอำนาจในการคิดและตัดสินใจด้วยตัวเอง

ทั้งหมดทั้งปวงเกี่ยวกับ การเข้าใจ เจตนาของผู้บังคับบัญชา และ การ ไปให้เห็นกับตา ของตัวเอง

โตโยต้า : เทพแห่งการไปดูให้เห็นกับตา

ในหนังสือระดับเทพ The Toyota Way (วิถีแห่งโตโยต้า) [Lik04] Jeffrey Liker

ได้เล่าเรื่องของหัวหน้าวิศวกรที่ได้รับมอบหมายให้ design รถโตโยต้า Sienna รุ่นปี 2004 ในมี ซึ่งเด็กต้องการที่จะพัฒนา design นี้สำหรับชาวทวีปอเมริกาเหนือ เพื่อให้เข้าถึงชาวทวีปอเมริกาเหนือ ให้ชีวิต ทำงาน และทำกิจกรรมด้วยรถยังไง หัวหน้าวิศวกรและทีมของเด็กๆ ได้ขับรถโดยตัวเอง Sienna ผ่านมัณฑุกฯ State ในประเทศสหรัฐอเมริกา ประเทศแคนาดา และประเทศเม็กซิโกสิ่งที่เด็กนับมีดังนี้:

- คนขับรถในทวีปอเมริกาเหนือ รับประทานอาหารในรถมากกว่าคนขับรถในประเทศญี่ปุ่น (โดยที่ระยะทางในการขับนั้น ส่วนมากจะสั้นกว่า) ด้วยเหตุนี้เอง คุณก็เลยจะพบถ้าดอยู่ตรงกลางระหว่างที่นั่ง และที่วางแก้วจำนวน 14 อัน เป็นมาตรฐาน ในรถโดยตัวเอง Sienna ทุกคัน
- ถนนในแคนาดาโค้งสูงขึ้นมาต้องลงมาหากว่าถนนในอเมริกา ดังนั้น การควบคุมการ刹ฟ์ฟในขณะขับรถนั้นสำคัญมาก
- ลมที่พัดในแนวราบกับพื้นใน Ontario มีความรุนแรงมาก ทำให้ การควบคุมรถให้มั่นคงเมื่อมีลมมาทางด้านข้าง กลายเป็นปัญหาที่ยากมาก ซึ่งถ้าคุณไปขับรถไปยังที่เดียวกันที่มีลมแรงพัดขนาดกับพื้น คุณก็จะเห็นได้ว่า Sienna รุ่นใหม่จะมั่นคงขึ้นมากและควบคุมได้ง่ายขึ้น

ถึงแม้ว่า คุณหัวหน้าวิศวกรจะสามารถอ่านถึงปัญหาเหล่านี้ได้จากการงานการตลาด เด็กคงจะไม่เข้าใจลึกซึ้งและมีความเข้าใจได้มากเท่ากับการที่เด็กได้ไปและเห็นกับตาด้วยตัวเอง

### ไปให้เห็นด้วยตาของคุณเอง

การที่เราเข้าใจว่าทำไม่เรามาอยู่ตรงนี้ในหัวสมองเรามันก็เรื่องนึง แต่การที่เราเข้าใจว่าทำไม่เราถึงมากอยู่ตรงนี้ นั้นเป็นอีกเรื่องนึงที่แตกต่างกันโดยสิ้นเชิง เพื่อที่จะเข้าไปข้างในหัวของลูกค้าและเข้าใจอย่างถ่องแท้ว่าเด็กต้องการอะไร คุณจะต้องไปดูให้เห็นกับตาด้วยตัวเอง

การไปดูให้เห็นกับตาเป็นการให้คนในทีมของคุณได้ขับกันออกมานอกไปที่สิ่งที่เด็กจะทำจะถูกนำไปใช้ ยกตัวอย่าง เช่น ถ้าคุณกำลังจะสร้างระบบคุณภาพงาน สำหรับบริษัทก่อสร้าง ซึ่งจะถูกนำไปใช้ที่ไซต์งาน ซึ่งเป็นเหมือนคุณต้องไปที่ไซต์นั้น ไปนั่งคุยกับพนักงานดูแลความปลอดภัย ไปดูรถ trailers ไปสังเกตสภาพการทำงานในสถานที่แบบ ขับเครื่องเน็ตไม่ค่อยเสถียร และสถานที่ที่ลูกค้าของคุณต้องทำงานอยู่ในนั้น ใช้เวลาที่ไซต์ขักรันนิ่ง และทำงานร่วมกับคนที่จะใช้ระบบของคุณทุกวัน “เข้าไปมีส่วนร่วม สามคำถาม และกล้ายเป็นลูกค้าของคุณ”

### ค้นหาความตั้งใจของคนสั่งการ

ความตั้งใจของผู้สั่งการมักมาเป็นประโยชน์หรือคำพูดสั้นๆ ที่สรุปเป้าหมายหรือวัตถุประสงค์ของโครงการหรือ mission ของคุณ ประโยชน์นี้ หรือ แสดงถึงน้ำทางนี้เอง ที่คุณจะใช้เป็นที่พึ่งพิงได้ ในช่วงเวลาคับขันของการต่อสู้ ที่จะช่วยคุณตัดสินใจได้ว่า จะยก

พวກไปปฏิศัต្តูก่อน หรือ จะหยุดรออยู่กับที่ ในหนังสือ Made to stick [HH07] คุณ Chip and Dan Heath เด่าถึงเรื่องที่ สายการบิน Southwest Airlines อกหักเดียงกันว่าจะเพิ่ม ชีชาร์สลดได้ ก็เข้าไปเป็นอาหารในเครื่องเครื่องนึงหรือไม่แต่เมื่อถูกถามว่า การทำอย่างนี้จะช่วยลด cost ของราคาค่าตัวหรือไม่ (ซึ่งเป็นความตั้งใจของ CEO Herbs Kelleher ที่เป็นคนสั่งการ) ก็ทำให้เห็นได้ชัดเจนว่า การเพิ่มชีชาร์สลดได้ นั้นไม่ make sense ความตั้งใจของคนสั่งการ สำหรับโปรเจกของคุณ ไม่จำเป็นต้องเป็นอะไรที่ยิ่งใหญ่หรือสร้างขึ้นโดยกำลังใจ มันสามารถที่จะเป็นอะไรจ่ายๆ และไฟฟ้าไปยังโปรเจกของคุณ จุดสำคัญสำหรับแบบฝึกหัดนี้ก็คือ การให้ทุกคนได้มารับเข้าคุยกันถึง อะไรที่เค้าคิดว่าเป็นเหตุผลที่ทำให้เค้ามาอยู่ที่นี่ และเค้าสิงเหล่านี้ไปHECKกับลูกค้าของคุณ เพื่อจะดูว่าสิ่งที่คิดนั้นใช่สิ่งที่ลูกค้าต้องการจริงหรือไม่

## 4.2 สร้างคำนำเสนอแบบราบรัด



### คำนำเสนอแบบราบรัด

- สำหรับ [ผู้จัดการงานก่อสร้าง]
- ผู้ที่ [ต้องติดตามดูชนิดของงานที่ทำในไซต์ก่อสร้าง],
- เจ้า [CSWP\*]
- เป็น [ระบบใบอนุญาตงานเพื่อความปลอดภัย],
- ที่ [ใช้ สร้าง ติดตาม และตรวจสอบ ใบอนุญาตงาน เพื่อความปลอดภัย].
- ไม่เหมือนกับ [ระบบกระดาษที่ใช้อยู่]
- ผลิตภัณฑ์ของเรา [เป็นระบบผ่านเว็บและสามารถเข้าจากที่ไหนก็ได้].

\*CSWP: Construction Safety Work Permit

มีเหตุผลล้านแปดที่จะทำโปรเจกของคุณ



เรื่วนี้ ผมได้ทำแบบฝึกหัดนี้ กับทีมซึ่งได้รับมอบหมายให้ทำระบบ invoice สำหรับฝ่ายใหม่ในบริษัท และก็ต้องอึ้งไปกับความแตกต่างในเหตุผลที่ทุกคนคิด ว่าทำไม่เค้าถึงมาอยู่ตรงนี้ บางคนคิดว่า มีความต้องการที่จะลดจำนวนหน้าของ invoice เพื่อลดการใช้กระดาษบางคนก็คิดว่ามีความต้องการที่จะทำให้ invoice ดูง่ายขึ้นเพื่อที่จะลดจำนวนคนโทรเข้ามาที่ call center และก็ยังมีบางกลุ่มที่คิดว่าสิ่งที่จะทำนี้เป็นโอกาสที่ดีที่จะทำแคมเปญทางการตลาดเพื่อเพิ่มยอดขายของ product และ service ซึ่งทุกคำตอบเป็นคำตอบที่ดี และแต่ละคำตอบก็สามารถจะเป็นโครงการได้ด้วยตัวของมันเอง แต่หลังจากทุกคนได้มามุดคุยกัน เราก็ยังคงทำงานเข้าใจกันนั้นแหละ ที่ทำให้เป้าหมายที่แท้จริงของโครงการได้แสดงตัวออกมาให้เห็น — ซึ่งก็คือการทำให้ invoice ดูง่ายขึ้น และลดจำนวนโทรศัพท์ที่เข้ามายัง call center นั้นเอง ผ่านงงง

เรื่อเข้า! นักลงทุนที่คุณอยากรู้มาตลอดสามเดือนเพียงเดือนเดียวมาในลิฟต์ คุณมีเวลา 30 วิ เพื่อนำเสนอบริษัท start-up ใหม่ แกะกล่องของคุณ ถ้าคุณทำสำเร็จ คุณจะได้ทุนมาเป็นเชื้อเพลิงให้บริษัทคุณก้าวหน้าต่อไป แต่ถ้าคุณทำพลาดก็กลับไปนั่งกินอาหารแข็งแข็งที่บ้านต่อหนอนก็คือที่มาของ คำนำเสนอบรรบราบัด — วิธีการสื่อสารแก่น้ำเดียวกับคุณในช่วงเวลาอันสั้น คำนำเสนอบรรบราบัดไม่ได้มีไว้สำหรับนักธุรกิจเท่านั้น เราสามารถเข้ามามาใช้ในการประชุมแบบสั้นๆ ได้อย่างดี กดตัวอย่างคำนำเสนอบรบราบัด จะทำให้เกิดสิ่งดีๆ เหล่านี้กับโครงการคุณ

### 1. ให้ความชัดเจน

แทนที่จะเป็นทุกอย่างสำหรับทุกคน คำนำเสนอบรบราบัด บังคับให้ทุกคนตอบคำถามยากๆ ว่าในプロジェクトนี้คืออะไร และทำไปให้คร่าวๆ

### 2. บังคับให้ทีมได้คิดถึงลูกค้า

ในการที่เราไฟกัสไปยังเรื่อง ผลิตภัณฑ์ที่ทำอะไร และทำไม จะทำให้ทีมได้เข้าใจอย่างลึกซึ้ง ว่าอะไรทำให้ผลิตภัณฑ์เราประสบความสำเร็จ และทำไม่ลูกค้าถึงอย่างจะซื้อมัน

### 3. เข้าประเด็น

เหมือนกับแสงเลเซอร์ คำนำเสนอบรบราบัด ส่องทะลุลึกลึกลงไปในรากฐาน เพื่อให้มองเห็นถึงหัวใจของโครงการ ความชัดเจนทำให้เราสามารถจัดลำดับความสำคัญได้ง่ายขึ้น และเพิ่มอัตราส่วน signal-to-noise ได้อย่างมาก

ที่นี่มาดู template ที่จะช่วยในการทำคำนำเสนอบอกคุณ

Template สำหรับคำนำเสนอบนแบบราบรัตน์

- สำหรับ [ ลูกค้ากลุ่มเป้าหมาย ]
- ผู้ที่ [ ข้อความบอกถึงความต้องการหรือโอกาสที่ใช้ ]
- เจ้า [ ชื่อผลิตภัณฑ์ ]
- เป็น [ ชนิดของผลิตภัณฑ์ ]
- ที่ [ ประโยชน์หลักและเหตุผลที่ทำให้อยากซื้อ ].
- ไม่เหมือนกับ [ คู่แข่งหรือตัวเลือกอื่น ]
- ผลิตภัณฑ์เรา[ ข้อความบอกถึงความแตกต่างหลักๆ ].

การทำคำนำเสนอบนแบบราบรัตน์ไม่ได้มีแค่ทางเดียว อันที่ผมชอบนี้มาจากการหนังสือของ Geoffrey Moore ชื่อ Crossing the Chasm [Moo91]

- สำหรับ [ ลูกค้ากลุ่มเป้าหมาย ] — อธิบายว่าไปรเเจนี้ทำมาเพื่อใคร หรือใครที่จะได้ประโยชน์จากการใช้งาน มั้น
- ผู้ที่ [ ข้อความบอกถึงความต้องการหรือโอกาสที่ ] — ลงรายละเอียดของปัญหา หรือสิ่งที่ลูกค้าต้องการจะแก้ไข
- เจ้า [ ชื่อผลิตภัณฑ์ ] — ให้ร่วงกับไปรเเจนของเราด้วยการตั้งชื่อใหม่ ชื่อเป็นสิ่งสำคัญ เพราะเป็นสิ่งที่สื่อสารถึงความต้องการ
- เป็น [ ชนิดของผลิตภัณฑ์ ] — อธิบายว่า การบริการนี้ หรือ ผลิตภัณฑ์นี้ คืออะไร หรือ ทำอะไร
- ที่ [ ประโยชน์หลักและเหตุผลที่ทำให้อยากซื้อ ] — อธิบายว่า ทำไมลูกค้าถึงอยากรู้จักผลิตภัณฑ์นี้
- ไม่เหมือนกับ [ คู่แข่งหรือตัวเลือกอื่น ] — พูดครอบคลุมถึงเหตุผลที่ว่าทำไมเราถึงจะไม่ใช่สิ่งที่มีขายอยู่แล้ว
- ผลิตภัณฑ์เรา [ ข้อความบอกถึงความแตกต่างหลักๆ ] — สร้างความแตกต่าง และอธิบายว่าการบริการของเราแตกต่าง หรือ ดีกว่า ของคนอื่นยังไง ข้อนี้สำคัญมาก เพราะเป็นสิ่งที่พิสูจน์ให้เห็นว่าทำไมเราถึงต้องเคารพมาลงกับไปรเเจนนี้
- ประโยชน์ข้างต้นของคำนำเสนอบนแบบราบรัตน์ รวมรวมทุกสิ่งทุกอย่าง ที่เราต้องการสื่อออกไปอย่างรวดเร็วถึงแก่นของไปรเเจนหรือไอเดีย ไม่ได้อย่างสวยงาม มันบอกให้เรารู้ว่า ไปรเเจนเราคืออะไร ทำไปเพื่อใคร และทำไม่เค้าถึงอยากที่จะซื้อมัน

มีหลายวิธีที่จะทำกิจกรรม “คำนำเสนอบนแบบรับรั้ด” กับทีมของคุณโดยคุณสามารถเริ่มด้วยการพิมพ์ template นี้ออกมา และให้ทุกคนเติมคำในช่องว่า “ด้วยตัวเองและหลังจากนั้นจึงให้มาช่วยคิดด้วยกันหรือถ้าคุณต้องการจะรักษา คุณก็สามารถเอา template ขึ้นไปเจคเตอร์แล้วให้ทั้งกลุ่มช่วยกันเติมคำในช่องว่า “ด้วยการเติมที่ลະส่วนของ template ໄล์ลงไปหลังจากที่คุณได้คำนำเสนอบนแบบรับรั้ดมาอยู่ในเมื่อแล้ว เจ้ามาเปิดก็อกรความสร้างสรรค์ในหัวสมองเราเพื่อออกแบบกล่องบรรจุผลิตภัณฑ์กันดีกว่า

### 4.3 ออกแบบกล่องบรรจุผลิตภัณฑ์



บางทีซอฟต์แวร์เป็นเหมือนสิ่งที่สำคัญที่จำเป็นสำหรับบริษัท แทนที่จะต้องมาสื่อสารความไม่แน่นอนของไปรษณีย์ฯ หลายๆ คนคงอยากรู้ว่าจะเดินเข้าไปในห้องสร้างสิ่นค้าใกล้บ้าน เอาเครื่องการ์ดออกมานะ ก็ต้องซื้อสิ่งที่ต้องการจะมีก่อสร้างจะอีกนาน ที่ซอฟต์แวร์ราคาเป็นล้านจะเขามาใส่กล่องของข่ายบนห้องในชุดเบอร์มาร์เก็ต ก็ยังทำให้มีคำถามที่นำเสนอใจได้หนึ่งคำถาม ถ้าเราสามารถซื้อซอฟต์แวร์จากชั้นในชุดเบอร์มาร์เก็ต กล่องบรรจุซอฟต์แวร์จะหนาตากว่าเดิม แล้วที่สำคัญกว่านั้น เราจะซื้อมันมั้ย การทำงานกล่องบรรจุผลิตภัณฑ์ให้ไปรษณีย์ของคุณ และถามว่า “ทำไม่ได้” คราวนี้อย่างจะซื้อมัน ทำให้ทีมของคุณได้ไฟกัสในสิ่งที่จะช่วยให้ลูกค้าสนใจที่จะซื้อผลิตภัณฑ์คุณ และประยิชน์ที่สำคัญที่เป็นภารกิจของผลิตภัณฑ์ของคุณ หั้งสองอย่างเป็นสิ่งที่ดีที่ทีมของคุณจะต้องนึกถึงในขณะที่ทำงาน

แล้วมันทำงานอย่างไร?

อ่า รู้นะคิดอะไรมุ “ชั้นไม่ได้เป็นคน creative ชั้นไม่ได้ทำงานด้านโฆษณา ชั้นจะไปออกแบบโฆษณาให้ผลิตภัณฑ์ได้ยังไง” ผมมีอะไรจะบอก คุณทำได้แน่นอน และผมจะให้คุณดูว่าคุณจะทำได้ยังไงภายในสามสัปดาห์

### สเต็ป 1: ระดมความคิดหาประโยชน์ของผลิตภัณฑ์ของคุณ

ไม่ต้องไปบอกลูกค้าของคุณหรือว่าผลิตภัณฑ์คุณมี feature อะไรบ้าง — เค้าไม่สนใจหรอก ถึงที่คุณสนใจ จริงๆแล้วก็คือ ผลิตภัณฑ์ของคุณจะช่วยให้ชีวิตเค้าง่ายขึ้นได้ยังไงบ้าง หรือพูดได้อีกอย่างก็คือ ประโยชน์ของผลิตภัณฑ์ของคุณ นั้นเอง

ยกตัวอย่างเช่น ถ้าเราทำสิ่งจะโน้มน้าวครอบครัวนึง ให้เห็นถึงข้อดีของการซื้อ mini-van เวลา ก็อาจจะลิสต์ feature ทั้งหมดให้เค้าดู หรือ เราจะทำให้เค้าเห็นว่า mini-van จะทำให้ชีวิตเค้าดีขึ้นได้ยังไง

ฟีเจอร์	→	ประโยชน์
เครื่องยนต์ 245 แรงม้า		ขับได้ง่ายบนไฮเวย์
Cruise control		ประหยัดเงิน
เบรค Anti-lock		เบรคปลอดภัยขึ้นเพื่อคนที่คุณรัก

### อย่าลืมเชื่อมฟีเจอร์กับประโยชน์เสมอ!

เห็นความแตกต่างมั้ย?

ดังนั้น สเต็ปที่ 1 ในการทำล่องบรรจุผลิตภัณฑ์ของคุณก็คือ การไป弄จดหมายที่มีของคุณและลูกค้า และระดมความคิด ถึงเหตุผลทั้งหมดที่คนจะอยากรู้ผลิตภัณฑ์ของคุณ แล้วก็เลือกสามอันที่ดีที่สุดออกอีก

### สเต็ป 2: เขียนสโลแกน

ปัจจัยหลักของสโลแกนที่ดีก็คือ พูดในสิ่งที่คุณพูดให้มากที่สุด โดยใช้คำให้น้อยที่สุด ผู้คนไม่ต้องบอกคุณหรือว่าบริษัทเหล่านี้มีจุดยืนอะไร เพราะสโลแกนของเค้า ก็ได้บอกทุกอย่างหมดแล้ว:

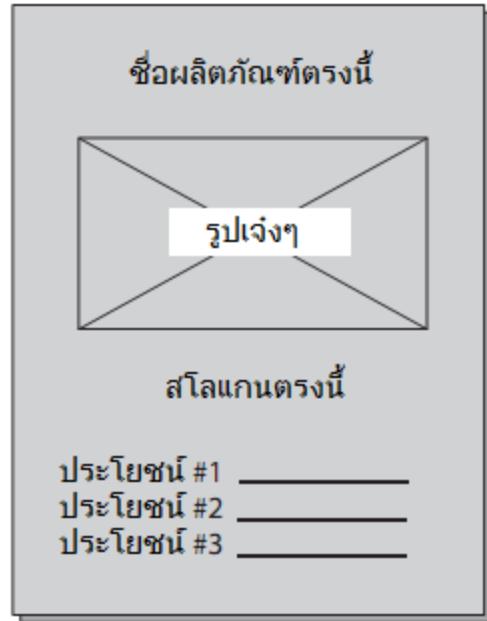
- Acura—The true definition of luxury. Yours. (คำนิยามแห่งความหรูหรา ของคุณ)
- FedEx—Peace of mind. (ความสงบสุขของจิตใจ)
- Starbucks—Rewarding everyday moments. (ให้รางวัลกับทุกช่วงเวลาของทุกวัน)

คุณรู้สึกถึงสิ่งที่ส่งมากับสโลแกนนี้มั้ย?

ใจเย็นๆ สโลแกนพวงนี้มันเทพมาก แต่สโลแกนของคุณไม่ต้องไปขนาดนี้ก็ได้ แค่รวมตัวกับคนในทีมคุณ จับเวลาประมาณสิบถึงสิบห้านาที ระดมความคิดหาสโลแกน แล้วก็สนับสนุนกับการใช้สมองในส่วน creative จำไว้นะว่า ไม่มีสโลแกนไหนเดียวกันไป

### สเต็ป 3: ออกแบบกล่องบรรจุ

เยี่ยม! ใกล้จะเสร็จแล้วล่ะ ด้วยเหตุผลที่ทำให้คุณสนใจข้อสามข้อ และสโลแกนติดนูญของคุณ คุณก็พร้อมที่จะเอาทุกอย่างมา รวมเข้าด้วยกันแล้ว



สำหรับแบบฝึกหัดนี้ ลองนึกภาพลูกค้าของคุณเดินเข้าไปในร้านขายซอฟต์แวร์เก็บบ้าน และเห็นกล่องบรรจุผลิตภัณฑ์ของคุณบนชั้นวาง และเมื่อเค้าหยิบมันขึ้นมา เค้าก็จะสังเกตดึงดูดให้ต้องซื้อไปใช้สิบกล่อง ไปให้ตัวเองและยังเอาไปฝากเพื่อนๆด้วย

เร็วเข้า ว่าดูปุกล่องนั้นขึ้นมา!

ไม่ต้องเป็นห่วงนะ คุณไม่ต้องผลิตผลงานขึ้นใบว์แดงระดับ *Mona Lisa* คุณแค่ต้องการกระดาษ flip chart ปากกาเมจิคสีๆ กระดาษแผ่นเล็ก กระดาษโพสติท และอะไรก็ได้ที่คุณสามารถหาได้ ตะไนสไลก์คุณออกมาก แสดงให้ลูกค้าคุณเห็นถึงประโยชน์ ใช้เวลาสิบห้านาทีออกแบบกล่องบรรจุผลิตภัณฑ์ที่ดีที่สุดที่คุณสามารถทำได้ ดีมาก! เห็นมั้ย ไม่ยากซักหน่อย ทำแบบฝึกหัดนี้ให้สนุก (ไม่บอยนักหรอกที่คุณจะได้เข้าสีเทียน หรือว่าดูปุกล่องใส่ผลิตภัณฑ์ที่น่าดึงดูดใจ) แบบฝึกหัดนี้สร้างความสัมพันธ์ที่ดีในทีม และเป็นเครื่องที่สนุก ที่ช่วยให้ได้คิดถึงเหตุผลว่า ทำไม เราถึงทำซอฟต์แวร์นี้ ในระยะคับขัน ต่อไป เราจะมาดูกันว่า เราจะสามารถทำอะไรได้บ้าง เพื่อตั้งความคาดหวังให้กับขอบเขตของโครงการ

#### 4.4 สร้างรายการ 'ไม่ทำ'

ในขอบเขต	นอกขอบเขต
<b>สร้างใบอนุญาตใหม่ แก้ไข/อ่าน/ลบ ใบอนุญาตที่มีอยู่แล้ว ค้นหา ทำรายงานแบบง่าย พิมพ์</b>	<b>ต่อเข้ากับระบบ ปิดถนนที่มีอยู่แล้ว ทำงานได้ต่อนไม่ต่อเน็ต</b>
<b>ปังไมรู้</b>	
<b>ต่อเข้ากับระบบ logistics ทำระบบรุดบัตรรักษาความปลอดภัย</b>	

เวลาตั้งความคาดหวังว่าไปรษณีย์คุณมีข้อบกพร่องแค่ไหน การบอกว่า เราจะ 'ไม่' ทำอะไรในนั้น สำคัญเท่ากับ การบอกว่าเราจะทำอะไรบ้างการที่คุณสร้างรายการ 'ไม่ทำ' จะทำให้เราบอกได้อย่างเคลียร์ว่าอะไรอยู่ และอะไรไม่อยู่ ในข้อบกพร่องของไปรษณีย์ การทำอย่างนี้ 'ไม่ใช่แค่ตั้งความคาดหวังให้กับลูกค้าที่ชัดเจนแล้ว แต่ยังทำให้มั่นใจว่า คุณและทีมของคุณจะโฟกัสกับสิ่งที่สำคัญจริงๆ และไม่ต้องไปใส่ใจสิ่งอื่นๆ'

แล้วมันทำงานยังไง?

รายการ 'ไม่ทำ' เป็นภาพที่ดีมากที่ทำให้เห็นว่าอะไรอยู่และไม่อยู่ในข้อบกพร่องของไปรษณีย์คุณได้อย่างชัดเจนว่ากันอย่างง่ายๆ ก็คือ คุณลูกค้าของคุณ และทีมของคุณมาร่วมตัวกันเพื่อเติมคำในช่องว่างซึ่งกันจะลดความคิดว่าต้องการเห็นฟีเจอร์อะไรบ้างในซอฟต์แวร์ในแบบกว้างๆ

ใน	นอก
 <p>ก้อนหินก้อนใหญ่ ที่เราต้องย้าย</p>	 <p>อะไรที่เราไม่ต้อง<sup>ไปเสียเงื่อนกับมัน</sup></p>
<b>ยังไม่รู้</b>	
<b>สิ่งที่เรา.yังต้องทำให้ชัดเจนขึ้น</b>	

ใน ประกอบด้วยสิ่งที่เราต้องการจะฟอกส ตรงนี้คือที่เราจะบอกว่า “นี่คือก้อนหินก้อนใหญ่ ที่เรากำลังจะเคลื่อนย้ายในโครงการนี้” สิ่งเหล่านี้อาจจะเป็นไฟเซอร์ในแบบกว้างๆ (แบบ ทำงานงาน) หรือเป้าหมายทั่วไป (แบบสามารถขยายการรองรับได้เหมือนกับช่อง)

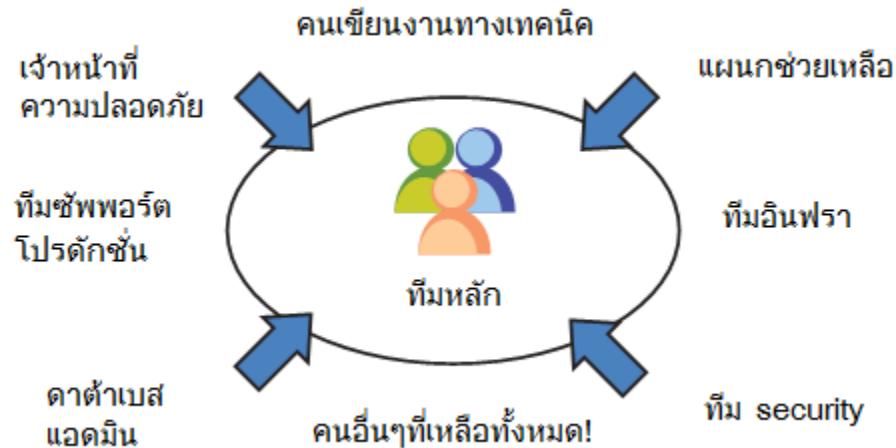
นอก ประกอบด้วย สิ่งที่เราไม่ต้องไปเสียเงื่อนกับมันอาจจะเป็นสิ่งที่เราอยากจะดันออกไป release หน้าหรือแค่เป็นสิ่งที่อยู่นอกขอบเขตของโครงการนี้ แต่ตอนนี้ เราจะไม่ไปกังวลกับมัน เอามันไปวางที่อื่นก่อน

ยังไม่รู้ คือสิ่งที่เราต้องทำการตัดสินใจส่วนนี้เป็นส่วนที่ตีมากเพรະมันทำให้เรามองเห็นความเป็นจริงในโครงการซึ่งมีผลลัพธ์ที่สำคัญส่วนใหญ่ ก็คือการที่คุณหลาย ๆ คน ต่างก้มองโครงการต่าง ๆ กัน—ซึ่งเป็นสิ่งที่เราต้องการหลีกเลี่ยง ท้ายที่สุด เราต้องการที่จะพยายามสิ่งในอยู่ในส่วน ยังไม่รู้ ของเรา ปยังส่วน ในหรือนอก

ความสวยงามของภาพนี้ก็คือ เราสามารถสื่อสารผ่านการมองเพียงแค่เดียว การที่เราได้ทำการ สิ่งที่ต้องทำใหญ่ๆ ที่อยู่ในขอบเขตทางด้านข้าง และสิ่งที่ไม่ได้อยู่ในขอบเขตทางด้านขวา และสิ่งที่ยังไม่รู้ทางด้านล่าง เพียงแค่มองແວ็บเดียว ทุกคนสามารถที่จะเห็นภาพได้อย่างชัดเจน ว่าขอบเขตของโครงการเราอยู่ตรงไหน

หลังจากที่มีขอบเขตที่กำหนดชัดเจนแล้ว เรา may ไปดูว่าในโครงการมีเพื่อนบ้านเป็นใครบ้าง

## 4.5 พับປະເພື່ອນບ້ານ



ເພື່ອນບ້ານທີ່ດີ ສາມາດເປັນເພື່ອນທີ່ດີທີ່ສຸດຂອງຄຸນ ພວກເຄົາຈະອູ່ຕຽງນັ້ນຕອນທີ່ຄຸນເຂົ້າບ້ານໄໝໄດ້ ພວກເຄົາຈະອູ່ຕຽງນັ້ນເມື່ອຄຸນ  
ຕ້ອງການເຄື່ອງມືອື່ອນບ້ານ ແລະ ມັນກີ້ວິສຶກດີມາກ ເມື່ອຄຸນຊ່ວຍເຄົາເຊື້ອ wireless ໃນບ້ານຂອງເຄົາ

### ຄໍາຖາມຮ້ອຍລ້ານ

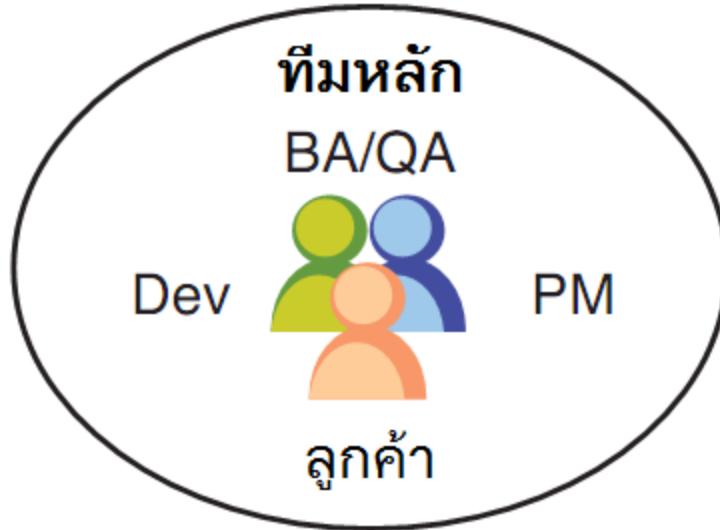
គັນນີ້ ທີ່ຜົມໄດ້ເຂົ້າວ່ວມ ຂັ້ນແໜ່ງການເຮີມຕົ້ນ ກັບບຣິຫັກສາຮາຮຸນປົກຂານາດໃຫຍ່ຂອງແຄນາດ VP ຂອງແຜນກົດາມເຂັ້ມາວ່າ ແລ້ວ  
ຮະບບປິ່ນນີ້ຈະຕ້ອເຂົ້າກັບຮະບບມເນັ້ນພົມທີ່ມີອູ່ໄດ້ຢັ້ງໄຟໃນທັງນັ້ນເງື່ອບະຈຸນຄຸນແທບຈະໄດ້ຍືນເສີຍເໝັ້ມຕກ VP ຜົ່ງທີ່ດຳອັນເປັນຄົນ  
ເຫຼັກເຊົາທີ່ຈະມີຜົດຕ່ອງຄວາມສໍາເວົງຂອງໂປຣເຈນນີ້ ໄນໄດ້ເຂົ້າໃຈເລຸຍວ່າ ຮະບບໃໝ່ໄໝໄດ້ຕ້ອເຂົ້າກັບຮະບບເກົ່າ ແຕ່ມັນຈະມາແກນທີ່ຕ່າງໆ  
ແລະເປັນພະຍາໄຕສ້າງຮາຍການ ໄນທຳເຂົ້ນມາເທົ່ານັ້ນແລະ ທີ່ຊ່ວຍໃຫ້ເຮົາລິກເລື່ອງການທີ່ຈະຕ້ອງມາດັ່ງຄວາມຄາດຫວັງໃໝ່ ທັງຈາກ  
ທຳໂປຣເຈນໄປນາແລ້ວ ມັນດີກວ່າທີ່ມາທຳຕອນນີ້ ແກນທີ່ຈະໄປທຳຕອນທີ່ເວີ້ມທຳໂປຣເຈນໄປແລ້ວ

ໄໝວ່າຄຸນຈະເຂົ້າໂຮື້ໂນກົດາມ ຄຸນກົມືເພື່ອນບ້ານໃນໂປຣເຈນຂອງຄຸນ ແຕ່ແກນທີ່ຈະເປັນຄົນທີ່ເກັບກຸ່ມແຈສໍາຮອງໄວ້ໃໝ່ ອົງໂຫຍມເຄື່ອງມືອື່ອນບ້ານ ພວກເຄົາຊ່ວຍດູແລດາຕ້າເບສ ທຳການຕຽບສອບ security ແລະ ທຳໃຫ້ເນັດເວົາກົດທຳການໄດ້ອ່າຍ່າງດີ

ການໄປປະເພື່ອນບ້ານຂອງຄຸນ ຈະທຳໃຫ້ຄຸນສາມາດສ້າງຄວາມສົມພັນທີ່ໄດ້ໄວ້ລ່ວງໜ້າ ເພື່ອຊ່ວຍຄຸນໄດ້ໃນອານາຄຕ ແລະ ມັນກີ້ວິສຶກດີມາຍາທີ່ດີ ທີ່ຈະທັກທາຍກັນອູ່ຕູລົດ ແກນທີ່ຈະຈົງໄປໜ້າເຄົາເວລາທີ່ຄຸນມີປົມຫາເທົ່ານັ້ນ ແລະ ສິ່ງທີ່ສໍາຄັນທີ່ສຸດ ທີ່ຈະເປັນຕ່ອງການສ້າງ  
ຮາກຮູ້ານສົດຂອງໂປຣເຈນທີ່ຈະປະສົບຄວາມສໍາເວົງ—ຄວາມໄວ້ໃຈ

ຄວາມຜິດພາດໃນການທຳໂປຣເຈນຮັ້ງໃຫຍ່ຮັ້ງແຮກຂອງຜມ

ເຈາທຸກຄົນເຄຍທຳຜິດພາດ ຄວາມຜິດພາດຮັ້ງໃຫຍ່ຮັ້ງນີ້ຂອງຜມ ຄືອຕອນທີ່ເປັນຫັວໜ້າທີ່ມີ ThoughtWorks ຕອນທີ່ເຈາທຳການ  
ບາງຍ່າງໃໝ່ Microsoft ມີປີທີ່ນັ້ນ ແລະ ເວີ້ມທຳໂປຣເຈນໂດຍຄືດວ່າ ສັນຄົມໂປຣເຈນຂອງເຈາເປັນຍ່າງນີ້:

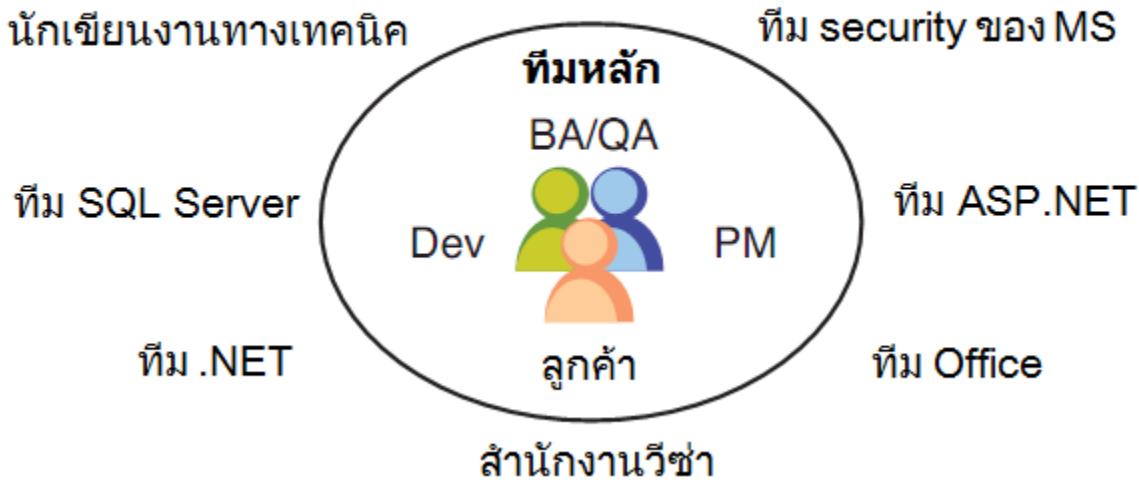


ตอนแรกๆ ทุกอย่างก็ยังดูดีอยู่ ทีมทำแอปฯ ใจล้ำ เราส่งมอบซอฟต์แวร์ที่ทำงานได้ด้อยู่ต่อลด ชีวิตก็มีความสุขดี แต่พอใกล้ๆ จะจบโครงการ บางอย่างแปลงๆ ก็เริ่มเกิดขึ้น กลุ่มคนที่ไม่เคยเห็นมาก่อน มาจากไหนก็ไม่รู้ ก็เริ่มมาขออะไรไร้สาระ จากผู้ผลิตและทีมของผู้ผลิต

- กลุ่มนึงต้องการตรวจสอบ architecture ของเรา (อย่างกะ architecture เราต้องการตรวจสอบจังหวัดแล้ว!)
- อีกกลุ่มต้องการจะทำให้มันใจว่าเราทำงานโดยภาย security ของบริษัท (บ้าป่า!)
- และอีกกลุ่มต้องการจะดูเอกสารของเรา (เอกสารอะไรเนี่ย!)

คนเหล่านี้เป็นใครกัน เค้ามาจากไหน และทำไมเค้าถึงต้องการที่จะมาทำให้กำหนดการเราพังด้วย ข้อขำคืน ลังๆ โปรเจคเด็กๆ น่ารักของเรา กลายจากทีมเล็กๆ ของคนหกคน เป็นอะไรที่ยิ่งใหญ่กว่าที่น้ำมาก

## สังคมใหญ่



ถึงแม้มอยากที่จะให้คนอื่นที่มาทำให้กำหนดการของเรางามมากแค่ไหน ความเป็นจริงก็คือ ผู้ไม่ได้ชานตั้งกับคำพูดที่ว่า สังคมไปริบเชิงคุณมักจะใหญ่กว่าที่คุณคิดเสมอ เท่าที่ควรด้วยการ “พบປະເພື່ອນບ້ານ” คุณต้องการที่จะหาดແຜນที่ “ว່າໃຈຮອຍໆ” ในสังคมไปริบเชิงคุณบ້ານ ให้ເດືອຍ່າງຍາຍໄດ້ເວດວົງຂອງคุณ และເຮີມສ້າງຄວາມສັມພັນນີ້ທີ່ດີ ກ່ອນທີ່คุณຈະຕ້ອງການມັນ ດ້ວຍວິທີນີ້ ເນື້ອເລານໜັ້ນມາດີ່ ດູຈະໄດ້ມີເປັນຄົນແປລກຫຼາ ແລະເດັກຈະສາມາດຮູ່ວ່າຍຸດໄດ້ດີຢຶ່ງໜີ້

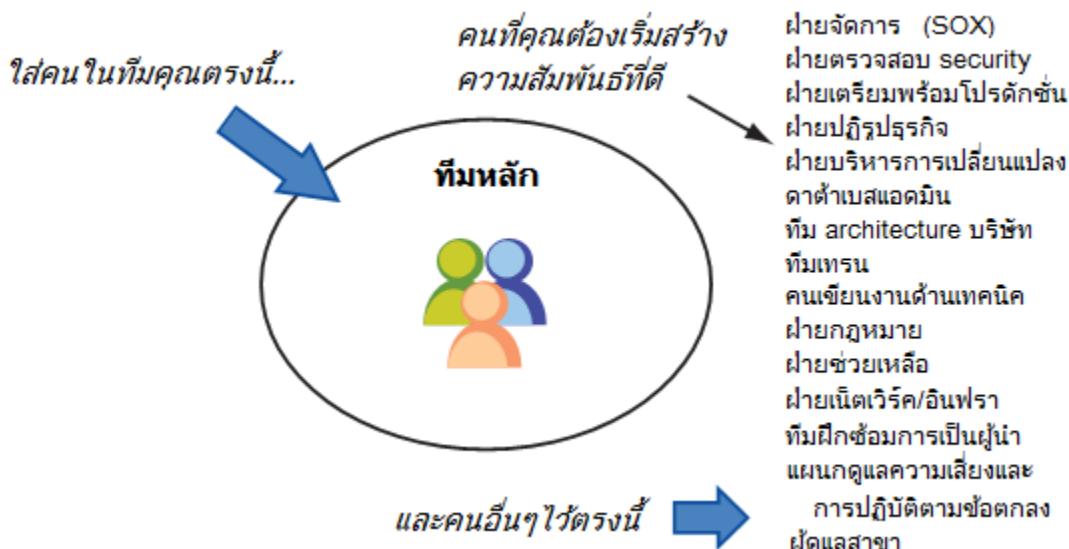
ມັນທຳງານຍັງໄໝ?

ຈະດີມຄວາມຄິດກັບຄົນໃນທີມຂອງຄົນ ເບີນຫົ່ວ່າທີມທີ່ຄຸນຄິດວ່າ ຄຸນຈະຕ້ອງຕິດຕ່ອງ ກ່ອນທີ່ໂປຣເຈົ້າຄຸນຈະອອກໄປສູລົກກາຍຸນອກ ຄົນໃນທີມທີ່ໄດ້ຍຸກັບບວິທີ່ມາເປັນເວລານານ ແລະຮູ່ຖິ່ນໂຍບາຍຕ່າງໆຂອງບວິທີ່ ແລະດ່ານຕ່າງໆຂອງອົງກົງກວດທີ່ຄຸນຈະຕ້ອງຂ້າມຝ່ານ ຈະມີປະໂຍ້ນ໌ ມາກໃນເວລານີ້

### ກາແພ ໂດນັກ ແລະ ຄວາມຈິງໃຈ

ເນື່ອພຸດຊື່ກາຮ້າງຄວາມສັມພັນນີ້ທີ່ໄດ້ຄວາມເຄາະພື້ນທີ່ກັນແລະກັນກັບເພື່ອນບ້ານຂອງເວາແລ້ວ ໃນເນື້ອໄຈດີໄປກວ່າ ກາແພດີ່ຫັກແກ້ວ່າ ແລະ ໂດນັກແສນອ່ວຍເຮົາເລື່ອກ ກາແພ ເພວະກາແພມາໃນຫຍ່ອກຮ້ອນໆ ແລະໃນຮ່ວງທີ່ເດັກກຳລັງດີ່ມີດໍາກັບກາແພ ເດັກຈະໄດ້ຮູ່ສຶກສິ່ງ ຄວາມຮູ່ສຶກອຸນທີ່ຄຸນອຸຍາກມອບໃຫ້ດ້ວຍເຮົາເລື່ອກ ໂດນັກ ເພວະໃນຂະນະທີ່ຄຸນກຳລັງບອກເພື່ອນບ້ານຂອງຄຸນ ວ່າຄຸນໜີ້ແກ້ໄຂນີ້ທີ່ ຄຸນມີພວກເດັກອູ່ຮອບຂໍ້າງ ເດັກຈະໄດ້ລື້ມຮສຄວາມຫວານຂອງໂດນັກ ແລະກົງຈະໄດ້ເຫື່ອມໂຍງຄຸນກັບຄວາມຫວານນັ້ນແຕ່ເຄື່ອງມືອັນສຸດ ຍອດ ທີ່ຈະສ້າງຄວາມສັມພັນນີ້ທີ່ດີກັບເພື່ອນບ້ານຂອງຄຸນ ກົດ່າວ່າ ຄວາມຈິງໃຈ ເພື່ອທີ່ເພື່ອນບ້ານຂອງຄຸນຈະໄດ້ຮູ່ວ່າຄຸນໜີ້ແລະເຫັນຄຸນຄ່າຂອງເດັກ ຄຸນຈະຕ້ອງໝາຍຄວາມຍ່າງນັ້ນຈິງໆ ພວກເດັກຈະມອງທະລຸດໍາ ຂົມແບບໄມ່ຈິງໃຈໄດ້ຍ່າງຈາກເຮົາ ຄວາມໜີ້ແລະຄໍາຂອບຄຸນທີ່ຈິງໃຈຈະພາຄຸນໄປໄດ້ໄກລ ໃນກາງທີ່ຈະໄດ້ຮັບການສັບສົນ ແລະຄຸນພໍ່ອມທັງໂປຣເຈົ້າຂອງຄຸນກົງຈະປະສົບຄວາມສໍາເລົາເພວະສິ່ງນີ້

ສັນຄົມໃໝ່



หลังจากที่คุณได้枉ฟังชื่อลงบนแผ่นที่แล้ว ให้คุยถึงแต่ละกลุ่ม และดูว่าคุณสามารถใช้ชื่อคนที่คุณจะใช้ติดต่อได้หรือไม่ โปรเจค เมเนจอร์ หรือครูกัตตามในทีมที่จะเป็นคนนำทีมในการสร้างความสัมพันธ์ภายนอกทีม จะได้ใช้สิ่งนี้ในการวางแผน ดึงกลุ่มคน เหล่านี้มาร่วมงานกับเรา



## อาจารย์ กับ นักเรียนอย่างไร

**ลูกศิษย์:** อาจารย์แบบฝึกหัดเหล่านี้ ต้องใช้เวลาของผู้ให้การสนับสนุนและผู้ที่เกี่ยวข้องทั้งหมด แล้วถ้าท่านเหล่านั้นไม่มีเวลา ว่างที่เหมาะสม หรือว่ายุ่งมาก ไม่สามารถมาตอบคำถามเหล่านี้เกี่ยวกับโปรเจคได้ เราจะทำยังไงดี?

**อาจารย์:** ถ้าอย่างนั้น เจ้าก็แสดงความยินดีกับตัวเจ้าเองได้ เพราะเจ้าได้ค้นพบความเสี่ยงขันใหญ่หลวงอย่างแรกของโปรเจค เจ้าแล้ว

**ลูกศิษย์:** ความเสี่ยงอย่างไรท่าน?

**อาจารย์:** การมีส่วนร่วมของลูกค้า ปราศจากการมีส่วนร่วมของลูกค้า โปรเจคของเจ้าก็ประสบปัญหาตั้งแต่ยังไม่ได้เริ่มเสียแล้ว ถ้าลูกค้าของเจ้าไม่มีเวลาจะมาบอกรวบรวมกว่า ทำไม่เจ้าจึงต้องทำซอฟต์แวร์นี้ บางทีมันอาจจะไม่ควรทำเลยก็เป็นได้

**ลูกศิษย์:** ท่านจะบอกว่า เราควรจะหยุดโปรเจคอย่างนั้นรึ?

**อาจารย์:** ข้ากำลังบอกว่า เพื่อที่จะทำโปรเจคให้ประสบความสำเร็จได้นั้น เจ้าต้องการการมีส่วนร่วมจากลูกค้าและผู้ที่เกี่ยวข้อง และถ้าไม่มีสิ่งนี้แล้วนั้น เจ้าก็คงต้องหยุดชะงัก ไม่ว่าเจ้าจะชอบหรือไม่ก็ตาม

**ลูกศิษย์:** แล้วถ้าเป็นเยี่ยมนั้นแล้ว ข้าควรจะทำยังไงดีท่าน?

**อาจารย์:** เจ้าต้องขอ匕าย อย่างชัดเจน และหนักแน่น ให้ลูกค้าของเจ้าฟัง ถึงสิ่งที่จะทำให้โปรเจคนี้ประสบความสำเร็จได้ การมีส่วนร่วมของลูกค้า เป็นสิ่งจำเป็นที่ต้องมี หรือนี่อาจจะไม่ใช่เวลาที่ควรจะทำโปรเจคนี้ก็เป็นได้ บางทีเข้าเหล่านี้อาจจะยุ่งมาก และมีสิ่งที่ต้องทำเบื้องต้นไป ถ้าเป็นเยี่ยมนั้นแล้วล่ะก็ เจ้าจะบอกราบรู้ทั้งพากษาพร้อม ก่อนจะถึงเวลา นั้น เจ้ายังมีลูกค้ารายอื่นที่ต้องรับใช้กันหลายราย

**ลูกศิษย์:** ขอบคุณท่านมากท่านอาจารย์ ข้าจะนึกถึงสิ่งเหล่านี้ให้มากขึ้น

แล้วไงต่อ?

ก่อนที่เราจะไปกันต่อ มาพักหายใจซักแป๊บ

คุณรู้สึกถึงมั่นเมี้ย?

คุณเห็นสิ่งที่กำลังเกิดขึ้นตรงนี้เมี้ย?

ในแต่ละแบบฝึกหัดของชั้นเรื่องต้นที่ผ่านไป จิตวิญญาณและขอบเขตของโปรเจคก็ค่อยๆ ดูเด่นขึ้นเรื่อยๆ

- เราได้รู้แล้วว่า ทำไม่ เรายังไงไปเจอกันนี่
- เราไม่สามารถเปลี่ยนอะไรได้
- เรารู้ว่ากล่องบรรจุผลิตภัณฑ์เราจะหน้าตาเป็นยังไง
- เราได้วางเสารอบๆ ขอบเขตของโปรเจค
- เราไม่มีความรู้ว่ามีใครเป็นเพื่อนบ้านของเราบ้าง

ตอนนี้ ผมรู้ว่าคุณกำลังคิดอะไรอยู่ บริบทมากพอได้แล้วน่า! เมื่อไหร่เราจะลงลึก และเริ่มพูดถึงว่าเราจะสร้างสิ่งนี้ขึ้นมา ได้ยังไงซักที? และคำตอบก็คือ ตอนนี้เอง

ในตอนที่ 5 ทำให้มันเป็นจริง

ในหน้าถัดไป เราจะเริ่มทำให้เห็นภาพของวิธีการแก้ปัญหาเชิงเทคนิคของโปรเจคคุณ และอะไรที่จะต้องมี เพื่อให้เราทำงานให้เสร็จได้

ดังนั้น เปิดหน้าถัดไปเลย และเตรียมตัวให้พร้อมกับการทำให้มันเป็นจริง

# ตอนที่ 5 Agile ไม่ใช่แค่คิด แต่ต้อง

## ทำได้จริง



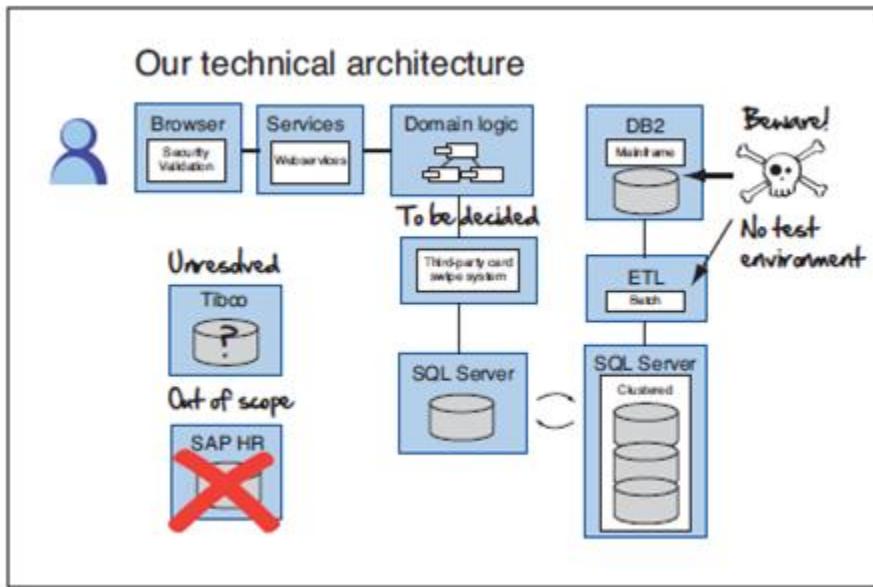
พดจิง(มามากพอแล้ว) ทำจริง(ได้เสียที)

เมื่อเราฟังแล้วว่าทำไม่เราถึงมากอยู่ตั้งนานี้ เราทำลังจะทำอะไรให้ใครแล้ว ต่อจากนี้เป็นขั้นตอนของคำว่า “ทำยังไง” ในบทนี้เป็นเรื่องของการสร้างกรอบความคิดเบื้องต้น เราจะเริ่มทำให้สิ่งที่เราคิดไว้เป็นสิ่งที่เป็นรูปธรรมและวางแผนรากฐานให้กับมัน สิ่งที่เราจะเริ่มต้นมันมีดังต่อไปนี้

- นำเสนอทางวิธีการทางด้านเทคนิค
- ทำการประเมินและประมาณถึงสิ่งที่ต้องทำ
- ตระหนักรองดีๆ เพื่อให้เข้าใจอย่างถ่องแท้
- อะไรมีที่ต้องใช้และต้องให้อะไรเป็นการแยกเปลี่ยน ( เพราะต้องให้ก่อนมันถึงจะได้นำในทุกๆ สถานการณ์ )
- แจ้งให้กับทางผู้สนับสนุนโปรเจคของเราเพื่อให้โปรเจคสำเร็จอย่างมีประสิทธิภาพ

แต่ก่อนอื่นมาเริ่มกันด้วยขั้นตอนเหล่านี้เลย

## 5.1 มี solution ดีๆ เอาจริงอุ่นใจให้ทุกคนดู (ว่าหน้าตามันเป็นยังไง)



การแสดงให้เห็นแบบติดตามจริงใจให้หลับตามึกภาพได้แล้วเข้าใจใน solution นั้นก็คือว่าให้ทุกคนเข้าใจตรงกันว่าในทางปฏิบัติ แล้วเราจะกำลังจะสร้างอะไรชี้รวมไปถึงวิธีการทำงานด้านเทคนิคด้วย ที่สำคัญที่สุดคือให้ทุกคนยอมรับถึงสิ่งที่เรากำลังสร้างภาพให้เค้าเห็นเข้าใจ (แบบว่าถ้าสร้างตีก 10 ขั้นอย่างน้อยก็ให้ทุกคนเห็นว่ามันมีเส้นเปลี่ยนแนวหน้าบ้างคงจะเป็นไปได้และมันจะสำคัญแน่ๆ) การนำความจริงเหล่านี้มาตีแผ่เพื่อให้ทุกคนในทีมและลูกค้าของเรารู้ได้เข้าใจตรงกันนั้นมีข้อดีอยู่มาก เพราะว่า

- เป็นการทำให้ความผันไม่เกินขนาดการและเข้าใจได้ว่าเราจะใช้เครื่องมือและเทคโนโลยีอะไร
- ทำให้มีความเข้าใจชัดเจนว่าอะไรคือขอบเขตและความรับผิดชอบของโปรดเจด
- ที่สำคัญเป็นการบอกถึงความเสี่ยงที่ควรระวัง

จากเหตุผลดังกล่าว ก็แน่นอนว่าซึ่งในการนำเสนอสิ่งที่เรามองเห็นกลับมาของทุกคนความถึงที่มีของเราและลูกค้าก็เป็นสิ่งจำเป็นอย่างหลีกเลี่ยงไม่ได้ แม้ว่าเราจะคิดว่าทุกคนเข้าใจตรงกันแล้วก็ตาม ถึงแม้ว่าอย่างมากกิจกรรมนี้อาจจะเป็นเพียงแค่การย้ำเน้นความเข้าใจว่าทุกคนเข้าใจตรงกันแน่ๆ แต่มันก็จะดีกว่าที่จะเดินไปข้างหน้าแล้วพบว่าเรากำลังเทหมดหน้าตักและลงทุนในสิ่งที่ไม่ตอบสนองความต้องการของใครเลย

### ต้องทำอะไรยังไงล่ะนั้น

จริงๆแล้วขั้นตอนของมันก็ง่ายๆเพียงแค่ว่าคุณนั่งสุมหัวกับทีมและผู้มีส่วนเกี่ยวข้องทางด้านเทคนิค เริ่มพูดคุยกันถึงสิ่งที่กำลังจะทำแล้วก็เริ่มวาดรูป Diagram ประมาณว่า Architecture Diagram นั่นล่ะครับ (เริ่มดูจะคุ้นตามั้ย เพราะเป็นแบบฝึกหัดง่ายๆ ที่ทุกๆทีมก็เคยทำกันอยู่แล้ว) และลองจินตนาการสร้างเหตุการณ์สมมติ (What-if scenario) เพื่อดู impact และเพื่อให้เข้าใจว่าสิ่งที่เรากำลังจะทำนั้นจะเข้าท่อนหรือต้องใช้แรงงานมากน้อยขนาดไหน

ถ้าหากว่าคุณมีเครื่องมือหรือ framework ที่เป็น open source ก็ควรที่จะให้เป็นไปตามมาตรฐานที่ยอมรับในองค์กรอาจจะมีการจำกัดเรื่อง open source (เพิ่มเติม ก็คือว่า การ manage เรื่อง tools เป็นสิ่งสำคัญมาก การ train และฝึก skill ของคนในทีมตลอดจนถึงการหาคนมา support นั้นเป็นปัญหาหลักอีกอย่างหนึ่งในการที่จะกำหนดว่าทีมของเราระจะใช้ tools หรือ framework อะไร)

#### คุณเลือก architecture ก็ต่อเมื่อคุณได้เลือกทีมของคุณแล้ว

เมื่อคุณนัดหมายก็จะทำอย่างนั้นแน่นอน เพราะเมื่อคุณมีค่อนอยู่ในมือคุณก็จะต้องตระหนูกันว่าทุกสิ่งทุกอย่างที่คุณเห็นมันจะกลายเป็นตะบูไปประหมด เมื่อคุณมีทีมที่เก่งเรื่อง SQL งานของคุณก็มีแนวโน้มที่จะทำโดยใช้เทคนิคทางด้านนี้ในการทำงานเป็นส่วนใหญ่ ในขณะที่ถ้าทีมของคุณเก่งเรื่อง Object Oriented มันก็แน่นอนอยู่แล้วว่าความซับซ้อนเหล่านั้นมันก็จะถูก implement ลงไปในงานคุณ

ซึ่งมันชัดเจนเลยว่าเมื่อคุณเลือกทีมของคุณ(ที่เก่งด้านไหน) มันจะมีผลอย่างยิ่งต่อ solution architecture ที่คุณจะใช้สร้างครับ ดังนั้นพยายามหาข้อสรุปทางด้านเทคนิคให้ได้ไวๆแนวทางที่เราจะสร้างจะเป็นแบบไหน ไม่ใช่ว่าเราจะต้องให้ได้มาซึ่ง solution ที่สมบูรณ์แบบ (จะเห็นว่าไม่ใช่ big design upfront นะครับ) หรือเพื่อให้ได้คำตอบทุกๆอย่างที่ชัดเจน (ก็ไม่ใช่เรื่องให้ได้ requirement ที่สมบูรณ์แบบอีกนั้นแน่นอน) แต่เป็นเพราะว่าคุณต้องการได้คุณที่มีความถนัดและมีความสามารถในงานที่คุณกำลังทำ (เพิ่มเติมนะครับ ยกตัวอย่างว่าถ้าคุณกำลังจะสร้างบ้านไม่คุณก็ต้องการงานช่างไม่ใช่ช่าง泥工 คุณจะได้มีไปจ้างช่างชาวบุญที่ช่าง泥工 แม้จะเป็นการสร้างบ้านเหมือนกันแต่แนวทางที่กำลังจะสร้างมันจะเป็นข้อสรุปว่าเราควรจะเลือกคนงานอย่างไร)

ครับทั้งหมดที่กล่าวมาให้สรุปสั้นๆก็คือว่า พยายามทำให้ทุกคนเข้าใจโดยใช้รูปและ Diagramอย่างง่ายๆว่าสิ่งที่เรากำลังจะทำคืออะไร ในเนื้องานบางทีมที่มีความเสี่ยงสูง(หมายถึงในเนื้องานที่อาจจะมีปัญหา เช่นการทำ integration เข้ากับระบบอื่นๆ) ตรงนั้นต้องทำให้เข้าใจตรงกันไปเลยว่าเรากำลังจะเจอกับอะไร และต้องให้แน่ใจว่าทุกๆคนเห็นด้วยกันกับเทคนิคที่เรากำลังจะใช้กับสิ่งที่เราทำ

## 5.2 ตั้งคำถามกับตัวเองว่าอะไรที่ทำให้เราเป็นกังวล



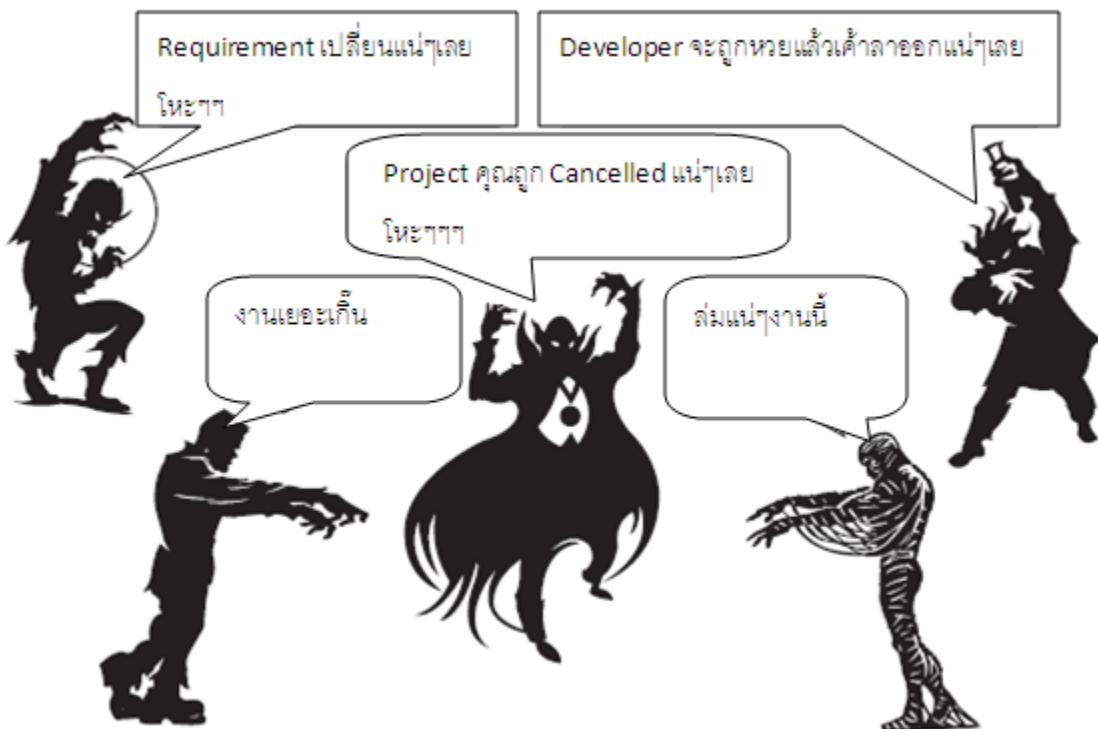
### Project Risks

- ผู้อำนวยการสร้างไม่ว่าง (แล้วใครจะมาช่วยคุมงบประมาณ)
- ทีมงานไม่ได้นั่งที่เดียวกัน (แล้วมันจะทำงานลำบากขนาดนั้น)
- มาตรฐานทางด้าน Security ที่ต้องนำมาใช้ (ก็ไม่เคยทำมาก่อนจะทำได้มั้ย)
- ระบบจัดการ Logistic ที่กำลังจะเข้ามาใหม่ (ไม่เคยใช้มันจะเสี่ยงไปมั้ยนี่)

มีผู้จัดการรายคนที่ต้องดูแลการพัฒนาซอฟต์แวร์แล้วไม่สามารถอนุมัติได้ในตอนกลางคืน (มันฟังดูคุณามั่ยครับ) เพราะเหตุผลมากมาย – ลองมาฟังเหตุผลดีๆ ดังต่อไปนี้ (อาจจะฟังดูเหมือนเรื่องของตัวเองมากขึ้นอย่างไม่น่าเชื่อ) บางครั้งเหล่าผู้จัดการทั้งหลายก็ประมวลการในแบบว่ามองโลกในแง่เดียว (สุปERVERMANTICA แนวโนนควรฟังดูเป็นเรื่องปกติอย่างมาก) บ่อยครั้งเหลือหลายที่ลูกค้าเปลี่ยนใจ หรือบ่อยครั้งที่เราต้องรับมืองานใหม่หลังเกินที่ต้องทำ (มากกว่าที่จะทำให้เสร็จภายในเวลาที่มีและก็เงินที่มีอย่างจำกัด) ครับแน่นอนสิ่งเหล่านี้ยิ่งฟังยิ่งดูคุ้นเคย

### **ปีดเผยแพร่และสร้างสรรค์(ติเพื่อก่อ(เรื่องให้เป็นเรื่อง))**

ทำไม่การพูดถึงความเสี่ยงและปัญหาเป็นเรื่องดี



การพูดถึงความเสี่ยงของโปรเจคนั้นเป็นสิ่งที่หลาย ๆ คนพยายามหลีกเลี่ยงและไม่ยอมพูดถึงเมื่อเริ่มต้น โปรเจค ไม่มีใครยกดูถูกว่าเป็นภาระต่ำต้นทุนว่ามัวแต่กังวลว่าโปรเจคยังจะล้มและเราต้องแยกแยะไปมีรอดแน่โปรเจคนี้แต่การพูดถึงความเสี่ยงนั้น เป็นทางออกที่ดีที่สุดที่จะทำให้ทุกคนได้รับรู้ว่าอะไรเป็นสิ่งจำเป็นและเป็นปัจจัยให้ Project ของเรางานได้ ขยายตัวอย่างเรื่องการใช้พื้นที่ในการทำงานระหว่าง (ให้ทุกคนนั่งอยู่ในพื้นที่เดียวกัน) สำหรับคนที่ไม่ได้ทำงานในการพัฒนาซอฟต์แวร์มาก่อน การให้คนทำงานมากันด้วยการอาจจะไม่ใช่เป็นเรื่องใหญ่มากนัก แต่สำหรับ agile project แล้วการนั่งรวมกันนั้นเป็นสิ่งที่สำคัญที่สุดและการพูดถึงความเสี่ยงนั้นก็เป็นเหมือนกับการบอกว่าถ้าสิ่งเหล่านี้เป็นจริงขึ้นมาและไม่ทำอะไรแล้วล่ะก็แน่นอนว่า Project นี้ไม่สำเร็จได้เลยแน่นอน

- ไม่ได้มีทีมที่นั่งที่เดียวกัน
- ไม่ได้มีลูกค้าที่ให้ความสำคัญและให้เวลาเพียงพอ กับ Project
- คุณไม่ได้เป็นคนควบคุม Development Environment – เพื่อติดสกนิดนะครับการที่เราจะใช้ Tool อะไรและพัฒนาด้วยของเราเป็นความเชี่ยวชาญของทีม ซึ่งคุณควรจะเป็นคนกำหนดได้
- หรือมีสิ่งที่เป็นปัจจัยอื่นๆ ที่สำคัญที่จะทำให้ Project ประสบความสำเร็จได้

### Bloomberg on risk

Michael Bloomberg อย่างน้อยต้องรู้ว่าเราบังเกี่ยวกับเรื่องความเสี่ยง เพราะในฐานะผู้ก่อตั้งบริษัททางด้านการเงินและยังเป็นนายกเทศมนตรีของมหานครนิวยอร์ก เค้าต้องตัดสินใจและทำอะไรที่มีความเสี่ยงสูงอยู่บ่อยๆ

ในหนังสือ Bloomberg by Bloomberg คุณ Michael ได้อธิบายเทคนิคที่เค้าใช้ในการจัดการกับความเสี่ยงดังต่อไปนี้

- เขียนทุกๆ อย่างที่มีความเป็นไปได้ว่าจะผิดพลาดได้
- แล้วใช้เวลานั่งคิดสกนิดว่าทำยังไงมันถึงจะไม่เกิด
- จัดการมันซะ

ปรัชญาของคุณ Michael นั้นง่ายๆ คือว่าคุณไม่มีทางจะเห็นทุกสิ่งทุกอย่างที่กำลังจะผ่านเข้ามาได้ทั้งหมด และแน่นอนว่าไม่มีแผนอะไรที่มันเป็นไปตามแผนของทุกอย่าง (ฟังดู Agile มากร) แน่นอนว่าชีวิตคุณมันก็เหมือนกันบางทีมันก็ไม่เหมือนกับที่เราซักซ้อมไว้ พยายามทำความคุ้นเคยกับสิ่งที่คาดไม่ถึง ไม่ว่าคุณจะรู้หรือไม่รู้ก็ตามว่าอะไรกำลังจะเกิดขึ้น มันก็แค่ยอมรับกับมัน และรับมือกับมันตามที่มันเป็นเท่านั้นเอง

นี่เป็นโอกาสอันดีที่จะพูดเรื่องความเสี่ยง เพราะว่ามันเป็นเวลาที่ดีที่สุดที่เราจะยืนยันมาแล้วกับทุกๆ คนถึงสิ่งที่คุณต้องการ และจำเป็นต่อความสำเร็จของ โปรเจค แน่นอนว่าคุณอาจจะไม่ได้ทุกอย่าง แต่อย่างน้อยก็ได้มีการทำให้ทุกๆ คนได้เข้าใจตรงกัน ว่าผลที่เกิดขึ้นมีค่าไม่เท่ากับสิ่งที่คุณคาดไว้ และมันสำคัญมาก (แค่ไหน) และต่อไปนี้คือเหตุผลดีๆ (คึกคัก) ว่าทำไม่เราควรพูด เรื่องความเสี่ยงหรือปัญหาที่อาจจะเกิดขึ้นในตอนเริ่มโปรเจค

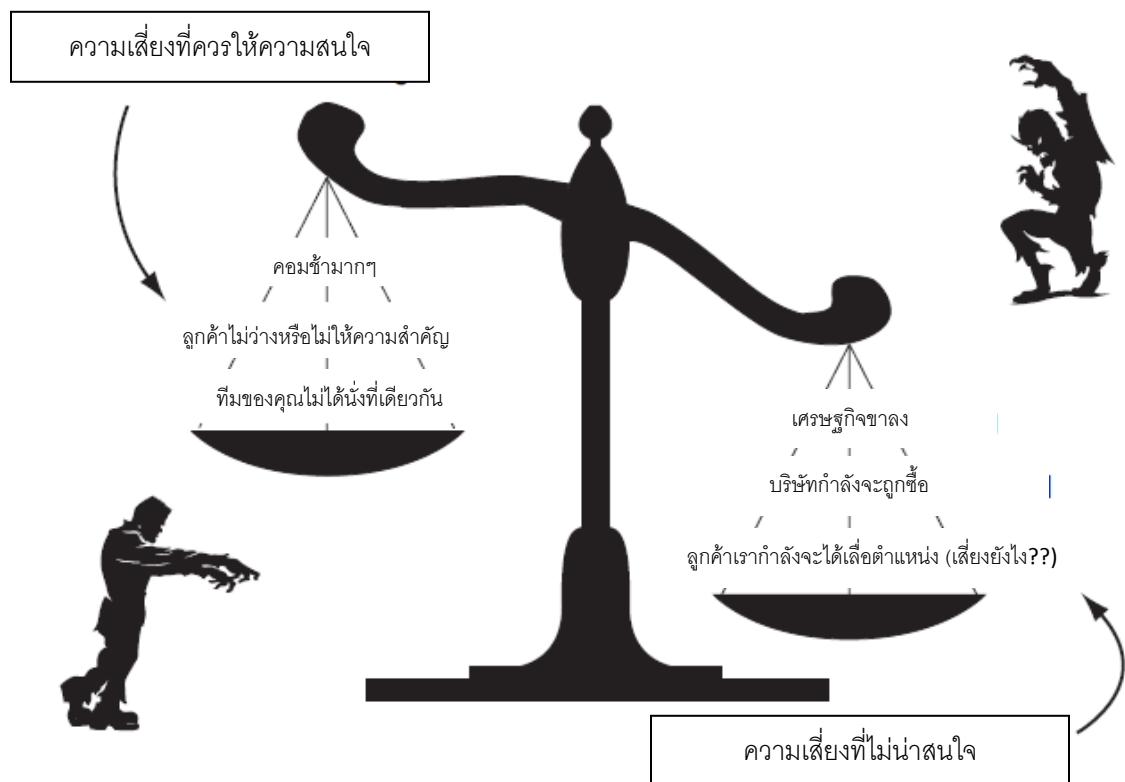
- แนะนำคร่าวว่าเป็นการเปิดประเดิมว่าเราจะทำมีเรื่องอะไรที่อาจจะเป็นปัญหาเสียแต่เงินๆ
  - เวลาที่เหมาะสมที่สุดที่จะพูดเรื่องนี้ก็คือตอนนี้แหล่งศตวรรษที่เริ่ม โปรเจค เพราะว่าถ้าปัญหามันเกิดไปแล้วมันก็สายไปล่ะ (ระเบิดเคราจะไปถูกต้องมั่นจะเปิดแล้วล่ะครับ ตอนนั้นเค้าเรียกว่าเก็บซากแล้ว) ดังนั้นถ้าคุณมีปัญหาหรือมีอะไรในใจที่คิดว่าจะเป็นสิ่งที่ทำให้ โปรเจค มันล้มลังกา พูดมันออกมามาเลย
- มันเป็นเวลาที่ทำให้เราได้บวิหารการคิดนอกกรอบ
  - ถ้าคุณได้เห็นความคิดที่หลากหลายในช่วงที่ทำ Inception Deck แล้วลังกา แนะนำว่านี่คือโอกาสที่คุณจะลอง บริหารสมองคุณในแบบเดียวกันเลย

- มันก็แค่แบบว่ารู้สึกตื่นเต้น (ที่จะทำ)

- มันเป็นความรู้สึกดีๆ ที่ได้บอกเล่าความกลัวและความกังวลของเราว่าให้กับคนอื่น (ลงใจในระดับนึง) มันเป็นโอกาสที่ทำให้ทีมมีความผูกพันและความเข้าอกเข้าใจกัน เป็นสิ่งดีๆ ที่จะได้เรียนรู้จากกันผ่านประสบการณ์ของผู้อื่น

อย่าลืมว่าคือตอนเริ่มต้นของ Project และมีสิ่งต่างๆ มากมายที่เราต้องทำเรียบร้อยแล้วก่อนที่จะทำการเข้าใจกับประเด็นสำคัญต่างๆ ที่เกี่ยวข้องกับ Project นี้ นำมันขึ้นมาบอกร่วมกันบอกร่องกันให้ทุกคนเข้าใจตรงกัน ระบบปัญหาที่สำคัญ(มากพอ) ที่จะให้ความสำคัญ

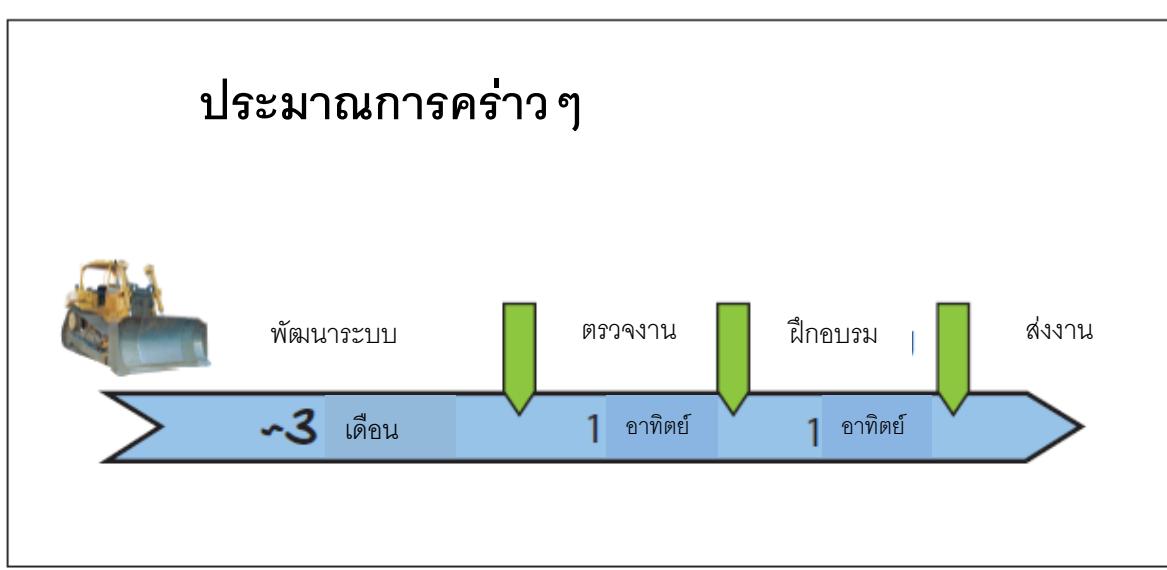
สิ่งที่สามารถทำได้่ายานเพียงแค่ว่าบ่นปรึกษากันภายในทีม (รวมถึงลูกค้าด้วย) เพื่อร่วมความคิดว่าอะไรบ้างที่เป็นความเสี่ยง และปัญหาที่อาจจะเกิดขึ้นกับ Project นี้ได้ในอนาคต เพราะคุณนั้นแหล่เบรียบเสมือนแขนขาเป็นเครื่องมือเป็นแรงงานเป็นมันสมองให้กับลูกค้าเพื่อให้งานของเด็กสาวลุล่วงไปได้ และอะไรก็ตามที่มันจะทำให้คุณไม่สามารถทำงานได้หรือทำงานได้ช้าลง คุณควรบอกให้ลูกค้าของคุณให้ได้ทราบถึงสิ่งเหล่านั้นเหมือนกันนะครับ  
อย่างแรกที่เราควรจะเริ่มต้นคือเอาริสิ่งที่เราได้เรียนรู้แล้วแล้วกว่าอะไรคือความเสี่ยงที่อาจจะก่อให้เกิดปัญหาเหล่านั้นมาจัดหมวดหมู่ใหม่โดยแบ่งเป็นสองประเภทคือ สิ่งที่คุ้มค่าพอที่จะให้ความสนใจ กับสิ่งที่ไม่ควรสนใจ



ลองมาดูตัวอย่างกันสักเรื่องนึงอย่างเช่นเรื่องเศรษฐกิจชาลุง แน่นอนว่ามีความเป็นไปได้ว่าเศรษฐกิจอาจจะดึงลงเหวแล้วเจาทั้งหมดอาจจะถึงขั้นตกงาน แต่ว่าสิ่งเหล่านี้เป็นสิ่งที่เราเก็ททำอะไรไม่ได้ ดังนั้นเราไม่ควรไปเดียวกันให้ความสำคัญ อีกด้วยอย่างนึงก็อย่างเช่นว่าหัวหน้าทีม Programmer ที่มีความสามารถ(มาก)ของเราว่าจะลาออกจากซึ่งเป็นธรรมดาก็ของอุตสาหกรรม IT ในปัจจุบัน สิ่งที่เราอาจจะพอทำได้ก็คือว่าพยายามให้มันใจได้ว่าทีมมีส่วนร่วมและร่วมมือในการถ่ายทอดความรู้ให้กันและกันเพื่อให้ความรู้ของเด็กที่เด็กมีปัญหามีอยู่กับทีมหรือองค์กร และเพื่อให้แนใจว่าไม่มีใครเป็นผู้ชำนาญการพิเศษในเรื่องใดเรื่องหนึ่งมากจนเกินไป (แบบประมาณว่าขาดเด็กแล้วบริษัทเวลาล่มสลาย) และก็อาจจะมีบางเวลาที่คุณรู้สึกเหนื่อยล้าหมัดแรงสิ้นหวังไว้กำลังในขณะที่จำเป็นต้องคิดว่าความเสี่ยงขันไหนหรือปัญหาอันไหนเป็นสิ่งสำคัญ อย่างน้อยมันก็ยังมีด้านดีๆให้มองเพื่อให้กำลังใจตัวเองนะครับ

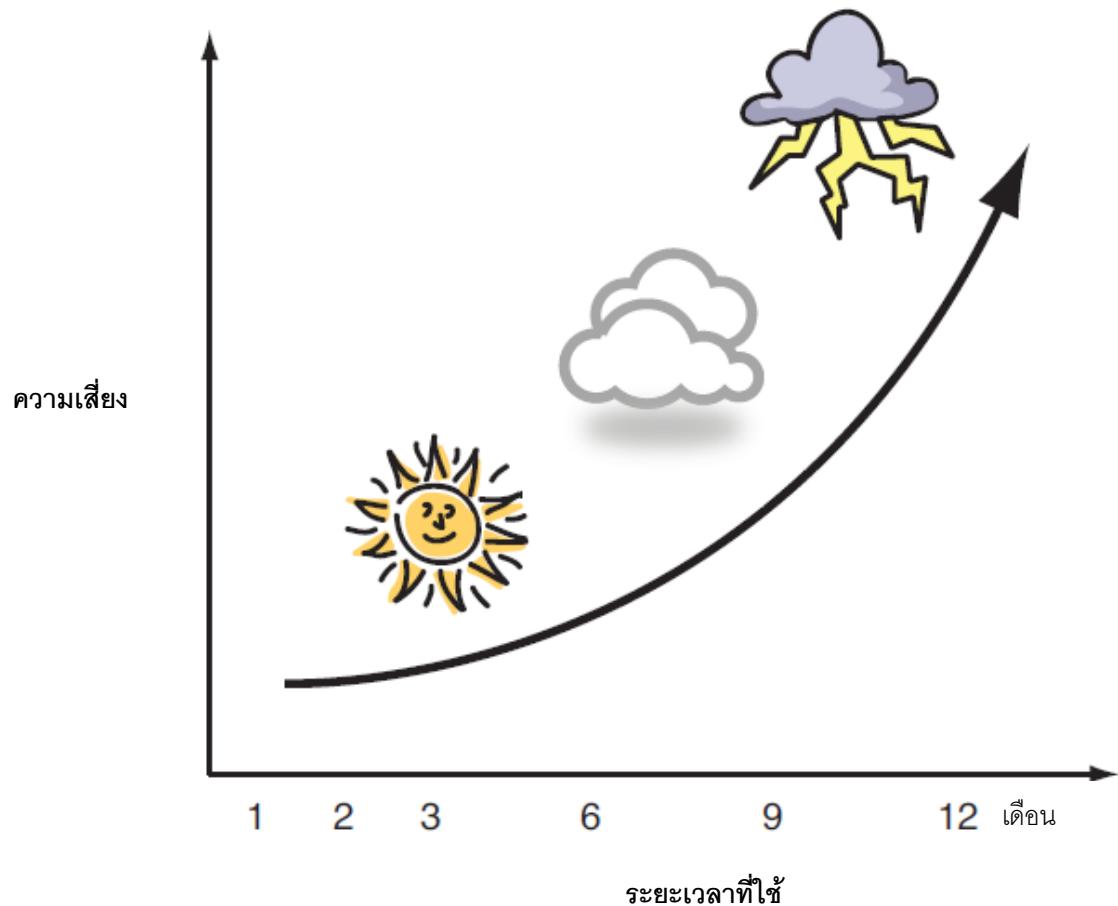


## 5.3 ประเมินและประมาณการ



แน่นอนว่าในขั้นตอนนี้คือการประมาณการว่าสิ่งที่เรากำลังจะทำมันคือ Project แบบ 1, 3, หรือ 6 เดือน เราคงไม่ได้คาดหวังว่าจะมีอะไรที่เปลี่ยนไปกวนใจหรอกนะในตอนนี้ แต่อย่างน้อยเราก็ควรจะสามารถตอบกลุ่มค้าของเราราได้ว่าตอนนี้เวลาที่เค้าจำเป็นต้องรอโน้มันประมาณเท่าไหร่

สำหรับเรื่องการประมาณนี้จะเจาะลึกในบทที่ 7 “Estimation” ศิลปะในการเดา แต่สำหรับตอนนี้ก็ขอสมมติว่าเราได้ผ่านขั้นตอนการประมาณไปแล้วและขอสมมติว่ามีผลลัพธ์อยู่ในเมื่อที่พร้อมใช้เป็นตัวอย่างได้เลย ก่อนที่จะไปตรงนั้นก่อนอื่นลองมาดูความสำคัญของการคิดในกรอบ(ขนาด)เล็กๆอย่าง่ายๆก่อนเลย



รูป 5.1 ความเสี่ยงของ Project จะเพิ่มขึ้นเมื่อระยะเวลาที่จะต้องใช้ในการทำ Project มากขึ้น

### คิดเล็ก

ชื่อของ Randy Mott อาจจะฟังดูไม่คุ้นหูหรือไม่เป็นที่รู้จักเลย แต่เค้าเป็นถึงดาวเด่นใน Fortune500 (ที่ได้ต้นในเรื่องการจัดอันดับบุคลากรบริษัทที่ร่วมราย...อย่างเป็นกิจการเป็นหมื่นคนต่อเดือน 1900) Randy เป็นคนช่วยสร้างระบบ Data Warehouse/inventory (พวกรากจักรการสินค้าคงคลัง) ให้กับ Walmart (ร้าน Mega Store ที่ใหญ่ยักษ์อันดับหนึ่ง) ซึ่ง

สามารถทำให้ผู้จัดการร้านค้าสามารถติดได้ในทันทีว่า ณ ขณะนี้มีสินค้าขั้นมุ่ง(อย่างเช่นเดย์)รสใหม่ขายดีที่สุด แนะนำของอย่างนี้เค้าไม่ได้ทำให้ที่เดียว เค้าก็ยังทำระบบอย่างเดียวกันให้ที่ Dell ซึ่งทำให้ Dell รู้ว่ามีสินค้าอะไรเหลือบ้างที่ยังค้างอยู่และสามารถนำไปลดราคาขายเพื่อระบายนิสต์ได้ทันท่วงที และในขณะนี้ที่เดาเป็น CIO ของ HP เค้ากำลังดำเนินการปรับสร้างแบบรีระบบใหม่หมด (ประมาณการว่าพันล้านเหรียญ!!!) เพื่อทำให้ระบบการจัดการคงคลังของ HP ดีขึ้น

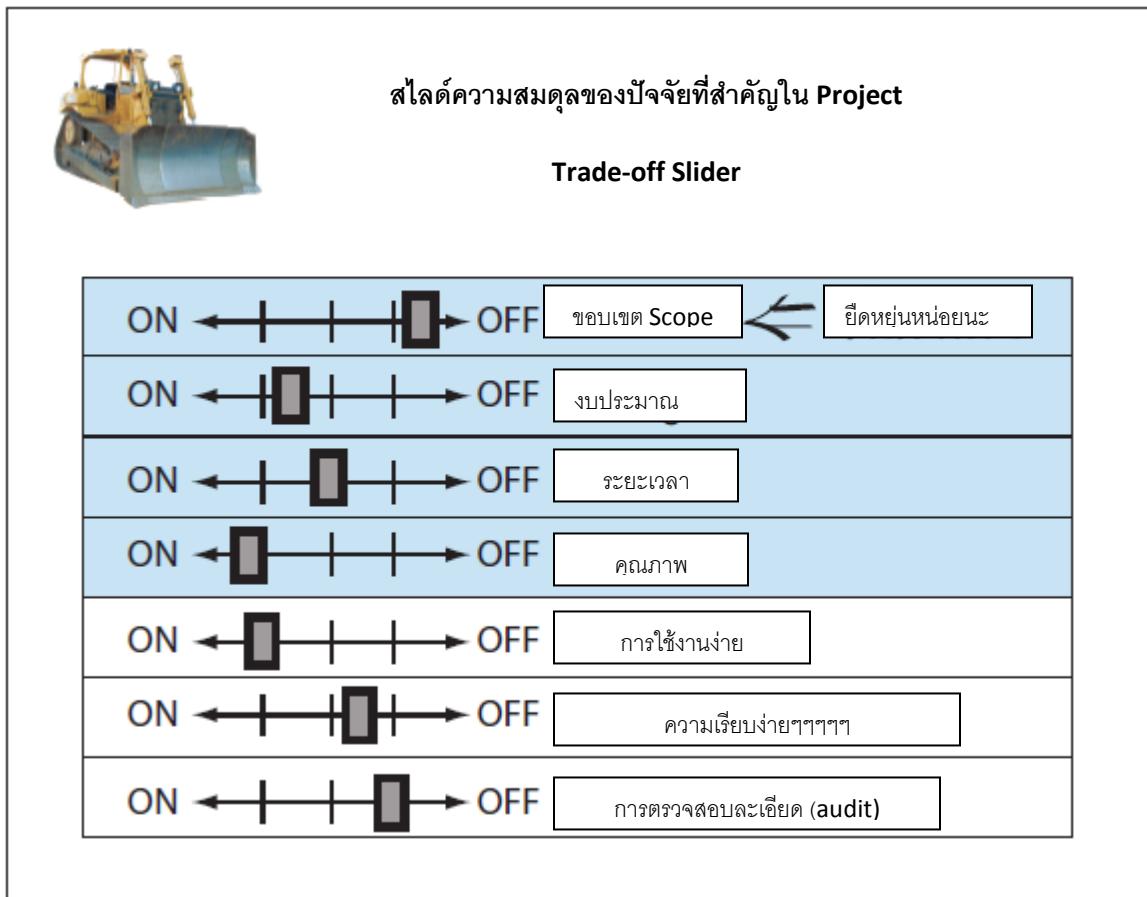
มันชัดเจนมากว่า Randy ต้องทำอะไรได้ถูก(ที่ถูกทาง)อย่างมากให้กับ Walmart, Dell และ HP แต่สิ่งหนึ่งที่ Randy มักจะบอกอยู่เสมอสำหรับกฎติกาหมายที่เค้ายื่นถือคือว่าเค้าจะไม่ยอมให้การพัฒนาของเค้ามันกินเวลามากเกิน 6 เดือน (รูป 5.1) ปัญหาอย่างยิ่งสำหรับ Project ที่มีขนาดใหญ่ปลายเปิดไว้กำหนดระยะเวลาที่เป็นจริงคือมันเหมือนกับว่าโปรเจคจะเดินไปช้าๆ กาลนานไม่มีวันจบสิ้นและป่วยครั้งที่ Project แบบนี้จะให้ความหวังกับลูกค้าว่าสิ่งที่เค้าจะได้มันมาก(เกิน)ในขณะนั้นที่ส่งให้ลูกค้าจริงๆ มักจะน้อย(เกิน) อย่างน้อยก็น้อยกว่าที่ทำให้ลูกค้าคาดหวังไว้แต่ต้น)

เวลาที่ถือว่าเป็นกรอบที่ดีที่สุดของ Randy ในการทำ IT Project นั้นคือไม่ควรเกิน 6 เดือน (อันนี้คือของ Randy นะครับ) อะไรที่นานกว่านั้นเค้ามองว่ามีความเสี่ยงมาก(เกิน) แต่นั่นไม่ได้หมายความว่าการทำ IT Project ที่เค้าทำนั้นจะทำให้เสร็จภายใน 6 เดือนสำหรับทุกๆ Project แต่อย่างน้อย Randy ได้เรียนรู้ว่ามันเป็นเรื่องหนักหนาสาหัสและเสียเวลามากสำหรับ Project ขนาดใหญ่ในการที่จะให้ได้ข้อสรุปว่าอะไรบ้างที่เค้าต้องส่งให้ลูกค้า (หรือว่าลูกค้าคาดหวังอะไรบ้าง) ในอีกทางหนึ่งก็คือเค้าจะแบ่งงานใหญ่ๆ ให้ลูกค้าทำในช่วงเวลาที่สำคัญที่สุด เช่น การทดสอบ(UAT), การฝึกอบรม(training), หรืออื่นๆ ที่มีความสำคัญก่อนที่ระบบเหล่านั้นจะถูกนำเข้ามาใช้งานจริง แต่หลักๆ ที่สำคัญที่สุดแล้วก็คือเพื่อบอกให้กับลูกค้าหรือผู้มีส่วนร่วมใน Project ทุกท่านเพื่อให้เข้าใจตรงกันว่าขนาดของ Project มีขนาดใหญ่ขนาดไหนแล้วมันจะใช้เวลาประมาณเท่าไหร่

แนะนำของ Project จริงๆแล้วก็คือการทำการประเมินและสามารถบอกกับลูกค้าและผู้มีส่วนร่วมใน Project แม้จะเป็นเพียงแผนงานคร่าวๆ ให้เข้าใจได้ แนะนำว่าคุณก็ต้องคำนึงถึงตัวแปรที่เป็นปัจจัยหลักๆ ของ Project ด้วย เช่นการทำ UAT (User Acceptance Test), training, หรืออื่นๆ ที่มีความสำคัญก่อนที่ระบบเหล่านั้นจะถูกนำเข้ามาใช้งานจริง แต่หลักๆ ที่สำคัญที่สุดแล้วก็คือเพื่อบอกให้กับลูกค้าหรือผู้มีส่วนร่วมใน Project ทุกท่านเพื่อให้เข้าใจตรงกันว่าขนาดของ Project มีขนาดใหญ่ขนาดไหนแล้วมันจะใช้เวลาประมาณเท่าไหร่ แนะนำว่าคุณมีทางเลือกอยู่บ้างในการนำเสนอแผนงานของคุณ โดยสามารถทำได้ง่ายๆ เมื่อนำเสนอลงไปโดยว่าคุณจะส่งงานวันไหน หรืออีกทางเลือกหนึ่งก็คือคุณจะบอกว่า features หลักๆ ที่คุณจะส่งมอบให้ลูกค้าว่าเค้าจะได้อะไรแต่ว่าให้ยืดหยุ่นในส่วนของวันที่คุณจะส่ง เราจะเจาะลึกในเรื่องนี้ในบทที่ 8.4

โปรดระวังไว้นิดนึง ไม่ว่าจะไปก็ตามอย่าทำให้ใครๆ ขอบคุณ เค้าไปคิดเห็นว่ามีอะไรแอบแฝงที่เราแสดงให้ดูในขณะนี้ถือว่าเป็นการทดลองผู้มั่นใจ ควรจะให้ทุกคนเข้าใจว่า นี่คือการประมาณการเพื่อให้เห็นภาพและเข้าใจตรงกันว่าเรากำลังจะทำอะไรใช้เวลานานขนาดไหน (ย้ำโดยประมาณ) แต่สำหรับค่าที่แม่นยำกว่านี้จะได้ก็ต่อเมื่อได้เริ่มงานและวัดผลแล้วนำมาสิ่งที่วัดนั้นมาประกอบเป็นข้อมูลปรับปรุงแผนงานอีกที

## 5.4 ยักษ์กันอึกท่าว่าอะไรบ้างที่ต้องยอมเสีย(เพื่อให้ได้อะไรมา)



แน่นอนว่ามันมีกฎและพลังผลักดันบางอย่างที่ทำให้เราต้องใส่ใจในเรื่องบางเรื่องอย่างมากใน Project แต่ละอัน ลองมาฟังสิ่งต่อไปนี้ซึ่งอาจจะฟังดูคุ้นหูบ้างครับ เงินที่มีให้ใช้อย่างจำกัดและกำหนดวันที่จะต้องส่งมอบดูเหมือนว่าจะคลาดเคลื่อนไม่ได้ ข้อบเขตของงานมักจะมีเพิ่มขึ้นตามความต้องการที่มากขึ้นแบบไม่ทันตั้งตัว แน่นอนครับคุณภาพต้องมาเป็นอันดับ 1 (เข้าวันนี่จะจ่ายราษฎร์ตามลักษณะของเงินเบนเนอร์กันเลยรึไงครับคุณลูกค้า)

ครับมันเป็นปกติโดยธรรมชาติของสิ่งที่เรียกว่า Project ที่ว่าสิ่งที่กล่าวมานั้นมันเหมือนจะขัดแย้งกันเอง(อย่างแรง) การให้น้ำหนักหรือความสำคัญในด้านนึงหมายถึงว่าต้องลดทอนความสำคัญของอีกอย่างลง(อย่างหลีกเลี่ยงไม่ได้) ถ้าเราทำให้สิ่งเหล่านี้อยู่ในภาวะสมดุลไม่ได้ แน่นอนปล่อยไว้นานๆ Project ก็คงล่มสถาปัตย์ไปในที่สุด ดังนั้นต้องให้ได้ข้อสรุปว่าอะไรบ้างที่เราต้องยอมเสีย...นั้นสิแล้วอะไรล่ะที่เราจะยอมเสียไป

ครัวบและอิกคั่ง Agile ก็มีทางออกให้คุณเพื่อทำให้แรงกดดันจากทั้งสิ่งทั้งปวงเหล่านี้มันบรรเทาเบาบางไป ผูกก็จะปล่อยให้เป็นหน้าที่ของอาจารย์ Sensei เพื่อสอนคุณละกันในเรื่องนี้

เราจะมาทำการศึกษาเพื่อทำความเข้าใจด้วยกันว่าในตัวแปรที่กล่าวมานี้อะไรบ้างที่เป็นปัจจัยสำคัญใน Project และอะไรบ้างเป็นสิ่งที่เราต้องเลกเพื่อให้ได้มาซึ่งปัจจัยสำคัญนั้น แล้วเราจะใช้พลังของปัจจัย(ที่สำคัญ)เหล่านั้นมาใช้ใน Project ได้ยังไง



1. ในสิ่งเหล่านี้ข้อใดเป็นแรงที่มีผลกดดันมากที่สุดใน Project ของเจ้า
  - a. คุณภาพ
  - b. เวลา
  - c. ขอบเขตของงานที่ต้องทำ
  - d. งบประมาณ
2. เมื่อเผชิญหน้ากับวิกฤติการทางด้านปัจมานงานที่ต้องทำ (มันแยกจนทำไม่ทันกันนะ) อะไรคือทางออกที่ดีที่สุด
  - a. ลดขอบเขตของงานลง
  - b. เพิ่มคนเข้ามาทำงาน
  - c. ขยายขอบเขตเวลา
  - d. ลดคุณภาพงาน

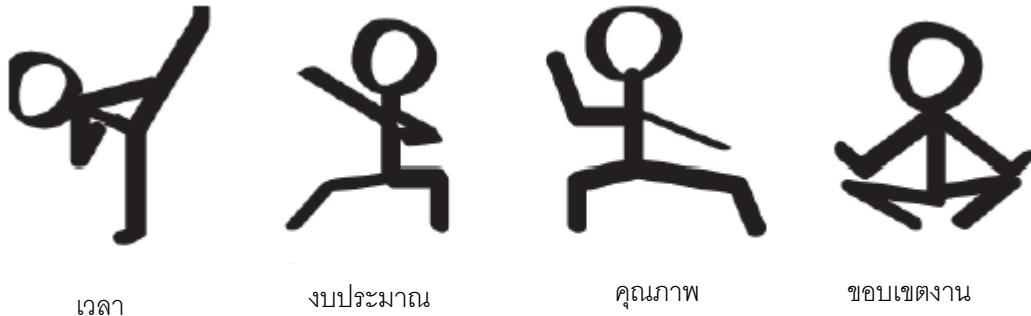
3. ข้อใดต่อไปนี้ทำให้คุณรู้สึกเจ็บปวดที่สุด

- a. เดินลุยฟ้า
- b. เดียวแก้วแตก
- c. เต้น Wonder Girls
- d. ขอเงินเพิ่มจากเจ้าของ Project

รู้สึกยังไงบ้างเมื่อต้องตอบคำถามเหล่านี้ ตอบได้หรือเปล่า หรือว่าคุณตอบได้ด้วย “ก็ขืนอยู่กับว่า...”

ในความเป็นจริงแล้วมันไม่มีคำตอบที่ถูกต้องหรือไม่ถูกต้องแบบตายตัวกับคำถามเหล่านี้ เพราะคำถามเหล่านี้แค่เพื่อแสดงให้เห็นว่ามีปัจจัยหลักๆอยู่สามอย่างที่เป็นแรงผลักดันอยู่ใน Project และการที่จะต้องสร้างสมดุลให้กับปัจจัยทั้งสามนี้ คงต้องเหนื่อยกันหน่อย

นี่เป็นเวลาที่เหมาะสมที่สุดที่จะได้เรียนรู้ถึงพลังจากปัจจัยสามัญเหล่านี้ และแนวทางที่จะควบคุมพลังเหล่านี้ให้สมดุลได้ เรามาดูรู้จักกับ คนเจ้าอารมณ์สี่คนนี้เลยดีกว่า (หรือคนเจ้าอารมณ์ทั้ง 4)



เวลา

งบประมาณ

คุณภาพ

ขอบเขตงาน

เมื่อทุกสิ่งทุกอย่างเริ่มต้น Projects จะถูกปักครื่องภายใต้กลุ่มคนสี่คน(ที่เจ้าอารมณ์นำดู)ซึ่งประสานงานร่วมกัน ซึ่งรู้จักกันในนามคนเจ้าอารมณ์ทั้ง 4 – เวลา, งบประมาณ, คุณภาพ, และ ขอบเขตของงาน แต่แน่นอนว่าเอกลักษณ์ระบบการปกครองโดยทั่วไป บอยครั้งที่คนทั้งสี่นี้มีภารผิดชอบทางอารมณ์และทำให้เกิดความไม่สงบร้อยกัน เพราะความเจ้าอารมณ์ของเด็กทำให้เกิดผลเสียกับ Project ของเรา

- บางทีเวลาบีบเราให้ยิ่งแย่ยิ่งขึ้นจนทำงานล้ม躉กไปเลย
- งบประมาณโ DIN มตัด
- Bugs เพิ่มขึ้นทุกวัน
- มีงานมากไปที่ต้องทำ

อีกเช่นเคยตามแบบคนเจ้าอารมณ์คนทั้งสี่ยอมดูร้ายเป็นธรรมดា แต่ว่าเราก็มีทางที่จะทำให้เขาเหล่านั้นสงบได้ ต่อไปนี้จะเป็นวิธีการที่จะทำให้พวกเด็กทั้งสี่ทำงานอย่างสอดประสานกัน

## เวลา

เวลาเป็นสิ่งที่มีจำกัด (แต่บ่อยครั้งผลก็ล้มไปเมื่อกัน) เราไม่สามารถสร้างเวลาหรือเก็บเวลาไว้ใช้งานได้ เราต้องใช้มันอย่างดีที่สุดเท่าที่เรามีและนี่เป็นเหตุผลที่ทำให้ทำในกรอบ Agile ทั้งหลายถึงชื่อของการวางแผนแบบ Time-Boxing (เป็นเทคนิคที่ใช้ช่วยในการบริหารเวลาอย่างนึง ซึ่งจะแบ่งเวลาออกเป็นช่วงๆอย่างแล้วทำงานภายในช่วงเวลาหนึ่ง...ไม่ว่าจะเสร็จหรือไม่เสร็จก็ตาม แทนที่จะทำงานนานนั้นๆ) นักรบ Agile รู้ดีว่าการต้องดึงที่จะขอเลื่อนการส่งงานออกไปนั้นเป็นการลดมูลค่าที่ลูกค้าของเราจะได้บันสิ่งที่เค้าได้ลงทุนไปและแน่นอนว่าังเป็นการเพิ่มความเสี่ยงที่จะไม่มีอะไรจะส่งให้ลูกค้าอีกด้วย (ยิ่งสายยิ่งไม่ได้งาน) ซึ่งเป็นโชคชะตาที่ย่ำแย่ที่สุดที่จะเป็นไปได้ของ Software Project (เบรียบเสมือนตายสนิท)

และนี่เป็นเหตุผลที่ว่าทำในกรอบ Agile ของเราระมองว่าเวลามันตายตัว

## งบประมาณ

งบประมาณเป็นเหมือนฝ่าแฝดของเวลา เพราะว่ามันไม่ยึดหยุ่นมีจำนวนจำกัด และไม่ได้มีให้ใช้อย่างฟุ่มเฟือย สำหรับลูกค้าของเราแล้ว มันเป็นเรื่องยากและน่าลำบากใจมากถ้าเดาต้องกลับไปหาผู้ให้การสนับสนุนทางด้านเงินทุนเพื่อขอเงินเพิ่ม อาจจะเห็นว่าเกิดขึ้นได้ในบางครั้ง แต่เชื่อเถอะมันไม่ได้เป็นประสบการณ์ที่ดีหากเราเพิ่งประสบการณ์ใหม่กว่าเดิม นักรบ Agile ปฏิบัติต่องบประมาณเช่นเดียวกันกับเวลาตายตัว

## คุณภาพ

มีบางคนที่มีความเชื่อที่ผิดว่าคุณภาพของงานสามารถถูกสังเคราะห์ให้ได้มาซึ่งเวลาที่เร็วขึ้น (ผิดมหันต์) การให้ได้มาซึ่งความเร็วในระยะสั้นโดยต้องแลกับคุณภาพนั้นเป็นความเห็นผิดอย่างร้ายแรงการลดคุณภาพลงก็เปรียบเหมือนกับการโยนมีด(ใหญ่)มากๆประมาณมีดสปาต้าล้มมัง) ที่ติดไฟเพื่อเล่นกอล์ฟกันที่มีหิมะตกหนัก เรายาจะทำให้มีเวลาอุ่นได้ในบางครั้งคราว แต่ไม่นานจากนั้nmันอาจจะบาดมือเราหรือไฟมือเราอย่างสาหัสและคุณภาพเป็นอีกอย่างที่ต้องพยายามไม่มีการละเลย

## ขอบเขตของงาน

สำหรับเวลา งบประมาณ และคุณภาพที่ไม่มีการลดหย่อนแล้ว นักรบ Agile ของเราก็เหลือแค่อย่างเดียวที่ทำให้มันยึดหยุ่นได้ – ขอบเขตของงาน ถ้ามันมีงานมากไปให้ทำ นักรบของเรายังทำน้อยกว่าที่มี ถ้าในความเป็นจริงมันไม่ตรงกับแผน นักรบของเราก็จะเปลี่ยนแผนนี้เป็นสิ่งที่ทำให้นักเรียนหลายคนรู้สึกไม่สบายตัวเลย หลายคนมาที่สำนักนี้โดยมีความรู้ติดตัวมา(เพราะลูกสอนมา)ว่าแผนคือสิ่งที่เปลี่ยนไม่ได้ ไม่ยึดหยุ่นไม่เปลี่ยนแปลง แต่ว่าไม่มีอะไรหนีจากความเป็นจริงได้หาก

วันเวลาอาจจะไม่ยึดหยุ่น แต่ไม่ใช่สำหรับแผนการก็ เพราะว่าพลังจากคนเจ้าอารมณ์ทั้งสี่ที่นักรบ Agile ต้องรับมือ เค้าจะทำให้เวลา งบประมาณ และคุณภาพตายตัวไม่เปลี่ยนแปลง แต่จะไปให้ความยึดหยุ่นกับขอบเขตของงานแทน เมื่อมานถึงตรงนี้ คุณก็พร้อมแล้วที่จะทำแบบฝึกหัดกับลูกค้าเพื่อให้ได้มาซึ่ง Trade-Off Slider หรือตารางความสมดุลที่แสดงให้ดูในข้างต้น

### การอบรม กับ การส่งงาน (Training VS Delivery)

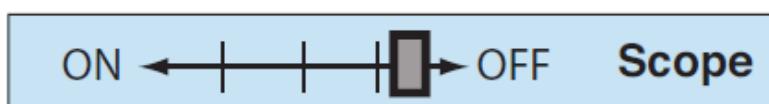
การอบรมกับการส่งงาน เป็นสิ่งที่เราต้องทำให้สมดุลอย่างมาก เชื่อหรือไม่ว่าที่ Thoughtworks (ซึ่งเป็นบริษัทของผู้เขียนเอง) แม้ว่าเค้าไม่ได้มองว่า Thoughtworks เป็นบริษัทที่ให้บริการทางด้านการฝึกอบรม แต่ว่าอย่างน้อยนี่เป็นจุดขายจุดเดียวที่ทีมขายของเราเอาไปใช้ได้เสมอ และเราก็ได้รับการต้อนรับจากลูกค้าด้วยเหตุผลเรื่องการฝึกอบรมอยู่ปอยครั้ง

แต่ยังไงก็ตาม การฝึกอบรมและการส่งงานมันเป็นสิ่งที่แยกออกจากกัน

โดยการใช้ slider board และถามคำถามง่ายๆเพื่อให้ลูกค้าของคุณได้จัดลำดับความสำคัญของปัจจัยสองอย่างนี้นั้น คุณสามารถจัดลำดับกับลูกค้าของคุณว่าสิ่งไหนสำคัญกว่าแล้วดำเนินการต่อไปได้

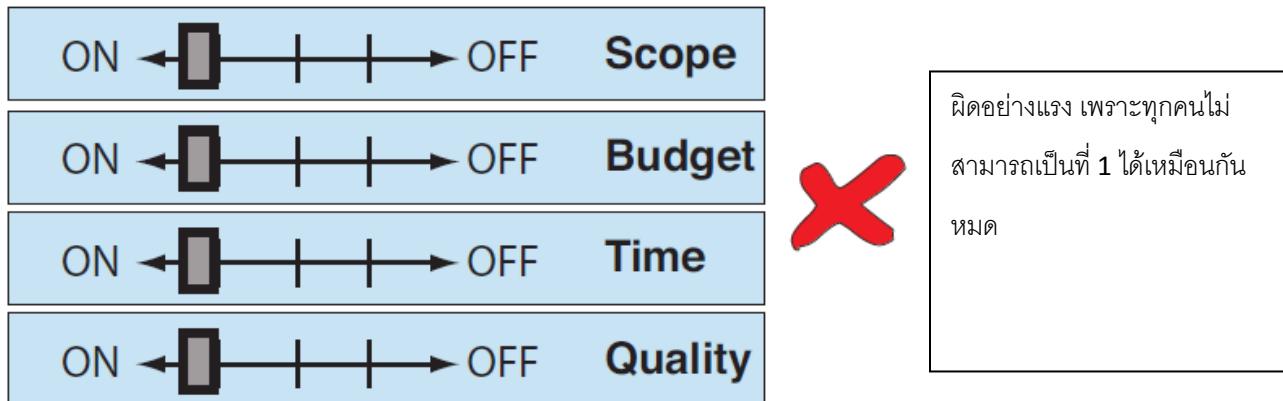
### สไลเดอร์ความสมดุล (Trade-off Slider)

สไลเดอร์ความสมดุลเป็นเครื่องมืออย่างดีที่มีนานานกาล โดยนักบุญ Agile ของเราสามารถใช้ในการระดมความคิดเพื่อหาข้อสรุป ถึงผลกระทบที่อาจจะเกิดขึ้นของพลังทั้ง 4 นักรบของเรามีความจำเป็นอย่างยิ่งที่จะต้องทำความเข้าใจอย่างลึกซึ้งว่าลูกค้าของเรานั้นให้ความสำคัญกับพลังทั้งสี่อย่างไรบ้าง และในทันของเดียวกันนักรบของเราก็ต้องการที่จะสร้างข้อกำหนดให้เข้าใจตรงกัน ถึงความสำคัญของความยืดหยุ่นในเรื่องของขอบเขตของงาน ความยืดหยุ่นทำให้เราไม่ยึดติดจนเกินไปว่าจะต้องทำงานแบบให้ได้ทุกอย่าง แบบว่าเขากูกๆ Features (User Stories) มาใส่เป็น To do List (Master Stories)

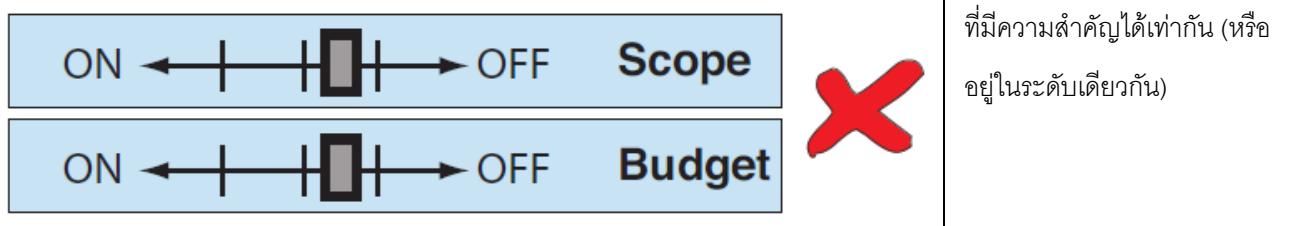


ให้แน่ใจว่าลูกค้าเข้าใจว่าเรา  
กำลังให้ความสำคัญกับความ  
ยืดหยุ่นบนขอบเขตของงาน  
(อาจจะมีบาง Features ที่เรา  
จะไม่ทำ ตามเวลาที่กำหนด)

เราจะหยิบเอาพลังแต่ละพลังมาถกคุยกันแบบถกนั่งลงกับลูกค้าของเรา เอกมาใช้วรรแบบให้เห็นกันจะเป็นสไลเดอร์ว่าลูกค้า  
มองว่าพลังแต่ละอันนั้นสำคัญอย่างไร (บันมาตราฐานเดียวกัน เช่น จากค่า 1-5 ถึง 1-5 เหมือนกันหมด) แต่กูที่สำคัญอย่างแรก  
ของการใช้สไลเดอร์คือ ทุกๆพลังของคนทั้ง 4 ไม่สามารถมีความสำคัญในระดับเดียวกันได้ (นั่นหมายความว่าจะไม่มีอะไรที่เป็น  
เบอร์ 1 มากกว่าสองคน)



ลูกค้าส่วนใหญ่จะสามารถทำความเข้าใจได้ว่าบางสิ่งบางอย่างต้องยอมเสียอะไรบ้างเพื่อให้ได้มาซึ่ง Project ลูกค้าของเราอาจจะรู้สึกลังเลและเป็นกังวล แต่เราต้องพยายามใช้ชัดเจนว่าสิ่งที่อยู่บนมาตรฐานนี้ทุกๆสิ่งเป็นสิ่งสำคัญมากและจำเป็น หรือให้มองอีกแง่มุมว่าคือว่าถึงแม้คุณภาพอาจจะมีความสำคัญน้อยกว่าเวลาแต่ไม่ได้หมายความว่าเราจะไม่สนใจคุณภาพ เพียงแต่เราต้องการจะบอกว่าเราจะไม่ยอมให้งานเสร็จช้าโดยเด็ดขาดไม่ว่าอย่างไรก็ตาม ดังนั้นเวลาจะมีความสำคัญกว่าในแห่งมุมนี้



แม้ว่าบุคคลทั้ง 4 มีความสำคัญมาก (เพราะผลกระทบของความผิดปกติของเค้าค่อนข้างจะรุนแรงเหลือเกิน) แต่นั่นไม่ได้หมายความว่าเราจะไม่พัฒอนิ่นๆจากปัจจัยอื่น ต่อจากนี้จะขอยกตัวอย่างให้ดูได้เห็นได้เข้าใจ

### ตรงเวลาและไม่มีปัญหาเรื่องการเงินอาจจะยังไม่พอ

ลองพิจารณาเหตุการณ์ต่อไปนี้

- จะมีประโยชน์อะไรถ้าเกมส์ที่สร้างมันเล่นไม่สนุก
- บริการหาคู่ผ่านเว็บจะอยู่ยังไงถ้าไม่มีลูกค้าสมัครใช้งาน
- พัฒนาแอปพลิเคชันเว็บไซต์ Online เล่นเพลินแต่ไม่มีใครพังอยู่เลย

เพราะว่ามีปัจจัยที่อาจจะมองไม่เห็นที่ลูกค้าของเรามิได้บอก(อาจจะลืมบอกหรือเรามิได้เข้าใจเอง) จะเห็นได้ว่ามันมีผลที่เราต้องสร้างความสมดุลเหมือนกันในด้านอื่น(ข้อนี้อยู่กับลักษณะงาน) เพราะว่าปัจจัยอื่นนั้นไม่ได้มีความสำคัญที่ยิ่งใหญ่กว่ากันเลย

อย่าลืมพิจารณาปัจจัยที่เป็นนามธรรม(จับต้องได้ยาก)



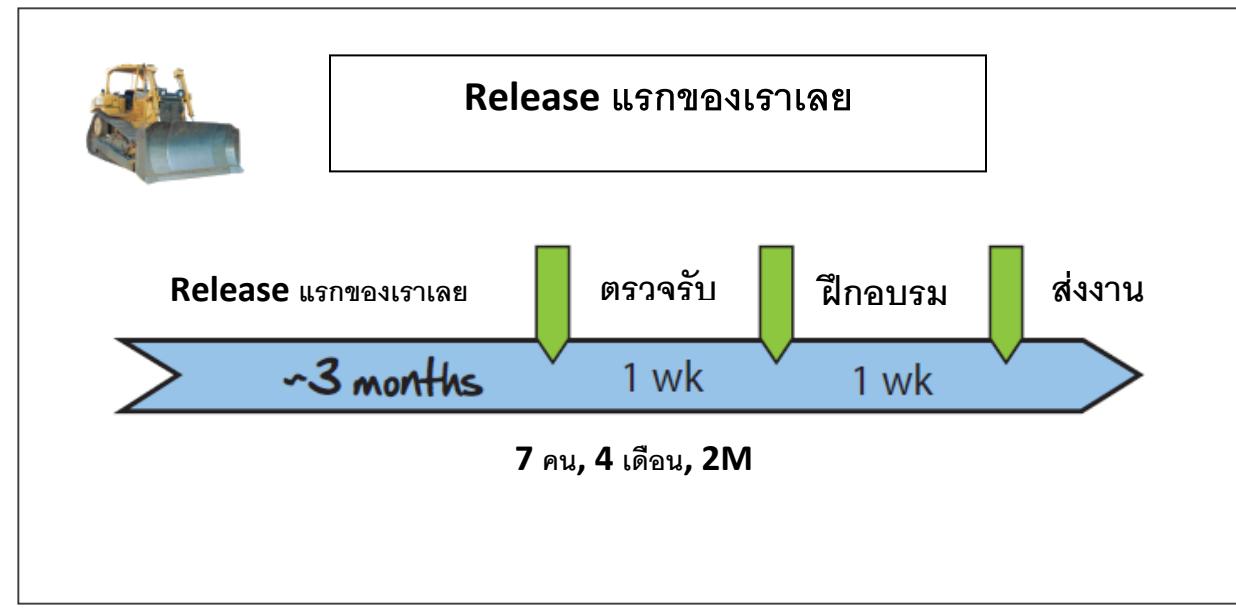
แม้เป็นอะไรที่จับต้องและมองเห็นได้ยากแต่รู้ว่าอาจจะเป็นสิ่งที่ทำให้ Project สำเร็จหรือล้มเหลวได้เลย

สิ่งเหล่านี้อาจจะเป็นสิ่งที่เราเริ่มได้กลิ่นเมื่อรู้สึกได้ในช่วงตอนทำ Inception Deck หรืออย่างน้อยก็ได้รับการบอเป็นนัยๆ จากลูกค้าของ我们在ช่วงของการเก็บ Requirement ที่เรียกว่า Your story-gathering workshops กับลูกค้าของคุณ ซึ่งเราจะมาดูกันในบทที่ 6.4

ในช่วงที่เราได้ทำการใช้ Sliders กับลูกค้าของเราอย่างน้อยเป็นสิ่งที่เราควรจำไว้เสมอว่าต้องเพื่อพื้นที่ด้านล่างตารางของเราเพื่อให้สำหรับปัจจัยที่จับต้องได้ยากที่จะมีผลต่อความสำเร็จ(หรือล้มเหลว)ของ Project ของเรา

เห้...ก่อนมาได้ด้วยดีถึงตรงนี้ เนี่ยมั้ย จากสิ่งที่เราได้เรียนรู้มาทั้งหมดต่อไปนี้ก็ได้เวลาที่จะรวมทุกๆ สิ่งทุกอย่างมาเสนอ ลูกค้าเราได้แล้วเพื่อแสดงให้เห็นว่าอะไรบ้างที่เรากำลังจะทำ (เรา = เรายากลับ - เพราะว่าลูกค้าคือหนึ่งในทีมของเรา)

## 5.5 มีอะไรบ้างที่จำเป็นต้องทำ (เพื่อให้สำเร็จ)



พอกมาถึงตรงนี้แล้ว

คุณก็ใกล้ความเป็นจริงขึ้นแล้วล่ะ!

อย่างน้อยก็มีวิสัยทัศน์!

อย่างน้อยก็มีแผน!

ขั้นตอนต่อจากนี้ก็มีเพียงแค่ต้องหาให้ได้ว่าอะไรบ้างที่เราต้องทำ แล้วมันมูลค่าเท่าไหร่

มาถึงขั้นนี้สิ่งที่คุณต้องทำการแนะนำให้คนที่เป็นผู้สนับสนุนทางด้านทุนให้กับ Project ได้เข้าใจว่า นี่จะคือทีมที่จะทำ นี่จะคือแผนของเรา และนี่จะคือมูลค่าของสิ่งที่เราจะทำ จันมาเริ่มกันที่ทีมเลยแล้วกัน

### ประกอบร่วมสร้างทีม

เมื่อมาถึงจุดนี้ได้แล้วอย่างน้อยเราก็ต้องมีความชัดเจนขึ้นมากแล้วว่าทีมแบบนี้ที่มีความจำเป็นสำหรับภารกิจนี้ ลองมาดูการจัดทีมตัวอย่างสักหน่อย

#	สมาชิก(ตำแหน่ง)	มีความรู้ความสามารถในด้านไหน
1	UX Designer	สามารถพัฒนา Prototype อย่างเร็ว (Rapid Prototypes), Wireframe and mockup (ตัวอย่างหน้าตาการใช้งานอย่างง่ายที่ทำได้รวดเร็ว), user flows (นำจะคล้ายๆ Role Playing บอกว่าคนกำลังต้องการทำอะไรแล้วจะทำสำเร็จด้วยการทำยังไงระบบที่เราสร้าง) และ ถ้าเป็น HTML/CSS จะดีมากๆ
1	Project Manager	สามารถรับมือกับความไม่ชัดเจนและลูกล้อเครื่อได้ สามารถทำงานได้แม่ไม่ต้องถูกสั่งงาน
3	Developers	C#, ASP.NET, มีประสบการณ์กับ MVC Unit testing, TDD, refactoring, continuous integration
1	Analyst	สามารถทำการวิเคราะห์ในแบบ Just-in-Time (Agile ไม่ได้แปลว่าไม่ต้องทำ Requirement แต่ว่าทำให้เพียงพอที่จะนำไป Develop ได้ใน Iteration นั้น) มีความคุ้นเคยกับ XP Story Card (เทคนิคในการคุยกับ On Site Customer และ Prioritize ว่าสิ่งใดที่ควรทำ) สามารถช่วยงานการ Test ได้
1	Customer	มีเวลาเพื่อตอบคำถามอย่างน้อย 1 ชั่วโมงต่อวัน สามารถประชุมร่วมกันได้อย่างน้อยอาทิตย์ละครั้งเพื่อให้ได้สามารถแนะนำ ซึ่งทาง และตัดสินใจในเรื่องที่เกี่ยวกับ Project ได้
1	Tester	มีประสบการณ์ในการทดสอบแบบอัตโนมัติ สามารถทำงานได้กับนักพัฒนาระบบและลูกค้า มีความเชี่ยวชาญในเทคนิค exploratory testing (มันคือการ test ที่มีระบบจะเบี่ยงไม่ลูกทุ่งแบบ script testing นะครับ)

และนี่เป็นเวลาที่เหมาะสมที่สุดที่จะพูดถึงเรื่องหน้าที่และความรับผิดชอบของคนในทีม (จากที่เคยกล่าวมาแล้วในบทที่ 2) และมายกยันว่าแต่ละคนถูกคาดหวังว่าจะต้องทำอะไรได้บ้าง มีตำแหน่งนึงในทีมที่ควรจะได้รับการใส่ใจเป็นพิเศษสักเล็กน้อยนั่นก็คือ ลูกค้า เพราะแ่นอนว่าไม่ใช่แค่ร่วมตำแหน่งนี้สำคัญมาก (เพราะว่าสำคัญมาก ๆ ๆ ๆ ๆ) ตำแหน่งนี้ส่วนใหญ่ไม่ได้มีในองค์กรของลูกค้าหรอก เราต้องให้มั่นใจได้ว่าลูกค้าเข้าใจจริงในการที่จะต้องมารับตำแหน่งลูกค้าใน Agile Project แบบนี้ เพราะว่า Project มีสิ่งเหล่านี้ซึ่งเป็นสิ่งที่เราต้องการจากลูกค้า

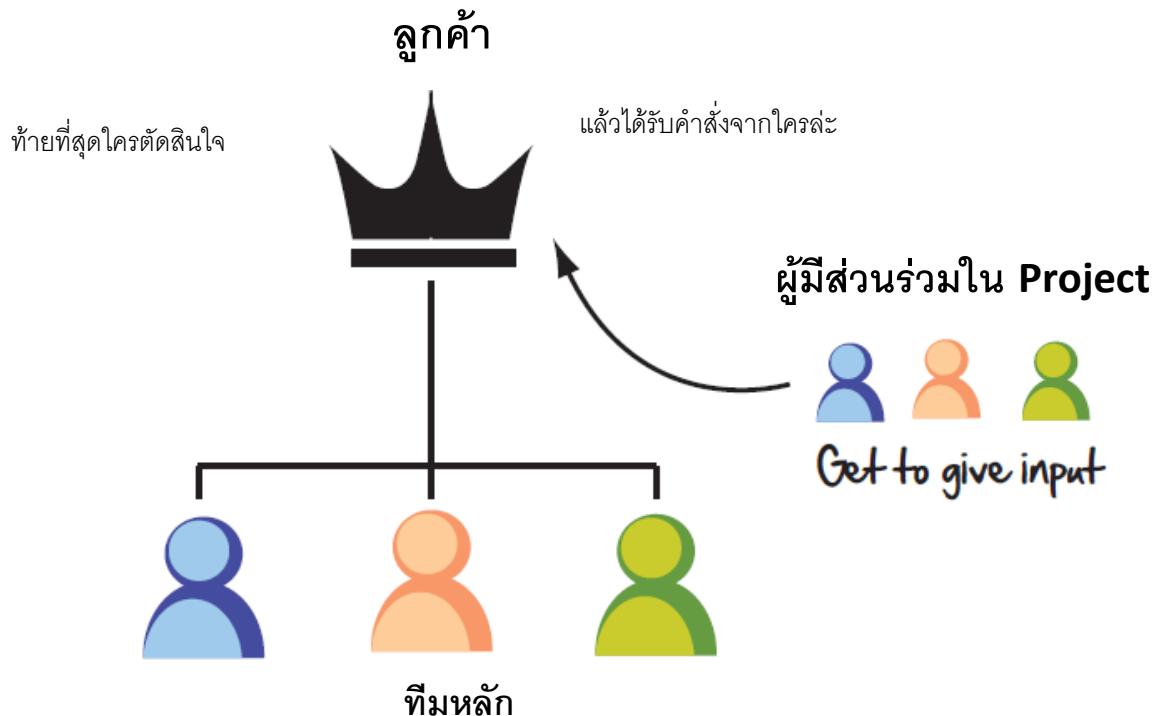
เวลา – ลูกค้ามีเวลาตามที่ตกลงกันไว้หรือไม่?

ลูกค้าสามารถชี้นำบอกแนวทางการพัฒนาของ Project นี้หรือไม่?

นักพัฒนาระบบ นักทดสอบ และนักวิเคราะห์สามารถรับมือกับหน้าที่ใหม่ๆ ของเด็กได้อย่างสบายๆ แต่ส่วนใหญ่แล้วลูกค้า Agile จะค่อนข้างใหม่ที่จะปรับตัว ดังนั้นอาจจะต้องเน้นกันเป็นพิเศษหน่อย อีกอย่างหนึ่งที่สำคัญมากก็คือเรื่องความชัดเจนในกรณีที่มีผู้มีส่วนร่วมใน Project อยู่เยอคุ เรายังคงต้องสรุปให้ได้สูดท้ายแล้วโครงการตัดสินใจ

### การให้ได้มาซึ่งผู้ตัดสิน(ใจ)

ไม่มีอะไรที่ดูจะสนับสนุนไปกว่าการที่ทีมไม่สามารถตัดสินใจได้ว่าทีมจะต้องรับคำสั่งจากใครว่าเราทำลังจะทำอะไรและเดินไปทางไหนลงมาฟังเรื่องแบบนี้(อาจจะพอกุ้น) เมื่อผู้อำนวยการทางด้าน IT ต้องการทดสอบเทคโนโลยี ในขณะที่รองประธานฝ่ายการกลยุทธ์ต้องการความรวดเร็วในการนำเสนอสิ่งที่ต้องการสู่ตลาด เดียวกัน...แต่รองประธานฝ่ายขายบอกว่าฟังออกช้ามากว่าบริษัทเราจะขอกรุณาระบุในต่อมาสหน้านี้เอง



มันเป็นไปไม่ได้เลยที่จะมีคนที่มีส่วนรวมในโครงการหลายคนที่จะพยายามบอกทีมว่าอย่างไรให้ทีมทำอะไร หรือควรเดินไปทางไหน อะไรมีความสำคัญและควรจะต้องทำอะไรต่อจากนี้

ในความเป็นจริง ถึงที่ต้องทำให้เข้าใจกันแต่แรกเลยก็คือว่า ใครเป็นคนนั้นที่จะตัดสินใจว่าจะพาทีมไปทางไหน ไม่ได้มายความว่าผู้มีส่วนร่วมคนอื่นๆจะไม่มีสิทธิ์พูดอะไรเลย แต่ว่าในทางกลับกันเราต้องการให้ลูกค้ามาพูดกับทีมด้วยความเป็นเสียงเดียวกัน และสอดคล้องกัน

เรื่องนี้เป็นสิ่งสำคัญที่ต้องพูดถึง ให้ขาดเดนตั้งแต่นั้น เพราะว่าจะทำให้ Project ของเรามีเมตัองแข็งกับความตุนวยและค่าใช้จ่ายแสนแพงสำหรับการแก้ไขหรือทำใหม่ (effort ที่ลงไปสูญเปล่า)

ถึงแม้ว่าเราจะรู้อยู่แล้วว่า ใครเป็นคนมีอำนาจในการตัดสินใจ แต่ก็ต้องพูดเรื่องนี้อยู่ดี (ตั้งแต่ตอนนี้) ไม่เช่นware ว่าเพื่อกำจัดข้อสงสัยหรือคดุมเครื่อง แต่เป็นการให้ทุกคนเข้าใจตรงกันและยอมรับว่า สุดท้ายแล้วเรื่องจะต้องไปจบที่ใคร

จบเรื่องนี้แล้วมารู้กันต่อเรื่องเงินๆทองๆติดกัน

ได้เวลามาประเมินว่า Project นี้ราคาเท่าไหร่

คุณอาจจะไม่เคยต้องพูดถึงเรื่องเงินเลยใน Project เพราะว่างบประมาณอาจจะถูกกำหนดตามด้วยตัวอยู่แล้วว่ามีเงินเท่านี้และนี่คือสิ่งที่คุณจะต้องใช้อย่างจำกัด (ฟังดูรู้สึกว่าคุ้นๆอย่างแรง)

แต่ว่าถ้าหากคุณจำเป็นจะต้องทำการประเมินเรื่องเงิน(อย่างคร่าว) ลองมาดูวิธีการอย่างง่ายๆและรวดเร็วเพื่อให้ได้ตัวเลขโดยประมาณ



(ขออภัยในภาษาไทย US Dollars นะครับ)

ง่ายๆเพียงแค่เอาจำนวนคนคูณด้วยจำนวนระยะเวลาและคูณด้วยอัตราค่าใช้จ่ายที่มีต่อ 1 ชั่วโมง (ต่อวันหรือต่อเดือนแล้วแต่ที่ใช้กัน เช่น ถ้าบวกกว่า Project ใช้เวลา 4 เดือน = 80-90 วัน วันนี้ใช้คิดค่าใช้จ่ายเท่าให้รึเปล่า) แนะนำว่าเราอาจจะมีค่าใช้จ่ายด้านอื่นๆยกตัวอย่างเช่น Software หรือบริษัทของคุณอาจจะมีวิธีทางบัญชีที่คิดค่างบประมาณรายจ่ายสำหรับสิ่งต่างๆที่แตกต่างไปจากนี้ แต่อย่างน้อยค่าใช้จ่ายส่วนใหญ่ในการพัฒนา Software Project มักอยู่ที่คนที่มานั่งพัฒนานี่แหละ ดังนั้นเมื่อเราเข้าใจทุกสิ่งทุกอย่างแล้ว ก็ถึงเวลาที่เราจะมาห่วยให้ Project สามารถตัดสินใจได้ว่าเราจะไปต่อหรือหยุดและ Cancel มันซะ

## สุดท้ายแล้วมีอนาคตอย่างมาประกอบร่างสร้างภาพ

นี่เป็นส่วนที่ทางผู้มีส่วนร่วมแต่ละคนอยากรู้อย่างใจจดจ่อมาก เพราะอะไร เพราะว่าทั้งหมดทั้งปวงที่เรากล่าวมาแล้ว มีคำถามเพียงแค่สองคำถามเท่านั้นที่ท่านๆเหล่านั้นอยากรู้มากมาย

1. เมื่อไหร่มันจะเสร็จ
2. แล้วมันราคาเท่าไหร่

แต่ให้ชัดเจน(อย่าลืมนะครับ)ว่าสิ่งที่เราบอกไปนั้นเป็นเพียงค่าประมาณการ(อย่างคร่าวๆ) แม้ว่าเราจะทำการบ้านของเรามาแล้ว(อย่างตี) และก็สามารถตอบคำถามพื้นฐานที่สำคัญ(สำหรับ Project) แต่ยังมีสิ่งต่างๆที่ยังถือว่าเป็นปัจจัยที่ยังไม่รู้ได้และไม่แน่นอน (อย่างเช่นทีมของเราจะสามารถทำงานได้ด้วยความเร็วเท่าไหร่) ดังนั้นเรายังไม่สามารถที่จะถือว่าค่าเหล่านี้ที่เราประเมินออกมานะเป็นอะไรที่ดีไปกว่าค่าประมาณ(แบบคร่าวๆ)

## บทสรุป Inception Deck

ยินดีด้วย!!! ตอนนี้คุณได้ผ่านพ้นช่วงที่ต้องได้ว่าสำคัญซึ่งถือเป็นก้าวแรกในการให้คำนิยาม Project ของคุณ รวมถึงการหาคนมาเข้าร่วมทีม และเริ่มก้าวแรกที่สำคัญของ Project ที่มีความเป็น Agile มาจากของคุณแล้ว

ลองมาดูภาพรวมและเรื่องราวที่คุณทีมของคุณ ลูกค้าของคุณตลอดจนถึงผู้ที่ให้การสนับสนุนด้านเงินทุนของคุณ สามารถเข้าใจตรงกันได้ โดยรวมแล้วสิ่งต่อไปนี้เป็นสิ่งที่เราได้ข้อสรุปจากกิจกรรมที่เราทำมาทั้งหมดครับ

- คุณกำลังสร้างอะไรและสร้างทำไม
- อะไรคือคุณค่าของ Project (อะไรที่ทำให้คนอื่นคิดว่ามีคือสิ่งที่ต้องทำ)
- อะไรคือสิ่งที่สำคัญหรือ Objective ของ Project นี้
- โครงสร้างที่มีส่วนร่วมใน Project ของเรางาน
- สิ่งที่เรากำลังจะสร้างหน้ามันเป็นยังไง
- อะไรจะเป็นคุปสรุคหรือปัญหาที่อาจจะเกิดขึ้นได้
- สิ่งที่เรากำลังจะสร้างนั้นมันเล็ก/ใหญ่ขนาดไหน
- มีปัจจัยด้านไหนบ้างที่เราพอจะยึดหยุ่นได้ใน Project
- แล้วสิ่งที่เราทำใช้เวลานานมั้ย และค่าใช้จ่ายประมาณเท่าไหร่



**อาจารย์ เช่นเช**

**และนักรับผู้มีความมุ่งมั่น**

**อาจารย์:** ไหนลองบอกมาซิลูกศิษย์ว่าอะไรบ้างที่ได้เรียนรู้มาจาก Inception Deck

**ลูกศิษย์:** ผมได้เรียนรู้ถึงความสำคัญในการถามปัญหาหากาในตอนเริ่มต้นของ Project และการให้ความสำคัญกับความเข้าใจ ตรงกันและความสอดคล้องกันของทุกๆคน

**อาจารย์:** ดีมาก มีอะไรอีกว่ามา

**ลูกศิษย์:** ตอนนี้ผมได้เข้าใจว่าในการทำ Project ขั้นตอนการระบุของเขตของงานและการวางแผนไม่จำเป็นต้องใช้เวลานานเป็นเดือนอีกต่อไป ด้วย Inception Deck เราสามารถทำให้ทุกคนเข้าใจตรงกันถึงสิ่งที่เราจะต้องทำโดยใช้เวลาเพียงไม่กี่วัน

**อาจารย์:** แล้วถ้าหากมีอะไรที่เป็นแผนหลักที่สำคัญ ขอบเขต หรือแม้กระทั่งทิศทางของ Project มีการเปลี่ยนแปลงไป เราควรจะต้องทำยังไง

**ลูกศิษย์:** แก้ไขปรับปรุง Deck ของเราตามการเปลี่ยนแปลงนั้น แล้วทำการอัปเดตตอน Inception Deck อีกครั้งกับทุกๆคนที่เกี่ยวข้อง เพื่อให้แน่ใจว่าความสอดคล้องและเข้าใจตรงกันยังอยู่ดีมีสุขใน Agile Project ของเรา

**อาจารย์:** เยี่ยมมาก ตอนนี้เข้าพร้อมแล้วที่จะเดินทางต่อไป

ด้วยคำถาม “ทำไม” ต่อจากนี้เราจะเจาะลึกในรายละเอียดเพิ่มเติม อย่างไร ที่คุณจำเป็นต้องใช้ในการวางแผน Agile Project การประเมิน,

รายการสิ่งที่ต้องทำ (แบบ Agile), หรือแนวคิดในเรื่องของความเร็วของทีม (ทีมสามารถทำงานได้เร็วแค่ไหน) ต่อจากนี้เราจะคุยกันในรายละเอียดของเรื่องเหล่านี้ทั้งหมด และแน่นอนว่าไม่มีอะไรที่จะดีไปกว่าการเริ่มต้นด้วย User Stories

# บทที่ 6 เก็บ User Story



ในภาคสามนี้มาลองดูเรื่องพื้นๆ ของการวางแผนแบบแคลใจล์กันบ้าง เราจะเริ่มจาก user story ตามด้วยการ estimate และจบด้วยการวางแผนแบบปรับไปเรื่อยๆ (adaptive) หลังจากสำเร็จวิทยาลัยหอการค้า requirement แบบ user story คุณจะเข้าไปซึ่งว่าแผนที่วางแผนแบบแคลใจล์นั้น จะได้รับการปรับเปลี่ยนให้เป็นปัจจุบันและมีแต่ข้อมูลล่าสุดเสมอ แผนยังไม่ต้องไปเสียเวลา กับสิ่งที่ไม่ประโยชน์ที่สุด ที่เราไม่จำเป็นต้องรู้ หรือที่เราอาจจะไม่สนใจ หรือที่เราอาจจะกันดีกว่า นั่นเท่านั้น! นั่นเรา มาเริ่มจากการดูว่าแต่ก่อนเราเก็บ requirement กันยังไง และต้องปวดหัวขนาดไหน เมื่อต้องทำเอกสารของเพนนิ่ม

## 6.1 ปัญหาของการทำเอกสาร

ถ้าคุณไม่ได้หลอกตัวเอง คุณจะยอมรับความจริงที่ว่า การเก็บ requirement อย่างละเอียดถี่ยบลงในเอกสาร จริงๆ แล้วมันก็ไม่ค่อยจะเดิร์คซักเท่าไหร่ใน software project ลูกค้ามักจะไม่ได้สิ่งที่เข้าต้องการจริง ทีมมักจะไม่ได้ทำในสิ่งที่ต้องการจริงๆ เวลา ส่วนใหญ่ก็มักแต่เอาไปเสียงกันว่าที่เขียนไว้มันหมายความอย่างนั้นอย่างนี้ แทนที่จะเอาเวลามาทำสิ่งที่ควรจะทำจริงๆ ลองมาดูปัญหาอื่นของการตีบบันทำเอกสารกันดูบ้าง

มักจะยอมรับความเปลี่ยนแปลงไม่ค่อยได้

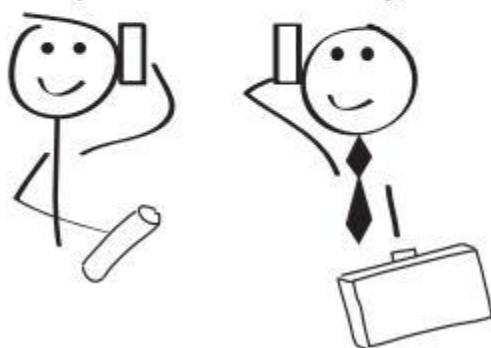
พี่จำได้ว่าพี่ขออะไรไปหนะไอ้น้อง  
แต่นั่นมันเมื่อ 6 เดือนที่แล้ว!

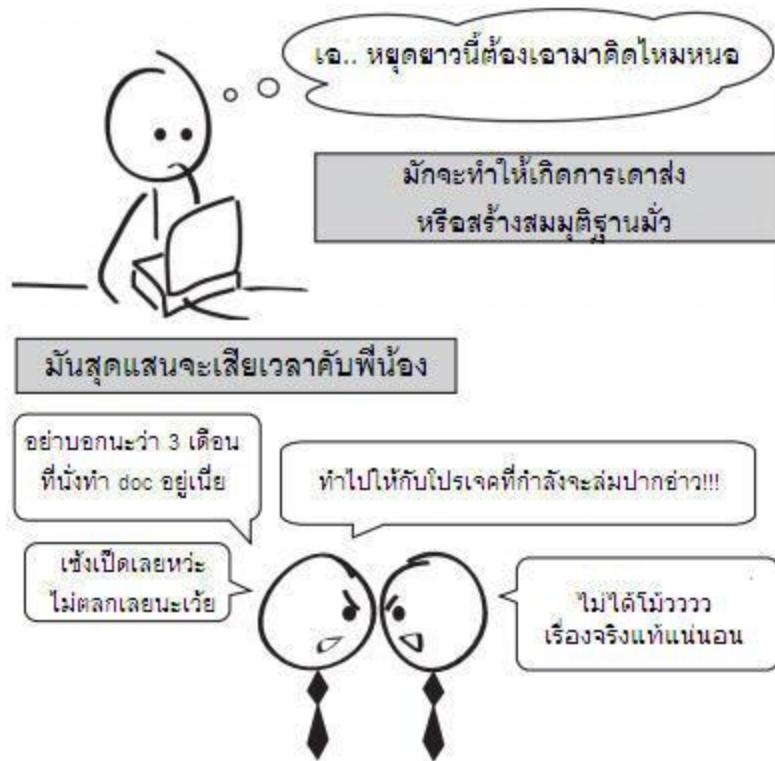


งานถูกทำออกตามสเปกเบื้องต้น  
แต่ใหม่มันไม่ยักระยะใจลูกค้า

ในที่สุดเรา ก็รู้แล้วว่า  
จริงๆ แล้วลูกค้าอยากรู้ได้อะไร

แหลมเลย แต่เก็บไว้ในใจก่อนนะ  
ตอนนี้ spec มัน freeze แล้วอีก





หรือจริงๆ แล้วเราต้องทำเอกสารให้มากกว่านี้?

ปัญหาของการเก็บ requirement ด้วยเอกสารไม่ได้อยู่ที่จำนวนว่าจะมากจะน้อย แต่มันอยู่ที่การสื่อสาร เอกสารมันพูดกับคุณ ไม่ได้ แรมโครงการต่อโครงการที่อ่านเอกสารเดียวกันก็อาจจะตีความไปได้ต่างๆ นานา

ผมไม่ได้พูดว่าเชือເຂອເເຈີນໄປ	
ผมไม่ได้พูดว่าເຂອເເຈີນໄປ	ผมไม่ได้เป็นคนพูดนะ
ผมไม่ได้พูดว่าເຂອເເຈີນໄປ	ผมใช้วิธีอื่นบอกต่างหาก
ผมไม่ได้พูดว่าເຂອເເຈີນໄປ	คนอื่นอาจจะເຂົາໄປ
ผมไม่ได้พูดว่าເຂອເເຈີນໄປ	ເຮືອແດຍເມື່ອເປັນໄປເຊຍາ
ผมไม่ได้พูดว่าເຂອເເຈີນໄປ	ເຂອຂອມໂນຍເຂົາໄວໃຈຜມໄປຕ່າງໜາກ

ตัวหนังสือมันดีนี่ได้นะ จะบอกให้!

Requirement เนี่ยมันควรจะเรียกว่า Requirement จริงๆหรือ

นักรบแอจใจลืมเมื่อติดกับ Requirement แค่ใช้คำว่า Requirement ก็ผิดแล้ว ปรมาการย์ Kent Beck ได้กล่าวไว้ในคำวิจารณ์ *Extreme Programming Explained : Embrace Change* ว่า

“การพัฒนาซอฟต์แวร์นั้นถูกทำให้หลงทางกันไปหมดก็ด้วยการใช้คำว่า Requirement นี่แหละ คำนี้ถ้าเปิดดิกดูก็จะหมายถึงอะไรที่เป็นข้อบังคับหรืออะไรที่ต้องทำ คำนี้สื่อความหมายถึงสิ่งที่เป็นที่สุด สื่อถึงสิ่งที่เป็นการถาวรสื่อถึงสิ่งที่เปลี่ยนแปลงไม่ได้ ดังนั้นการใช้คำว่า Requirement จึงผิดเต็มๆ

จากเอกสาร Requirement เป็นพันหน้า แค่ทำให้ได้ซัก 5, 10 หรือ 20 เปอร์เซนต์ก็มักจะพบว่าเราได้ตอบความต้องการทางธุรกิจได้เกือบหมดแล้ว และถ้าที่เหลืออีก 80 เปอร์เซนต์ที่ไม่ได้ทำหละ มันจะเรียกว่า Requirements คงไม่ถูกนัก เพราะมันไม่ใช่สิ่งที่ต้องทำ

ผมจำได้ว่า Martin Fowler เคยบ่นว่า ขนาดใช้เวลาเป็นปีๆ ทำหนังสือออกแบบมาให้อ่านกัน ท่านทั้งหลายก็ยังคุ้นเคยกันดี แต่ส่วนใหญ่แล้วปัญหาที่จะเกิดก็คือ การสื่อความหมายที่ผิดพลาด ทำให้ไม่สามารถบันทึกความต้องการที่แท้จริงของลูกค้าได้



ธรรมใจล์

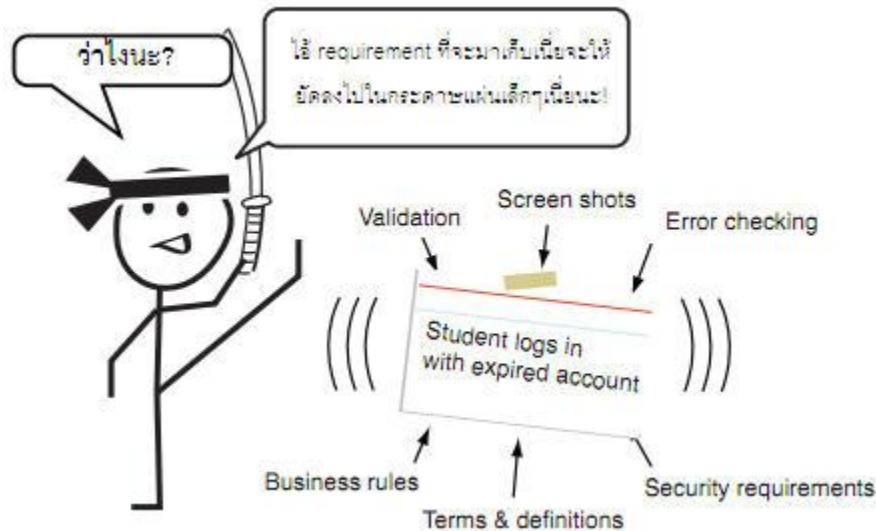
กระบวนการทบทวนอย่างสุดยอดแห่งการสื่อสารในภาษาที่มิโนร์ก็คือ การคุยกันชิ้งหน้า (face-to-face)

สิ่งที่เราต้องการ ก็คืออะไรซักอย่างที่สามารถทำให้เราคุยกันเรื่อง requirement จนสามารถเก็บเกี่ยวสาระสำคัญของมันไว้ได้ มันต้องเล็กพอที่จะสามารถวางแผน (ให้ทำเสร็จใน iteration) ได้แต่ก็ต้องมีความหมายพอที่จะเดือนให้เราจำได้ว่าเราคุยกันไว้ร่องไว

## 6.2 มาใช้ User Story กันดีกว่า

User Story ในแอจใจล์ คือคำอธิบายสั้นๆ ของคุณลักษณะของระบบที่ลูกค้าของเรา ผู้อยากร่วมมือในชั้กวันหนึ่ง ปกติจะเขียนในกระดาษแผ่นเล็กๆ (เพื่อจะได้ค่อยเตือนไม่ให้เราเขียนทุกอย่างลงไปหมด) และมีไว้เพื่อเป็นเครื่องเตือนใจให้เราไปคุยกับลูกค้าจริงในรายละเอียดปลีกย่อย

ในตอนแรกที่คุณได้เห็น User Story, คุณอาจจะมีข้อสงสัยว่า แล้วเนื้อหายไปไหน มีแต่น้ำ อาย่าเพิ่งตกใจไป เนื้องมือยัง  
แหละ เพียงแต่อาจจะไม่ได้อยู่ในที่ที่คุณคิดว่ามันน่าจะอยู่ก็ได้



มันก็ไม่ใช่หรอกรที่เราจะมาเขียน requirement ที่เก็บมาลงในกระดาษแผ่นเล็กๆแผ่นเดียว จริงๆแล้วเราไม่ต้องเขียนอะไรไว้มัน  
มากนักโดยด้วยข้อแล้วใจลืมอยากรีบเขียนไว้ก่อนแล้วก็ไปเล็กๆเพื่อเป็นการเตือนสติว่า เป้าหมายเบื้องต้นของการเก็บ requirement  
ไม่ใช่การเก็บรายละเอียด หากแต่เป็นเพียงการจดบันทึกความคิดหลักๆสองสามคำให้พอได้ใจเดียว เพื่อเอาไว้มาคุยกันใน  
รายละเอียดต่อวันหลัง ข้อ哪แล้วทำไม่ได้คุยกับรายละเอียดกันให้เสร็จๆไปเลยที่เดียวหละ ที่ไม่ทำก็อย่างนั้นก็ เพราะว่า ในความเป็น  
จริงเราไม่รู้ว่าเราจะได้ทำให้ได้คุยกันเนี่ยเมื่อไหร่ แผลๆอาจจะไม่ได้เข้าการทำเลยด้วยข้อ เจ้าอาจจะไม่ได้แตะ feature นี้อีกเลย  
เป็นเดือนๆและกว่าจะถึงตอนนั้น อะไรก็อาจจะเปลี่ยนไปหมดแล้ว ให้ทีมคุยกันก็ได้ประโยชน์ดังนั้น เพื่อเป็นการ  
ประยัดพลังงานลดโลกร้อน แทนที่จะมาชุดคุ้ยหารายละเอียด ให้ครบถ้วนกระบวนการกันตั้งแต่แรก เราจะเก็บเรื่อง  
รายละเอียดเอาไว้ทีหลัง (เพิ่มเติมใน บทที่ 9.4) ให้คิดซึ่งว่า user story คือคำสัญญาว่าเราจะมาคุยกัน ถึงวันนั้นเราจะล้างลับ  
ตัวแตกเอกสารให้หมดได้หมด พุง แต่เราอาจจะล้างจนกว่าจะหัวรัวถึงเวลาแล้วที่ต้องล้างแน่น

### 6.3 ส่วนประกอบของ User Stories ที่ดี

สิ่งแรกคือ มันต้องมีค่ากับลูกค้า แล้วอะไรล่ะที่แปลงว่ามีค่า? ค่าว่ามีค่าอาจหมายความถึงสิ่งที่ลูกค้าเข้าใจได้และรู้ว่าเขากำลัง  
จ่ายเงินไปเพื่ออะไร ลองคุยกับตัวอย่างด้านล่างว่าถ้าลูกค้ากำลังพิจารณาเลือกซื้อสินค้าที่ต้องรับจาก  
เดินดูเมืองหนึ่นวันแล้วลูกค้าจะเลือกร้านไหน

<b>Ernie's Tech Diner</b>	
C++	3 days
The system will be written in C++.	
Connection pooling	2 days
All database access will be handled by a database connection pool.	
Model-View-Presenter pattern	5 days
The system will separate presentation logic from business logic.	
<b>Sam's Business Pancake House</b>	
Create user account	3 days
Users will have individual, personalized accounts to log into your system.	
Notify subscribers of new listings	2 days
Subscribers will be notified every time a new house is listed in their market.	
Cancel subscription	1 day
Embedded in every email will be an unsubscribe option.	

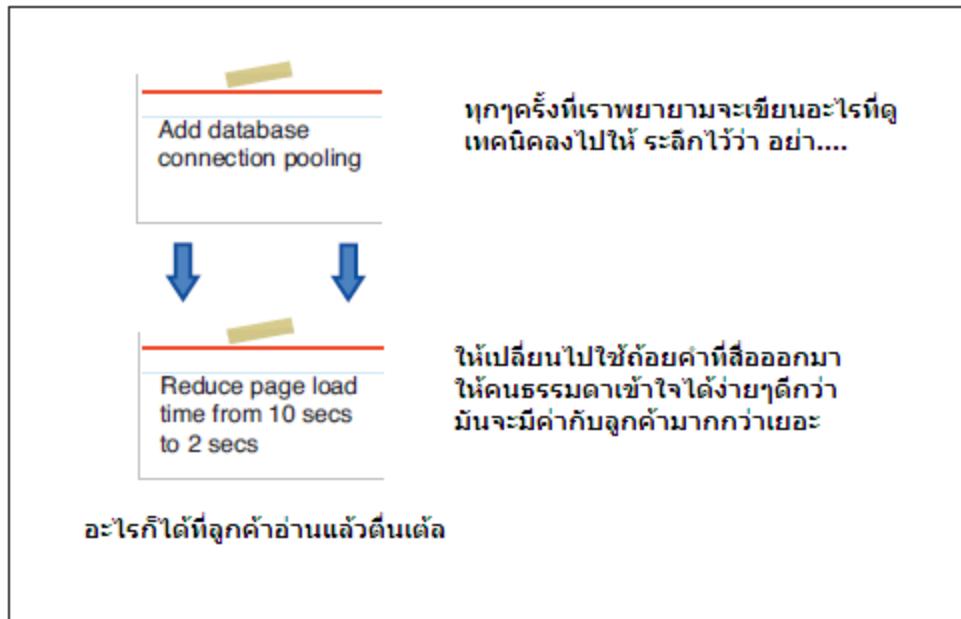
โว้มันคืออะไร หน้าไปก่อนละกัน



อะน่าสนใจขอลงลังสักหน่อย



ดังนั้นเราสามารถสรุปได้ว่า user stories ต้องเป็นอะไรที่เข้าใจง่าย คนทุกคนสามารถทำได้ ไม่ต้องมีความรู้ทางเทคโนโลยีมาก การใช้ connection pooling และ design pattern ของเหล่านี้ยังช่วยให้เราอย่างพยายามให้ลูกค้าเห็นว่าเป็นการดีที่สุด



ส่วนประกอบที่สองของการเป็น User Stories ที่คือมันจะต้องครอบคลุมหน้าจอดหลัง(end-to-end)? เมื่อแปลงเป็นภาษาอังกฤษ

ค้านล่าง



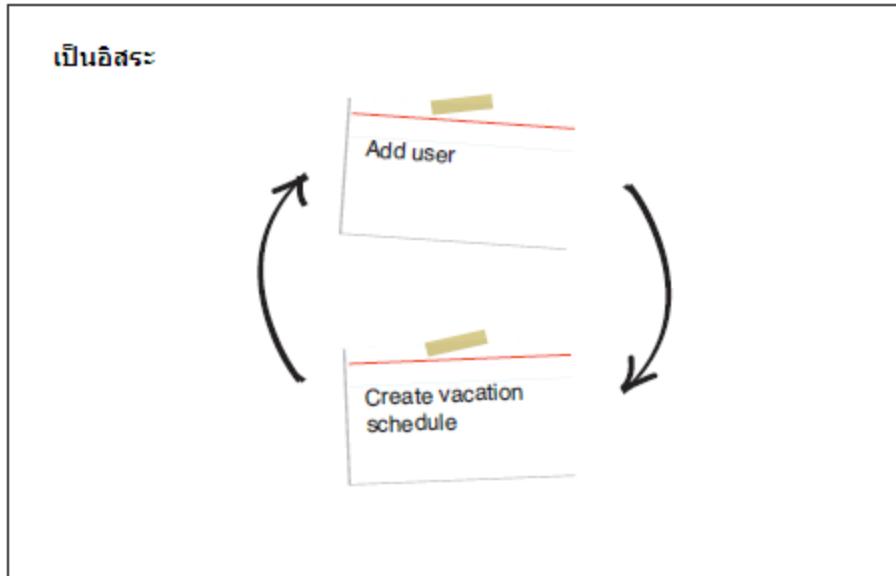
Use interface (HTML, Ajax, CSS)

Middle tier (C#, Java, Python)

Data tier (Oracle, SQL Server)

*UMMM CAKE ... OM NOM NOM*

ถ้าเปรียบกับเค้กแล้วคงไม่มีใครต้องการกินเค้กแค่ชั้นล่างชั้นเดียว ลูกค้าก็ เช่นกัน เมื่อจ่ายเงินแล้วต้องได้ต้มยำดังนั้น user stories ต้องมาแบบเต็มๆ ด้วย เช่น กัน มันต้องครอบคลุมทุกชั้นดังนั้นมันต้องมีคุณลักษณะดังนี้



อะไรก็เกิดขึ้นได้ในโปรเจกของเรา บางสิ่งที่เคยคาดหวังคัญขาดไม่ได้ขาดแล้วตาย พอข้ามมาอีกสปด้ามันอาจไม่มีความหมายเลยก็ได้ดังนั้นถ้า story ต่างๆมันผูกติดกันแน่นจนเกินไปการที่เราจะขยายหรือแยกมันกับสิ่งอื่นก็เป็นเรื่องยากขึ้นมาทันที

We don't always succeed (we need an application before we can create the reports), but slicing our stories from end to end and gathering them by feature enables us to treat the vast majority of our stories as independent and be flexible on scope when necessary.



อย่างไรก็ตามจาก requirement ที่เราได้จากลูกค้าเราสามารถสร้างซอฟต์แวร์ออกมาได้หลายระดับ (ตามราคา) เมื่อนักบุญลูกค้าต้องการรถยนต์หนึ่งคันและบอกความสามารถของรถมากว่าเราจะพบว่ามีรถหลายรุ่นที่ตรงตามที่ลูกค้าต้องการไม่ใช่จะเป็น Ford Focus, Honda Accord หรือแม้กระทั่ง Porsche 911 ดังนั้นถ้าเราเขียน User Story ออกมาให้มาในรูปแบบที่ต่อรองได้นั้นจะเป็นเรื่องดีกว่า เพราะจะทำให้เราสามารถหักหรือขยายจึงการสร้างออกมาให้เหมาะสมกับเงินที่เราได้รับมาด้วย

**ถูก Test ได้**

- Allow regular logins
- Re-direct expired logins
- Display appropriate error message
- Handle nonexistent user account

ถ้า Story ที่เราเขียนขึ้นมาสามารถถูก test ได้นั่นหมายความว่าเราเข้าใจความต้องการของลูกค้าได้ดี เนื่องจากเวลาอ่าน story จะต้องทำงานได้อย่างไรบ้างเพื่อให้ผ่านการทดสอบและเราจะสามารถเขียน test case ที่ล็อกกับ story ของเราได้เพื่อให้คนเขียนโปรแกรมสามารถเขียนได้อย่างถูกต้องตามที่มั่นใจจะเป็น

**เล็กและประเมินได้**

Think you boys can get this done in a week?

และเพื่อให้ขนาดของ Story ที่เราสร้างขึ้นนั้นเหมาะสมกับเวลาที่เรามีเราควรจะทำให้แต่ละ story มีขนาดอยู่ระหว่าง 1-5 วัน เพราะจะทำให้เรามั่นใจได้ว่าสามารถจัดการมันได้ในกรอบของ iteration ของเราระหว่าง 1-2 อาทิตย์ ดังนั้นคุณสมบัติของ Story ที่ดีนั้นควรจะเป็น Story ที่ INVEST ที่ย่อมาจาก “Independent, Negotiable, Valuable, Estimatable, Small, and Testable.”

User stories	Specifications & requirement docs
Lean, accurate, just-in-time	Heavy, inaccurate, out-of-date
Encourage face-to-face communication	Encourage guesswork (false assumptions)
Simplified planning	Complex planning
Cheap, fast, easy to create	Expensive, slow, hard to create
Never out-of-date	Always out-of-date
Based on latest learnings	Based on little or no learning
Enable real-time feedback	Disable real-time feedback
Avoid false sense of accuracy	Promote false sense of accuracy
Allow for team-based collaboration and innovation	Discourage open collaboration and innovation

ตอนนี้เราได้รู้ทฤษฎีเกี่ยวกับการสร้าง User Story ที่ดีไปแล้ว ต่อไปเราจะมาดูกันว่าเราจะเขียนไปใช้จริงยังไง

## Welcome to Big Wave Dave's Surf Shop



**BIG WAVE DAVE**

### มันก็ OK ที่จะช่วยลูกค้าเขียน Story

Don't take earlier agile books too literally when they tell you that customers should write all the user stories. That advice is right in spirit—but not in practice. It's true that customers should provide the content for our user stories (because they are the ones who know what they want built). In practice, however, it will be you who will be doing most of the writing.

So, don't be worried if you find yourself giving them a hand. Just make sure your customers are active participants in the process and you are capturing their needs in the cards.

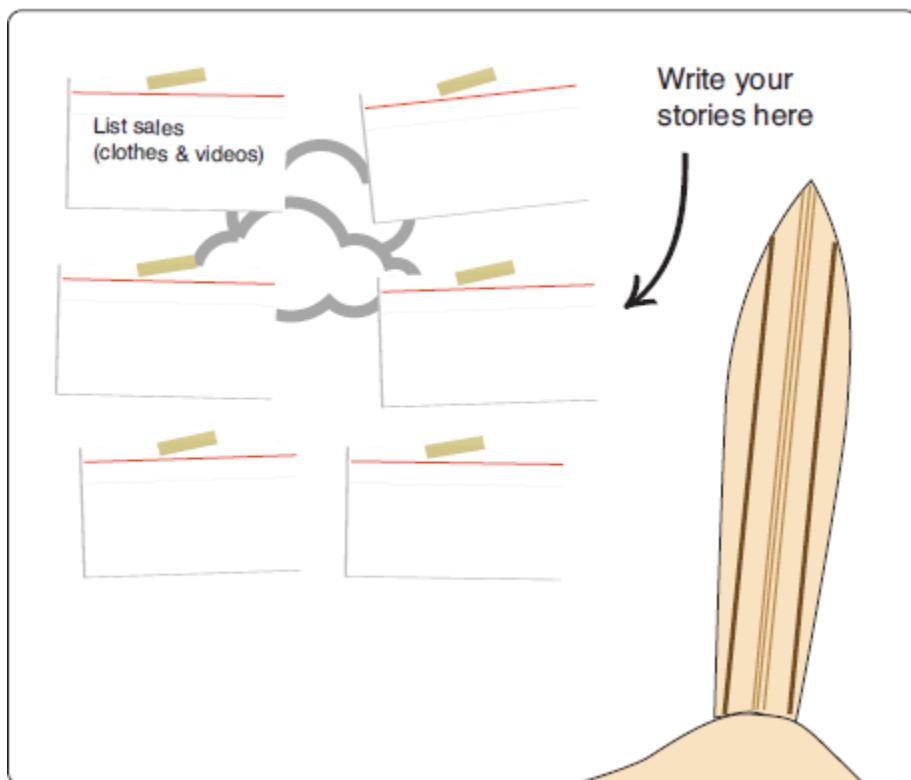
คุณเดฟ (ไม่ใช่โปรดแกรมเดฟ) จ้างบริษัทไก่ก้ามามีส่องสามเดือนที่แล้วเพื่อมาทำเว็บแต่หลังจากผ่านไปนานสิ่งที่คุณเดฟได้มาคือเอกสารเกี่ยวกับ requirement และไม่มีอะไรมากกว่านั้น ไม่มีการสร้างเว็บมาให้ดู (แรง) คุณเดฟทันไม่ได้จึงจ้างเจ้าไปแก้ปัญหา ดังนั้นสิ่งแรกที่เราเริ่มทำคือการหาให้เจอกว่าคุณเดฟต้องการอะไร? เราจึงตั้งคำถามไปว่า “พี่เจอร์หักๆ” ที่เดฟอยากได้คืออะไร และเรา ก็พบว่าสิ่งที่เดฟต้องการนั้นไม่ได้มีอะไรขึ้นชื่อเลยมันเป็นเรื่องที่แสนธรรมชาติ ข้อแรก, เดฟอยาให้เว็บนี้นำเสนอแหล่งท่องเที่ยวท้องถิ่น ที่เด็กๆ เก็บข้อมูลสามารถเข้ามาดูได้ว่ามีกิจกรรมเกี่ยวกับการ SURF Kavanaugh บังเอ็นการแข่งขัน การสอนการเล่น อะไรประมาณนั้น ข้อที่สอง, เดฟต้องการขายของเข่น board, suite, เสื้อผ้า วิดีโอ บังเอ็น แต่ระบบขายนี้ต้องใช้ง่ายนะและออกแบบแล้วต้องดูหล่อแหลมแจ่มแจ้ง

ข้อที่สาม, เดฟอยาให้เว็บนี้สามารถแสดงสภาพอากาศที่หาดแบบ real time ได้โดยการต่อ Web Cam และส่องไปที่หาดแบบๆๆ เพื่อให้คนที่สนใจเข้ามาดูแบบจริงๆ ได้เลย นั่นแปลว่าเว็บต้องเร็วส์



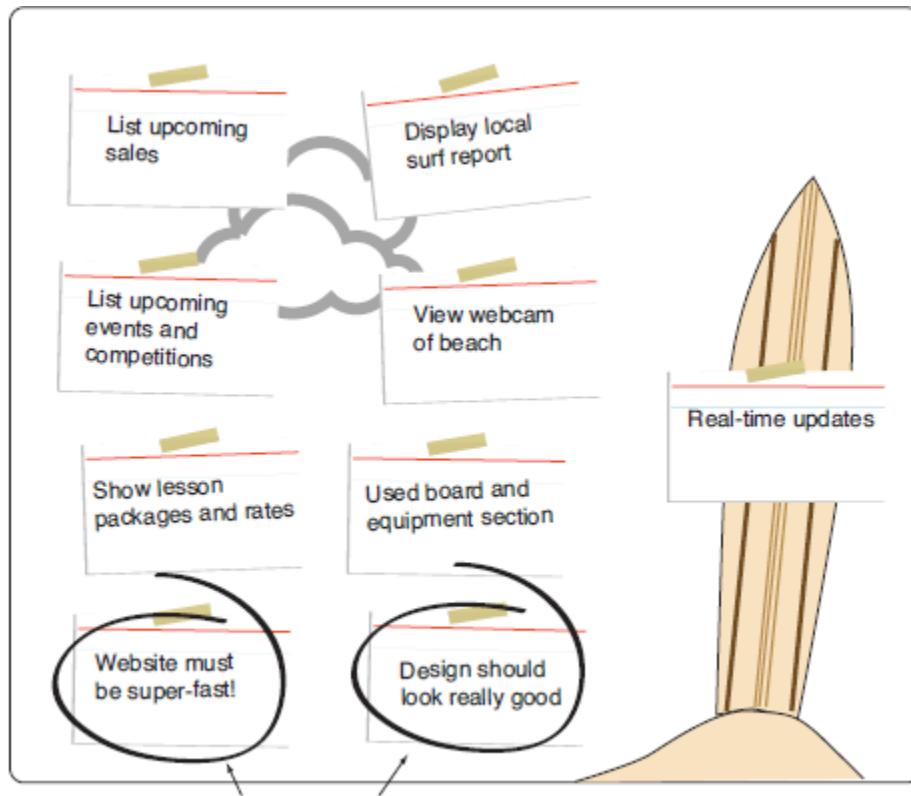
*Now you try*

จากความต้องการของเดฟข้างบนเราลองมาเขียนเป็น story ให้ได้สักหนึ่งกันใหม่ อย่างกังวลว่าจะผิด แค่ลองก่อนว่าจะออกมายังไง



เราลองมาดูว่า User Story แต่ละอันที่เราสร้างขึ้นมาสามารถตรวจสอบได้ด้วยเงื่อนไข INVEST ได้หรือไม่ (Independent, Negotiable, Valuable, Estimatable, Small, และ Testable)

เราสามารถกันตัวยการลดลงเขียนอะไรที่เราคิดว่าลูกค้าของเราเข้าใจมันได้ง่ายๆ สมมติว่าเราเขียนได้ทั้งหกเรื่องแล้วเราลองมาดูสิว่า มันออกมายังไง



หลังจากลองอ่านวนไปปานมา ไอ้สอง story Sud ท้ายมันดูเหมือนพิกัด

- The website must be super-fast!
- The design should look really good.

แล้วไอ้สอง story นี้มันไม่เด็ดดวงไหหน

คือถ้าเราจะให้ความสำคัญกับคำว่า super fast มันก็ไม่ติดหรอกเพียงแต่ว่า ไอ้ที่เรานำมันเว็บแค๊ดให้นอกจากนี้ ไอ้ที่ว่า Looking Good เนี่ยยังคงถึงเรียกว่า Looking Good การที่เราสร้าง Story แบบนี้ขึ้นมาันน้ำเรียกมันว่า Constraints เพราะสองสิ่งนี้เป็นสิ่งที่ลูกค้าต้องการจริงๆ แต่เราไม่สามารถทำมันให้เสร็จภายในหนึ่งหรือสองอาทิตย์ ดังนั้นถ้าเราได้ user story แบบนี้ออกมารา จะสามารถเปลี่ยนมันออกมานในอีกมุมหนึ่ง

เช่น เว็บของเราต้องเร็วประมาณนี้

All web pages must load in less than 2 sec

A constraint card

ความนี้เราก็รู้แล้วว่าเว็บส์เนี่ยมันเว็บแค๊ดให้นะและเราอย่างสามารถ test มันได้อีกด้วย

Constraints เป็นเรื่องที่น่าสนใจแต่เราจะไม่เอามันมาปนไว้กับ Story ทั่วไป เราจะเก็บมันไว้ในการดีสีอื่น เพื่อให้แน่ใจว่าคนอื่นๆ ในทีมก็ให้ความสำคัญกับมันและทำการทดสอบให้ผ่านทุกครั้งที่มีการ deploy

เพิ่มเพลตสำหรับ User Story

เพิ่มเพลทที่ประกอบไปด้วยคำจำกัดความต้องการอย่างด้านล่างจะช่วยให้เราสามารถเข้าใจ User Story ได้แม่นคืออะไร

As a <type of user>  
I want <some goal>  
so that <some reason>.



**who** is this story for  
**what** they want to do  
**why** they want to do it

ตัวนี้เมื่อเวลาลงเคาน์เพลท์ไปใช้กับร้าน Big Wave Dave เราจะเขียนออกมายังไง

As a surfer who likes to sleep,  
I want to check local surf conditions via a webcam  
so that I don't have to get out of bed if there is no swell.

As a land-locked Canadian hockey player,  
I want to sign up for some adrenaline-pumping lessons  
so I can feel the thrill of going "over the falls."

As a grommet looking for the latest surf wear,  
I want to see all the latest board shorts and designs  
so that I can look stylin' for the Sheilas this summer.

ข้อดีของ เพิ่มเพลท คือ มันจะช่วยให้เราสามารถตอบคำถามที่สำคัญได้สามเกณฑ์ who, what และ why แต่การได้มาซึ่งข้อมูลนี้ ต้องที่เราต้องแลกมาคือ คำอธิบายที่เดินเรื่องน้อย ตั้งน้ำกการที่มีคำอธิบายอยู่ในเพิ่มเพลททำให้บางคนมองว่า มันมากเกินความจำเป็น ภูมิใจตั้งน้ำถ้าเราไม่ลงมือใช้เราก็จะไม่รู้ว่าแบบไหนที่เราชอบ ตั้งน้ำให้ลองใช้งานดูแล้วจะพบวาวิธีการได้ที่เหมาะสมกับเราที่สุด — แต่ทั้งนี้ทั้งน้ำเราต้องเป็นที่จะต้องเลือกเข้าอย่างเดียว ทั้งนี้ เรายาจะให้มันทั้งคู่ได้พร้อมกัน เช่น เราอาจใช้ชื่อสั้นๆ อย่าง “Add user” สำหรับการเริ่มต้นแต่เราจะเขียนคำอธิบายโดยการใช้เพิ่มเพลทไว้ข้างหลัง

## 6.4 How to Host a Story-Gathering Works hop

หลังจากที่เราได้รวบรวม user story ทั้งหมดแล้ว เราควรจะเรียก user มาดูว่าสิ่งที่เราเขียนไว้ใน user story นั้นเป็นสิ่งที่เข้าอย่างได้จริงๆ หรือป่าวดังนั้นสิ่งที่เราต้องทำคือ การสร้างสิ่งที่เรียกว่า story-gathering workshop ซึ่งการจัด workshop นี้จะเป็นการพบกันและช่วยกันตรวจสอบและเขียน user story ระหว่าง developer และ user

The goal of the story-gathering workshop is breadth and depth of the story-gathering workshop. คือ ภาพรวม เพราะเราต้องการเป็นว่าจริงๆแล้วสิ่งที่ user ต้องการจากระบบมีอะไรบ้างແຕ้่มไม่ได้หมายความว่าเราจะทำมันทุกอย่าง. ภาพประกอบด้านล่างคือบริบทการทำการทำ story-gathering workshop ที่ดี

## Step 1: Get a Big Open Room

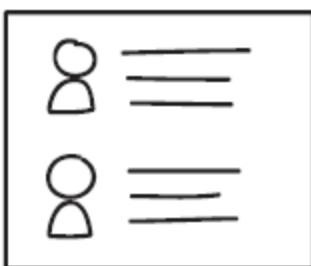
### Step 1: Get a Big Open Room



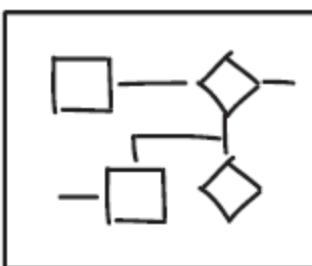
เราต้องการสถานที่ที่มีพื้นที่ให้เราได้ยกย้ายไปมาและเราต้องสามารถเข้ากระบวนการอันใหญ่ๆมาแบบได้สำหรับแยก story ออกมาเป็นกลุ่มเพื่อความง่ายในการเข้าใจ

## Step 2: Draw Lots of Pictures

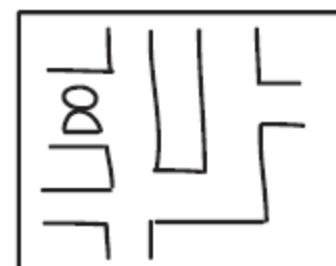
## Step 2: Draw Lots of Pictures



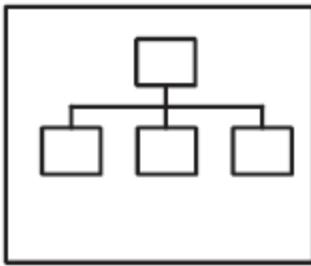
Personas



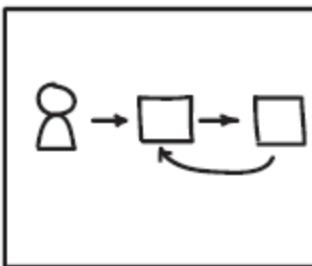
Flowcharts



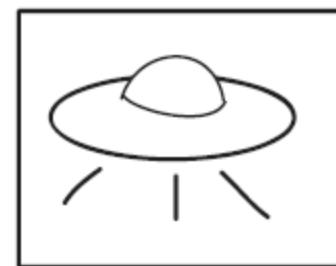
Scenarios



System maps



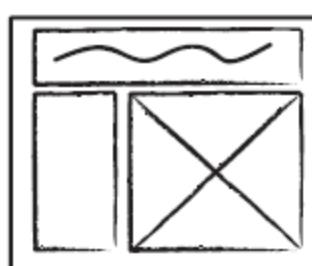
Process flows



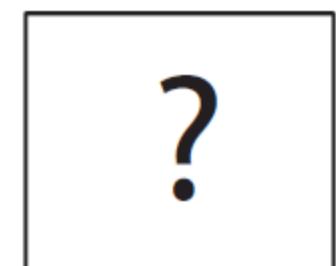
Concept designs



Storyboards



Paper prototypes

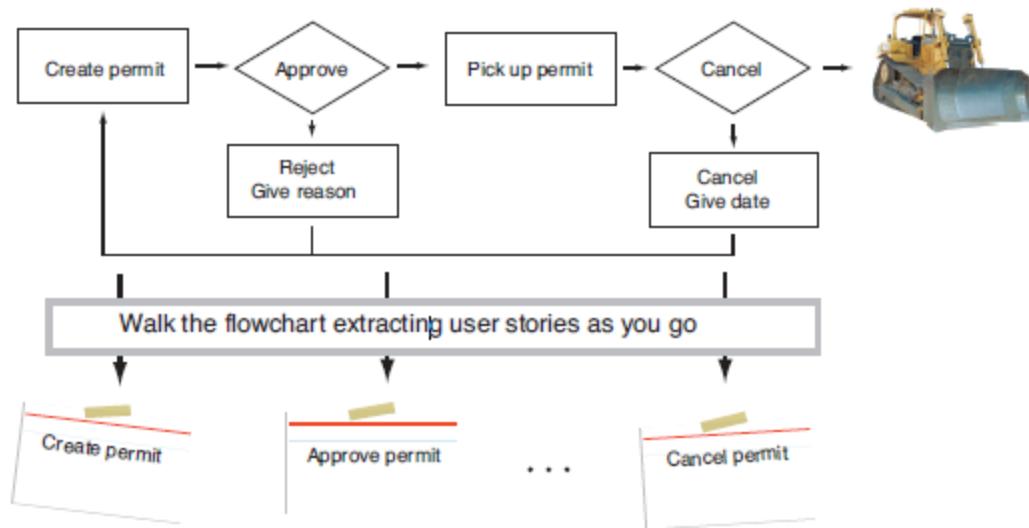


Your own

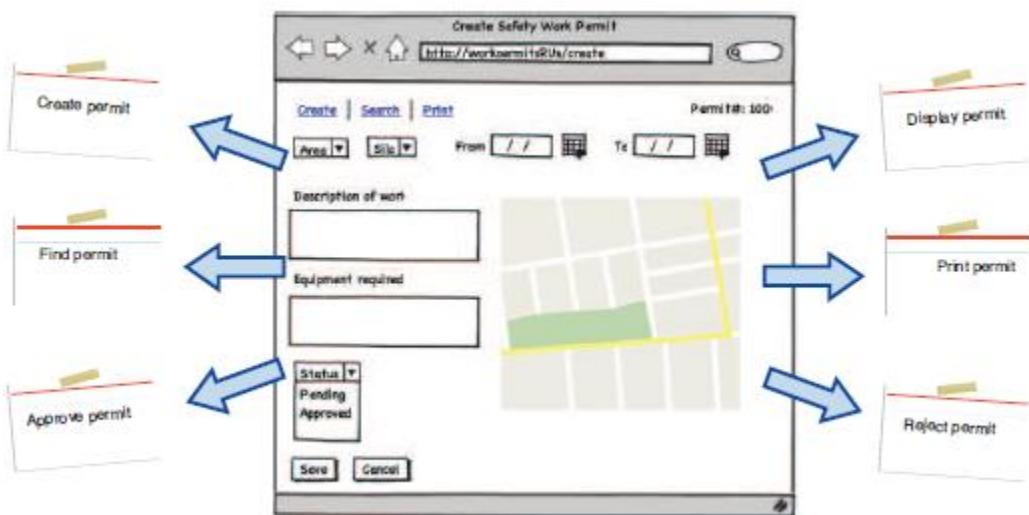
หนึ่งภาพแทนล้านถ้อยคำดังนั้นการเขียนภาพจะยิ่งทำให้เราเข้าใจอะไรได้ง่ายขึ้น เช่นการเขียน flow chart สามารถช่วยเราเข้าใจเรื่องเกี่ยวกับกระบวนการทำงานของระบบ System Maps และ Information Architecture จะช่วยให้เราสามารถจัดกลุ่มและแต่งงานของมาได้ Concept Design และ Paper Prototype เป็นวิธีทดสอบการใช้งานได้ระดับหนึ่ง แต่ว่าระหว่างทำต้องคิดอยู่เสมอว่าเราต้องการภาพในญี่ เราจะต้องไม่ลงไปในรายละเอียดให้มากเกินไป

## Step 3: Write Lots of Stories

หลังจากที่เราได้ diagram และภาพ มาแล้วจากการคุยกับ user เราจะเริ่มดึง user story ของมา



ถ้าเราได้ screen มาเราก็สามารถดึงเอา user story ของมาได้เท่านั้นโดยการดึง functionalities ต่างๆของ screen ออกมานะ



จาก flow-chart, lo-fi paper prototype เราจะได้เรื่องราวหลักๆของงานที่เราต้องทำแล้ว และเมื่อเราต้องดึงเอา user story ออกมานามาต้องคิดเสมอว่า user story ต้อง เล็กๆ, เป็นอิสระต่อกัน, ตอบโจทย์ได้แบบ end-to-end แต่ถ้า story นั้นทำไปก็ใหญ่ เราจะไม่เรียนมันว่า story เราจะเรียกมันว่า epic และ epic ก็คือ story ขนาดใหญ่ที่ต้องลงแรงประมาณสองอาทิตย์ เรายังการ์ดเหล่านี้ว่า Epic (การ์ดยกษัยกิจเวลาเป็นสัปดาห์ๆ) Epic เอาไว้ทำ high-level planning หรือเอาไว้เก็บงานใหญ่ๆที่ เรารู้ว่า น่าจะต้องทำแต่ยังไม่ชัวร์ ถ้าเอօวมีงานลักษณะนี้ (ใหญ่ๆ ยังไม่ชัวร์) ขอเอօวจงทำมันเหมือนการ์ด defect (report erratum, คือจดๆไว้คร่าวๆว่ามันเป็นยังไง แต่ยังไม่ต้องไปลงรายละเอียด) แล้วค่อยมาแทรกมันเมื่อถึงเวลาที่จะทำ

สุดท้ายแล้ว high-level การ์ดประมาณ 10 - 40 ใบจะเพียงพอที่จะเติมช่วงเวลา 6 เดือนใน plan ถ้าการ์ดขึ้นหลักๆอย่างเดียวจะก็แสดงว่าเชอร์วางแผนใกล้ไปหรือไม่ก็คงรายละเอียดเยอะเกินไป จำไว้ว่าเราทำสิ่งนี้เพื่อให้เราสามารถนำมันไปใช้ได้จริง ฉะนั้นเชอร์ว่างอย่าทำลึกเกินไป ไม่งั้นเสียเวลาเป็นสีปดาห์ๆกว่าจะได้ผลลัพธ์ตามนัดนี้จะหาว่าหล่อไม่เตือน

## Step 4: Brainstorm Everything Else

มาถึงขั้นนี้เราก็จะได้ของที่ต้องการมาทั้งหมดแต่ยังไม่หมดเราได้ user story ที่เป็นตัวบอกร่วมกับสิ่งที่ user ต้องการคืออะไรแต่สิ่งที่เราต้องทำยังต้องมีอีก เช่น data migration, load testing, เอกสาร SOX compliance, เอกสารสำหรับ support, เอกสารสำหรับการ train, เอกสารสำหรับ UAT และอื่นๆอีกมากมายเรื่องเหล่านี้ยังต้องถูกคิดเสนอและเราต้องให้ความสำคัญกับสิ่งเหล่านี้เท่ากับงานอย่างอื่นที่เราต้องส่งตอนจบโครงการ ถึงขั้นตอนนี้เป็นช่วงเวลาที่ดีที่เราจะคาดการณ์ได้ว่าเรายังต้องการพัฒนาบ้านมากแค่ไหนให้จับเพื่อให้การทำโครงการนี้สำเร็จได้ด้วยดีดังนั้นอย่ารีรอที่จะเขียนอะไรมากตามที่คิดออกไว้ให้ชัดเจน

## Step 5: Scrub the List and Make It Shine

หลังจากที่เราได้ list ของสิ่งที่ต้องการในทุกๆมุมมาอย่างเพียบ ตอนนี้ได้เวลาที่เราจะต้องมาันั่งหน้าแล้วว่าอะไรที่มันเข้า อะไรที่ยังขาดอยู่ จัดกลุ่มงานที่คล้ายๆกันไว้ด้วยกัน อะไรที่เขียนแล้วเข้าใจยากก็เขียนใหม่ให้มันเข้าใจง่ายๆ เมื่อทำสิ่งเหล่านี้เสร็จก็ได้เวลาที่เราจะทำ project plan แล้ว ก็จะส่องประกายเจิดจรัส

**ศิษย์:** ท่านอาจารย์ถ้าเข้าใจไม่ผิด การพูดคุยแบบเห็นหน้ากันนั่งใกล้ๆกันเป็นเรื่องที่ดีและมีประสิทธิภาพที่สุด ดังนั้นข้าต้องให้เวลาให้มากที่สุดในการนั่งคุยกับลูกค้าเพื่อหาว่าลูกค้าต้องการอะไร

**อาจารย์:** ถูกแล้ว

**ศิษย์:** นั้นแปลว่าข้าไม่ต้องเขียนอะไรมากในขณะที่เก็บ requirement?

**อาจารย์:** โว (แสรดดดดดด นี่ก็เกรียนเกินไป) การไปนั่งคุยไม่ได้แปลว่าเอ็งไม่ต้องเขียนเอกสาร ทำเอกสารเท่าที่จำเป็น เอกสารนั้นเป็นเครื่องมืออันทรงประสิทธิภาพในการ share ข้อมูลระหว่างเรากับลูกค้า

**ศิษย์:** นั้นแสดงว่าเอกสารบางตัวก็ทำได้ใช่ไหมครับระหว่างเก็บ requirement

**อาจารย์:** ถูกแล้ว แทนที่เราจะมานั่งจอมทำเอกสารเราต้องเพ่งสนใจไปที่สิ่งที่ลูกค้าต้องการ เราต้องรู้ว่าควรจะทำเอกสารเมื่อใด

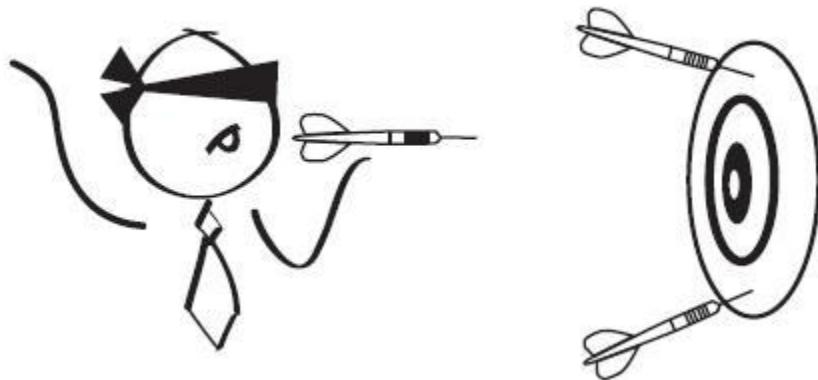
**ศิษย์:** ขอบคุณท่านอาจารย์

## What's Next?

Well done, amigo! Now that you see user stories are nothing more than short descriptions of features our customers would like to see in their software, you are one step away from being able to create your very first agile project plan. In the next chapter on estimation, we'll see how to size our stories so they can withstand the inevitable hiccups we encounter during delivery. So, onward and upward, as we demystify the art and science behind agile estimation.

# ตอนที่ 7 :ว่าด้วยการประเมิน :

## ศิลปะแห่งการคาดเดา



พร้อมพบกับความจริงอันน่าหวาดหวั่นอีกข้อที่เกี่ยวข้องกับการประเมินหรือยัง? เมื่องจากที่ผ่านมากการประเมินโปรดเจคแบบเดิมๆ ที่เราใช้นั้นนั้นห่วยแตกสิ้นดี มันไม่เคยเป็นจริง!!!

แต่เมื่อเวลาผ่านไปจากการประเมินแบบแอกล์ล์จะทำให้เราเลิกตามหาฟักทอง(ความต้องการและเที่ยงตรง ที่ไม่เคยเจอ)จากการประเมินแบบเดิมๆ แต่ในทางกลับกันเราจะหันมาสร้างแผนงานที่สามารถทำให้เราและลูกค้าสามารถทำงานร่วมกันอย่างมีความสุขและต่างเชื่อถือในสิ่งนี้

ดังนั้นในบทนี้เราจะได้เรียนรู้เกี่ยวกับการประเมิน user story และรวมไปถึงเทคนิคการประเมินอีกหลายแบบที่จะช่วยให้เรา gubern ขนาดของงานได้แบบพอดีอย่าง

## 7.1 ปัญหาของการประเมินเบื้องต้น

ปัญหาใหญ่ของภารกิจของเรา(พัฒนาซอฟต์แวร์)คือเรื่องของการทำการประเมินโครงการและความคาดหวังที่มีต่อการประเมินนั้น มักจะได้ผลลัพธ์ไม่ตรงกับความคาดหวัง

### The Point of Estimation

หลักการของการประเมินในการพัฒนาระบบงาน “ไม่ได้หมายถึงการทำนายผลสำเร็จของโครงการ มันเป็นการกำหนดเป้าหมายโครงการ ให้สะท้อนความเป็นจริง เพียงพอที่จะควบคุมโครงการให้สำเร็จตามเป้าที่วางเอาไว้”

—Steve McConnell, *Software Estimation: Demystifying the Black Art* (McC06)

มันไม่ได้มาจากการความผิดพลาดในการประเมินของเรา (แม้ว่าจริงๆแล้วมันจะผิดพลาดอยู่เสมอๆ lol ก็罢) แต่ความผิดพลาด มันเกิดมา “ความต้องการ” ของคนอื่นๆที่คาดหวังว่าจะได้แต่การประเมินนั้นมันไม่เคยให้ได้ นั่นคือ ความเม่นยำในการทำนายอนาคต

จบทันสั้น ช่วยส่งแผนการ  
ประเมินแบบละเอียดที่  
สามารถตอบสิ่งเหล่านี้ให้  
หน่อย



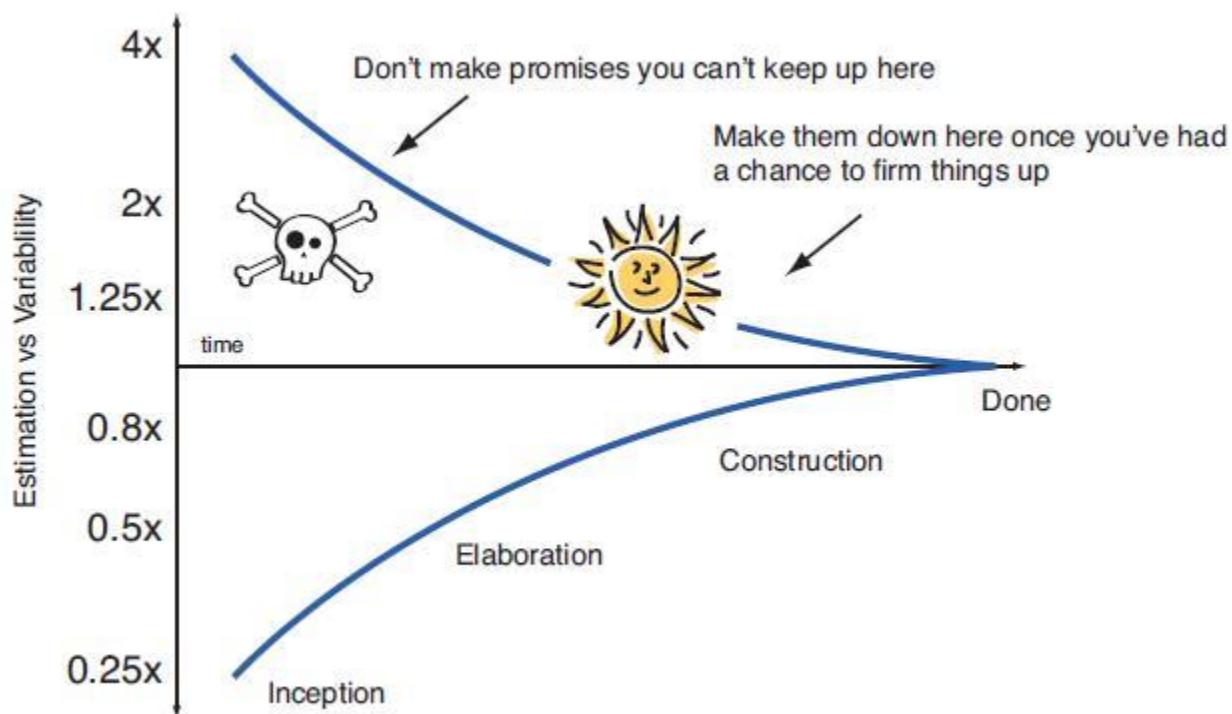
ระบบใหม่ ที่จะทำงานอยู่บนเทคโนโลยีใหม่กับทีมใหม่ ที่จะถูกสร้างขึ้นมาเพื่อตอบสนองแผนธุรกิจใหม่ ที่จะถูกสร้างขึ้นมาปีหน้า !!!!

หลังจากที่เราได้ทำการประเมินเบื้องต้นและทำงานกันไปสักพักก็ดูเหมือนว่า ผู้คนต่างหลงลืมความจริงที่สำคัญข้อหนึ่งไป นั่นคือ

## การประเมินเบื้องต้น (high-level estimates) เป็นเพียงการคาดเดา (และมันจะออกมาเลวร้าย เกินความจริงไปอย่างมากเสมอ)

และลิ่งที่เกิดขึ้นกับการทำงานจริงเสมอคือเรามักจะยึดมั่นเอาไว้การประเมินที่เกิดจากการคาดเดาสุดๆเหล่านี้มาเป็นคำมั่นสัญญาว่ามันจะต้องเกิดให้ได้ตามนั้น และนั่นก็คือที่มาของปัญหาอันใหญ่หลวง!!!

Steve McConnell ข้างว่า พฤติกรรมที่ผิดๆ ในทรงกรวยแห่งความไม่แน่นอน ( ภาพประกอบที่ 7.1 ในด้านล่าง ) ได้เดือนสติเราฯ ให้เห็นว่า ในช่วงต้นของการประเมินนั้น มีความเปลี่ยนแปลงมากถึง 400 % ในช่วงเริ่มต้นโครงการ และนั่นคือความจริงที่เรียบง่ายที่สุดเราสามารถกล่าวได้ว่า เป็นไปไม่ได้ที่การประเมินช่วงแรก (*up-front estimates*) จะให้ความถูกต้อง และเราจะต้องหยุดแล้วร่วงว่ามันมีความแม่นยำ




---

ภาพประกอบที่ 7.1 : ทรงกรวยแห่งความไม่แน่นอน แสดงให้เห็นถึง ความเปลี่ยนแปลงในช่วงต้นของการประเมิน มีความแตกต่างกันมากเพียงใด เมื่อเทียบกับช่วงอื่นๆ จนเสร็จสิ้นโครงการ

---

มีเพียงคำรามเดียว ที่การประเมินช่วงแรก สามารถให้คำตอบได้ นั่นคือ

## โครงการนี้มันเป็นไปได้หรือไม่? (เมื่อเทียบกับเวลาและทรัพยากรที่มีอยู่)

มีอะไรบ้างที่เราต้องการ เพื่อนำไปสู่การประเมินแผนงานตามด้านล่าง

1. เพื่อให้เรามีแผนงานสำหรับอนาคต
2. เตือนให้เราทราบว่าทราบว่าการประเมิน คือการคาดเดา
3. ตอบสนองให้เห็นความซับซ้อนในการพัฒนาระบบงาน

## 7.2 เปลี่ยนมะนาวให้เป็นน้ำมะนาว (Turning Lemons into Lemonade)

หลักแห่งแอจใจยอมรับว่าในช่วงแรกของโครงการ การประเมินเบื้องต้น ไม่มีความน่าเชื่อถือเพียงพอ แต่งานบางอย่างต้องถูกทำบนพื้นฐานความไม่เที่ยงตรงนั้นคือการจัดสรรจ์ทุนและความคาดหวังเป็นสิ่งที่ต้องกำหนดเพื่อให้โครงการเกิดขึ้น ดังนั้นต้องมีการสร้างอะไรสักอย่างที่เกี่ยวกับการประเมิน และให้อะไรสักอย่างนั้นต้องวัดผลได้ เพื่อใช้สิ่งนั้นในการตอบคำถามว่าจะใช้ระยะเวลาเท่าไร และใช้มันในการวางแผนเพื่อให้งานเดินต่อไปข้างหน้า เพื่อกวนนี้ เรายังคง 2 อย่าง

- Story ที่ถูกกำหนดขนาดให้ สัมพัทธ์ (relatively) กับ Story อื่น
- ระบบ point-based เพื่อการตรวจสอบความก้าวหน้า

ไปดูกันว่า แต่ละอย่าง มีรายละเอียดและจะช่วยในการวางแผนของเราได้อย่างไร

### ขนาดสัมพันธ์ (Relative Sizing)

ลองจินตนาการว่า เราใช้เวลา 10 วินาที ในการเดี่ยวคุ้กกีช็อกโกแลตชิป 1 ชิ้น และเราจะลองประเมินว่า ถ้าเราสถาปัม กองคุ้กกีขนาด 7 และ 14 ชิ้น (รวมถึงนมแก้วโต) เราจะประเมินออกมีได้ว่า?

If it takes 10 secs to eat  
one of these ...

how long should it take to  
devour these?



Om nom nom



7 cookies



14 cookies

10 secs

? secs

? secs

ในตอนนี้ เรายังคงนึกถึงบางสิ่ง บางสิ่งที่เรียบง่าย แต่บางที่เราไม่เคยลองทำมาก่อน เราจะใช้เวลาในการดำเนินการ งาน 4 งาน  
ง่ายๆ นี้ได้อย่างไร?

Roll a snake-eyes (two ones)  
three times using two die

? secs

Blow up six birthday balloons

? secs

Find the two missing cards  
in a deck of cards

? secs

Build a two-story house  
of cards

? secs

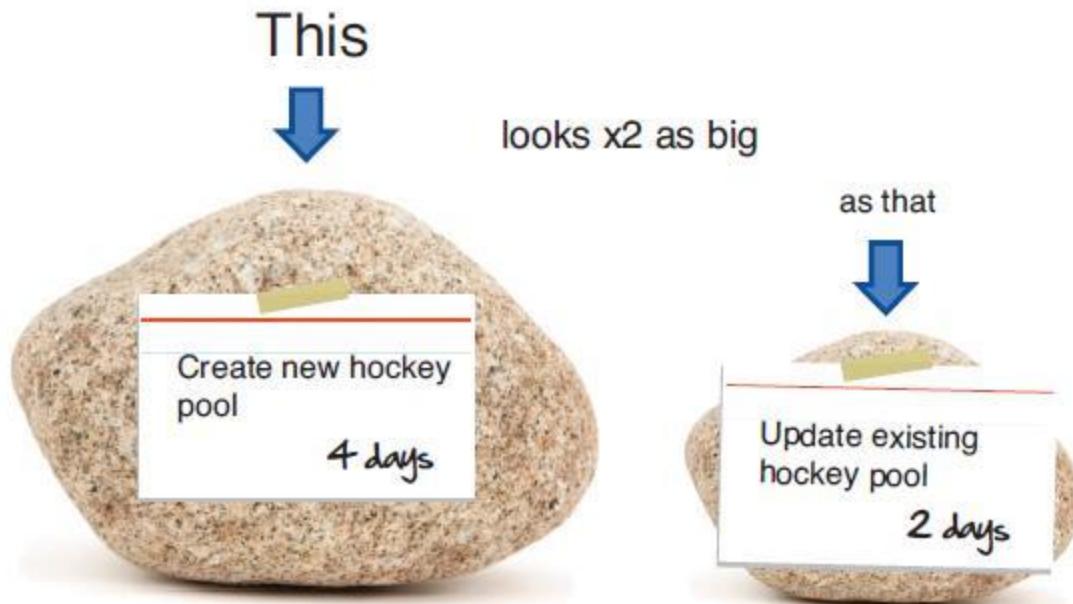
ถ้าเราเป็นเหมือนคนทั่วไป เราจะพบว่า การประเมินคุ้กคิ้วที่มีความสัมพัทธ์กัน ง่ายกว่า และงานอื่นๆ ยากกว่าอย่างแน่นอน

If one cookie = 10 sec

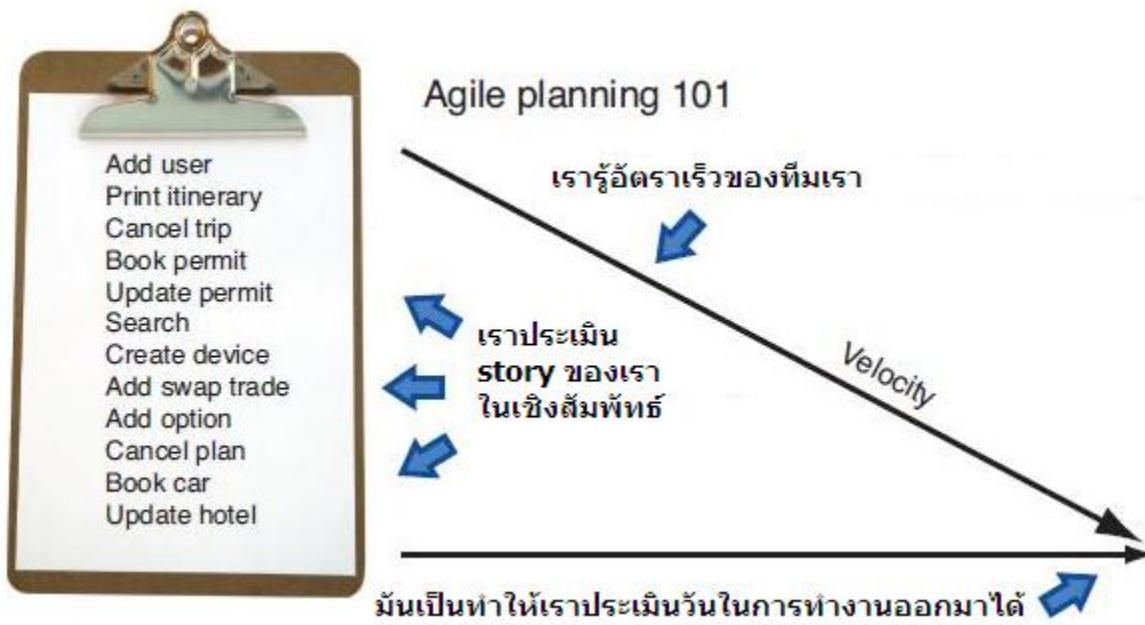
then seven cookies =  $10 \text{ sec} \times 7 = 70 \text{ sec}$

and fourteen cookies =  $10 \text{ sec} \times 14 = 140 \text{ secs} > \times 2 \text{ as big}$

ความแตกต่างระหว่าง แบบฝึกหัดทั้ง 2 นั้นคือ คุ้กคิ้นน์ เราสามารถประเมินได้จากความสัมพัทธ์ด้วยการจารนับจำนวน ได้อย่าง  
แน่นอนเป็นอีกครั้งที่วิทยาศาสตร์แสดงให้เห็นว่าการประเมินจากความสัมพัทธ์มีบางอย่างให้มนุษย์เราทำได้ค่อนข้างดี เช่นเวลา  
ที่เรามีวางแผน 2 ก้อนตรงหน้า เราสามารถบอกได้อย่างถูกต้องว่าก้อนไหนได มีขนาดใหญ่กว่ากัน



จากภาพก้อนหินเราพอจะประเมณได้ว่า ก้อนไหนใหญ่กว่า แต่ปัญหาคือการบอกว่า ไข่ที่มันใหญ่กว่าจะใหญ่กว่ากี่เท่า? จากตัวอย่างการกำหนดภาระก้อนหินข้างบนที่แสดงจะรวมด้านนี้เป็นเหมือนเดิมดังแนวคิดที่แสดงสำคัญของการประเมินและการวางแผนแบบแอจайл์ส่วนการที่เราสามารถคาดคะเนงานทั้งหมดให้สัมพัทธ์กันได้นั้นจะทำให้เรารู้ว่าเราสามารถทำงานได้เร็วขนาดไหน และนี่คือจุดเริ่งต้นของการประเมินแบบแอจайл์



ตอนนี้ ความท้าทายของเรากับการประเมินแบบสัมพัทธ์

คือ 1 วันในการประเมิน 'ไม่ได้เท่ากับ 1 วันทำงานจริงๆ'

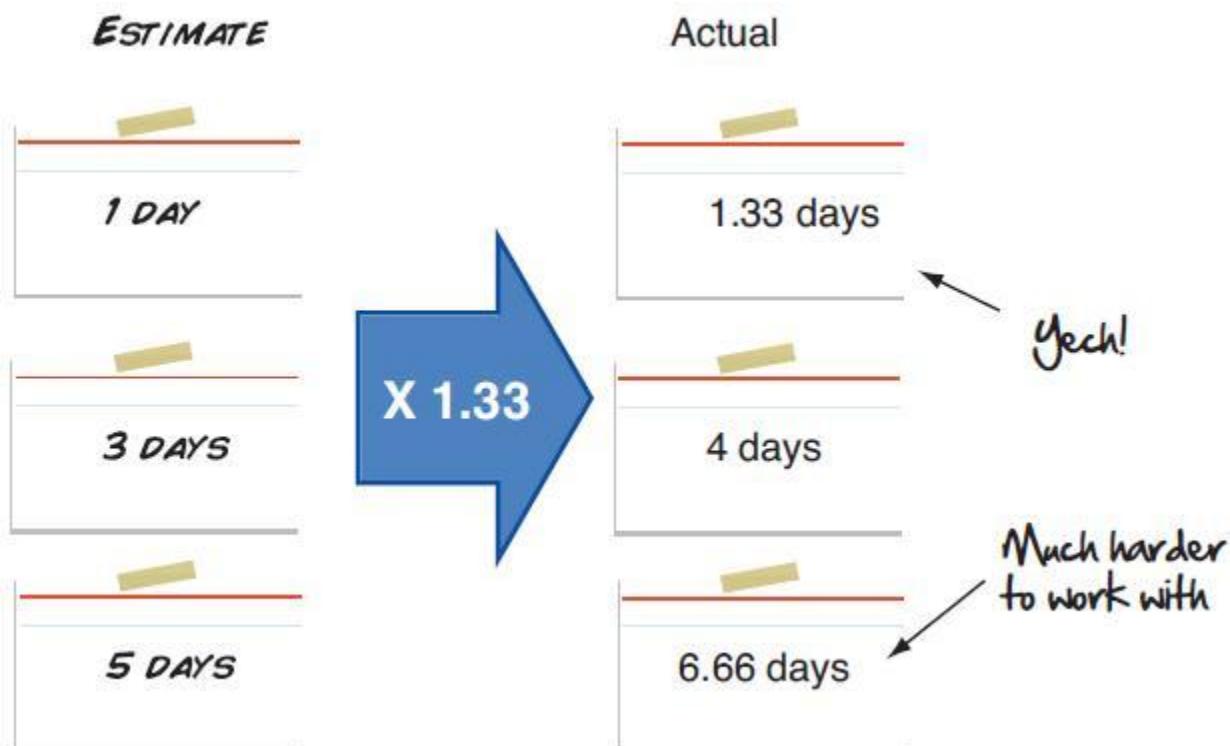
ดังนั้นทีมจะทำงานข้ากกว่าหรือเร็วกว่าการประเมินดั้งเดิมของเจาเสมอๆ ดังนั้นเพื่อหลีกเลี่ยงปัญหาที่จะเกิดขึ้นคือเรื่องของการย้อนกลับมาทำการประเมินใหม่อีกอย่าง การประเมินแบบแอจไลต์จะมีวิธีการประเมินอีกแบบคือการประเมินแบบแยกแต้ม

## ระบบ Point-Based (Point-Based Systems)

ระบบ Point-Based ทำให้เราสามารถตรวจสอบความก้าวหน้า และประเมินความสัมพันธ์โดยไม่ต้องกังวลว่าความเป็นจริงจะเป็นอย่างไร เมื่อเทียบกับสิ่งที่เราประเมินยกตัวอย่าง การประเมินเรื่องราวดูเดิมๆ เราใช้หน่วยเป็น 3 วัน ในขณะที่ความเป็นจริงมันจะลงที่ 4 วัน



เราลองปรับแต่งการประเมินของเรา เพิ่มขึ้นอีก 33 % ตามด้านล่าง



แต่ครกันเล่า? จะทำงานกับตัวเลขอย่าง 1.33 และ 6.66 วันได้? ไม่เพียงความแม่นยำจะผิดไปจากความรู้สึก แต่อะไรที่เราจะทำ ในเวลาหลังจากการส่งมอบงาน เรื่องราวนั้น จากที่เราประเมินไว้ 1.33 กล้ายเป็นใกล้กับ 1.66? จะปรับแต่งอีกรอบกันหรือ? เพื่อหลักหนี้ไปจากค่าคงที่ การเต้นไปมาของตัวเลขอย่างไม่รู้จบ แอจайл์ส่วนน้ำให้เราหยุดการประเมินไว้บันทึกฐานของความเรียบง่าย ใช้งานสะดวก และไม่มีข้อติดกับเวลาบนปฏิทิน



ด้วยระบบ Point-Based หน่วยวัดของเรามาไม่สำคัญอีกต่อไป การวัดจะใช้สิ่งที่สมพթกัน ไม่ใช่สิ่งที่เป็นค่าคงที่สมบูรณ์



เราทุกคนลองเก็บเกี่ยวๆ ความใหญ่โต (bigness) ของแต่ละงานกับจำนวน และขนาดที่มันสมพதกันกับเรื่องอื่นๆ ถ้ามันช่วยได้ เราลองมาคิดถึงการประเมินแบบแอจайл์ และลองที่จะจัดเรียงเรื่องราวด้วยในรูปแบบของขนาดเดือยีด เล็ก กลาง และใหญ่ ยิ่งกว่านั้น เราลองจับประเด็นจากการประเมินของเรา เมื่อสิ้นสุดในแต่ละวัน มันไม่สำคัญอีกต่อไป ตราบเท่าที่เราเรียงขนาดของเรื่องราวด้วยกันในทุกๆเรื่อง ในทุกๆเรื่องที่เราประเมินมากเกินไป มันก็จะมีเรื่องที่เราใช้เวลาน้อยไป เช่นกัน ดังนั้นแล้ว ผลทุกอย่างจะออกมาในที่สุดการใช้งานระบบ Point-Based ตามขั้นตอนลำดับเวลา และเรียนรู้ที่จะแสดงว่าเราได้อะไรรากๆ กลับมา ตามนี้

- เตือนให้เราทราบว่าการประเมินเป็นการคาดเดา
- ใช้การวัดจากขนาดบริสุทธิ์ (pure size) (มันไม่เสื่อมคลายไปตามกาลเวลา)

- เร็ว และเรียบง่าย



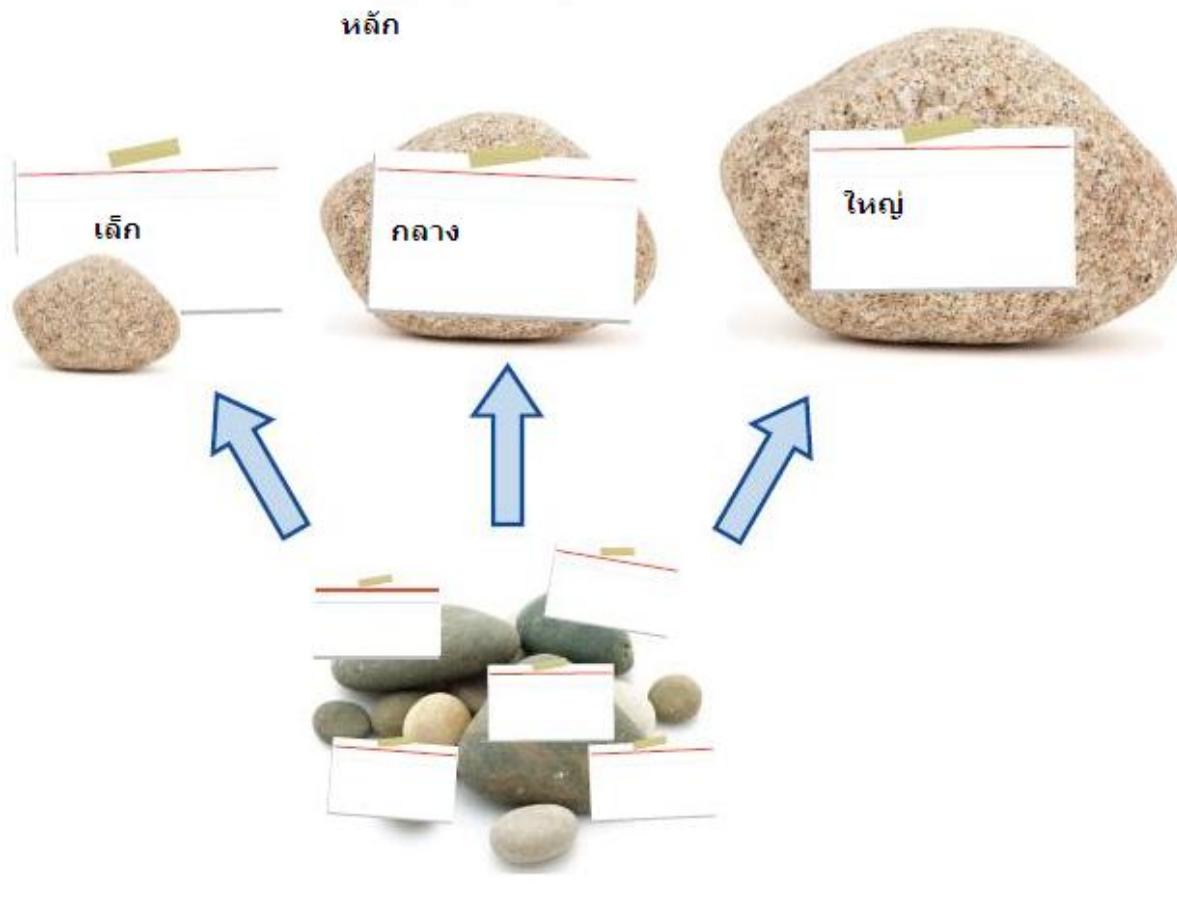
โคนจับได้แล้ว!! ก่อนที่เราจะมาถึงเรื่องวิธีการประเมินแห่งแอจайл์ด์ ในบทก่อนๆ ในเนื้อหาที่มีการเกี่ยวเนื่องกับการประเมินเวลา จะให้หน่วยของวัน (days) ในขณะที่จริงๆ แล้ว เราควรจะใช้เป็น หน่วย (points) และทำไม่ให้ “วัน” เราทำมันด้วย 2 เหตุผล เหตุผลแรก เราไม่เคยลองโอกาสที่จะพูดคุยกันกับแนวคิดของการประเมินโดยใช้ “หน่วย” ในภาระ และเหตุผลที่สอง มีบางแอจайл์ด์ที่มีการประเมินในหน่วยของวันอยู่ แต่วันของพากษาเรียกว่า วันในอุดมคติ (ideal days) วันในอุดมคติเป็นการวัดในรูปแบบที่ต่างออกไป วันในอุดมคติเป็นวันที่ไม่มีเหตุการใดๆ มาขัดขวางให้การทำงานเต็มเวลา 8 ชั่วโมงต่อวันดำเนินต่อไปได้อย่างปกติสุุ แต่นอน เราไม่เคยที่จะได้วันในอุดมคติในการทำงาน แต่บางทีมพบว่าแนวคิดนี้ใช้งานได้ดีวันในอุดมคติสามารถใช้ในการทำงานได้ แต่เรา秧จะแนะนำให้ยึดหลักของการใช้ “หน่วย” โดยมาก เพราะว่ามันทำให้ความจริงของการประเมินการมีความชัดเจน แต่อย่างไรก็ตาม การวัดด้วย “หน่วย” ทำให้เราไม่ต้องกังวลว่า วันในอุดมคติ ไม่เท่ากับความเป็นจริงอีกด้วย พักซักนิดกับหนังสือ อย่าเพิ่งตกใจถ้าเราเห็น “หน่วย” แทนที่จะเป็น “วัน” เรา秧ยึดหลักว่า “หน่วย” เป็นการลิ้งย้ำเตือนของหนังสือ แต่ถ้าเห็นคำว่า วัน นั่นหมายความถึงสิ่งเดียวกัน

### 7.3 มันทำงานได้อย่างไร? (How Does It Work?)

เนื้อหาในส่วนนี้จะเป็นส่วนของการที่จะใช้มันจริงๆ แล้ว เราเมื่อเทคนิคการประเมินร่างๆ 2 แบบ สำหรับคุณและทีมงาน เพื่อจะใช้ในการประเมินขนาดของ story ให้เหมาะสมกับภาระงานแผนงานตามวิธีแห่งแอจайл์ด์

## การทำให้เป็นสามเหลี่ยม (Triangulation)

Triangulation เกี่ยวกับการสร้างเรื่องราวตัวอย่างเด็กๆ เพื่อให้เข้ากังของ และกำหนดขนาดให้กับเรื่องราวนั้น ให้สัมพัทธ์กัน



เปรียบเทียบกับขนาดของเรื่องราวยังเหลืออยู่

มาดูตัวอย่างกัน ที่นี่คือร้านจัดภัณฑ์ที่พึงจะจัดจ้างระบบสินค้าคงคลังตัวใหม่ เราทำการบ้านเสร็จเรียบร้อยแล้ว และสร้างรายการที่ได้สำหรับเรื่องราวของผู้ใช้ (user - stories) และในตอนนี้ฝ่ายการประเมินจะเข้าไปป่วยวเหลือเข้าได้ที่ไหนบ้าง?

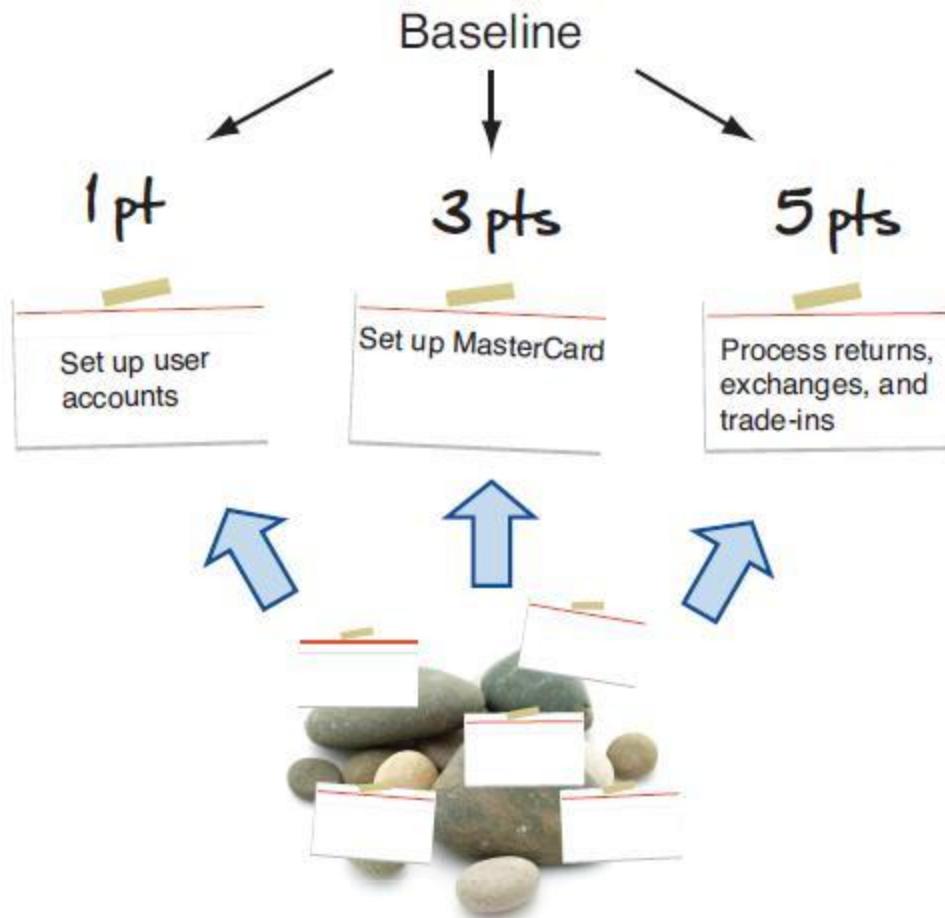
## Mike's Bike Emporium      Need a bike? Talk to Mike!



เริ่มเรียนรู้จากการรายงาน และดูว่ามีอะไรบ้างที่เป็นตัวเลือกที่ดี ที่จะเป็นเรื่องราวอ้างอิง ตามอุดมคติ เราต้องการบางอย่างที่ เล็ก บางอย่างที่มีขนาดกลาง และบางอย่างที่มีขนาดใหญ่พอก็จะสรุปใน 1 รอบการทำงาน (iteration) (โดยปกติจะใช้เวลา 1 ถึง 2 สัปดาห์) เราสามารถวิเคราะห์ได้จาก

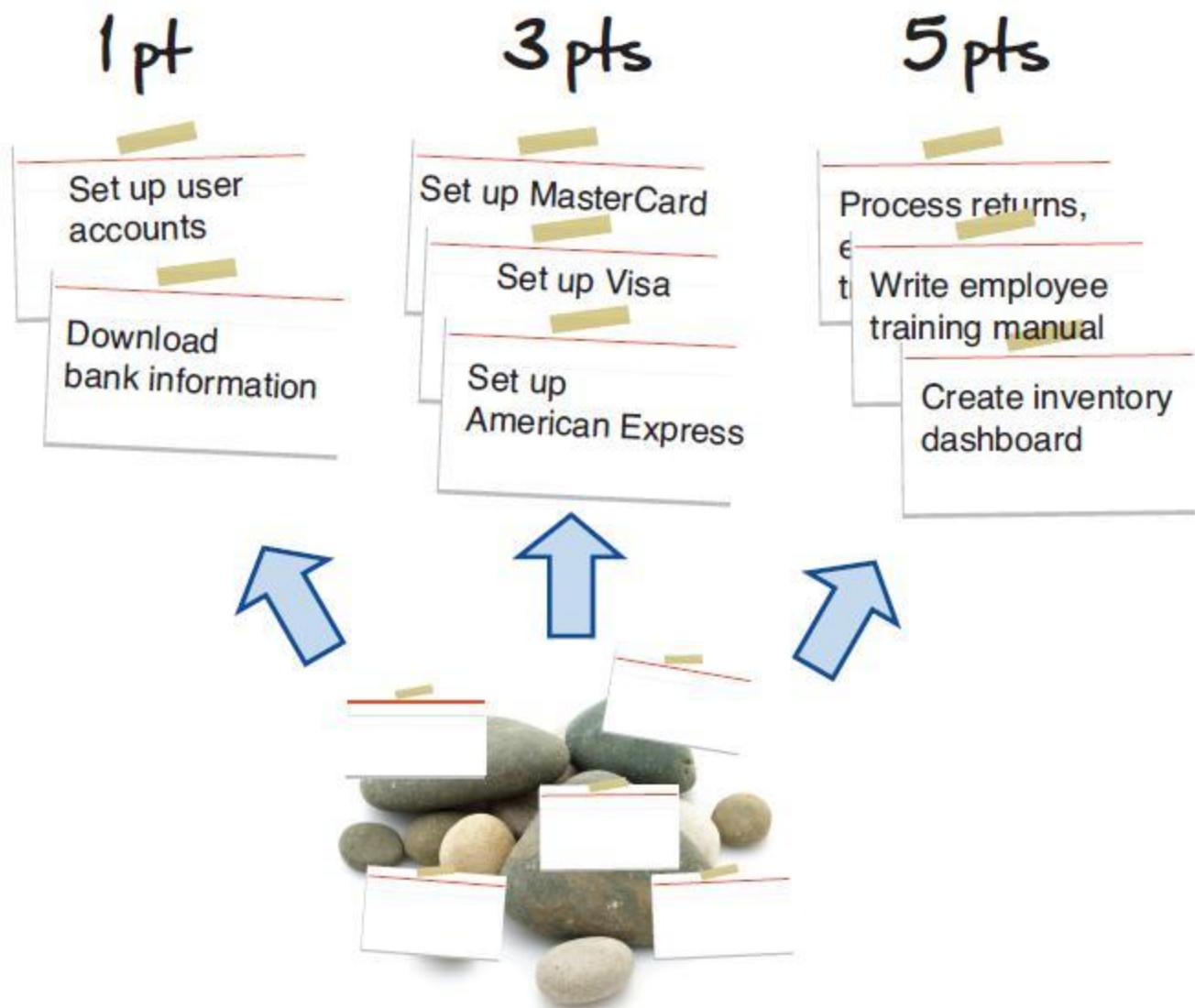
- ตระหง่านในการจัดกลุ่ม (Locical groupings)
- เรื่องราวดังแต่ต้นจนจบ (เพื่อแสดงสภาพของโครงสร้างจริงๆ)
- อภิปรายตามที่จะมาเป็นตัวอย่าง ที่เราเห็นว่าเกิดขึ้นในวงจรชีวิตของโครงการ

ในตอนที่เราค้นหาเรื่องที่จะมาเป็นตัวเลือก หลังจากตรวจสอบในรายการแล้ว มาดูกันว่าเราตัดสินใจเริ่มด้วย 3 เรื่องราวอ้างอิง (reference stories) ต่อไปนี้



### Remaining stories to be sized

ในตอนนี้ เราจะมีบางสิ่งสำหรับเปรียบเทียบแล้ว เราสามารถดำเนินการกับเรื่องที่เหลืออยู่และกำหนดขนาดให้บวกจากตัวเลือกที่มีอยู่



มาถึงตรงนี้อาจมีข้อสงสัยว่าจริงๆแล้วเราสามารถทำการกระหนาดของ story ใหม่ได้ไหม? คำตอบคือได้ เพราะคงไม่มีความสามารถอะไรได้ถูกทั้งหมดภายในครั้งเดียว ดังนั้นถ้าเราต้องการวิเคราะห์กระบวนการใดมันใหม่ให้ถูกต้องมากขึ้น แต่เมื่อเราทำการกระหนาดใหม่เรียบร้อยแล้วเราก็ควรที่จะไม่ไปยุ่งกับมันอีกปล่อยมันไว้อย่างนั้น เพราะถ้าเราวนเวียนเข้าไปเปลี่ยนงานบ่อยๆสิ่งที่จะเกิดขึ้นคือเราต้องมานั่งคำนวณอัตราเร็วของทีมใหม่ ซึ่งจะสร้างความสับสนอย่างมากเนื่องจากเราจะมีอัตราเร็วที่ไม่เท่ากันในการทำงาน

Also, if you ever run into something you've never done before and you don't know how to size it, do a spike. A spike is a time-boxed experiment where we do just enough investigation to come up with an estimate and

อีกเรื่องที่น่าสนใจเกี่ยวกับการ估算คือเมื่อได้ก็ตามที่เราเจอกับปัญหาหรืองานที่เราไม่เคยทำมาก่อน เราจะ估算ได้มั่นใจ? สิ่งที่เราต้องทำคือการทำ Spike แล้วไ้อี Spike มันคืออะไรถ้าเราแบบสั้นๆ มันคือการสละเวลาเพื่อค้นหาและทดลองเพื่อหาดูว่า ปัญหานั้นรูปแบบที่เราไม่เคยเจอนั้นจะมีผลกระทบอย่างไรแบบคร่าวๆ แล้วหยุด

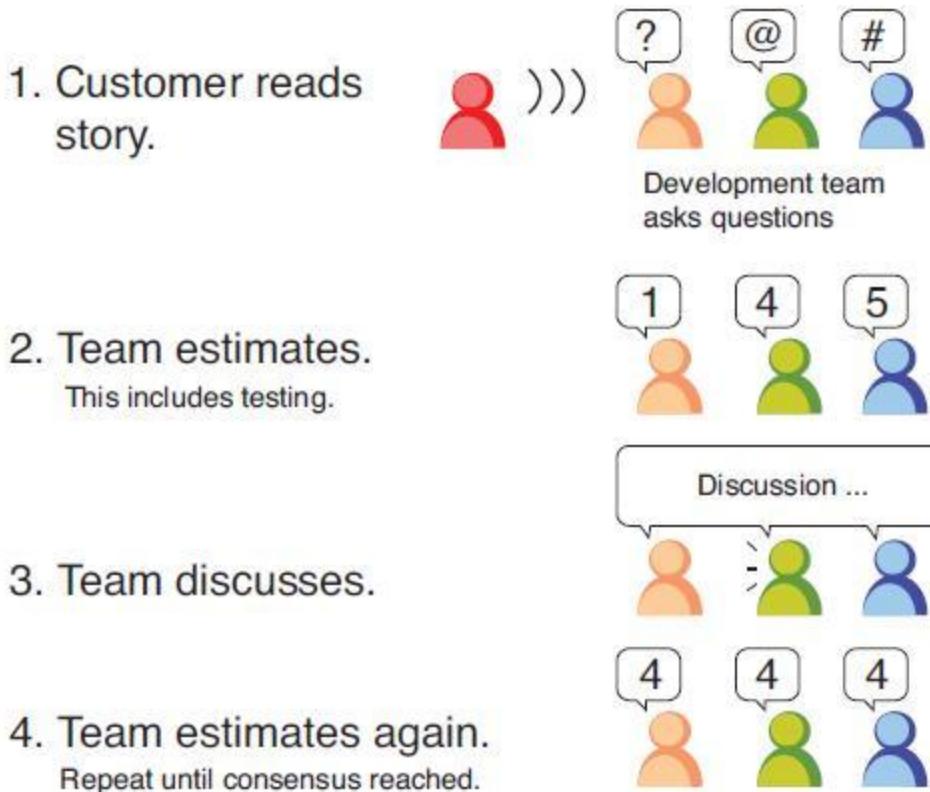
### The Wisdom of Crowds

James Surowiecki เขียนเรื่อง The Wisdom of Crowds เล่าไว้ ในปี 1906 ที่อังกฤษนักวิทยาศาสตร์ชื่อ Francis Galton ตั้นตะลึงกับผลลัพธ์ที่ได้จากการทดลอง ที่เขาดำเนินการที่งานริบบิ่งประจำเมือง (country fair) ความคาดหวังมีอยู่ว่า นักแล่นคุณเขี่ยราบจะมีความถูกต้อง ในการเดาจำนวนนกของวัวที่ถูกแล่ มากกว่าคนทั่วไป เช่นแปลกใจและตกลงใจกับสิ่งที่พูดได้จากผู้คนทั่วไป (simpleton) (ที่มีประสบการณ์เพียงเล็กน้อย ถึงไม่มีเลยกับการแล่นนี้) ซึ่งไม่เพียงแต่การเดาจำนวนนักสุดท้ายของสัตว์ได้เท่านั้น แต่ความแตกต่างที่เกิดขึ้นนั้น มีเพียงแค่ 1 ปอนด์ เมื่อเทียบกับนักล่าเนื้อผู้มีประสบการณ์ ความเชื่อที่ถูกแสดงให้เห็นว่าผู้ชายของ Sir Francis นั้นแสดงให้เห็นว่า ผู้เขี่ยราบจะให้ความถูกต้องเสมอ และ มีผลสำเร็จที่มากกว่าจากผู้คน ในเวลาที่เราเล่น planning poker มันคล้ายกับการที่เราควบคุมการตัดสินใจของกลุ่มคน ด้วยการเอาใจใส่กับการประเมินของเรา เราพนันกับกลุ่มคนที่จะช่วยให้ได้ผลลัพธ์การประเมินที่ดีกว่า ควรหนึ่งจะทำมันด้วยตัวคนเดียว

Spike โดยปกติจะทำกันไม่เกิน 2 วันและเป็นหนทางที่ยิ่งใหญ่ ในการที่จะลองบางอย่างให้มีผลของการเร็วๆ และมีข้อมูลเพียงพอที่จะบอกเล่าลูกค้าให้ทราบได้ว่า จะใช้เวลาเท่าไหร่ เขาเหล่านั้นจะได้ตัดสินใจความคุ้มค่าจากการลงทุนที่จะเกิดขึ้น ก่อนที่เราจะสรุปโดยทั่ย ยังมีเครื่องมืออื่นหนึ่งอย่าง ที่ควรรู้เกี่ยวกับการทำ ประเมินจากทีม (team-based estimation) และสร้างความคิดเห็นส่วนใหญ่ มันเรียกว่า *planning poker*

### Planning Poker

Planning Poker เป็นเกมส์ที่ทีมพัฒนาใช้ประเมิน story ตามแต่ละบุคคลก่อน (ใช้สำรับไพ่ที่มีหมายเลข คล้ายๆ กับ 1, 3 และ 5 หน่วย บนการ์ด) จากนั้นจึงนำมาเปรียบเทียบผล ที่รวมรวมมาด้วยกัน ถ้าทุกคนประเมินออกมาร้าวๆ กัน การประเมินจะถูกบันทึกไว้ ถ้ามีความแตกต่างกัน ทีมจะต้องพูดคุยกันและประเมินร่วมกันจนกว่าจะได้ข้อสรุป



Planning Poker จะใช้งานได้ดี เพราะว่าคนที่จะทำการประเมินนั้นเป็นคนกลุ่มคนทำงานชื่ร่วม นักพัฒนา, DBAs, นักออกแบบ, นักเขียนข้อมูลทางเทคนิค (technical writers) และครุภัติที่มีหน้าที่รับผิดชอบสำหรับการส่งมอบ ดังนั้นเราจะเป็นว่าการจะขนาดแบบนี้นั้นมันทรงพลังมาก เพราะมันจะเต็มไปด้วยบรรยากาศแห่งการทำงานที่ร่วมกันโดยเฉพาะในเวลาที่บังคับเสนอว่า เรื่องงานนั้นๆ จริงๆ แล้วมีขนาดเล็ก และบางครั้งกลับมองว่าจริงๆ แล้วมันใหญ่ มันไม่สำคัญว่าใครจะเป็นคนถูกหรือผิด (ซึ่งมันจะถูกจัดเรียงด้วยตัวมันเองในที่สุดอยู่ดี) คุณค่าของการหารืออยู่ที่การหาข้อสรุปสุดท้าย และนั้นคือสิ่งที่สำคัญจริงๆ เพื่อที่จะให้กระบวนการ planning poker ไม่ใช่ระบบของ (คูดีๆ นักพัฒนาจะต้องจับฉี่นี่ย์ 3 คน ไม่สามารถชนวนการให้หัวจากผู้เขียวชาญ 1 คน) แต่เม้นเป็นหนทางที่ให้บุคคลแสดงความคิดเห็น เพื่อที่จะไปยังการประเมินที่ถูกว่าในที่สุดและอย่างหลังผิดไปกับการใช้สำหรับไปจริงๆ ที่มีตัวเลข เช่น 8, 13, 20, 40 และ 100 เราไม่ต้องการมันขนาดนั้นหรือการทำให้มันเรียบง่าย ขนาดของเรื่องราวเล็กๆ (1, 3, และ 5 หน่วย กับมหาภัยในบางเรื่อง) และหลีกเลี่ยงความรู้สึกผิดปกติของความแม่นยำและสิ่งรบกวนที่ตัวเลขจะสร้างมันขึ้นมา



## Master Sensei and the aspiring warrior

**Padawan :** ท่านอาจารย์ เป็นความจริงหรือ? ที่วิถีแห่งแอจайл์ ไม่สนใจความถูกต้อง ในการประเมิน และทุกสิ่งอยู่ที่การกำหนดขนาดสัมพัทธ์ (relative sizing)?

**Master :** เมื่อถึงเวลาที่ต้องประเมิน ลิ่งหนึ่งที่เจ้าควรจะรำลึกไว้คือ จงประเมินอย่างดีที่สุดเท่าที่จะเป็นไปได้ ขอให้ตั้งแต่นี้ต่อไปความเข้าใจผิดๆ ในวิถีแห่งแอจайл์ ว่าไม่สนใจในความแม่นยำ จะไม่เกิดขึ้นกับเจ้าอีก

**Padawan :** ดังนั้นแล้ว เราควรจะพุ่งประเด็นไปทั้ง ความถูกต้องและความสัมพัทธ์ เมื่อต้องประเมินเรื่องราว อย่างนั้นหรือ?

**Master :** ถูกต้องแล้ว จงประเมินอย่างถูกต้องเท่าที่เจ้าสามารถ จงเข้าใจว่าเจ้าไม่ได้มีเพียงความแม่นยำ ตราบเท่าที่ที่เรื่องราวของเจ้าถูกกำหนดขนาดให้สัมพัทธ์กัน และเราวัดผลงานของพากพ้องได้ แผนงานของเรามีความแน่นอนและส่องสว่าง ดุจดวงอาทิตย์

**Padawan :** ดังนั้นแล้ว ศิษย์ควรประเมินอย่างที่ดีที่สุด แต่ให้เวลาอีกกว่า ในการระบุขนาดของเรื่องราวให้สัมพัทธ์กับ เรื่องราวอื่นๆ กระนั้นหรือ?

**Master :** เป็นเช่นนั้น การเสียสละแรงเพียงเล็กน้อยนี้ จะเกิดขึ้นตลอดบนหนทางแห่งการประเมิน จงอย่าอาศัยอยู่บนความไม่แม่นยำจากการประเมินของเจ้า จงระบุขนาดของเรื่องราวให้มีความสัมพัทธ์กัน ยอมรับในลิ่งที่มันควรจะเป็น และกำหนด ความคาดหวังร่วมกันในที่สุด

**Padawan :** ขอบพระคุณ ท่านอาจารย์ ศิษย์จะนำไปโปรดครวญ ให้ลึกซึ้งยิ่งขึ้น

### ลำดับต่อไป

ขอแสดงความยินดี!! เราได้เรียนรู้ กระบวนการ การประเมินเรื่องราวสัมพัทธ์ โดยใช้ระบบ Point-based เราเมื่อถูกอย่างพร้อมแล้ว ที่จะเป็นต่อการสร้าง แผนงานในแบบแอจайл์ ชั้นแรกการวางแผนโครงการโดยแบบแอจайл์ เราจะมีเครื่องมือทุกอย่างที่จำเป็นต่อ การพยากรณ์ การติดตามผลงาน และการสร้างแผนงานที่เราและลูกค้าของเรารู้สึกได้ร่วมกัน หลังจากนั้น ด้วยแผนงานในมือ และกองการ์ดเริ่มต้นข้างๆเรา (inception deck) เราพร้อมแล้วที่จะดำเนินการส่งมอบเนื้อต้มมันให้กับลูกค้า ด้วยการทำเนิน โครงการในแบบแอจайл์ พลิกหน้าต่อไปเพื่อเรียนรู้ถึงความลับในการวางแผนโครงการ ตามวิถีแห่งแอจайл์



# ตอนที่ 8 (DRAMATICA)



เขาวาให้ชน ไก่น้องเอี่ย กวักของ เมอร์ฟ (คืออะไรก็ไม่รู้) ใช้จับนักโทษไม่ได้มีเงื่อนไขกับ disrupting the best-laid plans ดังนั้นถ้าเราไม่มีแผนการรับมือกับ change ที่จะเกิดระหว่างทำโครงการฯ กำลังเดินหน้าเข้าหาความพยายามอย่างซ้ำๆ ในบทนี้เราอาจจะได้เรียนรู้รายละเอียดเกี่ยวกับการวางแผนที่เชื่อถือได้ซึ่งสิ่งนี้จะเป็นหลักยึดเหนี่ยวให้กับทีมไปตลอดการทำงานจนจบโครงการ และเมื่อเราได้เรียนรู้วิธีการสร้างแผนแบบแข็งๆ จัดๆ ทำให้เราผ่อนคลาย เพราะแผนของเราจะทันสมัย ติดเกะกะกับสถานการณ์เสมอส่งผลให้เราสามารถกำหนดสิ่งต่างๆ ได้อย่างมีเหตุผล และหนีอื่นใด change ไม่ใช่เรื่องน่ากลัวสำหรับเราแต่ในทางกลับกัน change จะเป็นตัวผลักดันเราไปข้างหน้าอย่างมั่นคงต่างหาก

## 8.1 ปัญหาของแผนแบบ สติกตี้ (โคตรไม่ make sense, static)

คุณเคยพบปัญหาเหล่านี้หรือไม่ เมื่อเริ่มโครงการฯ เริ่มตัวยังท่าสายสัมภាម เราก็โคตรทึ่ม เราก็เทคโนโลยีที่มีหักดุรย์ เราไม่แผนที่สมบูรณ์แบบ แต่ผ่านไปสองอาทิตย์ หม้อข้าวยังไม่ทันดำเนินอย่างเริ่มพัง มันไม่ได้สายอย่างที่คิด เรากำลังออกทะเลแล้วก็กากกากกาก

**ทีมโคนเปลี่ยน โคนแยก งานอื่นสำคัญกว่า งานเรา....**

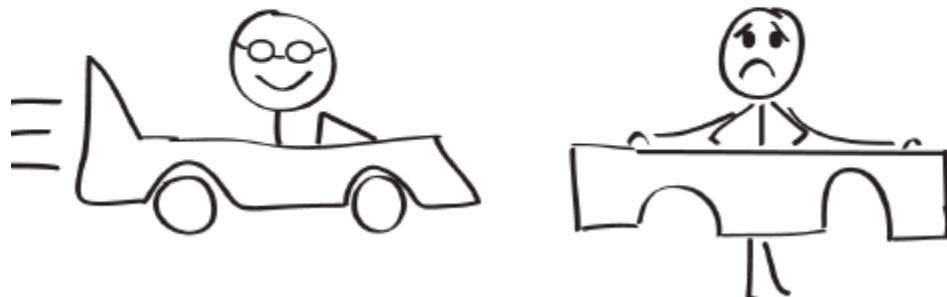


หัวหน้าทีมเดล็อกป้ำของเรากูกดึงตัวข้ามไปเพื่อจัดการกับโครงการอื่นที่สำคัญกว่า (คลาสสิก โครงการอื่นมีความสำคัญทางยุทธศาสตร์สูงกว่า) โโคเครายังมีเวลา "เรากิดในใจ" เราสามารถจัดการปัญหานี้ได้ แต่ยังไม่ทันไว นักประจำอยังไม่กินน้ำดูมุมมองของคนอื่น

**เรารู้ด้วยแล้วว่าเราไม่สามารถทำงานได้เร็วตามที่เราวางแผนไว้**

**แผนของเราราจะเร็วเท่านี้**

**นี่ในเราไปได้เร็วที่สุด เท่านี้**



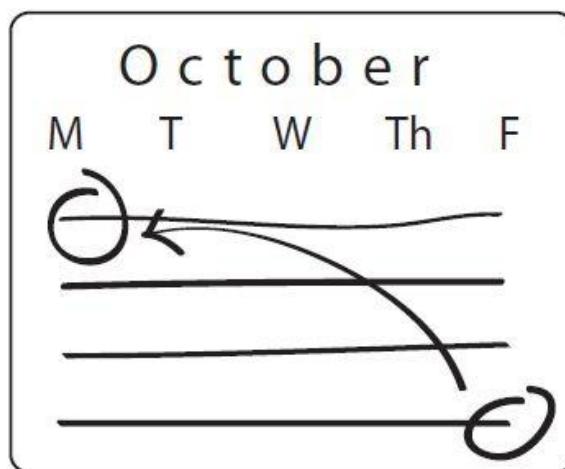
เพราะเราเพิ่งจะรู้ว่าไ้อีสิ่งที่ต้องทำกับสิ่งที่ทีมเราทำได้นั้นมันช่างห่างไกลกันเหลือเกิน และเมื่อเวลาเหลือบไปดูเวลาแล้ว เสือกอก กก ผ่านไปครึ่งทางแล้ว

หันใจนั้นลูกค้าของคุณจะได้พบว่า  
จริงๆแล้วสิ่งที่พวกรต้องการนั้นมันคือ . . . .



จาก Web Application ที่เคยดูเหมือนง่ายก็กลายมาเป็นอะไรที่ยุ่งยากและขับข้อน絮ๆ จากสิ่งที่ดูเหมือนปลอกกลั้วยเข้าปาก กลายเป็นสิ่งที่เป็นไปไม่ได้เดียวกับเวลาและสิ่งเรามีอยู่ในขณะนี้ และแล้วก็ปัญหาที่ไม่เคยคาดไว้ก็ปรากฏตัวให้เห็น ด้วยเหตุผลทางด้านธุรกิจทำให้มีความต้องการแอปพลิเคชันของคุณเร็วขึ้นแทนที่จะเป็นหลังจากนี้ (ตามกำหนดเดิม) การประชุม ล่วงเวลาเร่งด่วนเกิดขึ้น, ข้อตกลงการทดสอบโปรแกรมถูกตัดออกไป คนภายในทีมถูกขอให้ทำงานในวันหยุดยกเว้นวันหยุดต่างๆ และแล้ว(หลังจากอดทนทำงาน) แอปพลิเคชันก็ถูกนำไปใช้งานแบบคุณภาพแย่ๆ ไม่มีความสามารถใช้งานมันได้ เหล่านี้นำมาซึ่ง การล่าช้าอย่างอื่น, งบประมาณเกิน และสุดท้ายไปร์เจค์ fail

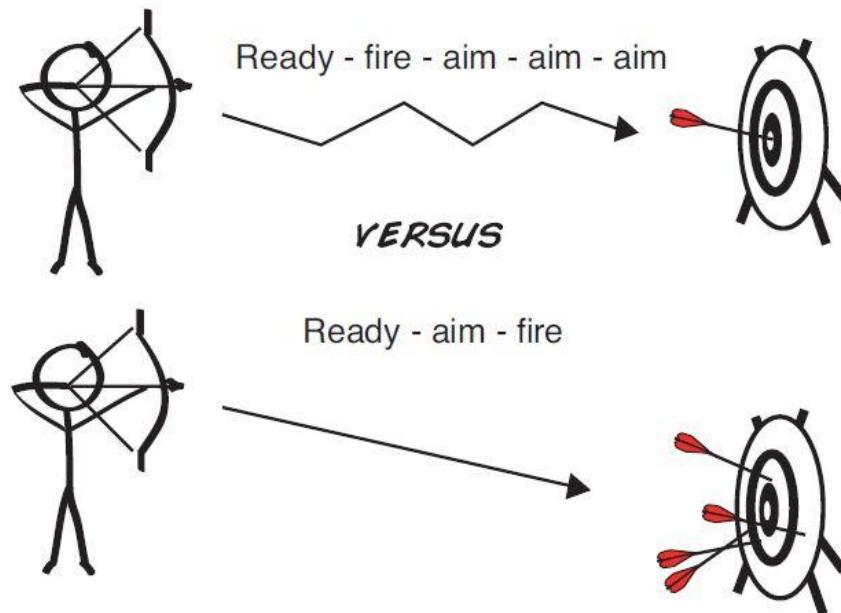
เราไม่มีเวลาพอที่จะทำโปรเจคให้เสร็จ



ถ้าคุณเคยเจอเรื่องเหล่านี้มาก่อน ทำใจให้สบาย – “You are not alone” การเปลี่ยนทีม, ลดTHONเวลา ของไปร์เจค, หรือแม้แต่ กระบวนการเลื่อน requirements เป็นเรื่องที่ปกติมักมากสำหรับทุกๆไปร์เจคในทาง software การที่เราจะเผชิญกับความไม่แน่นี้ได้นั้น เราต้องวางแผนที่จะเผชิญกับมัน แบบนี้:

1. ส่งมอบสิ่งที่มีคุณค่าสุดๆให้กับลูกค้า

2. จริงใจ, เปิดเผย และซื่อสัตย์ให้มากที่สุด
3. ตกลงสัญญาเฉพาะในสิ่งที่เราทำได้
4. อนุญาติให้มีการเปลี่ยนแปลงหรือปรับปรุงแผนเมื่อมีความจำเป็น



ด้วยการกำหนดสิ่งแวดล้อมให้พร้อมเชิงกับการเปลี่ยนแปลงตอนนี้เราก็พร้อมกับวิธีวางแผนงานในแบบ օเจล์ฯ กันแล้ว

## 8.2 เริ่มวางแผนในแบบแอลไจล์ฯ

มันเป็นอะไรที่ง่ายมาก การวางแผนงานแบบแอลไจล์ฯ ไม่มีอะไรมากกว่าการคำนวณ (measuring) หากความเร็วของทีม ที่จะสามารถแปลง Stories แต่ละ Story ให้กลายมาเป็นงาน (software ที่พร้อมใช้งานได้) หลังจากใช้สิ่งนี้แล้วทำนายว่าเมื่อไรงานจะเสร็จทั้งหมด (แมพ)

## กาลครึ่งหนึ่ง(นานมากแล้ว) ผู้คนคุกคอกให้ถอนตัวออกจากโปรเจค



ครึ่งหนึ่งผู้คนเข้าไปทำงานที่ใช้ลูกค้าเชื่อพากเราทำลังพยาภรณ์ทำระบบ **Gas Accounting System** มูลค่าสองล้านดอลลาร์ ด้วยเงินจำนวนจัดแสดงดอลลาร์ และเห็นได้ชัดว่างบประมาณที่มีให้นั้นเป็นแค่ครึ่งหนึ่งของสิ่งที่มันควรจะเป็น บริษัทได้ทำการบันทึกอี้ยวนหัวใจให้เราทำงานล่วงเวลาและในวันหยุดเพื่อให้โปรเจค "เสร็จทันเวลา"

ทุกคนคงจินตนาการกันได้ว่าเกิดอะไรขึ้นบ้าง ทุกๆครึ่งที่เราประชุมวางแผนล่าหรือ **Iteration** พากเข้าใจให้เราเพิ่ม **velocity** ของเราระบบเป็นสองเท่าจากของเดิม ที่เราเป็นอยู่....แน่นอนพากเราปฏิเสธที่จะทำ

เมื่อวันหนึ่งมาถึง เขาได้ตั้งผู้ผลักดันโครงการและบอกว่า ด้วยการที่ไม่รับทำงานให้มากขึ้น พากผุดให้ทำลายสิ่งที่จะสร้างความน่าเชื่อถือกับลูกค้า และขอวิสัยของผู้ไม่เป็นที่ต้องการ ล่าหรือโปรเจคแล้ว

และเมื่อวันสุดท้ายมาถึง ผู้ทำให้ลูกค้าผิดหวัง พากเราท่านางสิ่งผิดพลาด (อาทิเช่น ไม่ได้ทำ **inception deck** ในตอนเริ่มต้น และไม่ได้รีวิวการแผนงานแบบอิจลล์ฯ) แต่วัฒธรรมที่เป็นอยู่ก็สำคัญ ไม่ใช่ทุกคนที่จะชอบสิ่งที่มองเป็นได้และความโปรดปรานที่อิจลล์ เป็นคุณต้องทำให้แน่ใจก่อนว่าลูกค้าของคุณเข้าใจการวางแผนงานในแบบอิจลล์ฯ และตรงไหนที่คุณจะทำการปรับเปลี่ยน มีความเป็นจริงและแผนที่วางไว้ไม่ได้ไปด้วยกันแล้ว

## เราต้องทำอะไรกันบ้าง



ในแอ็ลไจล์โปรเจคคลิสต์ของสิ่งที่เราต้องทำนั้นจะเรียกว่า “Master story list” ภายในลิสต์จะประกอบด้วยฟีเจอร์ทั้งหมดที่ลูกค้าต้องการจะเห็นในซอฟต์แวร์ของพวากษา ความเร็วที่เราจะสามารถเปลี่ยน user stories มาเป็นซอฟต์แวร์ที่ทำงานได้จะเรียกว่า “team velocity” พวากเราจะใช้สิ่งนี้ในการคำนวนหาผลงานของทีมเราและใช้ในการประมาณวันที่จะทำการส่งฟีเจอร์ต่างด้วย (Delivery dates) กลไกที่จะทำการขับเคลื่อนงานงานเสร็จสมบูรณ์คือ Agile Iteration (ช่วงประมาณ 1-2 สัปดาห์ ของการแบ่งส่วนงานที่จะสามารถเปลี่ยน user stories ให้กลายมาเป็นซอฟต์แวร์ที่พร้อมทำงานได้)

ตัวอย่าง ໄໂດຍในการการคำนวนวันส่งงานแบบคร่าวๆ เริ่มจากหาผลรวมของเวลา(total effort) ที่ต้องใช้ทั้งหมดในโปรเจคนั้นๆ หารด้วยค่าประมาณของ team velocity และนำมาหาว่าเราต้องใช้จำนวน Iteration เท่าไร ที่เราคาดกว่าจะพร้อมส่งโปรเจคนั้นๆ ออกไป ซึ่งสิ่งเหล่านี้จะกลายมาเป็น แผนดำเนินงานโป๊กเจคของเรา

$\#iterations = \text{total effort} / \text{estimated team velocity}$

ตัวอย่าง เช่น

$\#iterations = 100 \text{ pts} / 10 \text{ pts} \text{ ต่อ iteration} = 10 \text{ iterations}$

สิ่งที่สำคัญ คือ ต้องเข้าใจก่อนว่าแผนดำเนินงานในโปรเจคครั้งแรก ไม่ใช้ข้อตกลงสุดท้ายที่ไม่สามารถเปลี่ยนแปลงได้ มันเป็นแค่การคาดเดาเท่านั้น พวากเราไม่สามารถจะรู้ได้เลยว่าความเร็วในการทำงานของทีมเป็นเท่าไร ในตอนเริ่มโปรเจค และแม้ว่าเราจะสามารถสร้างตัววัดที่จะนำมาคำนวนว่าโปรเจคใช้ระยะเวลาเท่าไร พวากเราก็ยังไม่สามารถถึงวันจริงๆ ที่เราต้องการหาได้ การจัดแผนดำเนินงานเริ่มต้นแบบด้วยตัวเป็นสิ่งที่ทำลายโปรเจคตั้งแต่ก็ที่มันจะเริ่มจะอีก

ตอนที่เรากำลังจะส่งผลงาน หนึ่งในสองอย่างจะต้องเกิดขึ้นเสมอๆ เราจะพบว่า

a) เราทำได้เร็วกว่าที่เราประมาณ b) เราทำได้ช้ากว่าที่เราประมาณ

เร็วกว่าที่เราคาดไว้หมายความว่าคุณและทีมของคุณทำงานได้เร็วว่าเวลาที่กำหนดได้ ช้ากว่าที่คาดไว้ (มากกว่าที่ประมาณไว้) หมายความว่า คุณและทีมของคุณมีหลายสิ่งที่ยังต้องทำ และมีเวลาไม่พอที่จะทำมันเมื่อต้องเผชิญกับงานจำนวนมากที่ต้องทำ ทีมแอ็ลไจล์จะทำงานน้อยลง (เหมือนกับสิ่งที่คุณและผม ทำเมื่อต้องเผชิญกับสุดสัปดาห์ที่แสนจะยุ่งยาก) แทนที่จะยึดติดกับแผนงานเดิม พวากเราจะปรับปรุงมัน โดยปกติด้วยวิธีลดโซลูชันของงานลง

### 8.3 Scope ของงานสามารถเปลี่ยนแปลงได้เสมอ



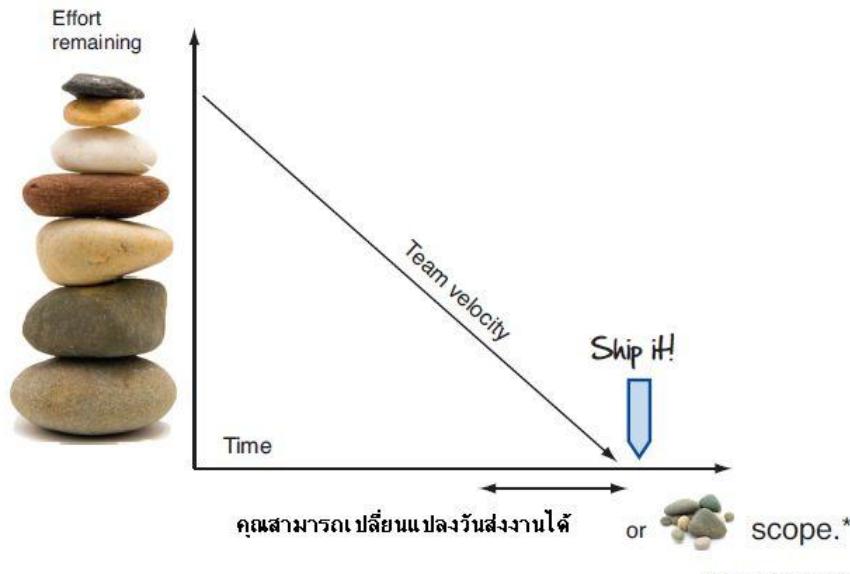
การทำงานแบบที่ Scope สามารถเปลี่ยนแปลงได้นั้น เป็นวิธีบริหารโปรเจคแบบแคลใจลื้อในการปรับปรุง แผนการให้ดีที่สุด เสมอโดยการยืนยันให้ลูกค้านำ Story เก่าออกในทุกๆ ครั้งที่จะนำ Story ใหม่เข้ามา, ทีมแคลใจลื้อจะทำงานภายใต้ ค่าเฉลี่ยเวลา ของโปรเจค และอนุญาติลูกค้าสามารถเปลี่ยนแปลงอะไรบ้างอย่างได้ (ในราคากูาก)



#### Agile principle

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

โดยวิธีนี้จะช่วยให้ลูกค้าจะรู้สึกว่าพวกรเข้าจะต้องเสียเวลา กับการตรวจสอบ requirement (เสียเวลาน้อยลง) และช่วยให้พวกรเข้าและทีมเรียนรู้ว่าที่จะเริ่มต้นแน่นที่จะพยายามทำให้ทุกสิ่งทุกอย่างสมบูรณ์แบบ ตั้งแต่แรก ถ้าจะพูดในความเป็นจริงแล้ว ลูกค้ามักจะไม่จำเป็นต้องยกเลิก requirement เก่าๆ เพื่อที่จะทำ requirement ใหม่ๆ เข้ามาเสมอ อาทิเช่น ถ้ามันเป็นฟีเจอร์ (feature) ที่พวกรเข้าต้องการมันจริงๆ และพร้อมที่จะจ่าย สำหรับฟีเจอร์เหล่านั้นซึ่งพวกรเข้าก็ต้องเพิ่มจำนวนวันขึ้น



แล้วอะไรล่ะที่ลูกค้าไม่สามารถทำได้ นั้นก็คือการใส่อะไรบ้างอย่างเข้ามาในลิสต์งาน แล้วคาดคะดิว่า สิ่งที่ได้ออกไปจะต้อง ตั้งกันเป็นๆ มันเป็นความคิดแบบสมบูรณ์แบบ ที่ไม่มีพื้นที่ให้การวางแผนในแบบแอลไจล์โดยเมื่อเวลา มาถึงเราต้องเลือก กันระหว่าง เลื่อนวันส่งงาน กับลดScopeของงาน โดยทั่วไปคนที่เป็น Agilelist มักจะเลือกอย่างหลัง(ลด Scopeของงาน) การเลื่อนวันส่งงาน เป็นสิ่งที่ไม่สามารถรับได้ในเชิงธุรกิจ แม้ว่าสิ่งที่พากเราจะทำได้ไม่ดีเลยคือสิ่ง software ให้ตรงเวลา ไม่ว่าเราจะ ทำงานด้วยกำหนดวันตามตัวหรือว่าทำงานภายใต้งาน(feature)ที่เป็นหัวใจของระบบ การปรับเปลี่ยน scope เป็นแนวคิดที่ว่าคุณ และลูกค้าของคุณต้องเริ่มที่จะรักษาแผนงานให้เป็นไปตามความเป็นจริงและป้องกันทีมของคุณจากสิ่งที่พากเขามาไม่สามารถทำ มันได้ บางครั้งคุณอาจจะแปลงใจเมื่อพบว่าลูกค้าไม่ต้องการที่จะทำงานแบบscopeที่สามารถปรับเปลี่ยนได้ และยังยืนยันจะให้ คุณและทีมทำงานให้มากขึ้น

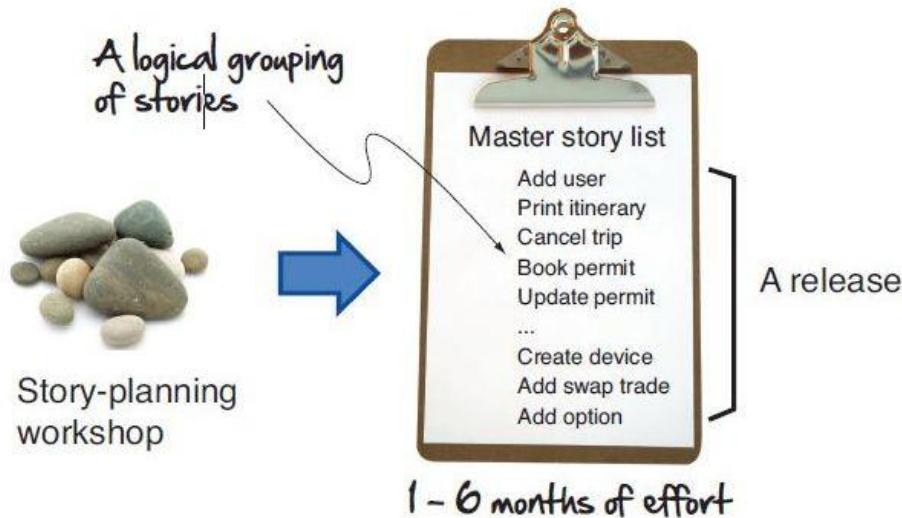
#### คุณสองทางให้เลือกดังนี้

เริ่มจาก โภนกดตัวเอง ปิดหน้าปิดตา ทำงานแผนแรกเริ่มโดยไม่สนใจอะไรมีอนาคตอื่นทำกัน หรือ ไม่คุณก็ประเมินเวลาให้มากๆเข้าไป มองข้ามความเจริญของทีม(team velocity) จะ สามารถตัวเองว่า ทุกสิ่งจะจบลงได้ด้วยดีในที่สุด(ปฏิหารเท่านั้นที่ช่วยท่านได้) หรือ เมื่อทุกอย่างล้มเหลว คุณควรที่จะซึ่งแจ้งข้อเท็จจริง บอกถึงสิ่งที่มันเป็นอยู่ จากนั้นก็หยุด นั่งลงรออย่างเงียบๆ จนกว่าทั้งพากเข้ารู้ว่าพากคุณจะไม่โดดเข้าลงไปในหลุมนั้น คุณจะไม่ทำแบบเดิมต่อไป และคุณไม่ยินดีที่จะเข้าร่วมเป็นส่วนหนึ่งของการโภน ก็เกิดขึ้นในกระบวนการผลิต software ของเรา ที่ผ่านมา 40 ปีไม่มีครบอกว่าการเป็น Samurai เป็นเรื่องง่าย เมื่อพร้อมแล้วเรามาดู วิธีการสร้างแผนงานแบบแอลไจล์กัน

#### 8.4 Your First Plan

การที่จะสร้างแผนงานในแบบแอลไจล์ไม่ได้ต่างจากการเตรียมตัวสำหรับสุดสัปดาห์ที่อยู่ๆ เลย มันเริ่มจากการทำลิสต์ของงานที่ ต้องทำขึ้นมาสักลิสต์

### Step 1: สร้าง Master Story List ของคุณขึ้นมา



"Master Story List" คือลิสต์ของ Story(Feature) ที่ลูกค้าต้องการจะเห็นใน Software ของพวกรเข้าทั้งหมด พวkmn จะถูกกำหนดด้วยความสำคัญด้วยตัวของลูกค้าเอง และถูกประเมิน(เวลาหรือ point) ด้วยทีมของคุณ และมันเป็นสิ่งที่ใช้เริ่มต้นในการวางแผนงาน โครงการของคุณ "Master Story List" ที่ดีควรจะใช้เวลาในการทำงานประมาณ 1-6 เดือน ไม่มีความหมายสมออะไร ที่เราจะติดตามความคืบหน้าของลิสต์ของ story ที่ยากกว่านั้น สาเหตุ เพราะว่า a) คุณไม่สามารถคาดการณ์ได้ล่วงหน้าว่าอีกหกเดือนต่อจากนี้จะเป็นอย่างไรบ้าง b) คุณก็ไม่ได้จะยึดติดกับแผนนี้ตลอดไป มันต้องมีการปรับเปลี่ยนอยู่เสมอแล้ว แล้วคุณจะสนใจทำไม่ล่ะ

บางทีคุณอาจจะสามารถที่จะทำและลงทุกอย่างที่อยู่ในลิสต์ได้ แต่ทั่วไปมันไม่เกิดขึ้นหรอกครับ มากจะมีอะไรที่ต้องทำมากกว่าที่เงินทุนและเวลาจะเอื้ออำนวย เช่น ดังนั้นในการกำหนดกรอบของสิ่งที่คาดหวังกับสิ่งที่อยู่นอกเหนือ scope และใจลึกทีมของ story ย่อยๆ จาก Master Story List และจัดกลุ่มนั้นเป็นชุดๆ ที่เรียกว่า Release กำหนด Release ของคุณ

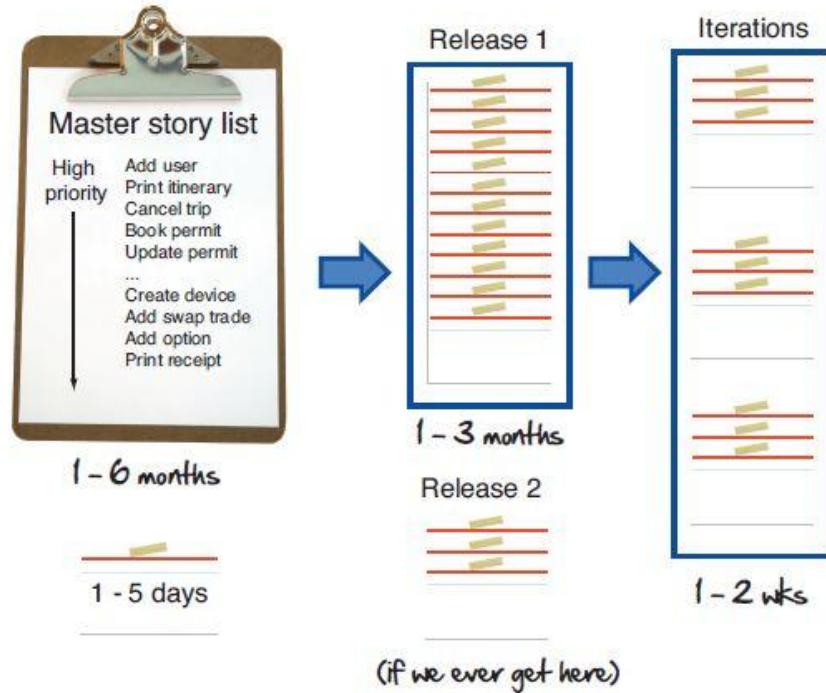
Release คือชุดของ Story (ที่ลูกค้าเข้าใจได้) สามารถทำเสร็จสิ้นลงได้ – software ดีๆ ที่ทำเสร็จ และพร้อมที่จะส่งมอบได้ ในบางครั้งมันอาจจะหมายถึง ชุดของงานที่น้อยที่สุดที่มีค่าพอที่ลูกค้าจะใช้งานได้ – minimal marketable feature set (MMF). MM ตัวแรกของ MMF (minimal) คือสิ่งที่เคยเตือนเรา ว่าเราต้องการเพียงที่จะส่ง สิ่งที่มีค่าให้กับลูกค้าให้รวดเร็ว (80% ของ software ของคุณมาจาก 20% ของฟีเจอร์ทั้งหมดเท่านั้น)



### Agile principle

Simplicity—the art of maximizing the amount of work not done—is essential.

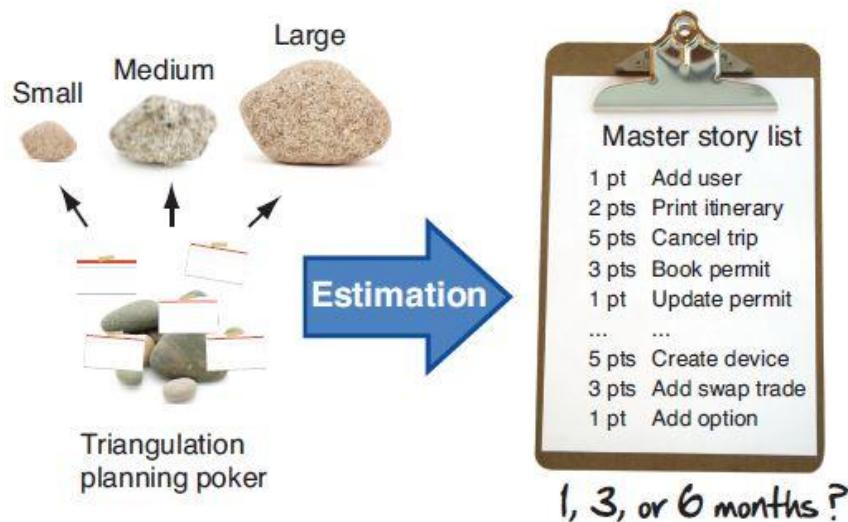
M ตัวที่สอง (maketable) คือสิ่งที่ค่อยเตือนเราส่งงานที่มีคุณค่ากับลูกค้าให้กับลูกค้าในแต่ละ release หรือไม่ (หรือว่าพวกเขายังไม่ใช้มันเลย) ดังนั้น minimal และ marketable คือคุณเจลสำคัญในการเลือกว่า story ที่จะทำก่อนใน release และของคุณ



เอกสารเริ่มเป็นรูปว่างขึ้นมาแล้วหลังจากที่เรามีทั้ง Release และ Master story List ขึ้นต่อไปเราต้องทำกันคือ การกำหนดขนาดของแต่ละสิ่งที่เรามีอยู่กัน

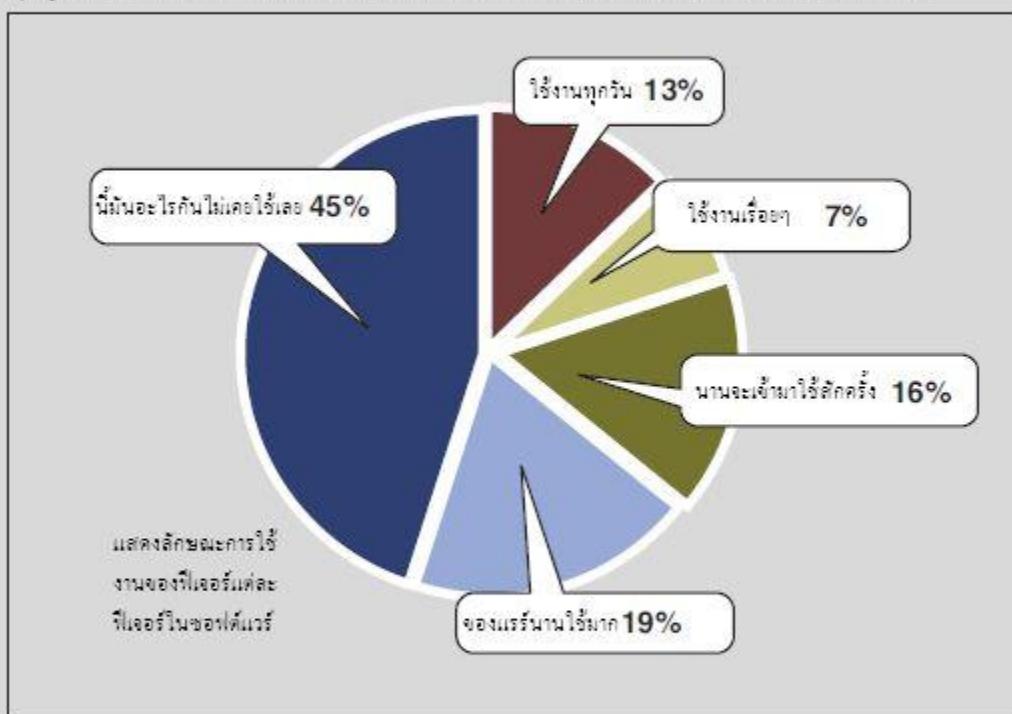
### Step 2: กำหนดขนาดของแต่ละสิ่งที่เราจะทำ

ในบทที่ 7 Estimation (การประเมินงาน): The Art of Guessing (ศิลปะในการเดา) เราได้เรียนวิธีที่ทีมสามารถใช้เทคนิคการประเมินงาน เพื่อที่จะกำหนดขนาดของ story ในแบบแอลไจล์มาแล้ว



## ที่มาอันดับหนึ่งขององค์สืบพิพากษาไปรบก

คุณรู้หรือเปล่าว่ากว่า 64 เปอร์เซ็นของที่เจอร์นั่นนั้นเสียเปล่าหรือไม่เคยได้ใช้งานเลย จริงๆนะ



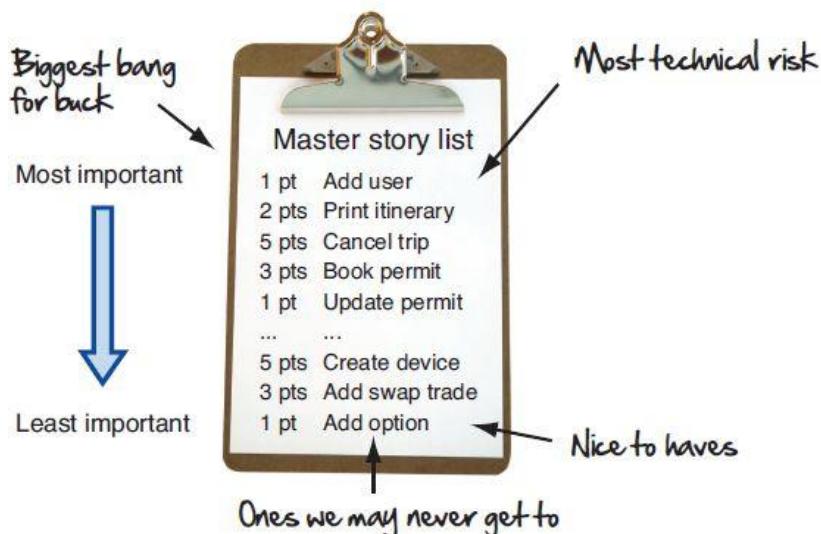
ลองคิดดูว่า คุณใช้งาน MS Word 5%? 10%? อาจถึง 20% คุณเป็นคนที่ใช้งานอะไร

โดยการซื้อให้ลูกค้าไปเก็บกันถึงที่บ้านเป็นจริง-และหักลิ่งอีก 10% ให้ก่อน พากเพียบการประท้วงให้ทั้งเวลาและเงิน  
ที่ซื้อมา กับจัดซื้อที่ไม่ใช่พากเพียบใช้งานให้เสร็จ

ที่นี่คุณต้องเข้าใจว่าสิ่งที่คุณกำลังทำอยู่มันใหญ่แค่ไหนและคุณกำลังจะให้เวลา กับมันแค่ไหน 1 เดือน 3 เดือน หรือแม้กระทั่ง 9 เดือน เอาจริงๆ ถ้าจัดสรรขนาดแล้ว ต่อไปเราจะมาพูดเรื่องความสำคัญ(Priority)ของแต่ละ story กัน

### Step 3: ลำดับความสำคัญ(Priority)

พิพากษารถผ่านได้ทุกเมื่อขึ้นได้ไปเจอกับคุณก็สามารถถูกยกเลิกหรือลดเวลาได้ทุกเมื่อขึ้นนั้น ดังนั้นพิพากษาเรื่องที่จะเริ่มทำให้สิ่งที่สำคัญก่อน การให้ลูกค้าของคุณให้ความสำคัญของ story ใน Master story list จากมุมมองในเชิงธุรกิจเพื่อให้มั่นใจว่าพิพากษาจะได้สิ่งที่ดีที่สุดสำหรับพิพากษา



แม้ว่าลูกค้าของคุณจะพูดว่าอะไรที่ต้องเริ่มทำก่อนและเมื่อไร แต่คุณก็ต้องมีคำแนะนำว่า story ไหนที่เป็นตัวเลือกที่ดีที่จะเริ่มต้นทำก่อน เพื่อที่จะลดความเสี่ยงในด้าน Architecture ภายหลัง ตัวอย่างของ story ที่ควรจะเป็นตัวเลือกที่ดีที่จะเริ่มทำก่อนคือ story ที่มีความสำคัญกับลูกค้า และหมายเหตุทางด้าน architecture ด้วยการเขียนต่อจุดต่างๆ ของ software ตั้งแต่เริ่มจนกระทั่งถึงจบ คุณจะสามารถประเมินความเสี่ยงต่างๆ ไปพร้อมกับการเพิ่มคุณค่า (ที่สามารถมองเห็นได้) ให้กับ software ด้วยวิธีที่ดีที่สุด ที่จะทำระบบ software ขึ้นมา ดังนั้นอย่ากลัวที่จะตระหนอนอกจาก ในสิ่งที่คุณรู้ หรือในสิ่งที่คุณเขียนช่วย ไม่ว่าจะอะไรมากมาย ด้วยการกำหนดความสำคัญร่วมกัน เราจะได้ลิสต์ของงานที่ประเมินเอาไว้แล้ว พิพากษาใกล้จะได้เริ่มต้น คุยกันกับวันที่เริ่มทำงานกันแล้ว แต่ก่อนหน้านั้น คุณต้องเดา ความเร็วในการทำงานของคุณ และทีมของคุณด้วย (team velocity)

## Velocity – มั่นเปี่ยมเรื่องของทีม

เมื่อพากเราเริ่มสร้างแผนการทำงานที่มิใช่ฐานมาจากความเร็วในการทำงานของทีมเราทำข้อตกลงกัน เป็นทีม พากเราทำสิ่งจะพูดว่า พากเราเป็นทีมที่จะส่งสิ่งที่มีค่าให้กับลูกค้าในทุกๆ Iteration

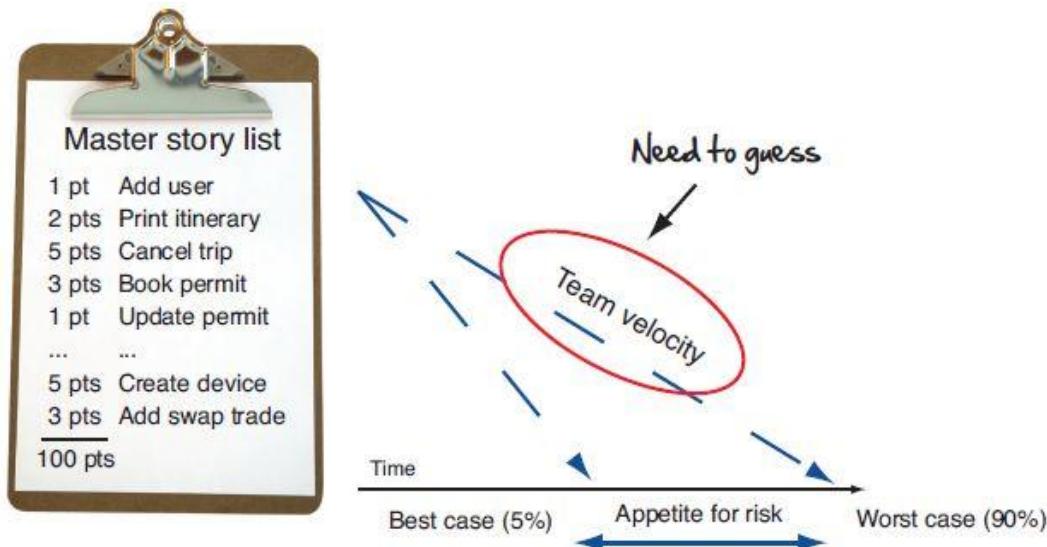
มั่นต่างจากการประเมินความสามารถแต่ละคนในทีม -- ซึ่งนำไปสู่ได้มีดของบริหารแบบโปรเจคฯ (Project Management)

ถ้าต้องการแค่บึกมากขึ้นๆ, ทำงานช้าๆ มากขึ้น, การถือสารพัดความมากขึ้น, การพูดคุยเนื้อหา, หักห้ามทำงานน้อยๆ และการเบ่งปั่นความรู้น้อยๆ คุณมาถูกทางแล้วเริ่มจากให้ความสำคัญ, เมื่อถึงขั้นและให้ร่วงวัลลัฟรับภาระผลงานเฉพาะแต่ละคนของDeveloper ได้โดย

เพียงแค่คุณต้องเข้าใจว่าการที่เราชนะนั้น คุณกำลังจะทำลายทุกสิ่งที่เป็นความรู้สึกและพฤติกรรมที่ พากเราต้องการสนับสนุนและส่งเสริมให้เพ้าสู่โปรเจคของเรา: แบ่งปันไอเดีย, ช่วยเหลือซึ่งกันและกัน แล้วก็ทำให้ได้เคนมองด้วยทั้งสองคนทั้งกับหน้ากับตา

### Step 4: ประเมินความเร็วของทีมงาน (Team's Velocity)

การวางแผนงานแบบแอ็ลจีล์ส์ได้ เพราะพากเราจะวางแผนงานของฟีเจอร์ บนลิสท์ที่พากเราพิสูจน์แล้ว ว่าพากเราสามารถทำมันได้จากในอดีต และถ้าเราไม่รู้ว่าความเร็วในการทำงานของทีมเราเป็นเท่าไร ในตอนเริ่มต้นโปรดเวกก์แคลคูเลเตอร์



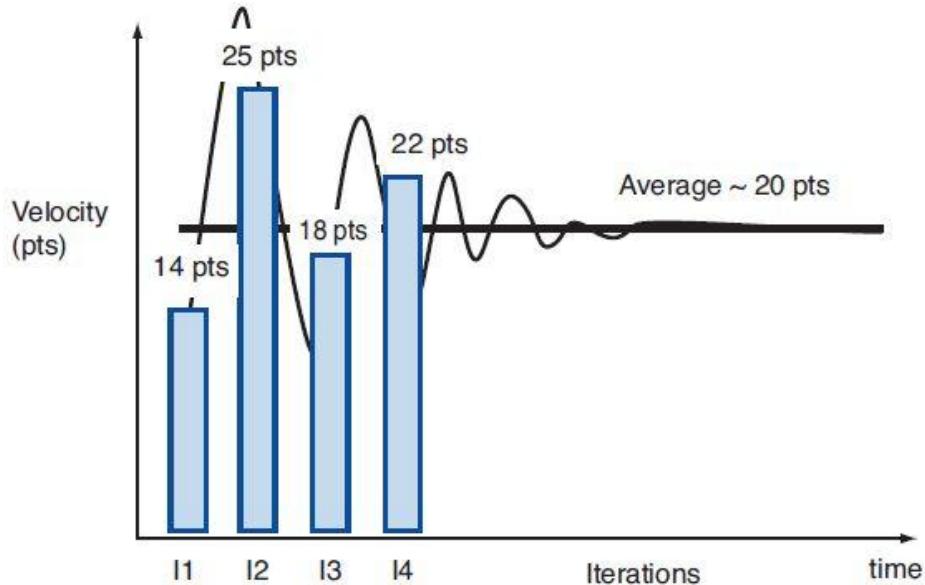
ถ้าทุกๆ Story ของคุณมีขนาดเท่ากันหมด ง่ายมาก ก็แค่

$\text{Team velocity} = \text{stories completed/iteration}$

ส่วนใหญ่มันไม่เป็นอย่างนั้น แต่ละ story มักจะมีขนาดไม่เท่ากันเสมอไป ในกรณีนี้ เราสามารถคำนวณ ได้จากสูตร

Team velocity = story pts completed /iteration

ในตอนเริ่มต้นโปรเจคของคุณ velocity มักจะแก่กว่าขั้นๆ ลงๆ อย่างก้าวไจ นี้เป็นสิ่งที่ปกติมาก ระหว่างที่ทีมกำลังจัดระเบียบของทีม และค้นหาวิธีที่ดีที่สุดที่จะทำงานร่วมกัน



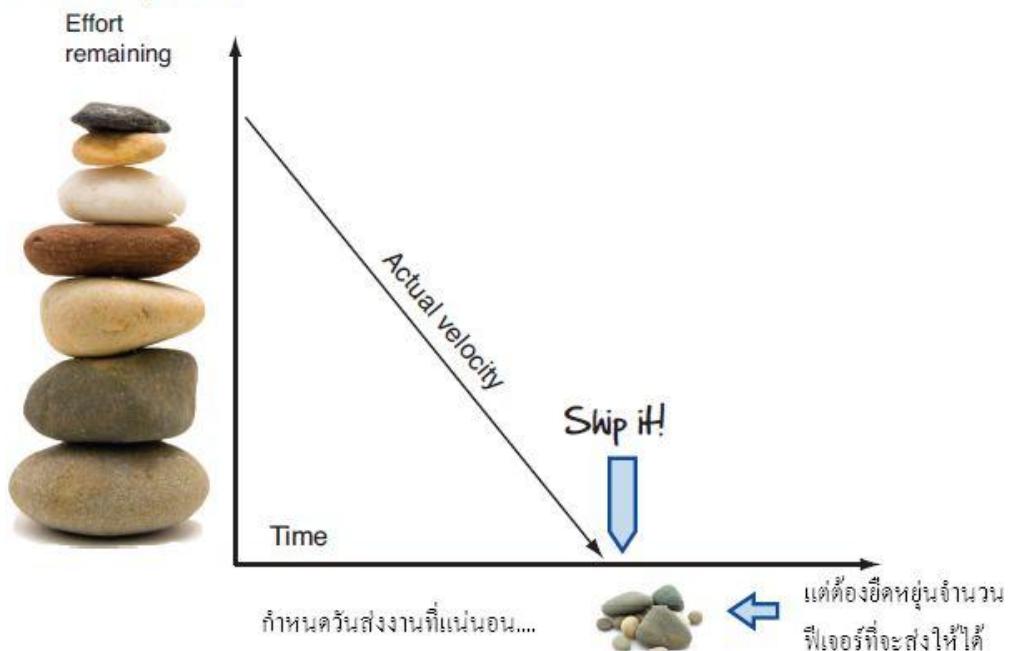
แต่หลังจาก 3-4 iteration ผ่านไป velocity ของพากคุณจะเริ่มเสถียรขึ้น และคุณจะเริ่มรู้แล้วว่าความเป็นจริง ที่ทีมของคุณจะสามารถทำได้ ไม่มีวิธีที่จะทำให้สามารถประเมินความเร็วในการทำงานของทีม (team velocity) ได้อย่างแม่นยำและรวดเร็ว ตามทีมของคุณว่าจะอะไรบ้างที่พากเข้าคิดว่าพากเข้าสามารถทำมันเสร็จได้ในแต่ละ iteration และต้องแนใจว่าสิ่งที่จะทำนั้นเป็นสิ่งที่ลูกค้าต้องการจริง และทีมพร้อมที่จะร่วมมือหรือเปล่า

รวมถึงต้องย้ำกับทีมว่าความหมายของคำว่า 'ทำเสร็จแล้วคืออะไร' (Section 1.3 Done Means Done) และการส่งงาน (story) ในแบบแอลจิล์มายถึงการผ่านทั้ง analysis, testing, design และ coding และทั้งหมดจะต้องได้รับการประเมินงานในตอนเริ่มต้น ความลับของการที่จะได้ความสุขมา คือคาดหวังให้น้อยๆ และถ้าคุณคาดหวังมากจะทำให้มันยากในการพูดคุยกับการคาดหวังไว้น้อยๆ ดังนั้นจึงควรพูดคุยกันมาก และคิดอยู่เสมอว่ามันเริ่มจากกระบวนการคาดเดาและประเมินผลในทุกวันตั้งแต่วันแรก ด้วยลิสต์ของงานที่คุณมีอยู่ในมือและการประเมินความเร็วในการทำงานของเรา พากเรา Kirk พร้อมแล้วที่จะประเมินวันในการทำงาน

## Step 5: จัดสรรวัน

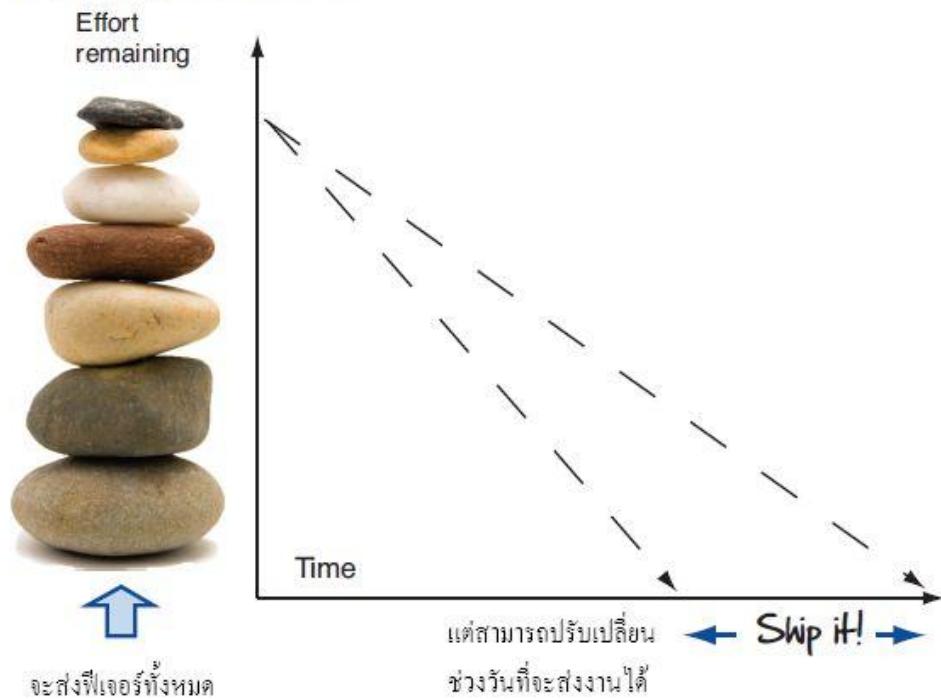
ทางเลือกในการประเมินวันมีสองทาง ส่งงานแบบกำหนดวันส่ง หรือส่งงานแบบกำหนดเป็นชุดๆ ของไฟเจอร์

### **Deliver by Date** (ส่งงานแบบกำหนดวัน)



ส่งงานตามวันที่กำหนดก็คือการขีดเส้นอย่างคร่าวๆ แล้วบอกว่า เราจะส่งงานชิ้นงานวันนี้ ไม่ว่าจะเกิดอะไรขึ้นก็ตาม เมื่อพบว่า story ใหม่ๆ ที่มีความสำคัญถูกนำเข้ามา story เก่าๆ ที่สำคัญน้อยกว่า ที่มีขนาดเท่ากัน ก็ควรถูกนำออกไปด้วย สิ่งนี้เป็นการบังคับให้คิดตัดสินใจและสิ่งที่ต้องแลก (ประเมินสิ่งต่างๆ อาทิ เช่น scope) เพื่อที่จะเพิ่มงานเข้ามา อย่างเร่งด่วนและให้ทุกคนรับรู้ว่า พวกรายจะทำอะไรต่อไป

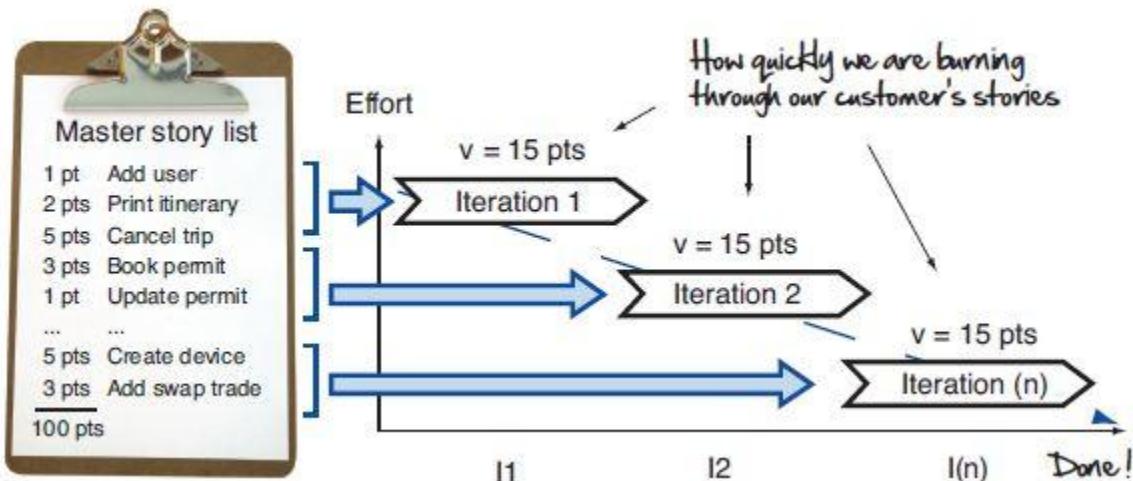
### **Deliver by Feature Set**



ถ้าคุณต้องการให้วันที่จะส่งงานยืดหยุ่นได้ และต้องการที่ส่งไฟเจอร์แบบเป็นชุดๆ คุณก็สามารถที่จะเลือก ส่งเป็นแบบชุดๆ ของไฟเจอร์ได้ วินิจฉัยทำการเลือกชุดของไฟเจอร์ที่สำคัญขึ้นมาทำ ทำงานระหว่างพากมันเสร็จแล้วจึงส่ง การยืดหยุ่นในเรื่อง scope ของงานก็ยังเป็นส่วนหนึ่งของวินิจฉัย (คุณยังคงจะพบไฟเจอร์ใหม่ว่างทางอยู่ดี) แต่ถ้ารู้สึกว่ามันเป็นสิ่งที่ไม่ต้องการที่จะทำมันส่งจริงๆ และคุณพร้อมอยู่แล้วที่จะปรับเปลี่ยนวัน เพื่อที่จะทำไฟเจอร์หลักๆ สามารถจัดส่งได้ ข้อได้เปรียบของการส่งงานด้วยชุดของไฟเจอร์คือคุณจะได้ไฟเจอร์หลักๆ และลดค่าใช้จ่ายในการเดินทาง จนถึงจุดที่พร้อมได้ ความเสี่ยงที่รับได้ เป็นเท่าไหร่นั้นขึ้นอยู่กับ การตัดสินใจร่วมกันของคุณ ลูกค้าของคุณ และคนจ่ายเงินตัดสินใจร่วมกัน และแล้วคุณก็ได้สร้างแผนงานแบบแอลจีล์! คุณได้สร้างการประเมินงาน, กำหนดความสำคัญของแต่ละ story ใน Master story list, ประเมินความเร็วในการทำงานของทีม และเลือกวันที่ส่งงานก่อนที่พากเราจะไปกันต่อ มีสิ่งพิเศษอีกหนึ่งอย่าง ที่คุณต้องรู้ก่อนที่เราจะจบ เรื่องการวางแผนงานแบบแอลจีล์: Burn-Down Chart.

## 8.5 Burn-Down Chart

แม้ว่าเราจะไม่ได้มีการพูดถึง Project burn-down chart อย่าเป็นทางการมาก่อนหน้านี้ แต่พากเราจะได้ผ่านๆ ตามมานมาบ้างแล้ว มันก็คือกราฟที่แสดงถึงความเร็วของเรา (ซึ่งหมายถึงทีม) ว่ามีความเร็วในการทำให้ story ของลูกค้าเสร็จเร็วมากแค่ไหน และมัน กับบอกพากเราว่า เมื่อไรที่เราคาดได้ว่าไปรอดจะเสร็จ

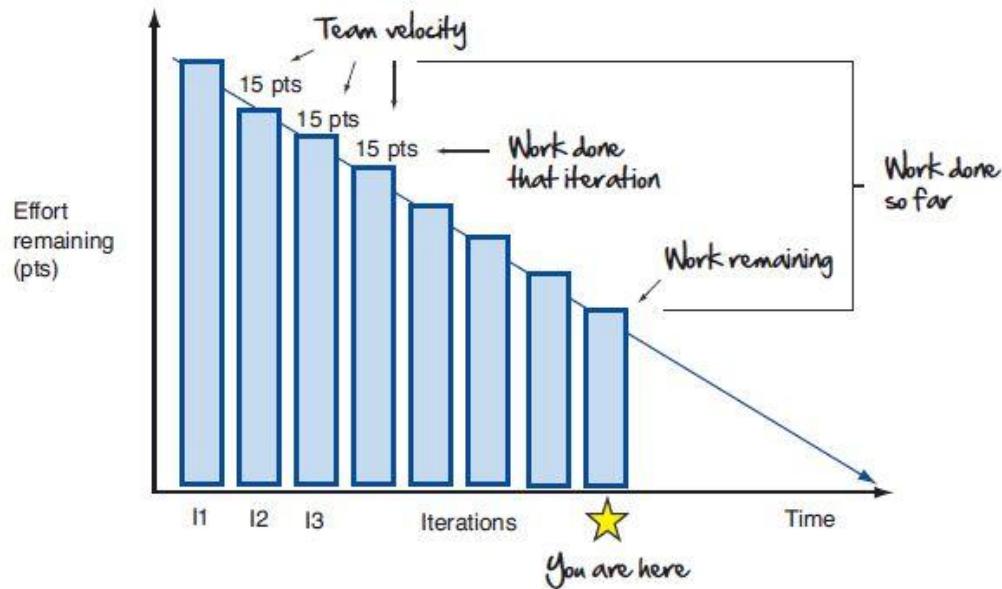


เมื่อแกน y ของเรามีจำนวนที่ยังคงเหลืออยู่ (จำนวนวันของงาน หรือ จำนวน pts) และแกน x เวลาในหน่วย iteration โดยทั่วไปเราจะบันทึกจำนวนของงาน(pts) ที่เหลือทุกๆ iteration และทำการใส่ลงในกราฟ เส้นสูตรของกราฟก็คือความเร็วของทีมนั้นเอง (จำนวนของงานที่ทีมทำเสร็จในแต่ละiteration)

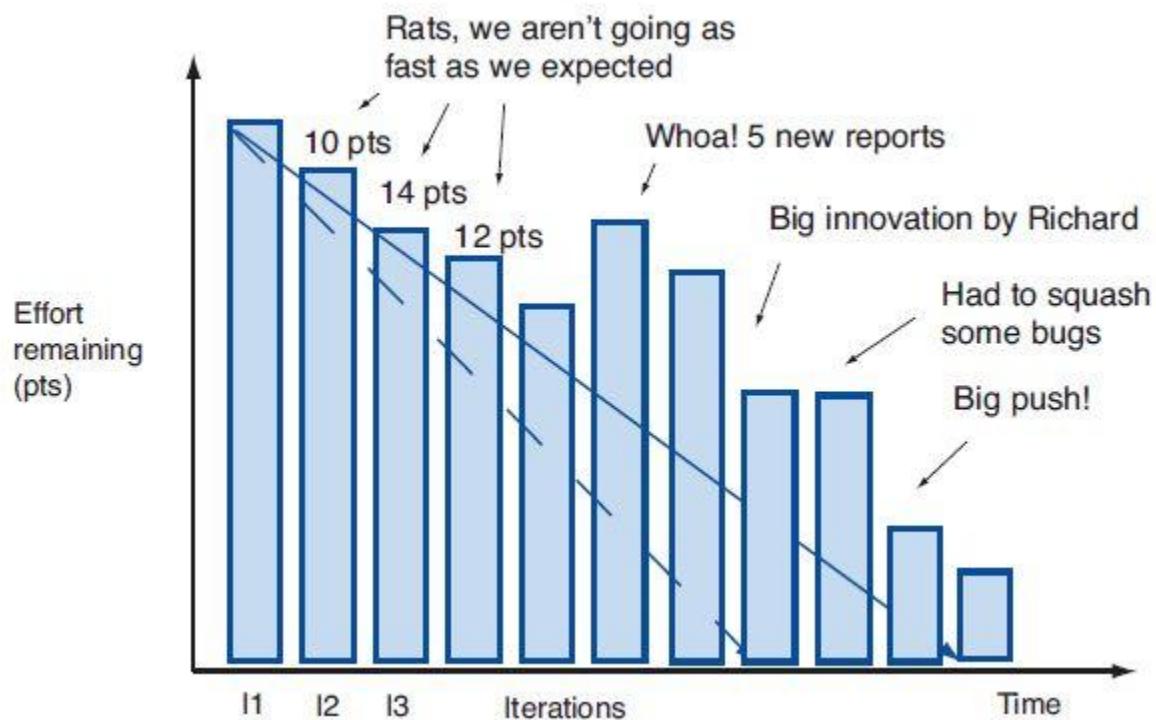
Burn-Down chart เป็นกลไกสำหรับแสดงให้เห็นถึงสถานะของโปรเจกของคุณ แต่เหลือบมอง คุณก็คงได้แล้วว่า:

- โปรเจกของคุณเสร็จไปแล้วเท่าไร
- เหลืองานเท่าไรที่คุณต้องทำ
- ความเร็วในการทำงานของทีม
- วันที่คาดว่าโปรเจกจะเสร็จ

แต่ละคอลัมน์(iteration)ที่อยู่บนกราฟจะแสดงถึงจำนวนของงานที่เหลืออยู่ในแต่ละโปรเจก เราจะปิดโปรเจกได้ก็ต่อเมื่อคอลัมน์ของ Burn-Down เหลือศูนย์



โลกแห่งอุดมคติ(...)ความเร็วในการทำงานควรจะเป็นค่าคงที่ มันควรที่จะเริ่มต้นที่ 15 pts และค่อยๆ ลดลง จากซ้ายไปขวา และเป็นอย่างนั้นไปจนจบโปรเจก ในโลกแห่งความจริง Burn-Down หน้าตาจะประมาณนี้



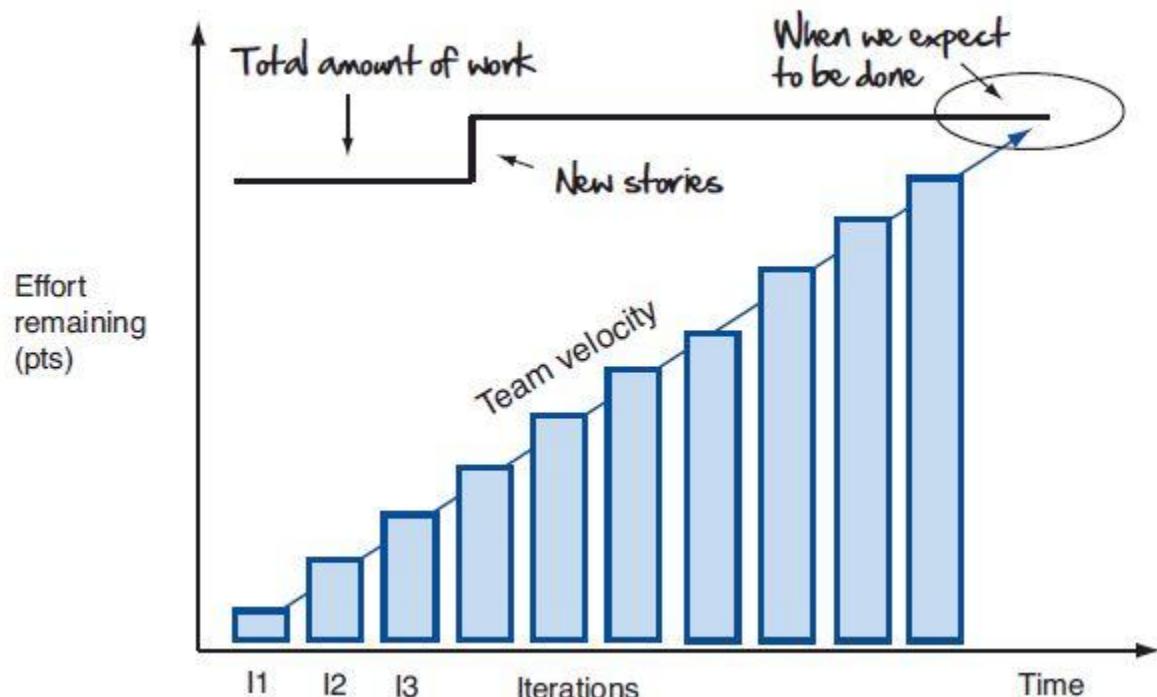
มันไม่ได้เป็นไปตามแผนที่วางแผนไว้เลย ความเร็วของทีมก็ขึ้นๆ ลงๆ มี story ใหม่เกิด story เก่าไปแทนที่

Burn-Down chart ทำให้คุณสามารถทราบได้ถึงเหตุการณ์เหล่านี้ ถ้าลูกค้าตัดสินใจที่จะเพิ่ม scope ไปเจอก็จะสามารถเห็นผลกระทบได้ทั้งที่ที่จะปรากฏในวันส่งงาน ถ้าทีมกำลังทำงานได้ช้าลง เพราะว่าคุณสูญเสียคนสำคัญของทีมไปมันก็จะแสดงให้เห็นเลยว่าความเร็วของทีมลดลงทันทีเมื่อไอนกัน

Burn-Down chart สามารถบอกเรื่องราวต่างๆ ผ่านตัวเลขเท่านั้น เมื่อมีบังสิ่งเกิดขึ้นกับ Burn-Down chart มันสามารถช่วยคำนวณความสะดวกให้เราในการพูดคุยกับเจ้าของเงินลงทุนของเราเกี่ยวกับสิ่งที่เกิดขึ้นกับโปรเจคและผลกระทบที่เกิดจากการตัดสินใจทำอะไรในโปรเจค

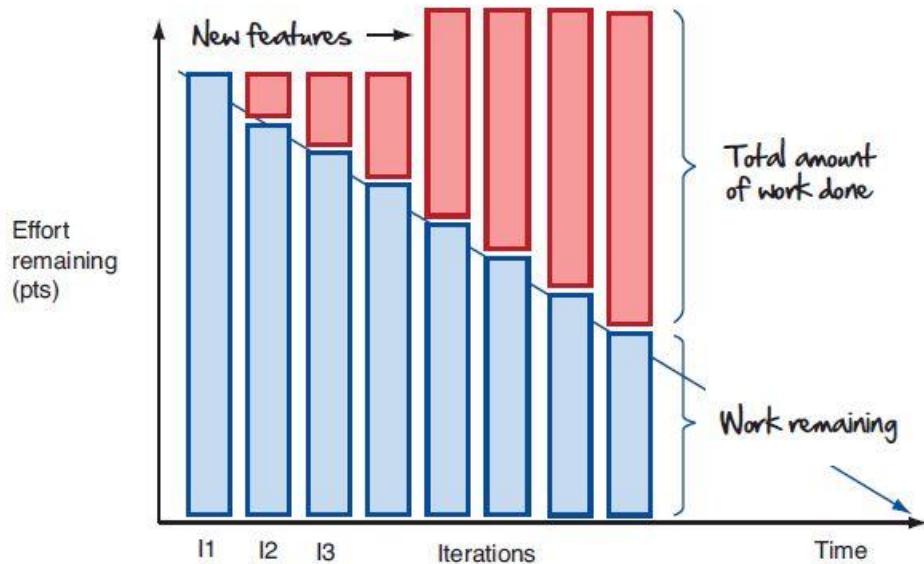
Burn-Down chart ของโปรเจคจะบอกสิ่งที่โปรเจคเป็นอยู่ นี่คือส่วนที่ทำให้แอลใจล เป็นโปรเจคแบบโปรดักส์ เรายังไม่รู้ว่า หรือปิดบังอะไรหรือปั่นแต่ความจริงให้ดูสวยงาม โดยปกติที่เราจะรีวิว burn-down chart ร่วมกับลูกค้าของเราระยะสามารถกำหนดความคาดหวังได้อย่างเปิดเผยอย่างชัดเจน และยังแน่ใจได้ว่าทุกคนเข้าใจดีกว่าเมื่อไรที่พากเพียรคาดว่างานจะเสร็จสิ้น

The Burn-Up Chart กราฟอีกกราฟที่ได้รับความนิยมไม่แพ้ Burn-Down chart ซึ่งมันก็เป็นแค่ Burn-Down chart กลับข้าง



คนบังกลุ่มขอบที่จะใช้ Burn-Up สามารถบอกได้ทันทีเมื่อมี Story ใหม่ๆ เข้ามา เส้นคงที่อยู่บนสุดของกราฟ เมื่อมีอะไรเพิ่มขึ้นใน scope มันก็จะเห็นได้ในทันที ช่วยให้ตรวจสอบได้ง่าย เมื่อเวลาผ่านไป

ถ้าคุณชอบที่จะมองเห็น scope งานในแบบ Burn-Up แต่ยังคงต้องการความเรียบง่ายและคอนเซ็ปต์ของ Burn-Down ก็รวมทั้งสองอันมันเลย ด้วยการหารดูรวมของงานที่ลดลงในแต่ละ iteration บนกราฟ Burn-Down ไปพร้อมกับจำนวนของงานที่เหลืออยู่

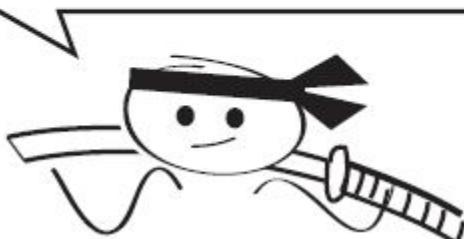


กราฟ Burn-Down หรือว่า Burn-Up ที่สุดแล้วก็อยู่ที่ตัวคุณ ก็แค่ต้องแน่ใจว่าคุณมีวิธีที่ง่ายๆ และสามารถได้จ่ายที่จะใช้ในการคาดการณ์ในเรื่องจำนวนงานที่เหลืออยู่และวันที่คาดว่าจะเสร็จงาน

## 8.6 เปลี่ยนโปรเจคต่างๆให้กลายมาเป็นแอลไจล์

ทำยังไงดีถ้าต้องเข้ารับช่วงต่อโปรเจคยุ่งๆสักโปรเจค?

ทำอย่างไงที่จะเปลี่ยนเป็นอิจล์เม้ออยู่โปรเจคเริ่มไปแล้ว?



มีวิธีมากมายที่จะเปลี่ยนมาเป็นแอลไจล์แม้ว่าโปรเจคของคุณจะเริ่มไปแล้วก็ตามสาเหตุที่คุณต้องคิดว่าจะทำแบบนี้ก็ เพราะว่า:

a) อะไรก็ตามที่คุณเป็นอยู่ขณะนี้มันไม่เวิร์ก หรือไม่ก็

b) คุณต้องการที่จะเริ่มต้นสิ่งที่เร็วกว่า

ถ้าปัญหาของคุณคือหนึ่งในสิ่งข้างบน ให้เริ่มจากการสร้าง Inception deck (บทที่ 3, How to Get Everyone on the Bus)

คุณอาจจะไม่ต้องทำDeckทั้งหมดแต่คุณต้องแน่ใจว่าทุกคนรู้ว่าสิ่งเหล่านี้

- ทำไม่พากคุณถึงมากอยู่จนนี้
- อะไรที่พากคุณพยายามที่จะทำให้เสร็จ
- ใครคือลูกค้าของคุณ
- อะไรคือปัญหา(หิน)ก้อนใหญ่ที่พากคุณต้องการแก้

- โครงการเป็นคนกำหนดภาพรวม

ถ้ายังคงมีข้อสงสัยเกี่ยวกับเรื่องเหล่านี้หรือว่าคำถ้ามื่อนั่นที่เกี่ยวข้องกับ Inception deck, วิธีการเล่นการ์ด Inception deck ที่เหมาะสม, การถ้ามั่นคำถ้ามากๆ และการเริ่มจัดอะไรบางอย่าง

ถ้าคุณต้องเลื่อนบางสิ่งบางให้เร็วขึ้น บ่อนแผนงานปัจจุบันของคุณทั้งหมด และสร้างแผนงานใหม่ ที่คุณสามารถจะเข้าถึงได้ เมื่อcion กันกับที่คุณสร้างแผนงานแบบแอลจีล์ สร้างลิสต์ของงาน กำหนดขนาดของงานแต่ละงาน กำหนดลำดับความสำคัญของงาน และ ส่งมอบงานจำนวนที่น้อยที่สุดของพังก์ชัน เพื่อที่ให้ได้ผลลัพธ์ของมา

ถ้าคุณต้องการที่จะใช้วาให้เห็นถึงความคืบหน้าของงาน แต่ต้องทำงานภายใต้ขอบเขตของแผนงานเดิม ให้เริ่มจะการส่งชิ้นงานที่มีคุณค่าในทุกสิ่งที่ เลือกมาสักหนึ่งหรือสองฟีเจอร์ที่มีคุณค่าในแต่ละสิ่งที่ แล้วทำมันให้เสร็จสมบูรณ์ เมื่อคุณได้แสดงให้เห็นแล้วว่า คุณสามารถจัดส่งงานได้ (และได้เรียกความไว้วางใจคืนกลับมาได้) ค่อยๆ ปรับแก้ไขแผนดำเนินงาน และกำหนด release ที่อยู่บนพื้นฐานความเร็วของทีม และจำนวนของงานที่ยังคงเหลืออยู่ สุดท้ายให้ส่งมอบงานจนกว่าคุณได้สิ่งที่คุณสามารถปิดมันได้ ปรับปรุงงานให้สอดคล้องกับสิ่งที่คุณเป็น, กะตือรือล้นในการปฏิบัติงาน และใช้ความรู้สึกที่เมื่อion กับว่าคุณ พัฒนิ่งที่มากข้างคุณออกจากทางของคุณ มาดูกันว่าอะไรที่ดูเป็นการผิดหัดที่แท้จริงกัน

## 8.7 ปฏิบัติจริง

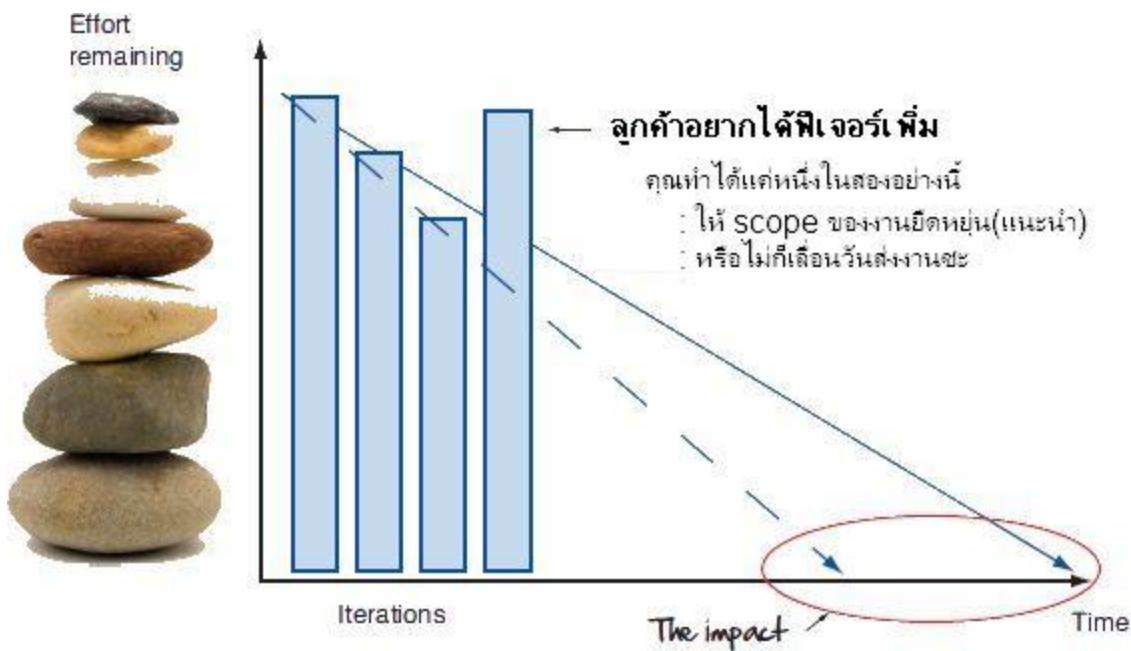
เราได้ผ่านเรื่องยากๆ กันมาแล้ว คุณรู้สึกว่าทั้งหมดแล้วตอนนี้ มาเริ่มนำทฤษฎีมาปฏิบัติจริงๆ กันดีกว่า และเผชิญกับความท้าทายทั้งสี่อย่างที่เราเคยเจอกตอนเริ่มของบทนี้อีกรัง และลองดูว่า เราสามารถควบคุมพวงมันได้อย่างไรบ้างด้วยการวางแผน ดำเนินงานแบบใหม่ แบบแอลจีล์

### การไม่รับรู้โดยเป็นความสุข



ครั้งหนึ่งผมจำได้ว่าเคยถ้ามั่นหัวหน้า(VP)ว่าเขาแนวคิดของแอลจีล์คืออะไร เขาตอบว่า “มันเป็นสิ่งที่ทึ่งนารักและน่าเกลียดไปในตัว” ด้านหนึ่งเขาชอบที่แอลจีล์ทำให้โปรเจค มีความโปร่งใส แต่ในทางกลับกันเขา ก็ไม่ชอบที่มันทำให้โปรเจค มีความโปร่งใสด้วยเช่นเดียวกัน ก่อนหน้านี้เขาทำได้ พิจารณาช่องโหว่ในหลุมพรางและแลรังทำเป็นว่าทุกอย่าง เป็นไปได้ด้วยดี แต่ตอนนี้มัน(แอลจีล์)อยู่ต่อหน้าแล้ว ทุกๆ วัน มันข้ามมาเผชิญหน้าเขาทุกวัน สถานะที่แทนจริงของโปรเจค มันทำหน้าที่ของมันอย่างตื่นตัวพากษาต้องทำการเพิ่ม ประสิทธิภาพของทีมอีกเท่าไร เพื่อที่จะให้ได้สิ่งที่ขายกับมันเป็นงานที่ดี

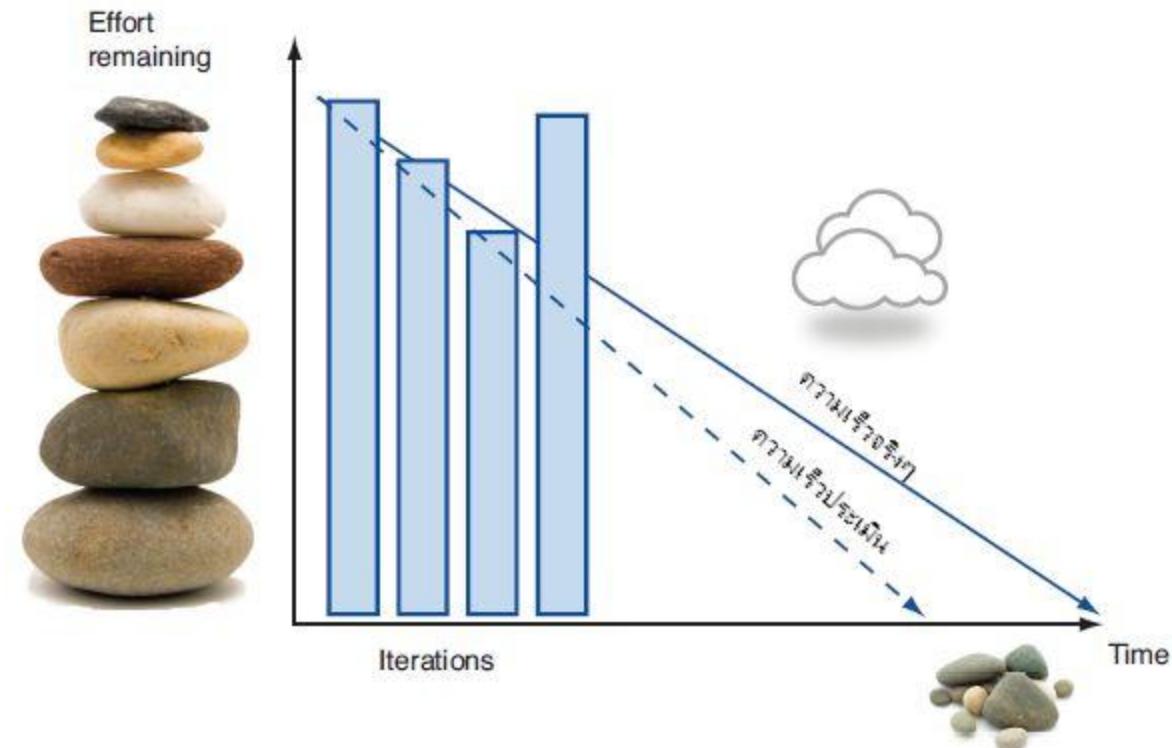
Scenario 1: ลูกค้ารู้สึกตัวว่าอย่างใดเพิ่ม



เมื่อลูกค้าของคุณพบว่า มีอะไรบ้างอย่างที่เข้าอย่างได้เพิ่มเติมในซอฟต์แวร์ ให้ถามพากษาว่า พากษาอย่างให้รับมือกับมันอย่างไรในเรื่องนี้ คุณสามารถเลื่อนวันที่ release ออกไปได้ (ซึ่งคุณกำลังจะบอกลูกค้าว่า เราต้องการเงินมากขึ้นด้วยสำหรับไฟเจอร์ใหม่นั่นๆ) หรือไม่คุณสามารถเอาไฟเจอร์ที่มีความสำคัญน้อยกว่า ออกจากลิสต์งาน (ซึ่งแนะนำให้ทำแบบนี้) อย่างไรก็ตามนี่ เมื่อคุณต้องเจราเจ่องเหล่านี้ มันไม่ใช่สิทธิ์ของคุณที่จะเรียกร้องให้ทำหรือไม่ทำอะไร ตัวคุณเป็นได้เพียงแค่เครื่องมือสื่อสารว่า อะไรเป็นอะไร และจะเกิดผลอะไรขึ้น ด้วยใจที่เป็นธรรมที่สุด (ไม่บวกความโน้มในคำตอบ) สิ่งที่คุณต้องรับผิดชอบคือ ทำให้พากษารู้สึกถึงผลกระทบ ของสิ่งที่พากษาตัดสินใจทำมันลงไป และให้ชี้มุมพากษาให้มากที่สุด เพื่อที่เป็นองค์ประกอบในการตัดสินใจ

ถ้าลูกค้ายังต้องการทั้งหมดจริงๆ ให้สร้างลิสต์ของไฟเจอร์ที่มีก็จะดีมาก(nice-to-have list) และต้องบอกพากษาว่า ถ้ามีเวลาพอที่จะทำมันเมื่อกลับมาเจอกับไฟเจอร์พากนี้จะเป็นไฟเจอร์แรกๆ ที่คุณจะจัดการทำมัน แต่บอกให้เดียร์ดวยว่า nice-to-have ณ ขณะนี้คุยกันอยู่ พากษามันมีสถานะคือ ไม่ทำและมันไม่ได้เป็นส่วนหนึ่งของแกนหลักของแผนงาน

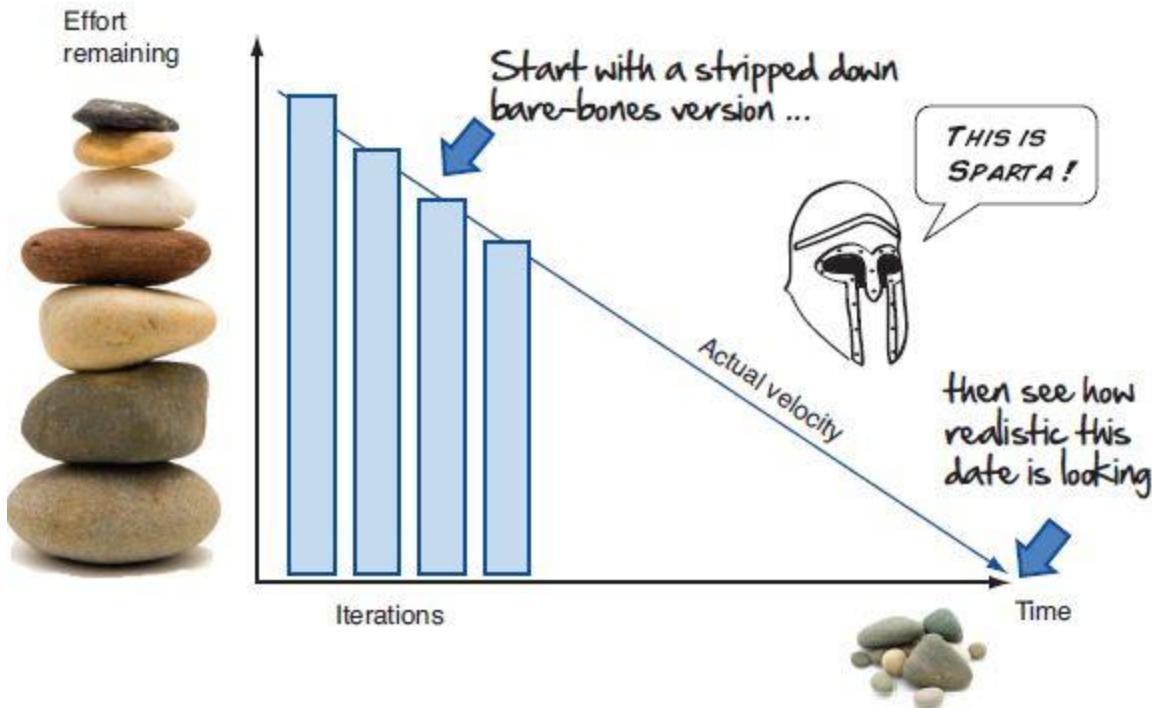
Scenario 2: คุณไม่สามารถไปได้เร็วเท่ากับที่คุณคาดหวังเอาไว้



เวลาผ่านไป 3-4 iteration แล้ว คุณก็พบว่า ความเร็วที่คาดไว้ กับความเป็นจริงมันไม่เป็นไปอย่างที่คิด อย่าตกใจ เราหักน้อยแล้วว่า มันเกิดขึ้นได้ เรื่องแบบนี้นั้นเป็นสาเหตุที่พากเราต้องกำหนดความคาดหวังร่วมกัน (กับลูกค้าด้วย) พร้อมทั้งบอกกับลูกค้าด้วยว่า อย่าเชื่อแผนงานเริ่มต้นให้มากนัก ข่าวดีคือ เราหักน้อยแล้วว่า มันน่าจะเกิดขึ้นได้ และสามารถปรับเปลี่ยนสิ่งต่างๆ ได้เสมอ ตามความจำเป็น การที่เรา vague scope ไว้แบบยืดหยุ่นได้อยู่แล้ว นั้นคือการที่เราเตรียมเครื่องมือ สำหรับเรียกความ สมดุลของโปรเจคคืนมาอยู่แล้ว คุณอาจจะมองหาคนทำมาเพิ่ม (ซึ่งแรกมันจะทำให้คุณช้ำลง) หรือว่าขยายวันที่จะส่งงานออก (หรือทำทั้งสองอย่าง) ลิงค์คุณที่สุดคือต้องพูดคุยและเสนอทางเลือกให้กับลูกค้า ให้แล้วมันอาจทำให้คุณรู้สึกอึดอัด แต่คุณไม่สามารถที่จะยื่น สิ่งเหล่านี้ได้ ข่าวร้ายนี้ แหล่งคือวิถีแห่งแอลจิล ตอนนี้เราเปรียบเหมือนไร้สิ่งป้องกันใดๆ ทั้งสิ้น เมื่อถึงตอนนั้นมันจะเห็นได้ ในทันทีว่าเราไม่สามารถพอยใน การทำโปรเจคหรือไม่ ทางเดียวที่จะทำให้พูดออกมากได้อ่ายแปลง “มีสิ่งที่เราต้องทำมากกว่า เวลาที่เรามี” คือคุณต้องมาถึงจุดนั้นด้วยความซื่อสัตย์ โปร่งใส ที่สุด

#### วิถีแห่งนักบุญ Spartan

วิถีแห่ง Spartan ตั้งอยู่บนหลักฐานเป็นหลัก ถ้าเราไม่สามารถนำเสนอ minimalist version of application ได้ด้วยเวลาและ จำนวนคนที่เรามีอยู่ และดูเหมือนว่าแผนที่เตรียมไว้จะผิดพลาด และต้องการการเปลี่ยนแปลงแล้วล่ะก็



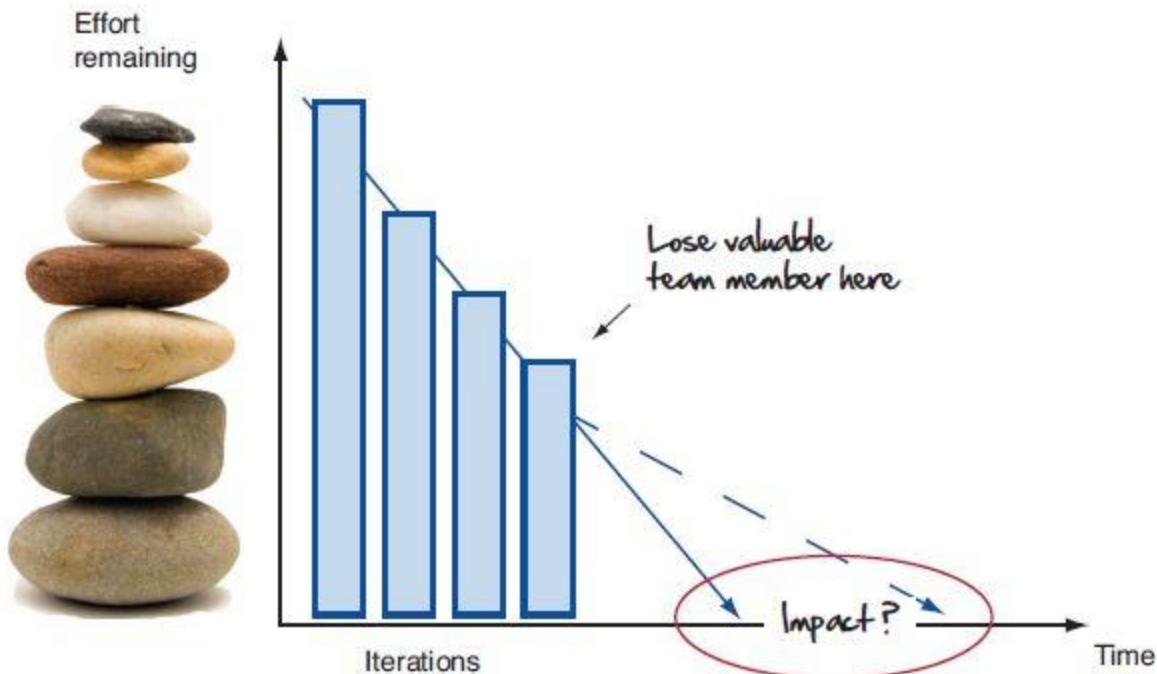
กลไกในการทำงานของมังกี้คือ: ให้นำเอาไฟเจอร์ที่สำคัญของโครงการของคุณ (สิ่งที่เป็นหัวใจของระบบ ที่จะเกี่ยวข้องกับทุกๆ ส่วนของซอฟต์แวร์) และคำนวนเวลาที่ จะใช้ทำมันให้เสร็จสิ้นแบบไม่เกี่ยวกับส่วนอื่นๆเลยออกแบบ, minimalistic version of feature จากนั้นให้นำสิ่งนั้นไปเทียบกับขนาดของ story ที่มีอยู่ว่าด้วยเวลาและจำนวนคนที่มีอยู่ เป็นไปได้ไหมที่จะทำให้ minimalistic version of application เสร็จสิ้นลงได้

ถ้าดูแล้ววันที่ส่งงานพอกับความต้องการ โอเคไปกันต่อได้เลยบนวิธีนี้

ถ้าดูแล้วมันทำจะeasy ไม่พอແນ່ງໆ เยี่ยมเลยอย่างน้อยก็รู้ตั้งแต่ตอนนี้ล่ะกันว่าไม่พอແນ່ງໆ

บนวิธีของ Spartan นั้นอนุญาตให้คุณสามารถมีการเจรจาเพื่อเปลี่ยนแผนงานกันได้ตามสภาพความแข็งแรง และความสมบูรณ์ มันไม่ได้ตั้งอยู่บนความ平凡ๆ ไม่มีการใช้อารมณ์นำทาง มันเป็นแค่เพียงการเผชิญหน้า มันยอมตีกว่าແນ່ງໆที่จะรู้ว่าจะเกิดอะไรขึ้นตั้งแต่ตอนนี้罷罷 ที่จะเป็นหลังจากนี้และด้วยข้อมูลเหล่านี้คุณจะลูกค้าจะสามารถตัดสินใจได้อย่างถูกต้องว่า ไฟเจอร์ได้ที่ควรอยู่ร่วมกันกับวิธี Spartan หรือว่าอันไหนที่ต้องการการน้ำลายและการขัดอีกนิดหน่อย แล้วคุณก็จะสามารถรับจูนแผนงาน โครงการเพื่อที่ส่งงานที่ดีได้ ทุกสิ่งขึ้นอยู่กับวิธีที่คุณใช้งานมัน

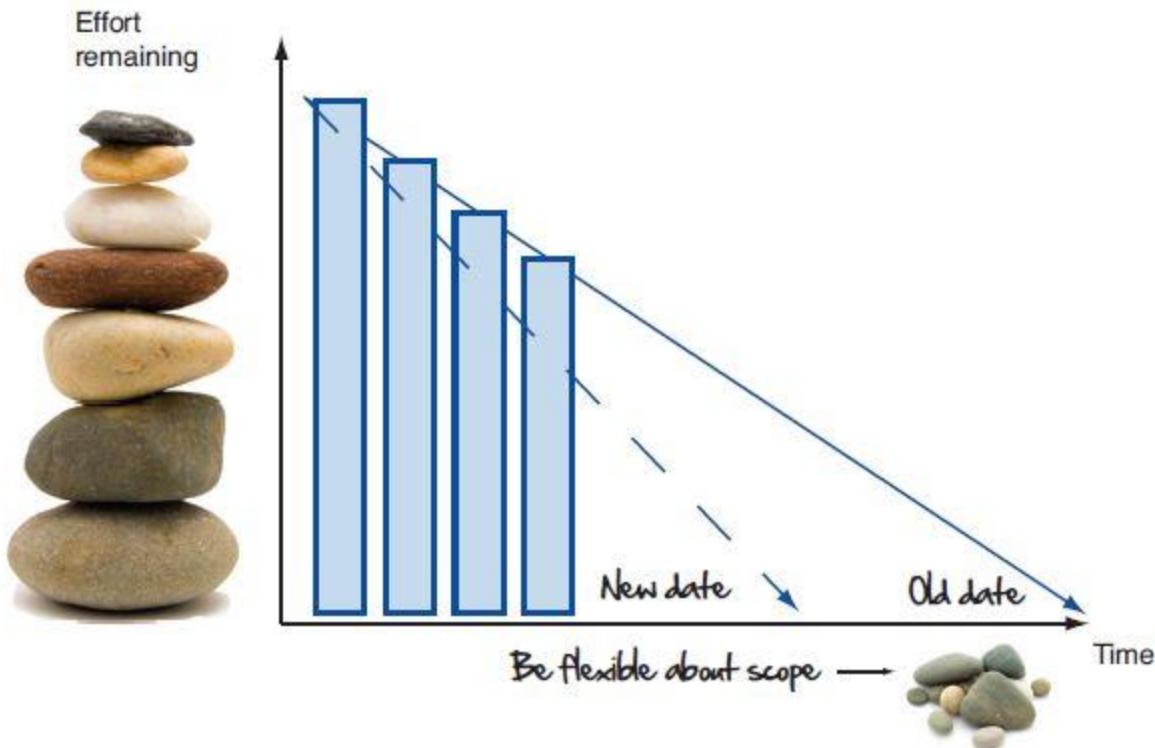
### Senario 3: คนในทีมหายไปลักษณะ



การรับมือกับผลกระทบเมื่อมีคนที่มีค่ากับทีมหายไปสักคนเป็นเรื่องไม่ง่ายเลย คุณก็รู้ว่าคุณจะเลือเข้ากับมันสักวันหนึ่ง มันเป็นเรื่องยากว่ามันเจ็บปวดแค่ไหนที่ต้องศูนย์เสียเงาไปเมื่อมันมาถึงกำหนดความคาดหมายเรื่องงานเมื่อมีการเปลี่ยนแปลงภายในทีมคุณไม่ต้องอาศัยหลักการทำงานวิชาการอะไรทั้งนั้น เดินเข้าไปบอกลูกค้าของคุณเลยว่าโปรดเชื่อของคุณกำลังมีปัญหาแน่ๆ (เดาว่าคุณสามารถทำมันได้) และต้องการมีการเปลี่ยนแปลงตัววัด และประเมินผลกระทบของความเร็วของทีมกันใหม่ (โดยมากใช้เวลาประมาณ 2-3 iteration) คุณจะเป็นผู้ที่สามารถบอกพวกเขาว่าผลกระทบมีมากน้อยแค่ไหน

แน่นอนหัวหน้าของคุณอาจจะหันหน้ามาแล้วบอกคุณว่าจะหาคนใหม่มาให้ เขาจะจ้างคนใหม่มาที่เดิมกัน (หรือแม้กระทั่งดีกว่า) คนเก่ามาให้แล้วคุณจะไม่รู้สึกเลยว่าความเร็วในการทำงานของทีมลดลง บางทีอย่าคาดหวังกับสิ่งนั้นมากนัก คนใหม่อาจจะไม่เหมาะสมก็เป็นได้ หรือเขาอาจจะมีการตุกติก ในขั้นตอนเขียนมาทำงาน สมภาษณ์ พร้อม resume ขั้นเทพและการจับมือจับไม้ เชื่อในสิ่งที่คุณเห็นเท่านั้น จนกว่าจะถึงตอนนั้นเราต้องแก้ความสงสัยด้วยพยายามร่วมกันไปก่อน

#### Scenario 4: โดนลดเวลา



คำตอบที่อยู่ในหนังสือนี้คือให้กำหนด scope ในแบบที่มันยืดหยุ่นได้ ถ้าคุณลดเวลาลงครึ่งหนึ่ง คุณก็ต้องลดจำนวนของพีเจอร์ลงครึ่งหนึ่งด้วย มันเป็นธรรมชาติมากอีกคำตอบหนึ่ง (ไม่ได้อยู่ในหนังสือ) แต่อย่างไรก็ต้องไปคุยกับลูกค้าของคุณดูสิ และเสนอทางเลือกใหม่ๆ ที่สร้างสรรค์ที่จะช่วยเข้าช่วงเวลาที่อาจจะมีงานบ้างงานก็ได้ที่จะสามารถส่งได้ในแบบเปลือยๆ/Spartan หรือว่า บางที่ หน้ารายงานแบบเดติค20หน้า อาจจะสามารถลดแทนด้วยหน้ารายงานแบบเดนามิตสักหน้าหนึ่งก็เป็นได้ การช่วยให้พวกเขารายงานออกได้ทันเวลาันั้นสิ่งที่ต้องการคือบัญญัติเพื่อให้ดำเนินการไปได้นั้นคือ การสร้างความสมพันธ์ที่ดีกับลูกค้าด้วย คุณต้องถูกมองจากลูกค้าว่าสามารถให้คำแนะนำที่ไว้วางใจได้ และวิธีที่จะทำอย่างนั้นได้ก็คือการให้ทางเลือกกับลูกค้าแต่อย่างรับหรือยอมทำหรือตกลงในอะไรก็ตามที่คุณไม่สามารถทำมันได้ นั่นไม่ได้ทำให้เครียดหรือ กการติดต่อสื่อสารกันจะต้องเป็นแบบสองทาง ซึ่งสัตย์เข้าไว้ และบอกพวกเขาว่าอะไรที่มันกำลังจะเกิดขึ้น

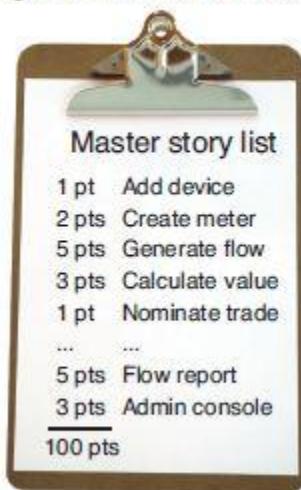
ท่านประจารย์ต้องการที่จะพูดคุยกับคุณที่โรงฝึกแอลไลส์เพื่อที่จะดูว่าคุณได้เรียนรู้อะไรมาบ้าง



## Master Sensei and the aspiring warrior

ยินดีตอนรับลูกศิษย์ของข้า ข้าดีใจมากที่ได้เห็นเจ้ายังคงมีชีวิตอยู่ เขายังคงเป็นจิตวิญญาณ โลกของความเป็นจริงกันล่ะ เหตุการณ์ที่หนึ่งในบริราชาติชัยได้ไปรับประสบการณ์จริงจากสมภูมิมา Scenario(เรื่องมือญี่ปุ่น): ทุกสิ่งทุกอย่างห้ามเปลี่ยนแปลงคงที่ตามตัวนิ่งๆตันๆหมดในโปรดเจคนี้ ไม่สามารถเปลี่ยนแปลงอะไรได้เลยสักอย่าง

Highly regulated  
government-run utility



*CHUGGA CHUGGA  
CHOO! CHOО!*

Change not  
welcome!



Fixed scope

Fixed date

Fixed budget



*How can this project  
be run agile?*

*Change*

ประมาณาร์ย์: นี่คือโปรดเจกของหน่วยงานในญี่ปุ่นรัฐ และเพรัวะว่ามันใช้เงินภาษีของผู้เสียภาษี โปรดเจคนี้จึงต้อง McDon ตรวจสอบอย่างใกล้ชิด และไม่สามารถรับได้กับการเปลี่ยนแปลงใดๆทั้งสิ้น ซึ่งหมายความว่า scope, ค่าใช้จ่าย และวันส่งงาน ทุกอย่างต้องตามตัวหมวด โปรดเจคนี้สามารถที่จะนำแอลไอลที่ใช้ในการประเมินและดำเนินงานได้หรือไม่

ลูกศิษย์: ถ้า scope, วันที่ส่งงาน และงบประมาณ ที่ต่างๆ และพวกเขาก็ไม่สามารถที่จะเปลี่ยนแปลงแผนอะไรได้เลย ข้าก็ไม่เห็นวิธีที่แอลไอลจะสามารถนำไปใช้งานได้เลย ท่าอาจาร্য

**ปรมาจารย์:** เจ้าคิดอย่างนั้นรึ? เมื่อโปรเจคได้ก็ตามที่พยายามจะกำหนดให้ เวลา, งบประมาณ, scope และคุณภาพตายตัว ในไม่ช้าพวกเขาก็จะพบว่า การควบคุมทั้ง4อย่างนี้อย่างเคร่งครัด ไม่สามารถทำได้ตลอดไปหรอก ต้องมีบางสิ่งเปลี่ยนแปลงเสมอ เพราะการเปลี่ยนแปลง จะปรากฏตัวขึ้นมาเสมอ พวกเขายังคงเดินต่ออยู่แล้ว ก็ต้องเลือกว่าพวกเขานั้นต้องการที่จะซ่อนมันไว้หรือแสดงมันออกมาก

**ลูกศิษย์:** แต่จะทำอย่างไรที่จะสามารถแสดงถึงความเปลี่ยนแปลงนั้นออกมาได้ และเมื่อยังต้องปฏิบัติตามคำสั่ง ที่ว่าห้ามเปลี่ยนแปลงอีกต่างหาก

**ปรมาจารย์:** นั้นคือที่ที่ยอดนักរบจะต้องใช้ประสบการณ์และทักษะทั้งหมดของพวกเขาระบุ ใจเป็นอย่างไรถ้าสร้างที่ที่ไว้เก็บ story เก่าที่ไม่ได้อยู่ใน scope อีกต่อไปเพื่อเพียงพอกับการติดตาม และตรวจสอบย้อนหลังได้ว่าการเปลี่ยนแปลงอะไรบ้างที่เกิดขึ้นกับโปรเจคนี้? สิ่งนี้จะช่วยให้พวกเขารู้สึกว่า ในความสามารถที่จะติดตามเพื่อที่จะแสดงถึงความต่างระหว่างแผนเก่าและแผนงานที่ดำเนินอยู่ ในขณะที่ไม่มีการสูญเสียของความสมบูรณ์ของแผนเดิมเลย

**ลูกศิษย์:** ท่านอาจารย์ ท่านกำลังจะบอกว่าไม่ว่าพวกเขารู้สึกอย่างไรก็ตาม แผนงานก็จะมีการเปลี่ยนแปลงเสมอ

**ปรมาจารย์:** ให้(ใช่)

**ลูกศิษย์:** และด้วยการจดบันทึกการเปลี่ยนแปลงเหล่านั้นเอาไว้ พวกเขาก็จะมีความสามารถที่จะตอบโจทย์ความต้องการของคนตรวจสอบได้ ในขณะที่ก็กำลังทำงานที่ตอบโจทย์ในสิ่งที่ลูกค้า ของพวกเขาร้องขอให้พร้อมๆกัน

**ปรมาจารย์:** แน่นอน

**ลูกศิษย์:** ขอบคุณมากท่านอาจารย์ ข้าจะตรึกตรองให้มากกว่านี้

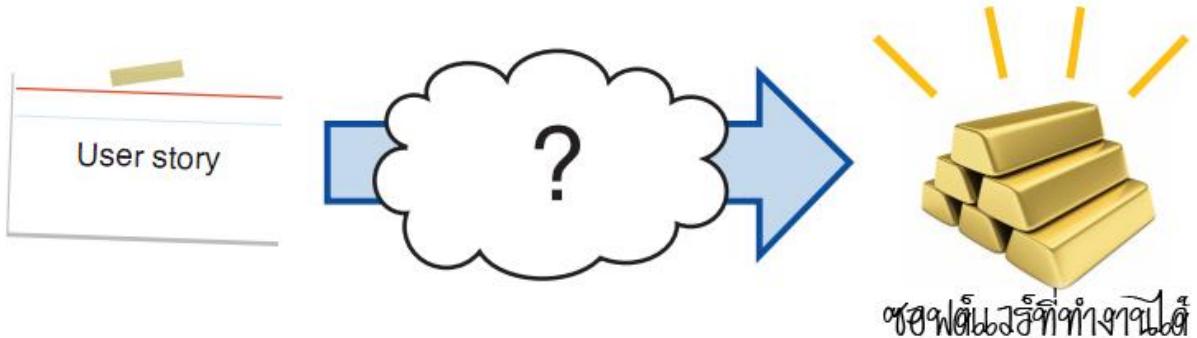
**Lesson(บทเรียนนี้สอนว่า):** ความเปลี่ยนแปลงมีอยู่ในทุกที่ บางครั้งเราแค่ต้องการความคิดสร้างสรรค์นิดหน่อยในการที่เราจะแสดงมันและบริหารมัน

**What's next?**

เอกสารคุณรอดมาได้จาก inception deck คุณมีทักษะของปรมาจารย์ทั้งศาสตร์และศิษย์ในการประเมิน story และคุณก็เรียนรู้ที่จะนำมันรวมกันแล้วสร้างแผนงานในแบบแอลจีล็อกตอนนี้คุณพร้อมแล้วที่จะก้าวออกไปผจญภัยกับการปฏิบัติงานในโปรเจคในรูปแบบแอลจีล์ ในขั้นตอนต่อไปคุณจะได้เรียนรู้ว่าจะเปลี่ยนให้ iteration และแผนงานดีๆ กลยุทธ์เป็นความจริงได้อย่างไร -- working, test ซอฟต์แวร์ที่พร้อมใช้งานและทั้งหมดจะเกิดขึ้นได้อย่างที่ไม่ทันได้ตั้งตัวเลยด้วยซ้ำ

# ตอนที่ 9 การจัดการ iteration

## ทำให้มันเกิดบ!:)



ยินดีต้อนรับสู่ส่วนที่ 4 ที่เรียกว่า Agile Project execution หรือ การดำเนินการไปริบเดกแบบแอจайл์ส ถึงขั้นตอนนี้เราจะนำแผนที่เราวางไว้ทั้งหมดในส่วนที่ 2 และ 3 มาใช้ และจับมันแปลงให้เป็นสิ่งที่ลูกค้าคำน้ำไปใช้งานต่อได้ คือ ซอฟต์แวร์นั้นเอง ในบทนี้จะมีการพูดถึงสิ่งที่เรียกว่า iteration เวลาจะพากุณไปดูเบื้องหลังและแสดงให้คุณเห็นกันแบบ จะๆ ว่า โปรเจกแอจайл์สำเร็จได้ด้วย พลังของ iteration !!! และจากนั้น ในตอนที่ 10 เรื่องการสร้างแผนงานติดต่อสื่อสารแบบแอจайл์ ในหน้า 179 คุณจะได้เห็น ตัวอย่างว่า iteration ในแอจайл์ทำงานได้อย่างไร การประชุมในแบบต่างๆ และการทำให้พอยท์ในทีมแอจайл์สอดคล้องกัน เพื่อให้ทุกอย่างดำเนินไปข้างหน้า และในตอนที่ 11 เรื่องการสร้างพื้นที่ทำงานแบบสมร้อน ในหน้า 91 คุณจะค้นพบว่า ทำไม่การเปลี่ยนแปลงสิ่งเล็กๆน้อยๆ ในพื้นที่ทำงานจะทำให้งานมันขัดเจอนมากขึ้น

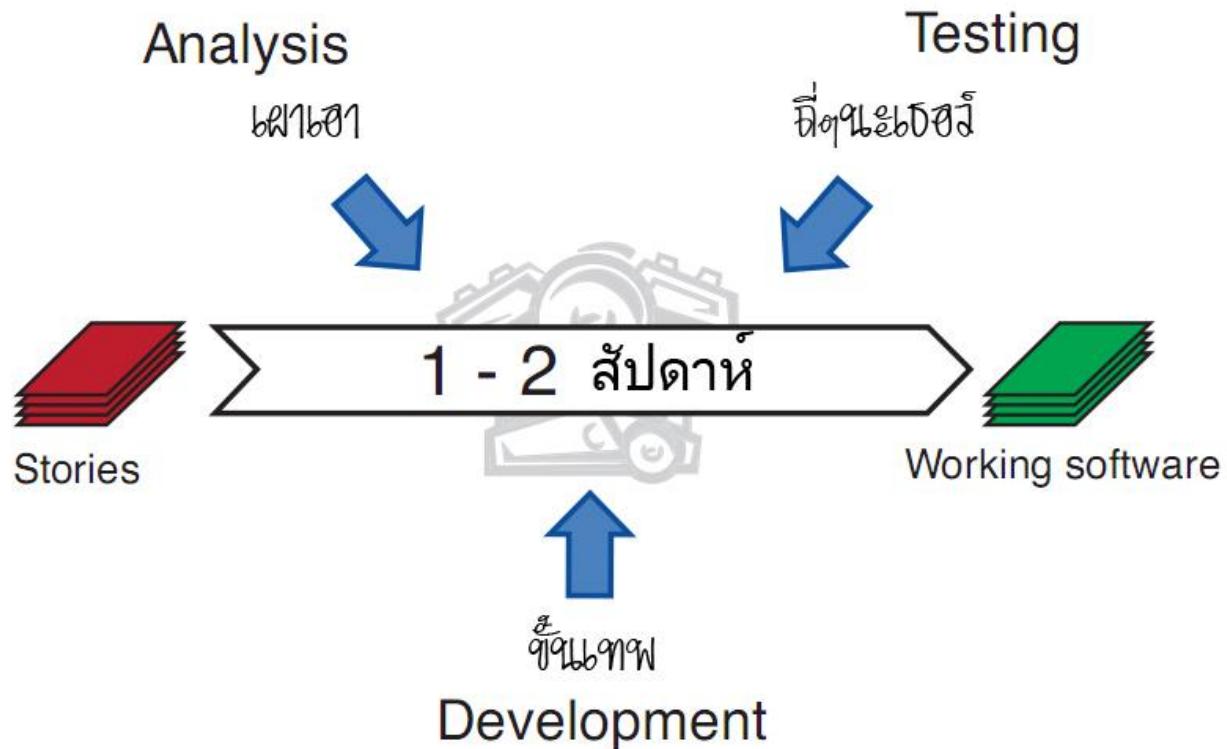
### 9.1 ทำอย่างไรจึงจะสามารถส่งงานที่มีคุณค่าแก่ลูกค้าได้ในทุกๆ สัปดาห์ ?

อืม..คุณมีแผนแล้วสินะ คุณรู้มั้ยว่าทำไม่มีคุณภาพจริงๆ ตรงนี้แล้ว? เพราะคุณพร้อมที่จะดำเนินการแล้วนะสิ! แล้วงัยต่อ? คุณจะทำยังจัยในการที่จะแปลงการ์ดต่างๆ ที่มีคำอยู่ไม่ถูกคำ ให้กลายเป็นสิ่งที่พร้อม สำหรับการเป็น software?

เอกสาร ขั้นแรก คุณไม่มีเวลาที่จะเขียนทุกอย่างออกมากให้หมดหรอก คุณต้องการแนวทางที่ทำให้เกิดการวิเคราะห์ที่ไม่เยอะ จนเกินไปทำเท่าที่จำเป็นมีความถูกต้องและให้ในสิ่งที่คุณต้องการได้ แค่ ณ เวลาที่คุณต้องการเท่านั้น

ขั้นที่สอง ทักษะการพัฒนาซอฟต์แวร์ของคุณต้องอยู่ในระดับเทพ คุณไม่มีเวลาที่จะกลับไปยังโค้ดเดิม และแก้บักที่เกิด มันควรจะทำงานได้อย่างดี ซึ่งหมายถึงมีดีไซน์ที่ดี มีการทดสอบอย่างดี และรวมกันกับโค้ดที่อยู่ในโปรเจคได้อย่างดี (ไม่ใช่ว่าอยู่ที่เครื่องคนเขียนแล้วทำงานได้ไม่เป็นปก แต่พอไปรวมกับโค้ดอื่นแล้วพัง)

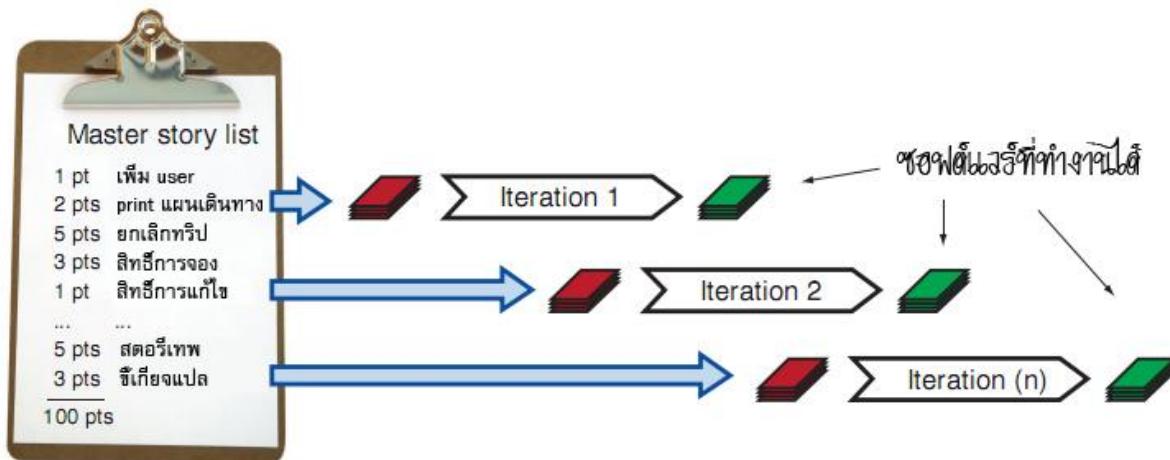
ขั้นที่สาม การทดสอบของคุณควรจะไปควบคู่กับการพัฒนา คุณไม่สามารถรอจนโปรเจคจบ แล้วค่อยดูว่าทุกอย่างทำได้จริงหรือเปล่า? คุณต้องดูและสุขภาพและความถูกต้องของโปรเจคตั้งแต่วันแรกเลย



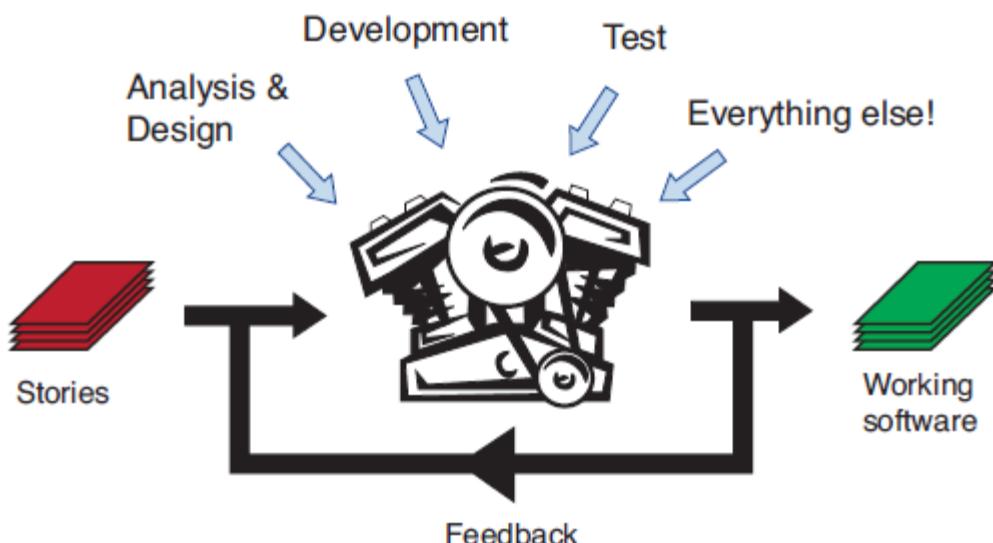
ถ้าคุณทำทั้งสามอย่างนี้ได้ คุณก็สามารถที่จะสร้างบางอย่างที่มีประโยชน์ในแต่ละสัปดาห์ได้แล้ว! และอีกสิ่งหนึ่งที่ดีคือ แบบแผนในการทำแบบดังกล่าว้นี้เพื่อการทำงานแบบ iteration อีกด้วย

## 9.2 iteration ในแอจайл

จนถึงตอนนี้แล้วคุณเริ่มมีไอเดียแล้วว่า iteration ในแอจайлจะหน้าตาเป็นอย่างไร มันคือกล่องของช่วงเวลาในการพัฒนาซอฟต์แวร์ (อาจจะเป็น 1 หรือ 2 สัปดาห์ต่อกล่อง) ที่เราจะเอาส่วนที่อยู่ด้านบน (หมายถึงเรียงตามความสำคัญ) มาแปลงให้เป็นซอฟต์แวร์ที่ทำงานได้



มันเป็นเครื่องมือของคุณที่ช่วยให้ทำงานในโปรเจกแบบแอกไซล์ได้สำเร็จ เป้าหมายคือการสร้างบางอย่างที่มีค่าต่อเวลาที่เราแปลงมันเป็นซอฟท์แวร์ นั่นหมายความว่าจะไก่ตามที่คุณสร้างเพื่อให้มันทำงานได้ มันจะต้องเป็นสิ่งที่ถูกทดสอบในระหว่าง iteration ด้วย (ก็ เพราะต้องแน่ใจว่าทำงานได้ด้วย)



iteration ยังเปิดโอกาสให้เราสามารถปรับเปลี่ยนลำดับการทำงาน เมื่อเกิดความจำเป็นได้ด้วย ถ้าลำดับความสำคัญของงานเรา\_men\_เปลี่ยนไป หรือว่าเกิดสิ่งที่เราไม่คาดหวัง เราสามารถปรับเปลี่ยนลำดับของสิ่งที่จะทำในอนาคตได้ในตอนจบ iteration โดยปกติเราจะไม่เปลี่ยนสตอร์กิกใน iteration (เพราะมันจะกระทบกับทีมมากเกินไป) แต่สิ่งที่คุณกำลังจะเห็นในตอนที่ 10 เรื่องการสร้างแผนงานการติดต่อแบบแอกไซล์ คุณจะพบว่าโอกาสในการเปลี่ยนแปลงเป้าหมายของโปรเจกมันเกิดขึ้นได้ถ้าคุณอยากรู้ว่ามันเกิด

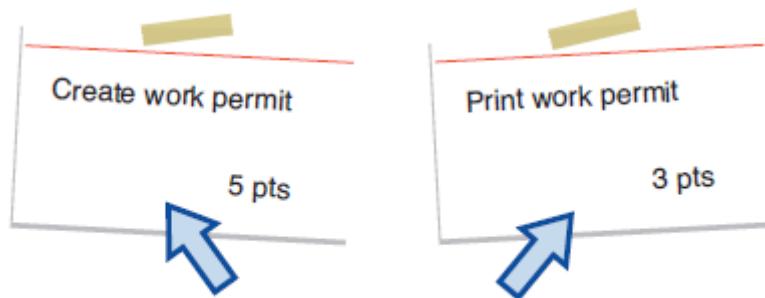
เราคุยกันพอแล้ว ทางที่ดีที่สุด ไปคุยกันดีกว่า iteration ทำงานอย่างไร เพื่อเห็นพูดติ่งรวม ตอนนี้เราเข้า user story มาดูกัน เดอะ ว่าจะมาแปลงเป็น product ที่พร้อมใช้งานได้อย่างไร

### 9.3 ช่วยด้วยyy !!!

ช่วยด้วย วันเริ่มของโครงการก่อสร้าง BigCo เพิ่งถูกเลื่อนขึ้นมาอีก 1 เดือน และเคลลี่ เพื่อนที่แสนดีของเรา ต้องการเว็บไซต์ที่ให้ผู้ทำสัญญาสามารถเข้าไปสร้างใบอนุญาตความปลอดภัยสำหรับการก่อสร้างได้



เราไม่มีเวลาสร้างเว็บไซต์ให้เสร็จทั้งอัน กายใน iteration เดียวอย่างแน่นอน แต่คุณเคลลี่ของเราก็จะดีใจมาก หากเราสามารถส่งงาน 2 สตอรี่นี้ให้เค้าได้อีก 2 สปดาห์ถัดไป



#### This iteration's stories

เพื่อทำให้มันเกิด เรา มี 3 ขั้นตอนที่ต้องดำเนินการกับสตอรี่แต่ละอันในการที่จะแปลงมันเป็นซอฟต์แวร์

1. วิเคราะห์และออกแบบ (ทำให้งานมีมพร้อมจะดำเนินการ)

2. การพัฒนา (ลงมือทำงาน)

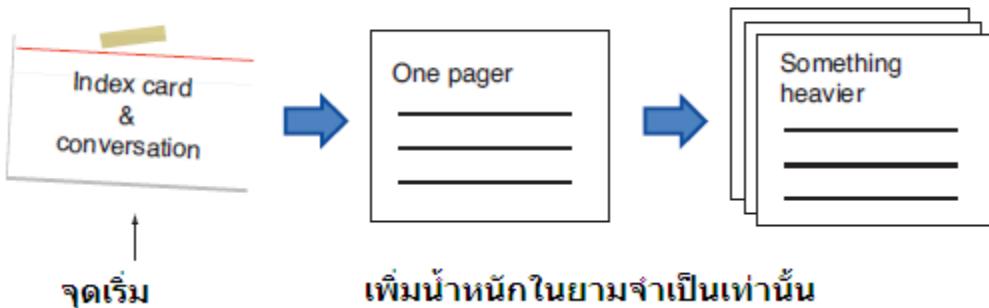
3. การทดสอบ (ตรวจสอบงาน)

มาดูกันใกล้ๆ ว่ามีอะไรเกิดขึ้นบ้าง ในแต่ละขั้นตอน

## 9.4 ขั้นที่หนึ่ง: การวิเคราะห์และออกแบบ: ทำให้งานมันพร้อมจะดำเนินการ

มีอยู่ 2 สิ่งสำคัญในการวิเคราะห์แบบแอ็จайл์ คือ just-enough (ทำเท่าที่ใช้) และ just-in-time (ทำในตอนที่ใช้-เผาเอกสาร) โดยการวิเคราะห์แบบ Just-enough การทำอย่างไรก็ได้ที่ให้เกิดความพร้อมที่จะทำงาน โดยที่ “ไม่มาก และ ไม่น้อย” จนเกินไป

### ท่าอย่างเพียงพอเพียง



สำหรับทีมเล็กๆ ที่ทำงานร่วมกัน และอยู่กับลูกค้าอย่างใกล้ชิด ไม่จำเป็นต้องสร้างเอกสารอย่างเป็นทางการ แค่มีการคิดและการสนทนา (ที่มีการอธิบายด้วยไดอะแกรมและรูปภาพที่ดี) ก็เพียงพอแล้ว สำหรับทีมขนาดกลางที่อาจจะกระจายกันออกไป (แต่ยังเดินไปมาหากันได้) อาจจะต้องทำอะไรมากขึ้นอีกนิด หากมีสักหน้าหนึ่ง ที่มีคำอธิบายสั้นๆ มีการแบ่งงานออกเป็น task และมีรายการของเงื่อนไขการทดสอบ ก็อาจจะเหมาะสมกับทีมมากขึ้น

**Story name: Create work permit**

**Description**

Before contractors can legally work on the construction site, they need a work permit. This permit is what they will take to the job site when they are ready to begin construction.

**Tasks**

1. Create permit page.
2. Save permit to database.
3. Add basic validation.
4. Ignore security (for now).

**Test criteria**

1. Requestor can save basic permit.
2. Permit gets saved to the database.
3. Invalid permits are rejected.
4. Permit defaults to next week's start date.

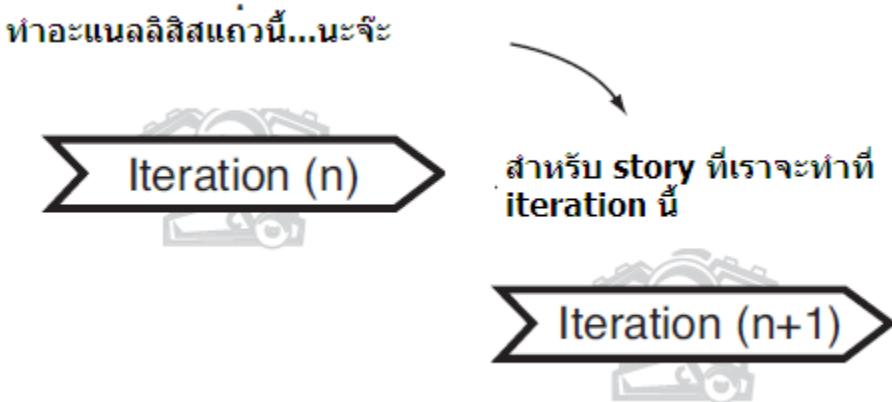
ไม่แปลนจะ มันดีอยู่แล้ว

สำหรับโปรเจคขนาดใหญ่ ที่มีทีมกระจายอยู่ตามที่ต่างๆ เช่น จีกาโก้ ลอนดอน สิงคโปร์ ก็ต้องทำบางอย่างมากขึ้นไปอีกเพื่อให้ทุกๆ คน ยังคงอยู่บนพื้นฐานความคิดเดียวกัน เพื่อให้ไปยังเป้าหมายที่ถูกต้อง ประเด็นคือ จริงๆ แล้ว ไม่สามารถบอกได้ว่าอย่างนั้นชัด การวิเคราะห์ระดับไหนที่ถูกต้องเหมาะสม ในแบบแอลจีร์ มันเขียนอยู่กับวันนั้นข้างกันได้กับคุณและโปรเจคหรือเปล่า? คุณจะใส่น้ำหนักให้มันในตอนหลังก็ได้ถ้าคุณต้องการ แต่การแบ่งของที่ไม่สำคัญไปด้วย มันจะยิ่งทำให้คุณเดินชั้ลง ดังนั้นตอนนี้เริ่มต้นแบบง่ายๆ เปาฯ แล้วค่อยเพิ่มน้ำหนักเมื่อคุณต้องการจะตีกว่าและยกแกนหลักหนึ่งของการวิเคราะห์แบบแอลจีร์ คือ just-in-time (ทำในตอนที่ใช้-เผาฯ)

### ลงรายละเอียดกิต่อเมื่อเราต้องการใช้มันเร็วนี้



การวิเคราะห์แบบ just-in-time คือการที่เราวิเคราะห์ลงไปในระดับลึกของสตอรี่เอน派ต่อนที่เราต้องการใช้มัน (ปกติจะทำการวิเคราะห์ story ของ iteration ถัดไปหนึ่ง iteration เมื่อไหร่ก็วันนั้น แนวๆ ตามลำดับของหน้า)

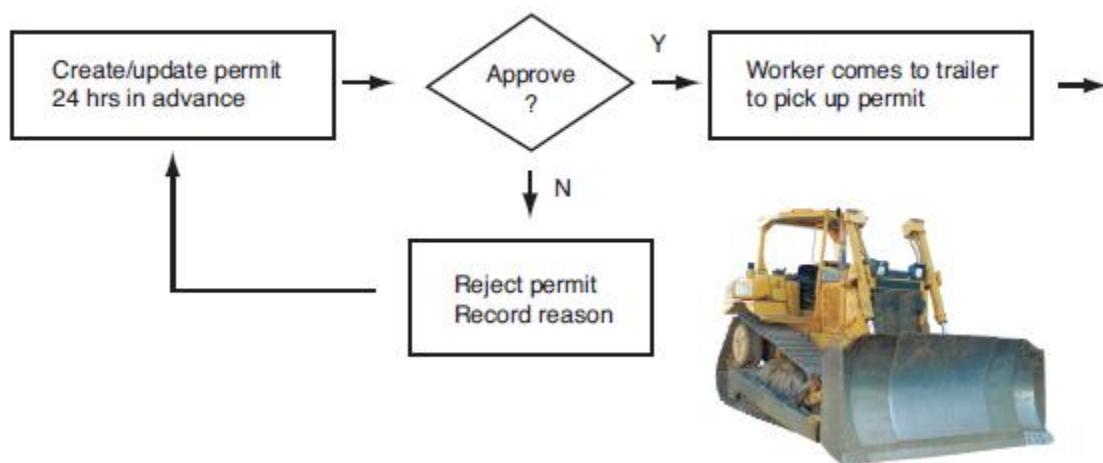


เพราเราไม่รู้หรอกว่าได้จะเปลี่ยนไปอย่างไรในหนึ่งเดือนข้างหน้านี้ ทุกอย่างเปลี่ยนแปลงได้ ดังนั้น การวิ่งอย่างเต็มฝีเท้าและพยายามที่จะทำให้ทุกอย่างถูกต้องดังต้น มันจะจบด้วยการผลิตขยะจำนวนมาก เสียเวลาและเนื่องด้วยสัดส่วน แทนที่เราจะทำแบบนั้น เราควรจะหยุดการวิเคราะห์สตอรี่แบบลึกๆ ไว้ก่อน จนกว่าที่เราจะต้องการมันอีกรึเปล่า ก่อมาวิเคราะห์มัน การทำแบบนี้ ทำให้แน่ใจว่า:

- การวิเคราะห์แบบละเอียดจะถูกทำโดยมีข้อมูลที่ใหม่ล่าสุดใช้ประกอบในการทำงาน
- คุณและลูกค้าให้โอกาสตัวเองในการเรียนรู้สิ่งที่กำลังจะเกิดขึ้น
- คุณหลีกเลี่ยงที่จะต้องทำงานซ้ำอีกรัง

ถ้าสิ่งที่คุณทำมันซับซ้อนมากและต้องการเวลาที่มากขึ้นก็ให้ทำไป ทำอะไรก็ได้ที่ทำให้งานมันพร้อมที่จะดำเนินการ แค่ค่อย่าเดินไปให้ไกลเกินไปแล้วจบด้วยการที่คุณต้องทิ้งทุกอย่าง เพราะมันมีอะไรอะเกินไปที่ต้องเปลี่ยนแปลงดังนั้น เราควรใช้อะไรนำเสนอกิจกรรมที่สตอรี่ เช่น “สร้างใบอนุญาตงาน” ?

### เริ่มต้นด้วยฟลัวชาร์ท งานฯ



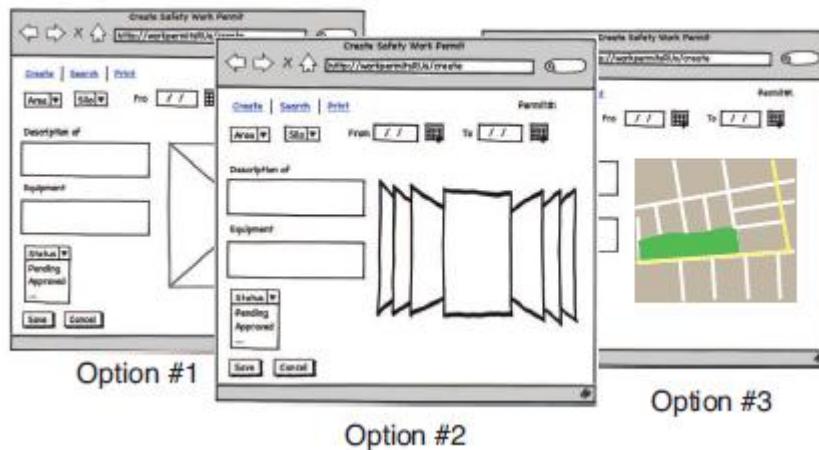
ฟลัวชาร์ทเป็นสิ่งที่สำคัญที่สุด เพราะมันมองง่าย ทำได้แน่นอน มันแสดงให้เห็นการทำงานของระบบ และง่ายให้เห็นขั้นตอนที่คนต้องการจะดำเนินการหรือถ้าต้องการรายละเอียดมากขึ้นก็สามารถเขียนคำอธิบายประกอบลงไปก็ได เพื่อเน้นถึงที่ต้องการในมุมมองของผู้คนนั้นเช่นเดียวกับการเปลี่ยนแปลงที่เราต้องการ แต่ความเข้าใจมากขึ้นว่าอะไรเป็นผู้ให้ระบบนี้ และรู้ว่าต้องทำอะไรกับสิ่งที่เรียกว่า personas



 <p><i>"Amanda"</i></p>	Administrator ต้องการมีสิทธิ์ที่จะเพิ่มและลบผู้ใช้ออกจากระบบใช้งานคอมพิวเตอร์ได้อย่างสบาย เป็นคนที่ทำงาน office ดำเนินการงาน
 <p><i>"Robert"</i></p>	Requestor ผู้จัดการหรือวิศวกรรมก่อสร้างผู้ที่จะขอใบอนุญาตทำงานให้กับลูกจ้างของตัวเอง เป็นคนที่รู้รายละเอียดงานรับผิดชอบในการร้องขอใบอนุญาตตามเวลาที่กำหนด
 <p><i>"Mr. Kelly"</i></p>	Approver ผู้ปฏิบัติการด้านความปลอดภัย รับผิดชอบเรื่องความปลอดภัยทุกๆ อย่างในงาน เป็นคนเชื่อนอนุญาตในใบอนุญาตงานก่อนจะนำส่งถ้าคนนี้เห็นก็แสดงว่าใบอนุญาตนี้ถูกต้องสมบูรณ์

Personas คือสิ่งที่ใช้อธิบายความแตกต่างของคนในแต่ละบทบาทของการใช้ซอฟต์แวร์ซึ่งช่วยให้ระบบเป็นไปตามลักษณะการใช้งานของคนเหล่านี้ด้วย คนเหล่านี้เป็นผู้ที่ใช้ระบบจริง เจอปัญหาจริง และการเข้าใจว่าพวกเค้ามาจากไหน เป็นอย่างไร จะทำให้เราสร้างระบบที่ตอบสนองความต้องการได้อย่างแท้จริง ดังนั้นในเวลาที่ออกแบบระบบจริง โลกนี้ก็เป็นแค่เรื่องหอยๆ ของเราและแทนที่จะต้องไปลงแรงกับดีไซน์ที่ต้องคิดหัวแบบแตก (ไอเดียแบบที่เคยเห็นในห้องทดลอง แบบที่เราคิดอยู่ในหัวว่าชับช้อนนั้นแหล่ะ) เรายังสามารถนำไปปรับเปลี่ยนมาอย่างรวดเร็ว สร้างหลายๆ อันที่ลักษณะแตกต่างกันออกไป (ก็ตามผู้ใช้) และราคาถูกขึ้นด้วย

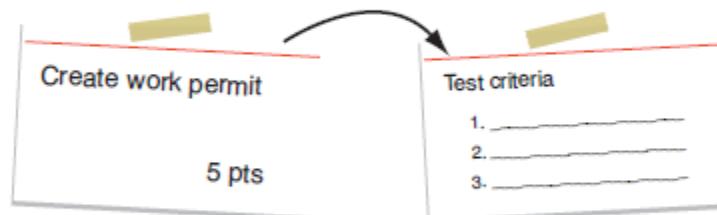
## การร่างต้นแบบบนกระดาษด้วยตัวเขียนที่แตกต่างกัน



สิ่งที่ดีในการที่จะให้ทีมทำงานร่วมกันได้บนกระดาษ คือ คุณมักจะได้ผลงานที่ดีกว่าเสมอ ดีกว่าการที่ให้แต่ละคนไปทำอยู่คนเดียวแล้ว ( เพราะทีมงานเดียวกันไม่ได้อยู่กับใคร คิดไปเองช่วยอะไร )

เมื่อคุณทำงานด้วยเครื่องเรียบหรือแม่เหล็ก ควรนั่งอยู่กับลูกค้าและเขียนออกแบบเป็นเงื่อนไขการทดสอบได้ลักษณะนี้ ตอนนี้ เรายังคงต้องรอจราจรจ่าจะแจ้งเกี่ยวกับสคริปต์รีนันได้ลักษณะนี้

### story นี้ผ่านเกณฑ์เมื่อไร? เขียนไว้หลังบัตรสักกะนิด



เขียนสักสามข้อที่คุณนึกออกว่าจะ test อะไร  
นึกไม่ออกลอง думแบบมีหลักการมา ก็ได้

สิ่งนี้เป็นที่ที่คุณนั่งคุยกับลูกค้าแล้วตามเค้าว่า เจ้าจะรู้ได้ยังไงว่าใช่สิ่งนี้มันจะทำงานได้?

คุณอยากรู้จะลงรายละเอียดเยอะๆ หรือว่า 'น้อยๆ ก็แล้วแต่คุณต้องการ' คุณสามารถเริ่มจากระดับบันๆ ก่อน และทำให้แน่ใจว่าทีมเข้าใจงานชิ้นหลักๆ ที่ต้องทำงานได้เพื่อให้ระบบสำเร็จ

### What About Pair Programming?

Few agile/XP practices have attracted more attention and controversy than this one. Pair programming is the act of two programmers sitting down at one computer and working together on a story. Seeing two valuable resources sitting down at one computer would understandably make any manager nervous. They think their team's productivity has just been halved, and that would be true if programming were merely typing. But it's not. And one good idea or innovation can often save teams a ton of work and rework later. With pairing you spread valuable knowledge and practices throughout the team, you catch more bugs early, and you increase code quality by having two people reviewing every line of code. It's not for everyone, and you have to respect how people work. But if your team is open to pairing (this applies to analysis and testing too), it can often more than pay for itself in return.

การทำงานแบบ **Pair Programming** ก็คือการที่เรานำเอาไปร่วมมือสองคนมานั่งทำงานด้วยกัน ช่วยกันสร้างโค้ดสำหรับ **Story** เดียวกัน

หรือว่า ถ้าสأتต่อว่าของคุณมันไปในทางเทคนิคและมีเงื่อนไขรายละเอียดทางด้านธุรกิจเฉพาะ คุณอาจจะต้องใช้เวลามากขึ้นและเสียเวลามากขึ้น (และอาจจะดีกว่าถ้าคุณเขียนมันออกแบบในรูปแบบที่จะทดสอบได้ในอนาคต)

มีเทคนิคหรือเครื่องมืออื่นๆ อีกมายที่ช่วยในการวิเคราะห์? แนะนำอยู่แล้ว มี **storyboards**, **concurrency diagrams**, **process maps**, **wireframes**, และเครื่องมือวิเคราะห์ที่มีประโยชน์อื่นๆ ที่เป็นที่รู้จัก (อย่างรู้วิธีก็ตามไปดู ตอนที่ 6.4 How to Host a Story-Gathering Workshop, on page 108)

จำไว้ว่า เราไม่สามารถไปโรงเรียนเพื่อให้มีคนสอนว่า ของพวนนี้ทำได้อย่างไร เราต้องจินตนาการด้วยตัวเอง มันบอกไม่ได้ว่าอันไหนถูก อันไหนผิด อะไร และถ้าคุณสนใจก็ลองลองกับสตอรี่ที่เกี่ยวกับการบูร์น์ มันอาจจะแปลไปในทางที่ว่าเราไม่ต้องการมันแล้ว การบูร์น์ที่บ่อนอกจากผ่านบริการเชอร์ฟจะดีเพียงพอแล้วสำหรับรีลีฟแรก ดังนั้นเราจึงตัดมันไปก่อน ดีจริง เราไม่ต้องเสียเวลา กับการวิเคราะห์มันแล้ว

เมื่อการวิเคราะห์ของเราเสร็จสิ้นแล้ว ก็ถึงเวลาเริ่มต้นทำงานซะที

## 9.5 ขั้นที่สอง: การพัฒนา: การดำเนินงาน

ตอนนี้ เรานำเอกสารวิเคราะห์ (ที่ทำเฉพาะตอนที่แล้ว) และแปลงมันให้เป็นทอง หรือในสถานการณ์คือ ให้มันเป็นซอฟต์แวร์ที่ทำงานได้จริง

Create work permit  
5 pts



```
if (userAccountExists)
    RedirectLoginPage();
else
    DenyAccess();
```

Java, C#, Ruby, Python, HTML, CSS



อะไรสักอย่างที่เราเอาไปติดตั้งได้

ตอนนี้การทำซอฟต์แวร์ที่ใช้งานได้ หรือทำท่องขึ้นมาันั้น ไม่ได้มาแบบพรีชา มันต้องใช้พัฒนาอย่างหนัก มีระเบียบวินัยอย่างยิ่ง และต้องมีความตระหนักรู้ในส่วนของโครงสร้างและตัวอย่าง เช่น ในแอปพลิเคชันมีบางสิ่งที่เราต้องทำ

- เราต้องเขียน automated tests
- เราต้องปรับปรุงดีไซน์อย่างต่อเนื่อง
- เราต้อง integrate code ที่เขียนเข้ากับ production อย่างต่อเนื่อง
- เราต้องแน่ใจว่า code เข้ากับภาษาที่ลูกค้าใช้เวลาที่พากเพียรในการอ่าน

เราไม่มีเวลาหรือช่องว่างที่จะครอบคลุมถึงวิธีการทางวิศวกรรมซอฟต์แวร์ที่ดีที่สุดหรือตอนนี้ แต่เราจะครอบคลุมถึงสิ่งที่เรียกว่า non-negotiables (หรือสิ่งที่ หากเราไม่มี เรายากจะตกลงได้ เพราะโดยลูกค้าจะต้องได้ในบทถัดไป เช่น Chapter 12, Unit

Testing: Knowing It Works, on page 203; Chapter 13, Refactoring: Paying Down Your Technical Debt, on

page 213; Chapter 14, Test-Driven Development, on page 225; as well as Chapter 15, Continuous

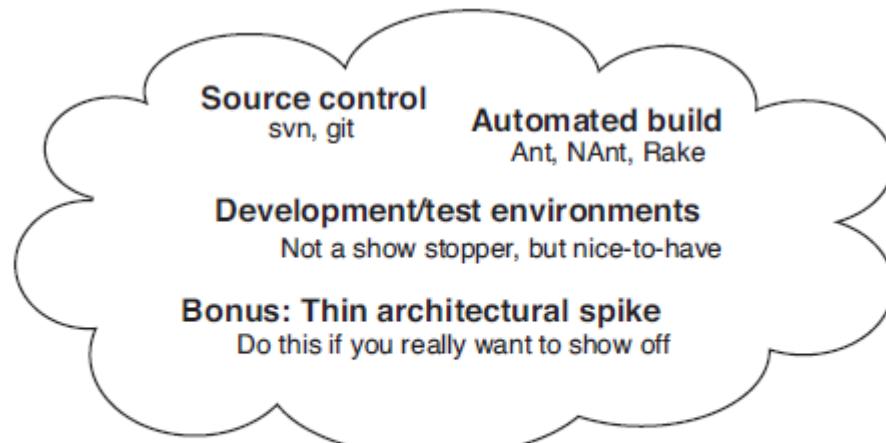
Integration: Making It Production-Ready, on

page 236 เราจะครอบคลุมรายละเอียดเรื่องการวิเคราะห์ TDD และการอินทิเกรตให้ดียิ่ง แสดงให้ดูว่าสิ่งเหล่านี้เป็นส่วนทำให้เกิดซอฟต์แวร์ที่พร้อมใช้งานได้อย่างไร

แต่ตอนนี้นักพัฒนาแค่เข้าใจไว้ว่า ไม่มีเทมนต์แอจайлที่ไหนหรอก จริงๆ มันก็มีแค่เรื่องพื้นฐานที่หน้าแน่นทางวิศวกรรมซอฟต์แวร์ที่อยู่เบื้องหลังทุกอย่างนั่นเอง ตอนนี้ไปดู iteration พิเศษสำหรับโปรเจกคุณกันดีกว่า คือ iteration แรก นั่นเอง (หรือบางคนเรียกว่า iteration 0)

ทำทุกอย่างให้พร้อมที่ iteration 0

มันก็ขึ้นกับว่าคุณจะมองมันยังไง แต่จริงๆ iteration 0 คือ iteration เริ่มแรก หรือเป็น iteration ก่อนที่จะเริ่มออกแบบ จริงๆ มันก็มีแค่เรื่องพื้นฐานที่หน้าแน่นทางวิศวกรรมซอฟต์แวร์ที่สร้างโปรเจกใหม่ มันก็ต้องมีบางอย่างที่โอนเตรียมพร้อมเอาไว้ก่อนที่เราจะเริ่มทำงานจริง มันก็เลยถูกเรียกว่า iteration 0



ของที่เราต้องหาก่อนที่เราจะเข้าสู่ สุด.or ของเรา

Iteration 0 เนื่องจากกับการสร้างบ้านตามคำสั่ง มันเกี่ยวกับการเตรียมทุกสิ่ง เช่น version control, automated build, สร้าง development และ test environment (หรือว่า production environment ด้วย ถ้าทำได้) เพื่อให้เราสามารถ deploy software ลงไปได้แต่ถ้าคุณอยากระหว่างไฟฟ้าแล้ว ก็ลองข้ามไปทำส่วนอื่น (อะไรก็ได้ที่ทำให้คุณได้ทำตั้งแต่เริ่มจนไปถึงการทดสอบ) เมื่องานที่ต้องพัฒนาทั้งหมดเสร็จเรียบร้อยแล้ว เวลาที่เก็บถึงเส้นชัยละ ลิ่งที่ต้องถัดไปคือ ตรวจสอบงานทั้งหมดที่ทำไป

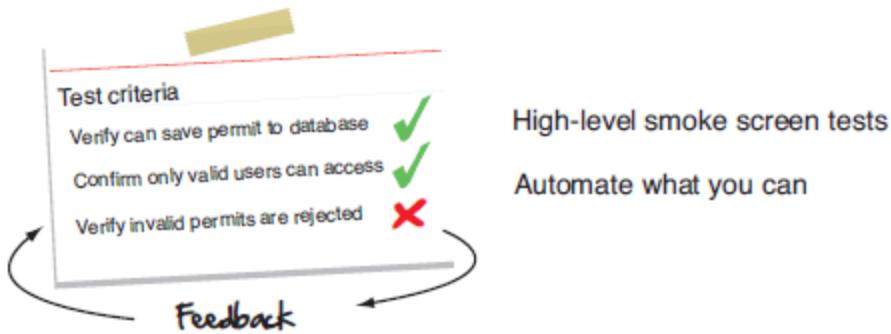
## 9.6 ขั้นที่สาม: ทดสอบ: ตรวจสอบงานที่ทำ

ตอนนี้มีคนจะเจอนามากถ้าเราทำทุกอย่างมากขนาดนี้แล้วด้วยเหตุผลดูว่า ลิ่งที่ทำมันเวิร์กจริงๆ การตรวจสอบงานคือ การที่เราทำให้แน่ใจว่าเราจะสามารถสู้สายไหมได้อย่างเต็มปอด เวลาที่ลูกค้ากำลังทดสอบงานของเรา

เรื่อง Collective code ownership

ไม่มีใครเป็นเจ้าของโค๊ดในแอจจาลลี่โปรเจค  
 เพราะโค๊ดมันเป็นของทีม หมายความว่า ทุกคน ณ เวลาใดก็ตาม  
 จะถูกคาดหวังและสนับสนุนให้เกิดการเปลี่ยนแปลง  
 เพื่อให้งานที่ทำลุล่วงสำเร็จ

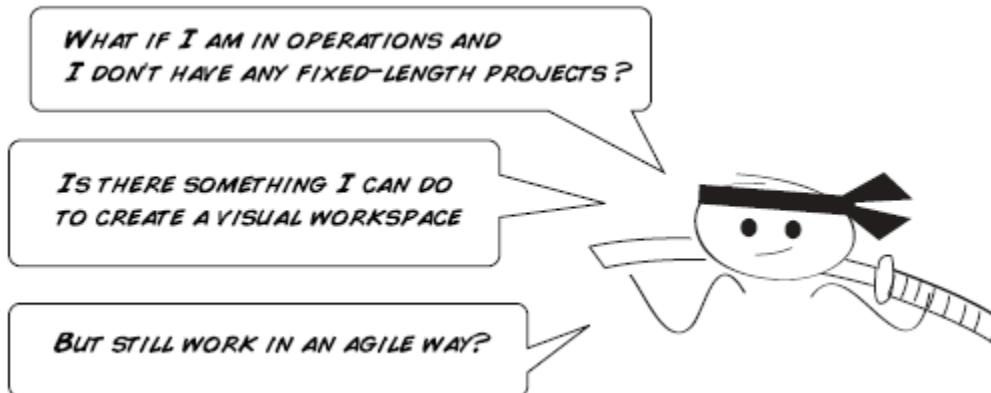
XP ใช้กลิ่งเหล่านี้ว่า เป็น collective code ownership และมันก็เป็นอีกแนวทางที่แอจจาลลี่โปรเจคใช้ในการซ่อมสื่อสาร  
 เกี่ยวกับสถาปัตยกรรมและมาตรฐานการเขียนโค๊ดภายในโปรเจค



การนำเสนอซอฟท์แวร์ให้ลูกค้าโดยการทำไปตามขั้นตอนของเงื่อนไขการทดสอบ (test criteria) ก็เป็นอีกทางเลือกที่น่าสนใจ แต่จริงๆ มันจะดีกว่านั้น ถ้าให้ลูกค้า เป็นคนแล่นเดินไปเองแล้วเราเก็บนั่งด้านหลังเพื่อดูว่า ลูกค้าใช้ซอฟท์แวร์ของเรายังงัย ตอนนี้ เรายังลุล่วงคุณกำลังคิด(บ้า)อะไร เราลงมือทำเทสท์กันจะยิบและด้วยทุกเทสท์ที่รวมมิในโปรดเจคแบบแอจайл์ รายังต้องการ User acceptance test (UAT) ตอนที่ production รีเปล่านะ? คำตอบคือ ใช่ เนื่องจากมันก็คือว่า...

เป้าหมายของคุณในฐานะผู้พัฒนา(คือใครก็ได้ในทีมพัฒนา) คือการทำให้ UAT มันเกิดได้อย่างปกตินั่นคือ ลูกคุณทำการทดสอบดี ให้ไฟด์แบคแก่ลูกค้าในตอนที่ส่งงาน นั่นคือสิ่งที่ UAT ทำทั้งสิ้น ผู้คนส่วนใหญ่มักจะติดอยู่กับ การตามหาความผิดพลาดของระบบ

บางทีมก็มีคุณภาพไปถึงขั้นดังกล่าวตั้งแต่ครั้งแรก (แต่หลายทีมก็ไม่เลย) ดังนั้น คำแนะนำผู้เขียนคือการให้ UAT มันอยู่รอบๆ จนกว่าเราจะจะพบสิ่งที่ทุกคนเห็นว่า คุณและทีมเขียนโปรแกรมที่มีคุณภาพได้เป็นปกติ การทำ UAT แบบเต็มที่จะไม่จำเป็น

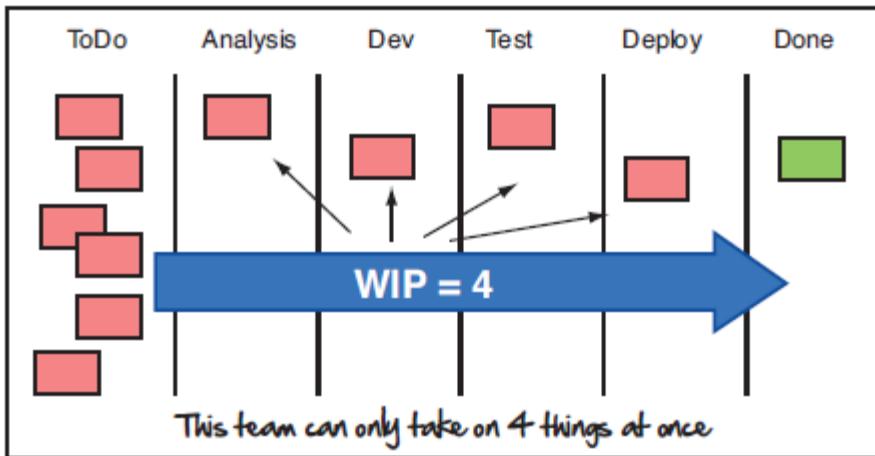


และแน่นอน มันมีแนวทางของแอจайл์ที่เหมาะสม ซึ่งเรียกว่า Kanban

## 9.7 Kanban

Kanban เป็น card-based signaling system (ระบบการให้สัญญาณด้วยการใช้การ์ด) ของโตโยต้า ที่ถูกพัฒนาขึ้นเพื่อเป็นกระบวนการการเติมเต็มชิ้นส่วนในสายการผลิต มันคล้ายๆ กับสตอร์บอร์ดของเรา แต่มีความแตกต่างกันเล็กน้อย

## Sample Kanban Board



No fixed iteration length  
No story/task limit size

\* WIP: Work in progress

งานของ Kanban

จะถูกจำกัดด้วยคุณภาพที่เรียกว่า work in progress (WIP) โดยจะอนุญาตให้ทีมทำงาน ในจำนวนขั้นที่จำกัดในขณะเวลา  
ได้เวลาหนึ่ง

จะเป็นอย่างไรถ้าคุณไม่ยอมให้มีการติดตามบิ๊ก?

ลองคิดดูว่าถ้าเราไม่ยอมให้มีการติดตามบิ๊กในโครงการ  
เราต้องทำอะไรในบังที่ต่างออกไป?

ก็จะ ติดดูว่า คุณอาจจะต้องเบ่งมันให้มัน ตีมมันให้ตายกัน ณ  
จุดที่คุณจะออมมันเลย เพราะคุณจะไม่สามารถกลับมาติดตามมันได้อีก

ถ้ามา คุณอาจจะต้องการหนทางที่จะทำ regression test  
ที่หากได้รับแล้วไม่แพง เพื่อแน่ใจว่าเมื่อคุณได้เบนบิ๊ก (เมื่อันเป็นสิ่ง)  
ตัวนั้นให้ตายไปแล้ว มันได้ตายไปจริงๆ และไม่กลับมาทำางานได้อีกในระบบ

ข้อสาม ถ้าคุณได้ผลลัพธ์บิ๊กจำนวนมากมากอกมา  
คุณอาจจะต้องทำงานช้าลงเพื่อหารือว่าอะไรที่ทำให้เกิดปัญหาเบื้องต้น  
แล้วก็แก้กันตั้งแต่ต้นของปัญหา

นี่เป็นวิธีการติดและพูดติกรรมที่คุณต้องดูแลในทีม  
มันไม่ใช่กรรมมาในเสียงกันเกี่ยวกับเรื่องระบบการติดตามบิ๊ก  
แต่เป็นเกี่ยวกับความติดที่จะเขียนซอฟต์แวร์อย่างไร  
ที่จะทำให้เราไม่ต้องการสิ่งที่ใช้ติดตามบิ๊ก

ตัวอย่างเช่น ถ้าทีมสามารถควบคุมสิ่งของได้ 4 อย่างในครั้งเดียว WIP ของทีมก็จะเป็น 4 สถานะอย่างอื่นที่จะต้องทำให้เสร็จก็จะ  
ถูกโยนออกไปที่ back burner และก็จะดำเนินความสำคัญ แล้วทีมก็จะไปทำมันเมื่อต้องทำ สิ่งอื่นที่แตกต่างออกไปอีกคือ  
Kanban ไม่ต้องการการทำงานแบบ iteration เราสามารถสามารถเลือกสิ่งที่สำคัญเพื่อมาทำด้วย ได้ง่ายๆ จากลิสต์ที่เรามี

แล้วดึงมันเข้ามาเวลาที่ทีมพร้อม เป้าหมายของ Kanban คือ flow เราต้องการให้มี flow ของสิ่งต่างๆ บนบอร์ดให้เร็วที่สุดเท่าที่จะเป็นไปได้ โดยการทำเฉพาะเพียงสิ่งเด็กๆ น้อยๆ ในหนึ่งครั้ง ประโยชน์บางอย่างของการทำงานลักษณะนี้คือ

- เราไม่ต้องมีความเครียดเกี่ยวกับ iteration
- ถ้าเราแยกแยะระหว่างการปฏิบัติการและการทำงานโครงการ เราจะไม่มีวันเครียดเกี่ยวกับการถูก interrupt ระหว่างที่ทำงานใน iteration เพราะมันไม่มี iteration เราสามารถทำงานได้จ่ายๆ โดยการเลือกชิ้นงานตัดไปที่จะทำเมื่อคุณพร้อมที่จะทำและไม่ต้องรีเซ็ตเป้าหมายของ iteration เวลาที่เราเลือกทำงานใหม่
- เราไม่ถูกจำกัดให้ทำงานที่ต้องทำให้เสร็จภายใน iteration
- แม้ว่ามันจะเป็นikoเดียที่ดีในการแยกของชิ้นใหญ่ให้ออกมารูปเป็นชิ้นเล็กๆ แต่มันก็อาจจะมีบางเวลาที่สิ่งของบางอย่างมันใหญ่จริงๆ และเราก็ต้องการเวลาประมาณ 2 สัปดาห์ในการทำให้มันเสร็จ
- เป็นทางเลือกที่ดีในการใช้เพื่อ manage expectation (จัดการความคาดหวัง)
- ทีมส่วนใหญ่ยังทำการ estimation หรืออย่างน้อยก็จัดขนาดของงานในบอร์ด kanban ตามขนาดที่สัมพันธ์กัน (อย่างน้อย มันก็ต้องทำ ถ้าเราอย่างจะตั้งความคาดหวังใน การที่คุณอย่างจะทำงานบางอย่างให้เสร็จ) แต่มันก็ยังมีความเรียบง่ายที่เห็นได้ชัดใน Kanban มันเหมือนกับ “เอ้ สุดหล่อ เรากำลังทำงานหนักมาก เราอย่างให้คุณทำงานให้เสร็จ แต่เราสามารถทำงานได้แค่ 4 อย่างในแต่ละครั้ง” มันไม่ใช่พ้ออยท์ ไม่มีการต้องอธิบาย estimation แค่เป็นการนับง่ายๆ เราสามารถจัดการอะไรมากๆ ได้ในแต่ละครั้ง ทั้งหมดพังดูแล้วบ้ามาก เพราะว่าเราเพิ่งพูดไปหยาๆ ในหนังสือเกือบทั้งเล่ม ว่าการที่เป็น iteration มันดียังงัย - - ” เข้อ.. โปรดทำใจให้สบาย..

iteration ของแอจайл์มันมีอนุภาพมาก และถ้าเรากำลังทำโครงการที่เป็นโปรเจค และมีเงื่อนไขในหลายอย่าง เช่น เวลา และเงิน ในทุกวันนี้ในโลกของอุตสาหกรรมที่มีการตั้งงบประมาณ การใช้ iteration ยังคงจำเป็นอยู่

แต่แอจайл์ก็เป็นมากกว่าแค่ iteration การเป็นแอจайл์หมายความถึงการทำงานอย่างไรก็ได้ให้คุณ ถ้าการทำงานแบบไม่มี iteration มันต้องกับคุณมากกว่า ก็ทำมันไปแบบนั้นแหละ Kanban มันก็ต้องงานที่เกี่ยวกับทีมปฏิบัติการ ทีมสนับสนุน (support team) ที่ต้องการตอบสนองให้รวดเร็วและไม่ต้องการความฟุ่มเฟือยในการทำงานแบบ iteration ที่มีการจำกัดความพยายามของเวลา

คำแนะนำผู้เขียน คือ คุณเพิ่งเริ่มต้นและทำงานเป็นลักษณะโครงการ คุณจะชอบแนวทางและความแม่นยำที่ได้มากจากการชอฟท์แวร์ ที่ได้ส่งมอบให้ถูกคำเป็นระยะในทุกสัปดาห์ ให้คุณเลือกใช้ การยืดมั่นการทำงานแบบ iteration ที่กำหนดความพยายามถ้าคุณทำงานแบบเป็นงานปฏิบัติการ งาน support ให้ลองใช้ Kanban หลักการก็เกือบจะคล้ายกัน การดำเนินงานไปต่อหากที่แตกต่างกันเล็กน้อย

เพื่อเรียนรู้ Kanban ล่าสุดและดีที่สุด ให้ลองเข้าดูที่เว็บไซต์นี้

<http://finance.groups.yahoo.com/group/kanbandev/messages>

ตอนนี้คุณพร้อมที่จะเรียนรู้วันท่านเจ้าสำนักแล้ว

ท่านเจ้าสำนักกับนักรบเจ้าผู้ทะเยอทะยาน

นักเรียน: อาจารย์ยะ พมกำลังทำงานเกี่ยวกับโครงการคลังข้อมูลอุ่นหัว และเราต้องผลิตรายงานด้านการเงินสำหรับผู้บริหารระดับสูง มันไม่มีทางไหนเลยจะที่เราจะผลิตบางอย่างที่มีค่า ในแต่ละสัปดาห์ได้ แค่เอื้อคลังข้อมูลอย่างเดียว ก็กินเวลาผอมไปอย่างน้อยเดือนนึงแล้ว呀ที่จะติดตั้ง ผมจะจัดการ **iteration** ของเผยแพร่ยังคงจะ อะๆๆๆๆๆ ท่านอาจารย์นั่นนน - -"

ท่านอาจารย์: เทคนิคในการส่งมอบบางอย่างที่มีค่านั้น ให้ฟ้าสู่ไปที่ สิ่งที่ทำงานได้เล็กๆ ที่ผู้ใช้งานได้ใน application แทนที่จะสร้างคลังข้อมูลอย่างเดียวในเวลาทั้งหมด ก็แยกส่วนเล็กๆ ของรายงาน และสร้างเฉพาะโครงสร้างของรายงานเล็กๆ นั้น นักเรียน: แต่ยังจ่ายก์ตามหลังจากนั้นอุ่นหัว เราต้องรับบางอย่างที่ใหญ่ๆ แล้วเราจ่ายเงินมาใส่ใน **iteration** เดียวไม่ได้ออยู่ดี นี่ ยะ (???) งึ่ง งึ่ง งึ่ง

ท่าอาจารย์: ถ้ามันไม่พอ มันก็จะไม่พอนั่นแหล่ะ! ก็ใช้จำนวน **iteration** เพื่อที่ต้องการในการสร้างโครงสร้างพอกนั้นแล้วก็ ดำเนินงานต่อไป แต่แค่เจ้าไว้ว่าเราต้องการให้ลูกค้าตกลงตามนี้ และบอกพวกเขาว่าเราจะหายตัวไป 3 เดือนตอนที่พวกเจ้า กำลังติดตั้งซึ่งทำให้พวกเขามาต้องรอ และมันก็ต้องหาทางเจ้าหนทางที่จะส่งมอบบางอย่างเล็กๆ ในแต่ละ **iteration** หลังจากนั้น ได้

นักเรียน: ขอบคุณยะท่านอาจารย์ พมจะคิดเรื่องนี้ให้มากขึ้นสักหน่อยนะ

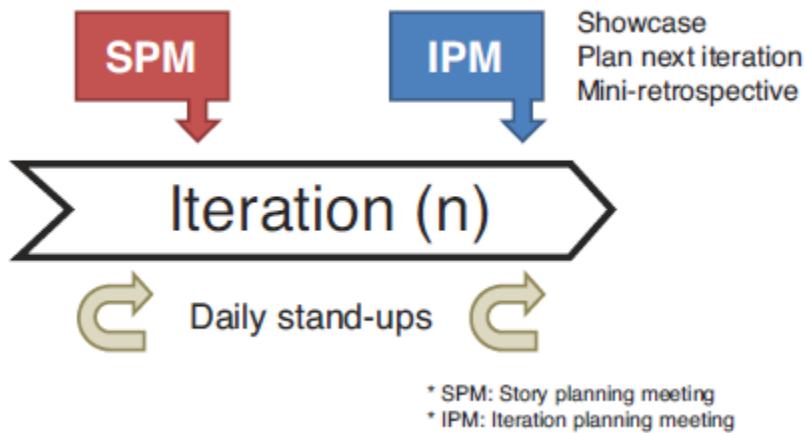
ต่อไปมีอะไร?

ตอนนี้คุณมีอะไร วิเคราะห์ พัฒนา ทดสอบ ทุกบทบาทที่ทำให้ส่งมอบบางอย่างที่มีค่าในทุกสัปดาห์ได้ จำไว้ว่า มันไม่ได้มีทางเดียว ในการทำสิ่งนี้ และวิธีการทำงานของคุณก็อาจจะเปลี่ยนไปในแต่ละโครงการ ดังนั้นไม่ต้องกลัวที่จะทดลองและพยายามในสิ่งที่แตกต่าง :)

ด้วยทุกสิ่งทุกอย่างเบื้องหลังเรานั้น เราพร้อมจะที่จะดูว่า แอจайлร์มีค่าติดต่อสื่อสารและร่วมงานกันในทุกกิจกรรม ที่เกิดขึ้น พร้อมกัน น้อยย่างไรในแต่ละ **iteration** มาดูกันเถอะว่า แผนการติดต่อสื่อสารในแบบแอจайлร์ ทำได้อย่างไร

..... (จบบทแรก คริๆ)

# ตอนที่ 10: การสร้างแพนก้าร สื่อสารแบบเวจิจล์



นอกเหนือจากการแนะนำให้รวมทุกคนในทีมแอจайл์ไว้ในสถานที่เดียวกัน ให้ทำงานอยู่ด้วยกันแบบตัวเป็นๆ และใช้วิดีโอไมโครโฟนหรือวิดีโอดูแลกันอย่างส่วนบุคคล แล้วก็สามารถแลกเปลี่ยนความคิดเห็นได้โดยตรง แต่การจัดการงานใน iteration หนึ่งๆ มันขึ้นอยู่กับคุณและทีมของคุณว่าอย่างไรจะจัดแจง สื่อสาร รับคำเสนอแนะ และทำให้ทุกสิ่งมันเข้าท่าอย่างไร ในบทนี้ คุณจะได้ค้นพบว่าสิ่งที่ขาดไม่ได้เลยในแผนการสื่อสารแบบแอจайл์คืออะไร คุณจะทำให้มันหมายความว่ากับคุณและทีมของคุณยังไง และเมื่อจบบทนี้ นอกจากคุณจะได้แผนการสื่อสารของคุณเองแล้ว คุณจะเริ่มรู้สึกได้ถึงท่วงท่าการทำงานและพิธีกรรมอันศักดิ์สิทธิ์เพื่อให้ได้มาซึ่งผลิตผลอันมีคุณค่าต่อโครงการของคุณอยู่ตลอดเวลา

## 10.1 สื่อถ่ายที่จะต้องทำใน Iteration ใดๆ

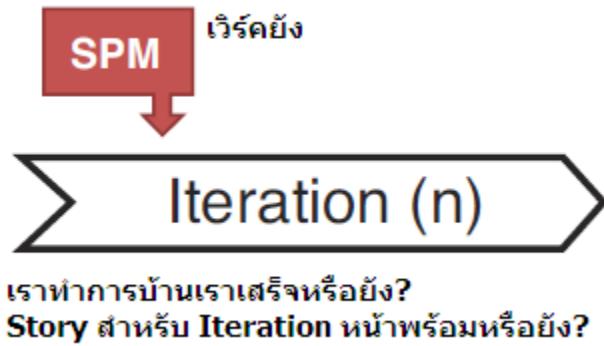
ในโครงการแบบแอจайл์ได้แก่ ตาม สองสิ่งที่ไม่เคยเปลี่ยนแปลงเลยคือการสร้างความคาดหวัง (ตั้งเป้า) และการเก็บ feedback การตั้งเป้าเพื่อสร้างความคาดหวังอย่างสม่ำเสมอ มันจำเป็นก็ เพราะหลายสิ่งมันเปลี่ยนแปลงอยู่ตลอดเวลา คุณจำเป็นที่จะต้องสร้างนิสัยที่ดีในการต้องพูดคุยกันอยู่บ่อยๆ เพื่อให้เค้าเห็นว่าตอนนี้สถานะของโครงการคุณเป็นอย่างไร และเพราะเรื่องง่ายๆอย่างการส่งมอบซอฟต์แวร์ให้ถึงมือลูกค้าคุณจะนำพามาซึ่งการเปลี่ยนแปลงของ requirement คุณก็จะต้องพยายามให้วังวนของการให้และการได้รับ feedback นี้เกิดขึ้นอย่างสม่ำเสมอ เพื่อที่คุณจะได้ไปถึงเป้าหมายที่ถูกต้อง ไม่อกซุนอกทาง และในวันนั้น สื่อถ่ายที่คุณจะต้องการเพื่อให้พิธีการศักดิ์สิทธิ์ของคุณเกิดขึ้นได้อย่างสม่ำเสมอในแต่ละ iteration คือ

1. การเตรียมงานสำหรับ iteration ต่อไปให้พร้อม (การตรวจความพร้อมของงาน - Story Planning Meeting)
2. การเก็บ feedback ของงานใน iteration ที่ผ่านมา (การนำเสนอผลงาน - Showcase)

3. การวางแผนงานสำหรับ iteration ต่อไป (การวางแผน iteration - Iteration Planning Meeting)
4. การพิจารณาหาจุดที่ต้องปรับปรุงอยู่ตลอดเวลา (mini-retrospective)

ขั้นตอนมาเริ่มจากการพิจารณาเตรียมงานสำหรับ iteration หน้าให้พร้อมกันเสมอ

## 10.2 The Story Planning Meeting การประชุมเพื่อวิเคราะห์เนื้องาน



นี่คือการจัดจุดตรวจเพื่อดูความพร้อมของตัวงานแบบทันเวลาพอดี ในการประชุมนี้ เราจะໄລຍ້ความเข้าใจในตัวเนื้องาน (exit criteria) ระหว่างเรากับลูกค้าว่าตรงกัน ทบทวน estimate กับนักพัฒนา เพื่อเป็นการให้แน่ใจว่าทุกคนได้ทำการบ้านและ เตรียมความพร้อมก่อนการเริ่มงานใน iteration หน้ามาดี

ยังไงคราวนี้ก็ต้องมีพลาดกันบ้าง - เพราะฉะนั้น อย่าได้กลัว

ผมได้เคยพัฒนาระบบการพิมพ์ให้กับโปรดักส์ร้างอันหนึ่ง และผมก็ได้เดินทางสายเดียวกันกับนักกรุบแห่งเมืองสปป.ต้าว (นั่นคือการทำให้มันทำงานได้ โดยไม่ได้ใจในรายละเอียด) ทันทีที่ผมสามารถให้กับลูกค้าดู ผมรู้ทันทีว่าพวกเค้าไม่ชอบ

คุณลูกค้าก้าวสู่ภาพเกินไปที่จะพูดที่อะไรก็ตาม แต่ผมเองก็รู้ดีว่า นี่ไม่ใช่งานที่ดีที่สุดของผม ถึงจุดนั้น ผม ก็ได้แต่กลับลืนฝืนหน และขอโอกาสเดียวให้ผมแก้ตัวอีกที ซึ่งพวกเขาก้อยินยอม

ถ้าผมไม่ได้ทำงานส่งพวกเขาย่างดูเดือดตลอด 7 อาทิตย์ที่ผ่านมา คำตอบที่เขามาให้อาจจะเป็นอย่างอื่น แต่ถ้าคุณทำให้ ลูกค้าของคุณตระหนักได้ว่าคุณได้ทำงานตัวเป็นเกลี้ยงเพื่อพวกเขาระบุในทุกๆอาทิตย์ที่ผ่านมา มันก็อยู่กับมีโอกาส มากกว่าที่เขาจะยกโทษให้คุณ และไม่ถือสาหากความมากนี้คุณทำอะไรพลาดในบางครั้ง

เพราะฉะนั้น อย่าได้กลัวที่จะลองอะไรใหม่ๆ การลองผิดลองถูกและการริเริ่มทำอะไรล้วนแต่เป็นส่วนหนึ่งของเกมส์ทั้งนั้น

บางที่คุณอาจจะพบว่างานชิ้นนี้มันใหญ่กว่าที่คุณคิด ไม่เป็นไร ก็แค่อยมันออกมาก่อนให้เป็นหลาๆก่อนที่เล็กลง เพื่อให้มันทำจบ ได้ภายใน iteration ปรับแผนใหม่ แล้วก้มุ่นหน้าทำมันต่อไป ข่าวดีก็คือ เรื่องอย่างนี้มันเกิดขึ้นได้ทั้งสองด้าน (บางที่เราจะ พบว่างานบางชิ้น “เล็ก” กว่าที่เราคิดไว้!)

คุณอาจจะไม่เห็นว่า SPM นี้ถูกกล่าวถึงในหลักการแอจайл์แบบทางการใดๆ นั่นก็เพราะมันเป็นแค่กลวิธีหนึ่งที่ผู้ผลและคนอื่นๆ เห็นว่ามันช่วยให้เราได้ยังไงให้เกิดการเสียเวลาไปกับการเริ่ม iteration ที่ยังไม่ผ่านการวิเคราะห์ที่ดีมาได้และนั่นคือความพยายามและความอดทนของแอกเจิต์ มันไม่มีสิ่งไหนที่ถูกที่สุด หรือทำได้แค่เพียงทางเดียวเท่านั้น ถ้าคุณต้องการอะไรก็อย่าง ก็ขอสร้างสรรค์มันขึ้นมาด้วยตัวคุณเอง (อย่าได้แคร์ว่าหนังสือเล่นไหนจะสอนนายังไง)

สิ่งต่อไปที่คุณจะอยากรีบมันทุก iteration นั่นก็คือการเก็บ feedback จากลูกค้าของคุณ

## 10.3 โซลูชันของ The Showcase

สำเร็จแล้ว!!! คุณได้ส่งมอบงานที่มีคุณค่าบางอย่างกับลูกค้า คุณรู้บ้างมั้ยว่ามีโปรเจคเยอะแค่ไหนที่คื้อทำกันเป็นอาทิตย์ เป็นเดือน หรือแม้เป็นปี แต่แล้วสุดท้ายมันก็ไม่ก่อให้เกิดคุณค่าอะไรเลย

การจัดแสดงผลงาน (Showcase) นี้ เป็นโอกาสที่คุณจะได้อวดผลงานและสิ่งที่คุณและทีมของคุณได้ทำให้โลกได้รู้ อีกทั้งยังเป็นโอกาสอันดีที่คุณจะได้รับ feedback อันแท้จริงจากลูกค้าของคุณ ในระหว่างการ showcase นี้ คุณและทุกคนในทีมคุณจะต้อง demo story ของ iteration ที่ผ่านมา นั่นหมายถึงการโชว์ code จริงที่ deploy ไว้ที่ test server มันไม่ใช่ว่าคุณต้องสร้างภาพที่สวยงาม แต่เป็นการโชว์ของที่คุณสามารถเอาไปใช้งานได้จริง และสามารถลงที่ production server วันนี้ได้ตั้งแต่ตอน แค่นั้น มันก็อัพเดตสมบูรณ์

การ showcase ควรเป็นอย่างไรที่สุดและเป็นวิธีที่ดีในการปิด iteration ของ!! หากนั่นหมายความ เหล้ายากไปปั้งมาเดียงกัน โชว์ออฟ เก็บ feedback ให้ลูกค้าของคุณเป็นคนขับเคลื่อนการ demo แล้วสังเกตดูเค้าใช้ software เราเป็นยังไงบ้าง

ที่นี่ลองมาดูการประชุมนึงที่ทั้ง Scrum และ XP แนะนำว่าคุณควรมี นั่นคือการวางแผน iteration หรือ Iteration Planning Meeting (IPM)

## 10.4 Iteration Planning Meeting

IPM คือช่วงเวลาที่คุณมาจับเข้าคุยกันกับลูกค้าของคุณและวางแผนงานที่จะต้องทำใน iteration หน้า คุณต้องดู velocity ของทีมคุณ สำรวจ stories ที่กำลังจะเข้ามา แล้วตัดสินใจด้วยกันว่ามีอะไรบ้างที่คุณและทีมคุณจะสามารถทำได้ใน iteration หน้า IPM ก็เป็นอีกช่วงเวลาหนึ่งที่หมายความว่าการเช็คสุขภาพของโปรเจค



### ห้องฟ้านจ์ ais

- ไปต่อได้เจว
- อายไรก่อหยุดเราไม่อยู่
- อายไรๆ ก่อคุณเมื่อจะเดินขึ้น



### มีเมฆในบ้านพื้นที่ ฝนอาจจะตก

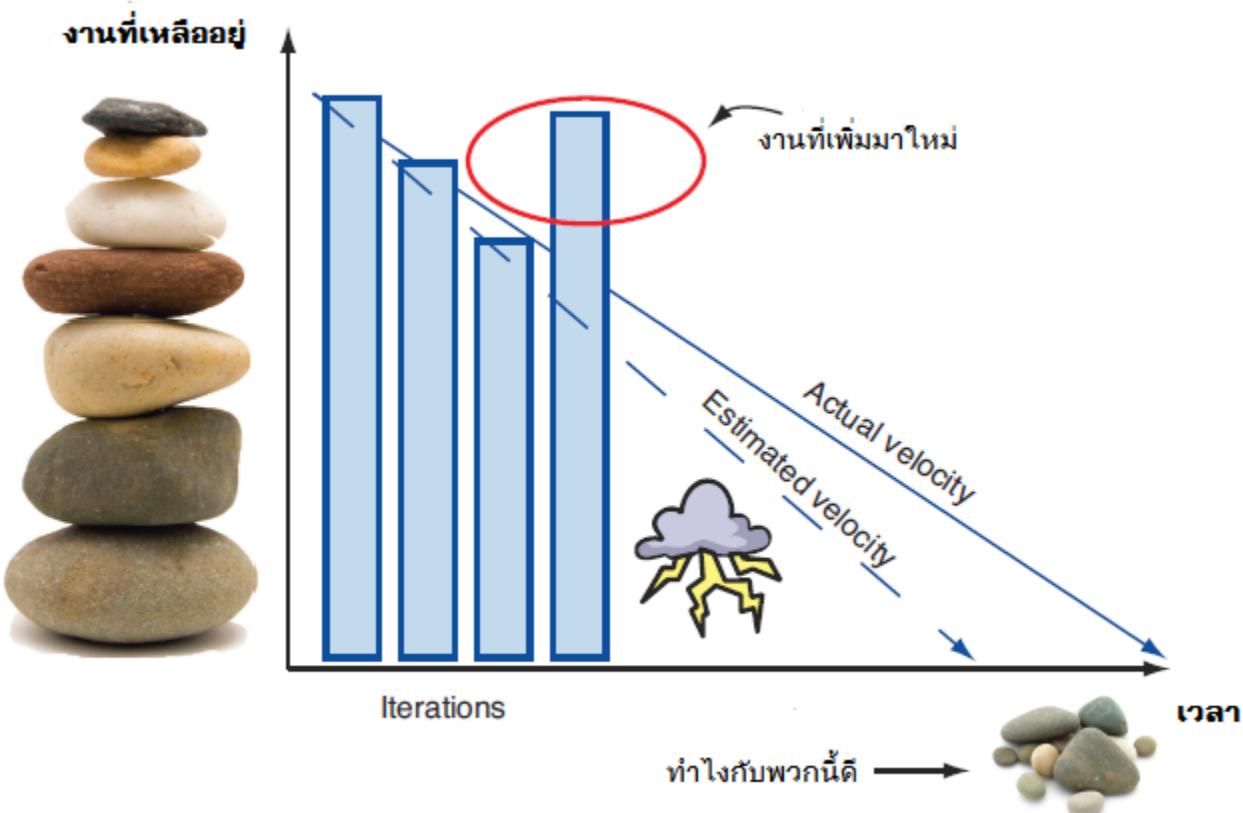
- ยังน่าจะไปถึงฝั่งอยู่
- อาจเจอกลุ่มเล็กน้อย
- แต่ก็ไม่มีอะไรที่เอาไม่อยู่



### ฝนฟ้าคะนอง

- งานเข้า เรากวยแส้ว
- มีหลากหลายมากมาย
- ต้องการความช่วยเหลือด่วน

ณ จุดนี้คุณสามารถพยากรณ์ได้ว่าใน gerade ของคุณอยู่ในสภาพไหน มีอะไรหรือปัจจัยที่คุณต้องจัดการ ถึงมันเดียวนี้ นี่คือโอกาสที่คุณจะได้ยกประดีนและเสนอตัวเลือก หรือแม้แต่เสนอแนวทางที่คุณอยากรับไปต่อ หากมีการพูดถึงวันที่ (ที่จะต้องส่งงาน) จะใช้ burn-down chart ของคุณให้เกิดประโยชน์ ถึงแม้มันจะเป็นความจริงใจขันให้ร้าย แต่มันก็ไม่ได้เป็นการหวั่นไหวไปกับอารมณ์ มันจะบอกคุณและลูกค้าของคุณได้ว่าวันที่นั้นมีความเป็นไปได้มากน้อยแค่ไหน



### ทำข้างในสิ่งจะได้ feedback ที่สร้างสรรค์และเกิดประโยชน์

การให้ feedback ทำได้สองวิธี คุณจะพูดตรงๆไปเลยก็ได้

“ใจ ผิดสังเกตว่าคุณทำ print module ใน iteration ที่แล้วได้มากเลย แต่ว่า unit test ยังขาดไปอยู่มากนน”

หรือจะเข้าไปมีดีไปอ่านน้ำผึ้งก่อนก็ได้

“ใจ เยี่ยมมากเลยงาน print module ที่คุณทำนั้นใช้ความปราณีตขั้นนักกับ unit test ของคุณสิ ไม่นานคุณไปประดับเจ้าโลกแน่!”

สังเกตเห็นความแตกต่างมั้ย แค่คุณเลี่ยงไม่ใช้คำว่า “แต่” มันก้อฟังและรู้สึกต่างกันพี๊บเหอะ

ผมไม่ได้หมายความว่าคุณควรจะใส่เข้าไปน้ำตาลในทุกการสื่อสารของคุณ แต่บางทีการเปลี่ยนน้ำคามบางอย่างของสิ่งที่จะสื่อสารเพียงเล็กน้อย ก็อาจให้ผลแตกต่างกันอย่างได้ชัดในการพยายามปรับเปลี่ยนพฤติกรรมของคน

ถ้ายังอยากรู้ศึกษาเพิ่มเติมเรื่องการสื่อสารอย่างไรจึงจะมีประสิทธิภาพ ผมแนะนำให้อ่านหนังสืออมตะของ Dale Carnegie ชื่อ *How to Win Friends and Influence People* [Car90].

นี่คือส่วนที่ทำให้เกิด visibility ใน agile เราต้องการความโปร่งใสมากที่สุดกับลูกค้าและผู้มีส่วนได้เสียในโปรดเจ็คของเรา การแจ้งข่าวร้ายแต่เนื่นานนี่แหล่ะคือวิธีแยกใจลสั่งสุดท้ายที่เราอยากระทำการก่อนที่จะจบ iteration คือการถามตัวเราเองว่ามีอะไรบ้างมั้ยที่เราอาจจะทำได้กกว่านี้

## 10.5 จัด Mini-Retrospective ยังไงดี

เราสามารถจัด Retrospective ให้ใหญ่โตหรือเล็กๆ อลังการ หรือเป็นงานทดลองอย่างหนึ่งเมื่อจบ release ให้ญาหือเมื่อ project จบ แต่นั่นไม่ใช่สิ่งที่ผมจะพูดถึงตอนนี้ Retrospective ที่ผมยกจะพูดถึงคืออันที่ทำกันแบบสั้นๆ 10 - 15 นาที คือช่วงเวลาที่คุณและทีมคุณมาลงคุยกันเพื่อพูดถึงสิ่งที่คุณทำได้เจํ และสิ่งที่ควรจะทำให้ได้กกว่านี้ กฎประการชิ้นของการจัด retrospective ที่ดีคือการทำให้ทุกคนรู้สึกปลอดภัย ถ้าคุณมีปัญหาภัยเรื่องนี้ ดึงเจตนารวมมือหลักของ retrospective (The Retrospective Prime Directive) ออกมายกเพื่อเดือนใจทุกคนว่าเราไม่ retrospective เพื่ออะไร

### The Retrospective Prime Directive

ไม่ว่าอะไรจะเกิดขึ้น พวกราเข้าใจและเชื่อว่า ด้วยข้อมูลที่มีอยู่ ณ จุดนั้น รวมถึงด้วยทักษะและความสามารถ ทั้งทรัพยากรที่มีอยู่ และสถานการณ์ ณ ขณะนั้น ทุกคนได้ทำเต็มที่แล้ว  
(หรือถ้าจะให้พูดอีกนัยนึง นี่ไม่ใช่ยุทธการล่าแม่มด!)

จากนั้นก็อุ่นเครื่องด้วยการตาม retrospective คำถามแรก

### 1. มีอะไรบ้างที่เราทำมันได้ดี

“มานะ unit tests ของคุณเจ่งมาก”

“ขอรับ แม้เดี๋ยวนี้ ได้ Style Guide ที่คุณทำขึ้นมา แล้วก้อที่คุณไปตามไล่ refactor style sheet ของเราให้มันเป็นไปในลักษณะเดียวกัน ที่นี่เรากำจดทำให้หน้าตาของ application ของเราดูดีและสดคล่องกันได้ง่ายขึ้น”

การสรุวภาพดิกรรมที่ดีเพื่อมาตรฐานนี้มีความเช่นนั้นมักก่อให้มีอนเติมถ่านให้กับกองไฟของทุกคน แม้จะส่งเสริมให้พัฒนารูปแบบที่นี่มีมากขึ้นเรื่อยๆ ในโปรเจ็คของคุณ

เมื่อพูดถึงด้านบวกแล้ว เรา ก้อต้องพูดถึงด้านลบกันบ้าง

### 2. มีจุดไหนบ้างที่เราสามารถทำให้ดีกว่านี้ได้

“ทุกคน ใน story ชุดที่แล้วนี่มีบักกหลุดออกจากมาตรฐาน เราแตะเบรกกันหน่อยดีมั้ย ใช้เวลาให้เต็มที่จนเรามั่นใจแล้วว่าทุกอย่างทำงานได้ แล้วก้อทำ unit test ให้ครอบคลุมด้วย”

“รู้สึกว่าตอนนี้มี code ข้าๆอยู่ในโปรเจ็คเรามากมาย อย่าลืม refactore ด้วยล่ะเวลาไปแพะแกลนนั้น”

“มองให้ชัด ใช้ print story ที่ผอมทำไปมั่นแย่มากๆเลย ด้วยขอผอมแท้ทัวร์ใน iteration นี้ รับรองเวอร์ชันหน้าจะดีกว่ามีมากมายແเน่นอน”

ไม่ก่อปัญหาจะเป็นอะไร การจัด retrospective และการมาช่วยกันคิดแก้ปัญหา และแบ่งปันข้อมูลกับทีมก็เป็นวิธีที่ดีที่สุดในการเติมไฟให้กับทีม และช่วยให้ทีมฟอกสกับจุดที่ต้องการการบันปูรุ่ง แล้วที่นี่คุณก้อสามารถที่จะสร้าง theme ในสองสาม iteration ข้างหน้าเพื่อที่จะติดตามความคืบหน้าของสิ่งที่ต้องการจะปูปูด้วยอย่างจะศึกษาให้ลึกซึ้งถึงการจัด retrospective ที่ดี ลองอ่าน Agile Retrospectives: Making Good Teams Great [DL06] เพิ่มเติมดู เอาล่ะ เรายาจะกันด้วยวิธีดีๆ ที่จะทำให้ทุกคนเห็นภาพเดียวกันก่อนการเริ่มวันทำงานทุกวันกับ Daily Stand-up กันเถอะ

## 10.6 อย่าจัด Daily Stand-Up แบบนี้

Daily Stand-Up คือการแรร์ข้อมูลที่สำคัญกับทีมในรูปแบบที่กระชับ รวดเร็ว แม้คือการประชุมที่จะทำให้การประชุมอื่นๆหมดความสำคัญ และมันใช้แค่เพียงห้าถึงสิบนาที ไม่จำเป็นต้องมีกราฟหรือตารางอะไรให้มากmany (เพื่อเตือนใจให้เราต้องทำการประชุมให้กระชับ) แค่บอกว่าสถานะว่าตอนนี้คุณกำลังทำงานอะไรชิ้นไหนอยู่ และข้อมูลอย่างอื่นที่คุณคิดว่าทั้งทีมต้องรู้ไว้หนังสือใจล้วนๆในตอนนี้ แค่บอกว่าสถานะว่าคุณต้องยืนเป็นวงกลมและหันหน้าเข้าหากันเวลาทำ stand-up และแต่ละคนจะต้องบอกทุกคนในทีมดังต่อไปนี้

- เมื่อวานคุณทำอะไร
- วันนี้คุณจะทำอะไร
- มีอะไรบ้างที่ขัดขวางการทำงานของคุณหรือทำให้คุณทำงานช้าลง

ก้อเป็นข้อมูลที่ดี แต่วันไม่ค่อยสร้างสรรค์แรงบันดาลใจหรือก่อให้เกิดการเปลี่ยนแปลงใดๆเลย

ที่นี่ ลองเปลี่ยนดูบ้าง ทุกครั้งที่พากคุณมารุมตัวกันในตอนเริ่มวัน ลองเซร์ฟิ่งต่อไปนี้ให้ทีมฟัง

- เมื่อวานคุณได้เปลี่ยนแปลงใดอย่างไรบ้าง

- วันนี้คุณจะทำอะไรเพื่อโลกได้บ้าง
- คุณจะฝ่าฟันกับข่าวกหนามที่มากับข่าวของคุณเหล่านั้นได้ยังไง

การตอบคำถามเหล่านี้มันเปลี่ยนความถูกของการทำ stand-up โดยสิ้นเชิง แทนที่จะยืนอยู่เฉยๆแล้วรายงานสถานะ ตอนนี้คุณกลับกำลังวางแผนกู๊ดกอกอยู่! หากคุณทำเข่นี้แล้ว สิ่งที่จะตามมา ก็มีอยู่อย่างสองอย่าง คือคุณจะทำตามแผนที่คุณวางแผนไว้ให้สำเร็จลุล่วง หรือคุณจะไม่ทำมัน ทั้งหลายทั้งปวงแล้วมันก็ขึ้นอยู่กับคุณทั้งหมด แต่สิ่งนึงที่ผมบอกคุณได้ก็คือ ถ้าคุณมีหน้ามายืนในวง stand-up และประกาศกล้าๆต่อทุกคนว่าคุณจะทำสิ่งนั้นๆภายในวันนี้ มันจะเพิ่มโอกาสที่คุณจะทำงานสิ่งนั้นได้สำเร็จอย่างน่ามหัศจรรย์!

## 10.7 ทำอะไรก็ได้ถ้ามัน work สำหรับคุณ

ตอนนี้ถ้าคุณสงสัยว่าเราต้องจัดการประชุมทั้งหลายทั้งปวงนี้ແยักษันหรือ หรือว่าเราสามารถรวมมันทั้งหมดไว้ใน การประชุมเดียวได้ คำตอบก็คือ แล้วแต่คุณสิ! เพื่อที่จะให้จำนวนครั้งในการประชุมลดลง บางทีมชอบที่จะรวมเอา showcase, iteration planning และ retrospective เข้าไว้ด้วยกันใน session เดียวแล้วเอาให้จบภายใน 1 ชั่วโมง (ซึ่งผลก็ชอบแบบนี้มากกว่า และได้นำเสนอในที่นี่ภายใต้ IPM 1 ครั้ง) บางทีมชอบที่จะแยกการ planning ออกจาก showcase แล้วก็อัด retrospective ให้สนุกๆตอนท้ายๆอาทิตย์ บางทีมสนใจที่จะถูกค้ามากๆ จนแทบไม่ต้องมี story planning meeting เลย เพราะพวกเค้าคุยกันทุกวันและมี design session กันได้ทันทีเมื่อมีความจำเป็น จำเป็นให้มีวาระเดียวในการทำอะไรพวกนี้ ถ้าอะไรทำไปแล้วมันไม่เกิดประโยชน์ ก็ไม่ต้องไปทำมัน ลองอะไรใหม่ๆแล้วดูว่ามันใช้ได้มั้ยสำหรับคุณ เอาเป็นว่าสิ่งที่คุณต้องทำแน่ๆ ณ จุดใดจุดหนึ่ง ใน iteration คือการเปลี่ยนถูกค้า เอา software ที่มันใช้การได้ไปให้ค้าๆ สร้างความคาดหวัง และหาวิธีที่จะปรับปรุง โฉมโฉม ถึงควรที่พิจารณาต้องการจะดูแล้วว่าคุณตามทันแเทคโนโลยีใหม่ๆที่มีอยู่ วิบัติภัยงานตัวที่สำนัก และขอให้โชคดีกับแบบฝึกหัดที่พิจารณาเตรียมไว้ให้นะ! ยินดีต้อนรับกลับบ้านนะเด็กๆ ฉันได้เตรียมบทวิชาไว้ทดสอบความพร้อมในการรับมือกับกลไกของ iteration ในชีวิตจริง ให้พวกเจ้าได้ลองดู จนถ้าน่าให้ล้มเหลว ก็อย่าลังเลที่จะลองดู

### จากที่ 1: Story ที่ไม่ครบถ้วนสมบูรณ์

**อาจารย์:** วันนี้ในระหว่าง iteration planning meeting ทีมได้ค้นพบว่ามี story ขันนึงที่ทำไปได้เพียงแค่ครึ่งนึง ด้วยความที่อยากรู้ให้ดูเหมือนมีความคืบหน้า project manager เลยเสนอว่าจะนับ point ครึ่งหนึ่งให้เป็น team velocity ของ iteration นี้ แล้วอีกครึ่งนึงจะเอาไปรวมกับ iteration หน้าเมื่อ story เสร็จสมบูรณ์ เจ้าคิดว่า นี่เป็นความคิดที่ดีมั้ย

**ศิษย์:** ข้าคิดว่าในเมื่อ story มันได้ถูกทำไปแล้วครึ่งนึงจริงๆ ก็ไม่น่าจะเสียหายอะไรที่เราจะแสดงถึงสถานะที่ถูกต้องของ story โดยการนับ point ครึ่งนึงให้เป็น velocity ของ iteration นี้และอีกครึ่งนึงให้เป็นของ iteration หน้า

**อาจารย์:** เจ้าคิดเข่นั้นหรือ ถ้าอย่างนั้น ตอบอาจารย์มาชิว่าช้าสามารถสามารถย้ายข้ามโดยใช้เกวียนที่มีล้อเดียวได้หรือเปล่า คนเราสามารถใช้ตะเกียบเพียงชิ้นเดียวทันทีได้ไหม แล้วลูกค้าจะเข้าใจที่มี feature เพียงชิ้นเดียวไปใช้ production ได้หรือปล่าว

**ศิษย์:** ไม่ได้ครับ พระอาจารย์

อาจารย์: สำหรับนักเรียนใจล้ำนั้น มันไม่มีสิ่งที่เรียกว่า сервисค้างๆ กลางๆ มีแค่งานเสร็จ หรือไม่เสร็จเท่านั้น และด้วยเหตุผลนี้ นักเรียนนับถือ point มาเป็น velocity ให้เฉพาะ story ที่สมบูรณ์และผ่านการ test มาแล้วเท่านั้น story ที่ยังไม่เสร็จสมบูรณ์ ก็จะต้องถูกผลักไป iteration หน้า

## จากที่ 2: เมื่อ Stand-Up ไม่ก่อให้เกิดประโยชน์

อาจารย์: ภาครังนี้ มีทีมนานี่ที่ต้องพยายามอย่างมากเพื่อที่จะให้สมาชิกทุกคนเข้าร่วม stand-up meeting ประจำวัน ทีมรู้สึกว่า stand-up มันไม่ได้ก่อให้เกิดประโยชน์อะไรขึ้นมาสำหรับพวกเข้า มันอาจจะดีกว่าถ้าพากเด็กๆ เอาเวลาไปทำงานแล้วคุยกันเฉพาะเมื่อจำเป็น ที่มีน้ำควรทำอย่างไรดี

**ศิษย์:** ผู้นำทีมควรที่จะย้ำเตือนทีมถึงความสำคัญในการทำให้ทุกคนเห็นภาพเดียวกัน และบทบาทสำคัญที่ stand-up meeting จะช่วยให้เราบรรลุจุดประสงค์นี้

อาจารย์: ถูกต้อง ทีมควรจะต้องมาต่อรองคู่ๆ จุดประสงค์ของการมี stand-up ประจำวัน และเหตุผลที่เราได้ตกลงที่จะมีมัน ตั้งแต่ที่แรก แต่กระนั้น ถ้าหากทีมยังคิดว่ามันไม่จำเป็นละ

**ศิษย์:** ข้าไม่เข้าใจ ท่านอาจารย์ การที่เราให้ทุกคนมาพร้อมหน้ากัน เพื่อที่พากเดียวจะได้เข้าใจตรงกันโดยใช้เวลาแค่เพียงนิดเดียว จะดีกว่าเป็นการเสียเวลาได้อย่างไรกัน

อาจารย์: ถึงแม้ว่าการมี stand-up meeting ทุกวันจะมีข้อดีมากนัย แต่นั้นไม่ได้เป็นเพียงวิธีเดียวที่จะให้ได้มาซึ่งผลลัพธ์ที่ต้องการ บางทีมีสมาชิกทุกคนอยู่ด้วยกัน เป็นทีมเล็กๆ ทำงานด้วยกันอย่างใกล้ชิดทั้งวันทั้ง夜 ในทีมด้วยกันเองและกับลูกค้า ทีมแบบนี้ stand-up meeting ก็อาจจะไม่จำเป็น

**ศิษย์:** อาจารย์กำลังจะบอกว่า บางทีมไม่จำเป็นต้องมี stand-up meeting กระบวนการหรือครับ

อาจารย์: ข้าต้องการเพียงจะบอกแต่ว่า อะไรก็ตามที่ทำแล้วไม่เกิดประโยชน์ จงเปลี่ยนแปลงแก้ไขมันซะ หรือไม่ก็เลิกทำมันไปเลย

## จากที่ 3: Iteration ที่อะไรก็ไม่เสร็จ

อาจารย์: ภาครังนี้ มีอยู่ทีมที่มีนึงที่ทำงานจนจบ iteration แล้ว ก็ไม่มีอะไรเป็นขั้นเป็นอนันท์ที่มีมูลค่าต่อลูกค้าเลย และความล้มเหลวนี้ก็เป็นเพราะพวกเขางานทั้งหมด เค้าไม่ได้วางแผน เริ่มทำงานก็อื้า แฉมยังขี้เกียจ ด้วยความที่รู้อยู่เต็มอกว่าการจะไปบอกลูกค้าเข่นนี้มันซ่างยากเหลือเกิน พวกเขายังยกเลิก showcase กับลูกค้า เจ้าคิดว่านี้เป็นสิ่งที่ควรทำใหม่

**ศิษย์:** ถึงแม้ใจนึงข้าคิดว่าพวกเด็กควรจะต้องได้รับอะไรที่สามารถกับการที่ทำงานไม่เสร็จ แต่ถ้าไม่มีอะไรที่จะไปนำเสนอให้ดูจริงๆ จะยกเลิก showcase กัน่าจะพอรับได้ ถึงอย่างไรก็ดี ข้าก็อยากรู้ให้พวกเด็กซื่อสัตย์และยอมรับว่าทำไม่ถึงยกเลิกมัน

**อาจารย์:** อา... เจ้าเริ่มฉลาดขึ้นแล้วสินะ การที่ทีมไม่ได้ทำให้อะไรที่มีมูลค่าให้ลูกค้าเลยใน iteration นี้มันก็เกิดขึ้นบ้าง แต่ส่วนใหญ่เนี่ยมันไม่ได้เกิดจากความตั้งใจ หรือพยายามไม่พอ เราจะแก้ไขนิสัยความเกี่ยวกับคร้านนี้ได้อย่างไร

**ศิษย์:** ท่านกำลังแนะนำว่าเราควรที่จะมี showcase อยู่ เช่นนั้นหรือ และเพียงหน้ากับลูกค้าทั้งๆที่ไม่มีอะไรใช้ให้เด็กดู

**อาจารย์:** ใช่! บางที่การทำให้ลูกค้ายังบังกับเป็นการสั่งสอนที่ดี การที่ต้องพูดกับลูกค้าแล้วไม่มีอะไรไปให้เดือนั้นเป็นประสบการณ์ที่ทำให้เสีย self อยู่มิใช่น้อย หากได้ลองเพียงครั้งนึงแล้ว ทีมก็คงไม่อยากให้มันเกิดขึ้นอีก

**ศิษย์:** ขอบพระคุณมากครับท่านอาจารย์ ข้าจะกลับไปต่อต่องในสิ่งที่ท่านสั่งสอน

อย่าพยายามที่จะเลี้ยงเหตุการณ์อันไม่พึงประสงค์ในไปเจอกับคุณ บางทีมันอาจจะเป็นครูที่ดีที่สุดของคุณ ยอมรับความผิดพลาด แบ่งปันสิ่งที่คุณได้เรียนรู้กับคนอื่น แล้วสู้ต่อไป

### มีอะไรอีก?

ตอนนี้คุณมีแผนการสื่อสารและความเข้าใจขั้นต่ำแล้วว่าการพัฒนาแบบ iterative นั้นทำงานอย่างไรแล้ว คุณก็อยู่ในสถานะที่ดีที่จะมาดูกันว่าสุดยอดแอ็ลใจล์ทีมมัน เด็กค่ายฯยังกันยังไงเมื่อเราต้องเริ่มทำงานกันอย่างดุเดือด

ในขั้นต่อไป คุณจะเรียนรู้เกี่ยวกับสุดยอดความลับแห่ง visual workspace และการนำมันมาใช้เพื่อให้คุณและทีมของคุณเต็มเปี่ยมไปด้วยพลังและความแนวหน้า!!!

# ตอนที่ 11 สร้างพื้นที่ทำงานที่

## มองเห็นได้ (visual workspace)



กระดานสถานะการบินเป็นสิ่งที่ดี แค่นั้นแค่เว็บเดียวคุณก็เห็นหมดแล้วว่าเที่ยวบินไหนกำลังมา อะไรกำลังจะเกิด และเที่ยวไหนดูยกเลิกไปแล้ว ในคราวเดียวแล้วทำไม่ໄง่แบบเดียวกันนี้กลับโครงการบ้างล่ะ?

ด้วยการเรียนรู้ว่าจะทำอย่างไรเพื่อสร้างพื้นที่ทำงานที่มองเห็นได้ คุณและทีมจะไม่มีวันพลาดสิ่งที่จะทำต่อไปหรือคุณจะเพิ่มคุณค่าให้มากที่สุดได้อย่างไร มันไม่เพียงแต่จะทำให้คุณมองเห็นทุกอย่างชัดเจน แต่การเพิ่มความชัดเจนจะช่วยในการตั้งความคาดหวัง (expectations) ในโครงการได้

## 11.1 ໂອ ຊາວ ເຂົ້າແລ້ວ

ມີການເປັນແປງຄວັງໃຫຍ່ທີ່ບວິເທັກ ບປປະມາລຸໂດນຕັດ ຮະຍະເວລາຖຸກວ່າເຂົ້າມາ ແລະ ທຳທຸກການໃຫ້ດີຂຶ້ນ ເວົ້າຂຶ້ນ ແລະ ຖຸກກວ່າເດີມ ດ້ວຍເຫດຸນີ້ ຄຸນຈຶ່ງຖຸກສ່ວ່າໃຫ້ທຳການໃຫ້ໄດ້ນັກກວ່າເດີມດ້ວຍທຽບພາກທີ່ນ້ອຍລັງ ຜູ້ບວິຫາວ່າຄົບທີ່ຈະໃຫ້ຄຸນສົມອບການປົວມາຄເທົ່າເດີມ ແລະ ພັ້ນກຳທຳການໃຫ້ໄດ້ທ່າເດີມ ດ້ວຍຈຳນວນຄົນໃນທີ່ມີຄົງເດືອຍ ທຳເສົ່າງຈ່າຍຕາງສັກເດືອນນີ້ ສໍາລັບໄວ້ແລ້ວແຕ່ (ຕລອດຄ່ວ່າ ແພີ!) ທຸກອ່າງຈະເຂົ້າມາອ່າງໜັກໜ່ວງແລະ ວັດເວົາ ແລະ ພຸ່ຽນໆພວກເຂົາທີ່ຕ້ອງກາຈະນັດປະຊຸມກັບຄຸນ ເພື່ອຢືນຢັນວ່າຄຸນມີແຜນທີ່ຈະ ຈັດກາກັບມັນແລ້ວເຢົກ! ຄຸນຈະຈັດກາຮັງຮັງ? ສິ່ງທີ່ພວກເຂົາຍາກໄດ້ຄື່ອສິ່ງທີ່(ໂຄຕຣ)ໄມ້ມີເຫດຸຜ ຄຸນກີ້ວິ ທີ່ມີກີ້ວິ ມັນເໝືອກັບວ່າ ມີແຕ່ ຜູ້ບວິຫາວ່າເຫັນນັ້ນແລ້ວ ທີ່ໄໝວ່າ! ຄຸນຈະທຳຍ່າງໄວເພື່ອແສດງໃຫ້ພວກເຄົາເຫັນວ່າ ດ້ວຍທຽບພາກທີ່ເໜືອແຄ່ຄົງເດືອຍ ຄຸນເອົາກົດຍາກຈະສົ່ງ ມອບຂະໄວໃຫ້ຄຸກຄ້າແທນທີ່ຈະໄມ້ມີຂະໄວເລີຍເໝືອກັນແລ້ວ(ເວົຍ)? ແຕ່ມັນທຳໄມ້ໄດ້

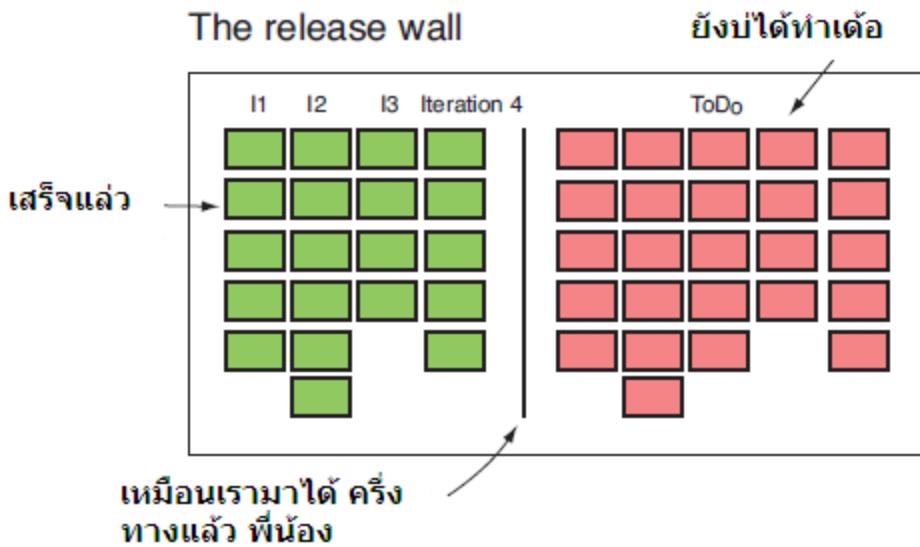
### ຕາມພວກເຮາໃຫ້ທັນຊີ ທ່ານຜູ້ບວິຫາວ່າ

ແທນທີ່ຈະນັດປະຊຸມແລ້ວນັ້ນໆຈຸດ powerpoint ກັນ ຄຸນເຫືຟຸຜູ້ບວິຫາວ່າລົງມາເຫັນຈາກພື້ນຖານຈົງຈຸບັນຂອງໂຄງການກັນຕີກວ່າ ຄຸນເຮີມດ້ວຍກາພາພວກເຄົາໄປປຸດ ຕີ່ໃຫ້ການເຮີມຕົ້ນ (inception desk) ຂອງໂຄງການ ທີ່ຄຸນໄດ້ແປະໄວ້ອ່າງຈ່າຍໆ ບນຸພັນຈັງ ຄຸນອອີຍາຍເກີຍວັນກັບໂຕະເຮີມຕົ້ນຂອງໂຄງການ ວ່າເປັນເຄື່ອງມືອ່ານຸ່ມທີ່ຄຸນແລະທີ່ມີໃຫ້ເພື່ອແນ່ໃຈວ່າ ຄຸນໄມ້ເຄີຍພລາດເປັນຫມາຍໃນໂຄງການ ເພື່ອໃຫ້ເຫັນໄດ້ອ່າງໜັດແຈ້ງ ຂອີຍາວ່າຄຸນຮູ້ຍູ່ເສມວ່າໄຄຣີຄື່ອຄຸກຄ້າ ແລະ ທີ່ສຳຄັນທີ່ສຸດ ທຳໄໝເຈິ່ງຕັດສິນໃຈທີ່ຈະລັງທຸນໃຫ້ກັບໂປຣເຈັກ ນີ້ໃນຄວັງແຮກ

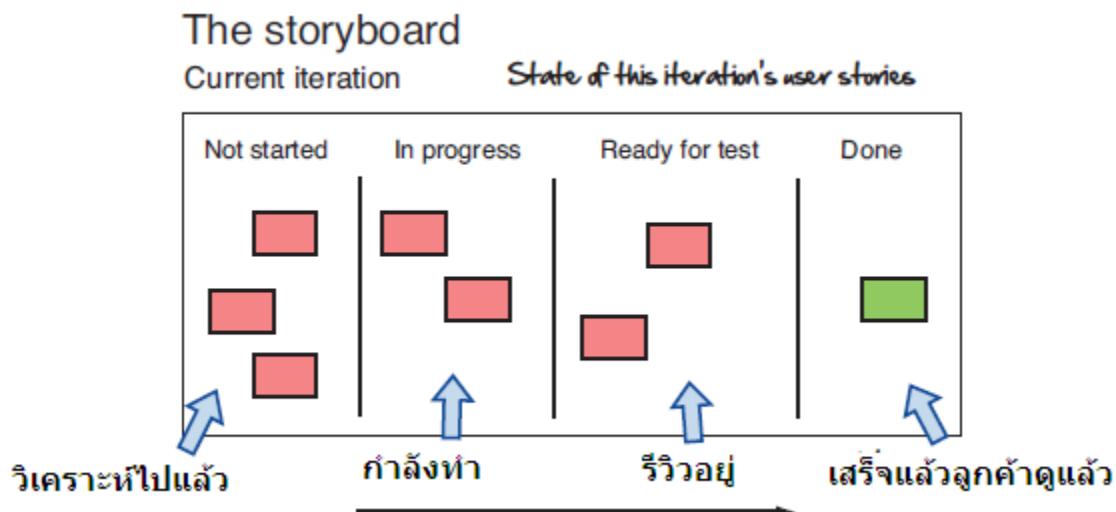


ເຮົາຍູ່ໃນ ເນື່ອຍ	Elevator pitch	Product box
ອ່າຍ່າຫາ	ເຈົ້າເພື່ອນ ນັ້ນ	ແສດງພລ ຈານ
ຕື່ນກລາງດີກ	ມັນໃຫຍ່ຂຶ້ນ	What's going to give
What's it going to take		

ນ່າປະກົດໃຈ ຜູ້ບວິຫາວ່າຈະເອີ້ນມາທາງຄຸນມາກົ້ນແລະ ຄຸນວ່າງານໄປລຶ່ງໃຫຍ່ແລ້ວ ເພື່ອຕອບຄຳດາມນັ້ນ ຄຸນກົດຈະຈຸງຄວາມສົນໃຈ ຂອງພວກເຄົາໄປຢັ້ງ release wall ຂອງຄຸນ



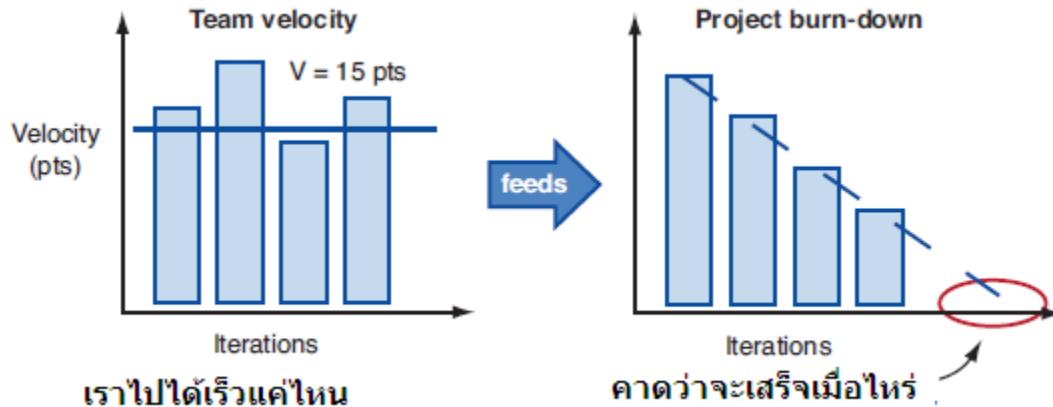
release wall คือที่ที่คุณและทีมติดตามสถานะว่ามีอะไรเสร็จและอะไรที่ยังทำอยู่ ด้านข้ายของ wall แสดงให้เห็นฟีเจอร์ที่ถูกวิเคราะห์พัฒนา ทดสอบและผ่านการยืนยันจากลูกค้าอย่างสมบูรณ์แล้ว (คือพร้อมที่จะส่งมอบแล้ว) และด้านขวาแสดงให้เห็นสตอรี่ที่ยังต้องถูกพัฒนาตรวจสอบเท่าที่ทีมทำงานใน iteration นี้ คุณก็คาดการณ์การจบสตอรี่ของ iteration



สตอรี่บอร์ดติดตามสถานะของฟีเจอร์ของ iteration นี้ (หรือที่เรียกว่า user stories) ฟีเจอร์ที่ยังไม่ได้พัฒนาอยู่ด้านข่าย ขณะที่สิ่งที่พร้อมจะให้กับลูกค้าแล้วจะอยู่ด้านขวา และสตอรี่ที่กำลังจะถูกพัฒนาเพิ่ม ก็จะโดยย้ายจากซ้ายไปขวา มีเฉพาะอันที่พัฒนาทดสอบ และผ่านการตรวจสอบโดยลูกค้าอย่างสมบูรณ์แล้วเท่านั้น จึงจะถูกย้ายไปที่คอลัมน์ Done

สังเกตการมองดูของผู้บริหาร พวกเค้าจะตัดบทแล้วถามคุณว่า คุณคาดหวังว่ามันจะเสร็จเมื่อไหร่

เพื่อตอบคำถามนี้ คุณนำพากเข้าไปยังแผนภาพสองแบบบนผนังที่คุณยังไม่ได้เคยให้พากเข้าเห็น คือความเร็วของทีมและแผนภาพ burn-down ของโครงการ คุณอธิบายว่า ความเร็วของทีม คือสิ่งที่คุณและทีมมีไว้เพื่อวัดระดับความ productivity ของทีม (Hyper-productivity seeker เด็กบอกว่า ไม่ต้องแปลหรอกคำว่า productivity) ทีมจะคาดคะเนได้อย่างถูกต้องเกี่ยวกับเวลาที่งานเสร็จ ด้วยการวัดว่าทีมทำอะไรเสร็จได้บ้างในแต่ละสัปดาห์ และใช้การวัดนั้นมาเป็นสิ่งที่ช่วยวางแผนในอนาคต ซึ่งลิ่งนี้แสดงอยู่บนแผนภาพ burn-down นั้นเองแผนภาพ



burn-down ของโครงการ นำความเร็วของทีม และประมาณการความเร็วที่ทีมจะ “เผา” งานที่ลูกค้าต้องการมาดำเนิน โครงการจะสำเร็จเมื่อทีมได้ส่งมอบทุกอย่างในรายการ หรือโครงการหมดเงินแล้ว (อะไรมีตามที่มาถึงก่อน)

ด้วยสถานะที่กล่าวมา ตอนนี้คุณก็มาถึงจุดที่จะแสดงให้อย่างรอบคอบ ให้ทุกคนในห้องเห็นอย่างชัดเจน ครั้งหนึ่งของทีมก็อาจจะตัด productivity ของทีมได้ครั้งหนึ่ง เช่นกันด้วยความน่าประทับใจกับการสั่งการของคุณในสถานการณ์นี้ ผู้บริหารขอบคุณที่คุณให้เวลาและพากเข้าไปประชุมโครงการอีกต่อไป อีกสองสามสัปดาห์ดีไป คุณจะได้อีเมลล์ที่อธิบายว่า เพราะว่าบริษัทมีคำสั่งเกี่ยวกับกลยุทธ์ใหม่ ทำให้โครงการของคุณโดนยกเลิกไป! (ชีวิตบางทีก็เป็นแบบนี้แหละ)

อย่างไรก็ตาม มันก็คือข่าวดีที่พากเข้าได้รู้ว่า คุณจัดการโครงการได้ดีอย่างไร พากเข้าต้องการให้คุณยังคงเป็นผู้นำในการเริ่มต้นใหม่! นี่เป็นเพียงแค่นึงด้วยอย่างที่วางไว้เพื่อแสดงให้เห็นว่า พื้นที่ทำงานที่มองเห็นได้มันช่วยให้คุณสามารถตั้งสิ่งที่คาดหวังในอนาคตกับลูกค้าและสร้างความเป็นจริงให้กับสถานการณ์ได้อย่างชัดแจ้ง แต่ที่มันจะส่องแสงสว่างคือตอนที่มันช่วยคุณและทีมในการดำเนินงานและเพ่งความสนใจในงานเราไปยังความคิดที่จะสร้างพื้นที่ทำงานสมมูลของคุณเองดีกว่า

## 11.2 สร้างพื้นที่ทำงานสมมูลได้อย่างไร?

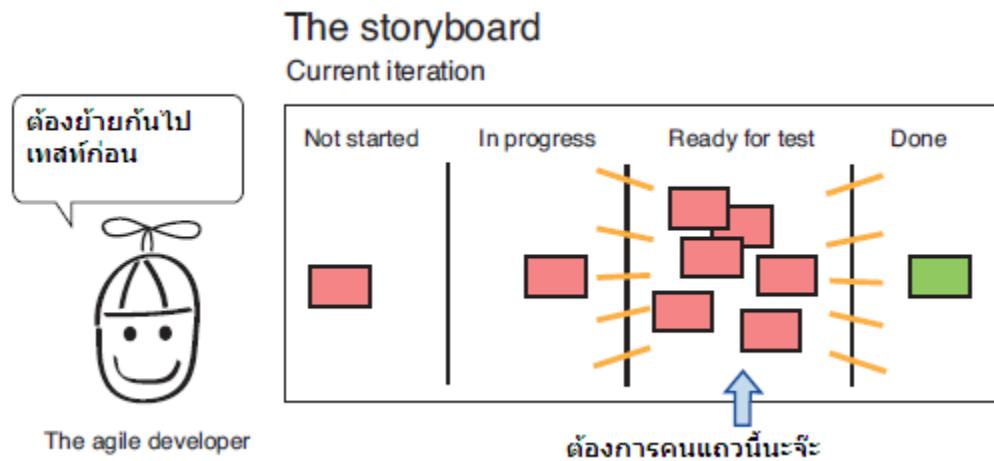
การสร้างพื้นที่ทำงานสมมูลที่ดีมันทำได้ตรงไปตรงมา สำหรับทีมแอจเจลใหม่ ผู้เขียนมักจะแนะนำให้เริ่มด้วย:

- A story wall (ผนังสตอรี่)
- A release wall (ผนังรีลีฟ)
- A velocity and burn-down graph
- An inception desk (โต๊ะทำงานเริ่มต้น) ถ้ามีเนื้อที่พอ

ใต้ระหบันเริ่มต้นคือสิ่งที่ดี เพราะมันเตือนให้ทีมรู้ว่าทำไม่พากเพาอยู่ตรงนั้น และสิ่งทั้งหมดตรงหน้านั้นเกี่ยวกับอะไร (ซึ่งมันง่ายมากที่เราจะหลุดได้เมื่อสมองเรามันจมอยู่กับโครงการ) ผนังสตอรี่เป็นสิ่งที่ดี เพราะทุกคนสามารถเดินเข้ามาแล้วรู้ได้เลยว่าอะไรที่ต้องทำให้เสร็จต่อไป



ผนังสตอรี่จะแสดงให้คุณเห็นคุณภาพที่คุณมีในระบบ และส่วนใดที่คุณอยากรับการทบทวน



ผนังรีลีฟคือสิ่งที่สวยงาม เพราะใครๆ ก็สามารถเดินมาที่ห้องของคุณและเห็นสถานะของโครงการ โดยมองแค่แก็บเดียว ว่ามีคือสิ่งที่ต้องทำ นี่คือสิ่งที่เหลือ ไม่จำเป็นต้องมีคณิตศาสตร์ยากๆ หรือไฟล์เอกสารมาช่วยอธิบายและเหมือนที่เราเคยคุยกันว่าเกี่ยวกับกระบวนการแผนakkumajjel แบบกว้างๆ ไม่มีอะไรที่จะตั้งสิ่งที่เราคาดหวังได้เท่ากับแผนภาพ burn-down อย่าลืมเอาเด็กน้อยคนนี้ไว้บนผนังด้วย แล้วคุณจะรู้ความจริงอยู่ตลอดเวลา ยังเหลือเวลาอีกเท่าไหร่ และตอนนี้เป็นอย่างไรและแน่นอนนี้เป็นแค่การเริ่มต้น

ถ้าคุณมีภารต่างๆ พวกรูปแบบจำลอง หรือได้อะแกรมที่ช่วยในการดำเนินงานให้กับคุณและทีม ก็สามารถเอาไปประวัติและทำให้ทุกคนมองเห็นมันได้ง่าย

### 11.3 แสดงเจตนาของคุณ

ข้อตกลงการทำงาน คือสิ่งที่เกี่ยวกับการปักเสานลักษณะพื้นที่ทีม และมุ่งว่า “นี่คือแนวทางที่ทีมจะทำงาน” มันเป็นแนวทางการตั้งสิ่งคาดหวังให้กับทุกคนในทีม เกี่ยวกับว่าทีมจะทำงานกันอย่างไร และควรจะคาดหวังอะไรจากคนในทีมถ้าหากเขาเข้าร่วมในงานนี้ เรื่อง shared values (ค่านิยมร่วม) ก็เหมือนกัน มันเป็นแค่เรื่องของความรู้สึก ถ้าทีมเคยเจ็บปวดมาก่อนหน้านี้เพราะว่า พวกรูปแบบลักษณะนี้ให้ประนีประนอมเรื่องคุณภาพงาน และไม่อยากถูกหักจิกในฐานะของทีมที่ใช้ทางลัดและเขียนซอฟท์แวร์ที่ใช้ประโยชน์ พวกรูปแบบสามารถเขียนค่านิยมร่วมและทำให้คนอื่นรับรู้ได้

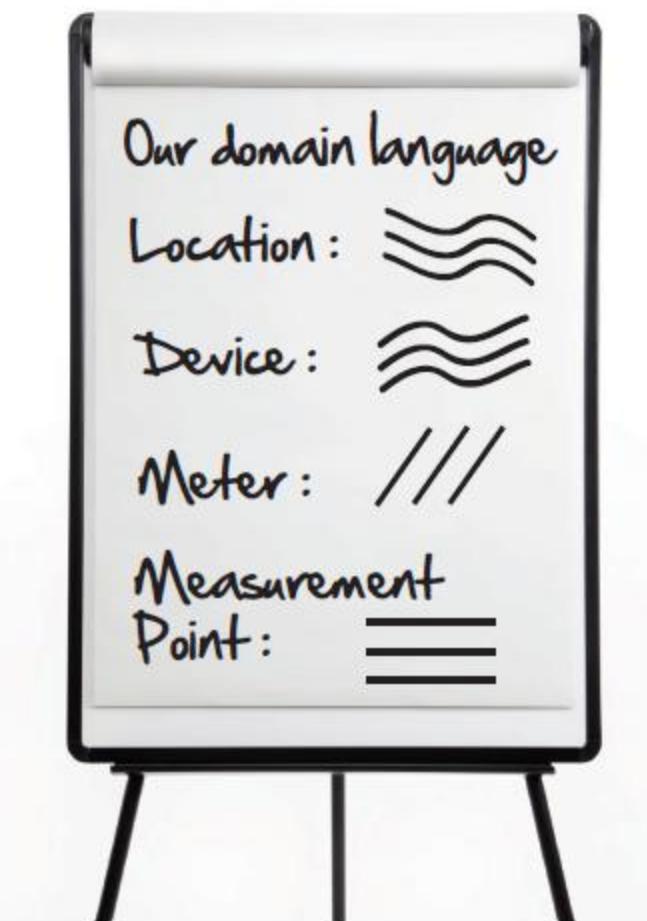
<b>ค่านิยมร่วม</b> <ul style="list-style-type: none"> <li>- พวกรูปแบบใช้พวกรูปแบบลักษณะ</li> <li>- ต้องไม่ปล่อยให้เกิดการแตกแยก</li> <li>- มันไม่เคยทำให้ไม่เห็นด้วย</li> <li>- พวกรูปแบบรักษาความจริงได้</li> <li>- อย่าเดา ให้คำนวณ</li> <li>- ถ้าเจอบัญหา ให้เขียนเทส</li> <li>- ชอบที่ดี</li> <li>- ตรวจสอบ อยู่ ขอตัวเองเสมอ</li> </ul>	<b>ข้อตกลงการทำงาน</b> <ul style="list-style-type: none"> <li>- เวลาฯ แห่งสักคือ 9 am - 4 pm</li> <li>- ประชุม stand-ups รายวัน ทุก 10 am</li> <li>- งานที่เหลือจดลงในกระดาษทราย</li> <li>- กิมมีครัต้องการความช่วยเหลือจากคุณให้ตอบว่า “ได้”</li> <li>- เดบันงาทุกวัน บังคับ 11 am</li> <li>- ฉุกเฉียวย่างต่อน 1-3 pm</li> </ul>
--	---

อีกนึง ที่คุณต้องแนใจคือ สิ่งที่คุณแบ่งปันออกไป เป็นภาษา(คน)

### 11.4 สร้างและแบ่งปันภาษาส่วนกลาง

เมื่อคำศัพท์ที่คุณใช้ในซอฟท์แวร์มันมีหลายคำ แต่ละคนเรียกไม่เหมือนกัน คุณอาจจะเกิดปัญหาได้

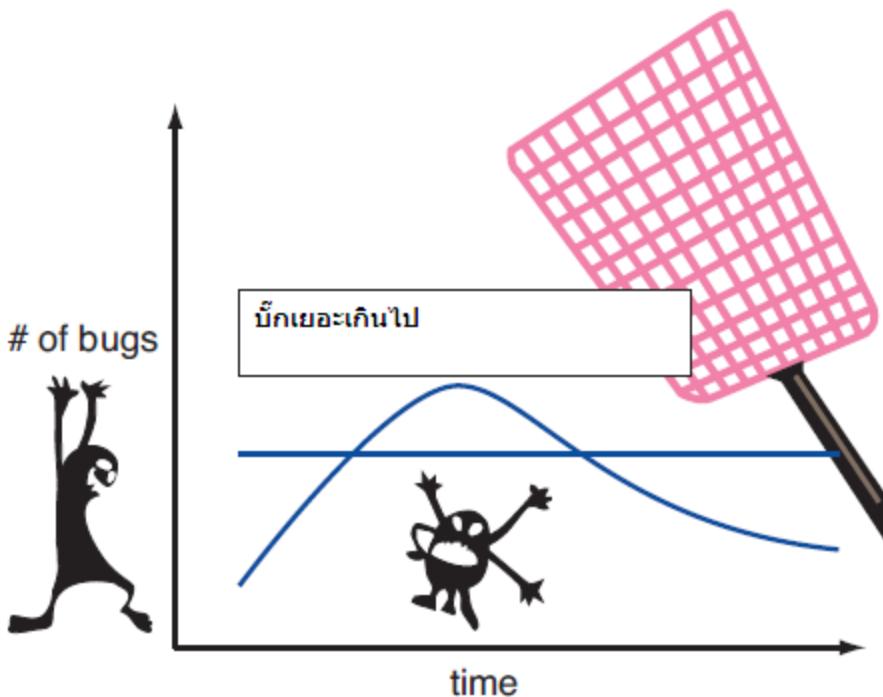
สร้างนามธรรมที่มีความหมายเดียวกันในซอฟท์แวร์ (ทางธุรกิจอาจจะมี locations มีความหมายอย่างหนึ่ง ในขณะที่ผู้พัฒนาอาจจะเปลี่ยนเป็นความหมายอย่างอื่น) ซอฟท์แวร์ภาษาไทยเป็นสิ่งที่ยากจะเปลี่ยนแปลง ( เพราะว่าคำที่ใช้บนหน้าจอมันไม่เหมือนกับคำที่ใช้ในฐานข้อมูล ) คุณجبด้วยการมีบัญชีและมีคำใช้ซ้ำๆ ในกรอบภาษาซอฟท์แวร์สูง ( เพราะว่าทีมมีงานเพิ่มในการที่ปรับเปลี่ยนซอฟท์แวร์ ) เพื่อหลีกเลี่ยงความผิดปกตินี้ จึงควรสร้างภาษาส่วนกลางที่คุณและผู้ช่วยกันใช้ร่วมกัน และให้มันอย่างเคร่งครัดใน user stories, โนเดล, รูป และ โค๊ด



ตัวอย่างเช่น ถ้ามีบางคีย์เวิร์ดที่คุณและลูกค้าใช้เมื่อคุณพูดถึงระบบ ให้เขียนมันลงมา แล้วนำมารวบรวมที่ชัดเจน ก็ยกับคำศัพท์นั้น และแน่ใจว่ามันเป็นความหมายเดียวกันกับที่อยู่ในซอฟต์แวร์ (นั่นคือ หน้าจอ และคอลัมน์ในฐานข้อมูล) การทำแบบนี้ ไม่เพียงแต่เป็นการลดบั๊กและการทำงานขับข้อที่จะเกิด แต่เป็นการสร้างการทำงานที่ง่ายขึ้นในการคุยกับลูกค้า เพราะโค้ดของคุณจะเป็นภาษาเดียวกันกับภาษาที่ใช้คุยกันในธุรกิจ  
เราไม่มีเวลาหรือเนื้อที่ที่จะคุยเรื่องนี้ต่อ แต่ก็มีหนังสือที่ดีมากๆ เกี่ยวกับเรื่องนี้ โดย Eric Evans คือ Domain-Driven Design: Tackling Complexity in the Heart of Software. [Eva03]. มันดีที่เดียวสำหรับที่จะอ่านเรื่องนี้  
สุดท้าย มาดูบัญชีของคุณกัน

## 11.5 ฝ่าดูบัญชีของคุณ

เพื่อแน่ใจว่าคุณและทีมจะไม่รู้สึกผายแพ้นมดหวัง หากว่าเจอบัญชีที่ไม่ได้คาดคิดก่อนที่จะขึ้นโปรดตั้งขึ้น ก็ให้กรุณาติดตามและเก็บข้อมูลของมันไว้ในแต่ละวันที่เจอกันในโครงการ



ถ้ามันช่วย ก็ขอให้สละเวลาสัก 10 เปอร์เซนในแต่ละ iteration เพื่อใช้ในการทดสอบบีกแลจ่ายหนี้ทางเทคนิคกันบ้าง แค่กำจัดมันในเวลาที่เกิด และอย่าให้มันกลับมาได้อีก



นักเรียน: นานะ ถ้า workspace ของผมมันไม่ยอมให้ผมสร้าง visual workspace ผมจะทำยังดียะ

ท่านอาจารย์: มันเป็นความจริงที่บางสำนักงานต่อต้านการให้ทีมเข้าของไปແປບแผนฯ ถ้าเจอกារต่อต้านแบบนี้ ก็ให้ยอมรับมัน และตัดสินว่าจะทำอย่างไรต่อไป

นักเรียน: ใช่แล้วนะท่านอาจารย์ แล้วผมควรจะต่อสู้เพื่อ visual workspace ปี๊ะ? หรือว่าแค่ยอมรับว่าผมจะไม่ได้ใช้มันแน่ๆ แล้ว (โขๆๆ)

ท่านอาจารย์: มันก็แล้วแต่เจ้า ถ้าเจ้าสามารถประนีประนอม ยอมทำงาน หรือเผชิญหน้า มันมีเวลาและสถานที่สำหรับทำสิ่งนี้ ค้นหาหัวใจตัวเอง หาพวก และตัดสินใจว่าการต่อสู้นี้มันจะคุ้มค่าความพยายามหรือไม่

นักเรียน: ถ้านี่เป็นทักษะที่สำคัญจริงๆ ควรทำยังไงให้เกิดการประนีประนอมนะ?

**ท่านอาจารย์:** ถ้าเผชิญหน้ากับสถานการณ์แบบนี้ นักศึกษาทั้งหลายมักจะพบว่า สดอร์บอร์ดแบบพับเก็บได้มันมีประโยชน์และทำให้พื้นที่ทำงานสะอาดอยู่เสมอ ในขณะที่ก็ทำให้ทีมได้สื่อสารกันอย่างเปิดเผยในระหว่างวัน บางครั้งก็เข้าพากเครื่องมือออนไลน์ หรือว่าสดอร์บอร์ดเสมือนในการแบ่งปันข้อมูลที่สำคัญ หรือการทำให้มีรู้เท่าๆ กัน

**นักเรียน:** ดังนั้น visual workspace ก็ไม่จำเป็นต้องจับต้องได้บนเดสก์?

**ท่านอาจารย์:** ใช่แล้ว การจับต้องได้มันก็ต้องสุด oranage แต่บางทีมันก็เป็นไปไม่ได้ ;P

**นักเรียน:** แล้วจะไร้ระยะที่ permova พื้นที่ permova หมายความยังไง?

**ท่านอาจารย์:** เจ้าสามารถเริ่มต้นจากการสร้าง visual workspace ง่ายๆ ใช้มันในแต่ละวันในโครงการ และหวังว่าสิ่งที่พูดแล้ว เรียนรู้มันจะมีประโยชน์

**นักเรียน:** แล้วถ้าไม่ล่าจะ?

**ท่านอาจารย์:** ดังนั้นต้นตอของปัญหาคือเรื่องของความรู้สึก มันอาจจะมีการต่อต้านมาจากการส่วนกลางในสิ่งที่เจ้าพยายามจะทำให้พยายามที่จะตอบกลับและเข้าใจแรงผลักดันนั้น บางทีในการพูดจากัน เจ้าอาจจะสามารถค้นพบวิธีการแก้ปัญหาที่มันรับได้กันทั้งสองฝ่าย เวลาและความอดทนเท่านั้นที่จะทำให้มันเกิด

ต่อไปมีอะไร?

การเดินทางของคุณใกล้จะสมบูรณ์แล้ว คุณได้นำทุกคนขึ้นรถบัสแล้ว (บทที่ 3 หวานคนขึ้นรถแอร์เจล์) และคุณก็มีแผนงานแล้ว (บทที่ 8 ตามมาหากาพ) และคุณก็รู้แล้วว่าจะไว้ใจต้องดำเนินการ

ส่วนต่อไปของหนังสือเล่มนี้ “วิศวกรรมแอร์เจล์ซอฟท์แวร์” เน้นไปยังส่วนหลักของทักษะด้านวิศวกรรมแอร์เจล์ซอฟท์แวร์ที่คุณและทีมจะต้องการ เพื่อให้แน่ใจว่าสิ่งที่เป็นแอร์เจล์ทั้งหลายจะเกิดขึ้นจริง

มันเป็นสิ่งที่ต้องอ่านถ้าคุณต้องการจะทำให้โค้ดสั้นลง แต่มันจะถูกแนะนำให้อ่านถ้าคุณเคยวางแผนในการควบคุมโครงการแบบแอร์เจล์ไม่มีอะไรในแอร์เจล์ที่จะทำงานได้หากมันไม่มีทักษะด้านเทคนิคที่แข็งแกร่ง และแม้ว่าสิ่งที่ได้ไปจะสามารถเป็นหนังสืออีกเล่มได้ด้วยตัวมันเอง แต่บทต่างๆ ในเล่มนี้ก็จะให้รวมชาติที่เพียงพอกับคุณ เพื่อจะเข้าใจทักษะการทำงาน และเข้าใจว่ามันสำคัญอย่างไร ที่จะแอร์เจล์

เราจะเริ่มด้วยการคุยวิธีหนึ่งในการประหยัดเวลาที่ดีที่สุดในการทำโครงการซอฟท์แวร์ นั่นคือการทำ automated unit testing (Hyper-productivity seeker บอกมาอีกแล้วว่า ไม่ต้องแปลหรอก)

(อ่าคริ อ่าคริ เสร็จແກ້ວ...)

# ตอนที่ 12 ยูนิตเทส: รู้สเมว่า

## works



จากเวลาทั้งหมดที่เราใช้ไปกับการวางแผน ตามแนวทางแบบแอกเจล์จะไม่ประสบความสำเร็จเลยถ้าสิ่งเหล่านี้ไม่มีการสนับสนุนจาก หลักปฏิบัติในการพัฒนาซอฟต์แวร์ที่ดีเยี่ยม ถึงแม้ว่าในบางแบบการปฏิบัติแบบแอกเจล์เหล่านี้บางอย่าง เช่น การทำ แพร์เพรแกรมมิ่ง (Pair Programming) จะยังเป็นสิ่งที่ต้องเรียนรู้ แต่ก็มีอีกหลายแบบการปฏิบัติแบบแอกเจล์ที่ได้เป็นที่ยอมรับกันอย่างแพร่หลาย เช่น การทำงานนิติเทศแบบอัตโนมัติ ในสิ่บหกสุดท้ายของหนังสือเล่มนี้ เราจะได้เรียนรู้เกี่ยวกับสิ่งที่ผมมักจะเรียกว่า เป็นหลักพื้นฐานของการพัฒนาซอฟต์แวร์แบบแอกเจล์ ซึ่งได้แก่

5. ยูนิตเทส
6. รีแฟกเตอร์ (Refactor)
7. ทีดีดี (TDD)
8. Continuous Integration

ในเนื้อหาของแต่ละบทนั้นเรียกได้ว่าเป็นหนังสืออีกเล่มหนึ่งที่มีด้วยตัวมันเองแล้วก็ว่าได้ แต่ด้วยการอธิบายเบื้องต้นในที่นี่ อย่างน้อยคุณก็จะได้เข้าใจอย่างดี วามันคืออะไร และทำงานอย่างไร ซึ่งเพียงพอที่จะให้ตัวคุณ หรือทีมสามารถที่จะนำสิ่งนั้นไปเริ่มต้นใช้งานได้

ตัวอย่างต่างๆ เหล่านี้จะเขียนด้วยภาษา Microsoft .NET C# ทั้งหมด ซึ่งจริงๆ แล้วโดยหลักการที่ไปแล้ว มันก็สามารถที่จะนำไปประยุกต์กับภาษาอื่นๆ ได้ และคุณก็ไม่ต้องกังวลไปนะครับ ถ้าคุณไม่ใช้โปรแกรมเมอร์ สิ่งที่อยู่ในโค้ดตัวอย่างนั้นจะมีจุดที่น่าสนใจ และสำคัญที่ผมจะเน้นให้เห็นไปด้วยตลอด

เราสามารถกันเลยกับเรื่องของการทำบัญนิตเทสอย่างเต็มรูปแบบ ซึ่งเป็นสิ่งพื้นฐานที่สนับสนุน หลักการอื่นๆ ของการพัฒนาซอฟต์แวร์แบบแอจайл์

## 12.1 ขอต้อนรับสู่เว็บครับทุกท่าน

คุณคือผู้โชคดีสุดๆ ที่เพิ่งจะเข้ามาร่วมทีมพัฒนาซอฟต์แวร์ที่กำลังจะเริ่มพัฒนาเกมจำลองการเล่นไพ่ Black Jack! งานแรกของคุณคือการออกแบบเบ้าเพื่อ 1 สำรับโดยด้านล่างนี้คือได้บางส่วนที่เขียนด้วยภาษา C# เพื่อแสดงถึงเพื่อ 1 สำรับ

```
public class Deck
{
    private readonly IList<Card> cards = new List<Card>();

    public Deck()
    {
        cards.Add(Card.TWO_OF_CLUBS);
        cards.Add(Card.THREE_OF_CLUBS);
        // .. remaining clubs

        cards.Add(Card.TWO_OF_DIAMONDS);
        cards.Add(Card.THREE_OF_DIAMONDS);
        // .. remaining diamonds

        cards.Add(Card.TWO_OF_SPADES);
        cards.Add(Card.THREE_OF_SPADES);
        // .. remaining spades

        cards.Add(Card.TWO_OF_HEARTS);
        cards.Add(Card.THREE_OF_HEARTS);
        // .. remaining diamonds

        // joker
        cards.Add(Card.JOKER);
    }
}
```

ก่อนที่คุณจะแก้ไขให้เขียนบัญนิตเทสที่ไม่ง่านก่อนเสมอ

ถ้าคุณเจอกับข้อผิดพลาดในซอฟต์แวร์ของคุณ คุณก็อาจจะรู้สึกอย่างที่จะกระใจเข้าไปที่ส่วนที่คิดว่าผิดและแก้ไขไปที่ได้ในทันที อย่างไรก็ตามคุณอย่าเพิ่งทำอย่างนั้น สิ่งที่ควรทำคือก่อนอื่นเลยคือ การทำให้บันทึกมาอยู่ในรูปแบบของบัญนิตเทสเสียก่อน และค่อยเริ่มพยายามแก้ไขได้เพื่อแก้ไขนั้น ซึ่งประযุชน์ของการทำงานแบบนี้คือ

- เป็นสิ่งที่เขียนโดยคนที่ได้รับมอบหมายมาแล้ว (เพราะคุณได้จำลองการทำเกิดของบันทึกนี้ได้แล้ว)
- เป็นสิ่งที่ทำให้คุณมั่นใจได้เลยว่าคุณได้ทำการแก้ไขบันทึกนี้ได้เรียบร้อยแล้ว (เพราะบัญนิตเทสผ่านแล้ว)
- เป็นสิ่งที่เข้าบังกันได้ว่าบันทึกนั้น จะไม่กลับมาหลอกคุณอีกในอนาคต

ได้ด้วยความต้องการตรวจสอบอย่างเคร่งครัด ทุกอย่างต้องไปได้ด้วยดี แต่แล้วในวินาทีที่ได้ด้วยความต้องการทำให้รับภาระไม่ไปลงที่โปรดักชัน ทีมทดสอบจะเขียนบันทึกตัวเองซึ่งระบุว่ามีไฟจิกเกอร์ในสำรับ คุณรับแก้ไขโดยการนำไฟจิกเกอร์ออกจากคลาสไฟทันที และให้ทีมทดสอบตรวจสอบอีกครั้ง จากนั้นก็จะนำซอฟต์แวร์ไปลงที่โปรดักชัน หลังจากนั้นสองสัปดาห์คุณได้รับข่าวว่าทางอีเมล์ จากผู้จัดการทีม

เทสนอกจากบันก์ที่รุนแรงมากในระบบเมื่อคืนนี้ ซึ่งทำให้ทางบริษัทด้องคืนเงินให้กับลูกค้าเป็นจำนวนเงินกว่าหลายแสนดอลลาร์ เพียงเพราะว่ามีบางคนเอาไฟโจรเกอร์เข้าไปในคลาสไฟ

“ว่าไงนะ” คุณพูด “ไม่มีทาง ผู้พิ่งแก้บันก์ไปเมื่อประมาณสองอาทิตย์ก่อนนี้เอง” หลังจากตรวจสอบโดยละเอียดอีกครั้ง คุณก็พบว่ามีน้องฝึกงานภาคฤดูร้อนคนนึงในทีมที่คุณคุ้นเคยอยู่ได้ทำการที่คุณบอกให้เขอข่าวตรวจสอบว่าในคลาสไฟนั้นมีไฟครับ เรียบร้อยตามที่ไฟที่เป็นไฟสำรองจริงๆ มีทุกใบ และเอกสารได้ทำการที่คุณสั่งทุกคำ และเหมือนว่าเขาจะทำการใส่ไฟโจรเกอร์กลับเข้าไปในคลาสไฟอย่างไม่ได้ตั้งใจ โดยเรื่องดีว่านั้นเป็นบันก์ของคลาสด้วยความรู้สึกผิดและอภัยน้องฝึกงานกูขอโทษคุณและทุกคนในทีม หลังจากนั้นเรือเข้ามาถามคุณว่า เขายังทำอย่างไรได้บ้างเพื่อเป็นการป้องกันไม่ให้สิ่งที่คล้ายๆ กับเรื่องนี้เกิดขึ้นอีก คุณจะตอบเขาว่าอะไร อะไรคือวิธีที่คุณหรือน้องฝึกงานจะทำได้เพื่อให้มั่นใจได้ว่าไฟโจรเกอร์จะไม่มีวันกลับเข้ามายังในลำรับได้อีก เลยตลอดไป ด้วยเหตุนี้ เรายกมาเริ่มนูนิตทดสอบเก็บไว้

## 12.2 เริ่มต้นทำยูนิตเทส

ยูนิตเทสเป็นเครื่องที่เล็ก และใช้ทดสอบอยู่ในระดับ Method ในทุกครั้งที่ Developers จะทำการแก้ไขปรับปรุงโค้ด จะเขียนยูนิตเทสขึ้นมาเพื่อทดสอบให้มั่นใจว่าสิ่งที่แก้ไขไปนั้นได้ให้ผลที่ถูกต้องตามที่ตั้งใจไว้ ยกตัวอย่างเช่นถ้าเราต้องการที่จะทดสอบว่าไฟ ในลำรับของเรานั้นมีไฟครับ 52 ใบ (ไม่ใช่ 53 ใบ) เราสามารถเขียนยูนิตเทสได้ดังนี้:

```
[TestFixture]
public class DeckTest
{
    [Test]
    public void Verify_deck_contains_52_cards()
    {
        var deck = new Deck();
        Assert.AreEqual(52, deck.Count());
    }
}
```

ขอเน้นตรงนี้นะครับว่าโค้ดที่เป็นยูนิตเทสข้างบนนี้ เป็นคนละส่วนกับโค้ดที่เราจะไปเขียน Production นะครับ ยูนิตเทสนี้ใช้เพื่อ ตรวจสอบว่าโค้ดที่จะเข้าทำงานจริงนั้น ทำงานได้ถูกต้อง เมื่อไหร่ก็ตามที่เราเกิดความสงสัยว่า โค้ดส่วนนั้น ส่วนนี้จะทำงานอย่างไร และที่สำคัญเป็นไปตามที่เราอยากให้มันเป็นหรือเปล่า ก็ให้เขียนยูนิตเทสเลยครับ (อย่างที่เราทำเป็นยูนิตเทสข้างบน ก่อนหน้านี้ เพื่อตรวจสอบว่าไฟในลำรับมีไฟอยู่ 52 ใบจริง)

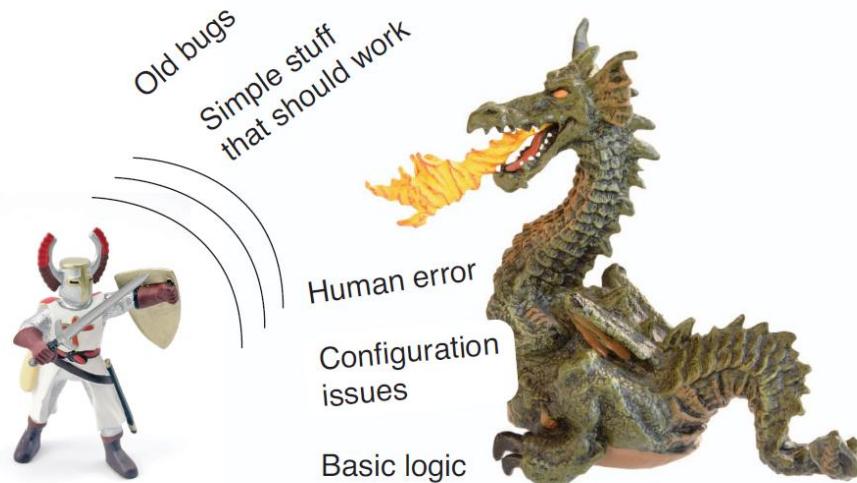
ถ้าเราทำให้ยูนิตเทสของเรานั้นสามารถวันแบบ automate และวันได้ง่ายๆ ได้ด้วยแล้วนั้นจะเป็นประโยชน์มาก เพราะเราจะสามารถติดตั้งทันทีเลยว่า ส่วนที่เราเพิ่งแก้ไขไปในโค้ดนั้น จะไปทำให้โค้ดส่วนอื่นทำงานผิดหรือไม่ (more on this in Chapter 15, Continuous Integration: Making It Production-Ready) ในแอคิลิปวิจฉัดทั่วไป จะมียูนิตเทสกันอยู่ในระดับจำนวนร้อย ถึงพันก็เป็นได้ ซึ่งก็จะรวมไปถึงการตรวจสอบโค้ดทั้งหมดไม่ว่า จะเป็นส่วนที่เป็นการคำนวน Business Logic ของระบบ ไปจนถึงการจัดเก็บข้อมูลของลูกค้าลงดาต้าเบส คุณประการต่างๆ ที่เราจะได้จากการเขียนยูนิตเทสอย่างมากมายกับโค้ดของคุณ ได้แก่:

- ยูนิตเทสสามารถอ่านได้ในบัดเดียวนี้
  - เมื่อไหร่ก็ตามที่คุณได้ทำการเปลี่ยนแปลงโค้ด และทำให้หนึ่งในยูนิตเทสของเรามาไม่ผ่าน คุณก็เก็บจะเรียกได้ว่ารู้โดยทันทีว่าสิ่งที่คุณแก้ไขไปนั้นต้องได้รับการแก้ไข ไม่ใช่ว่าจะรู้ก็ชั่ว Production ไปแล้วสามอาทิตย์

○ ทดสอบทุกจุด ที่คิดว่าเป็นจุดเสี่ยงที่จะเกิดปัญหาได้

- Extreme Programming (XP) มีคติพจน์ประจำใจข้อหนึ่งกล่าวว่า “จะทดสอบทุกจุด ที่คิดว่าเป็นจุดเสี่ยงที่จะเกิดปัญหาได้” ข้อความดังกล่าวจะช่วยเตือนนักพัฒนาว่า ถ้ามีจุดใดก็ตามที่คิดว่าจะเป็นจุดที่สามารถสร้างความเสี่ยงอย่างระบบได้ จุดต่างๆ เหล่านั้นก็สมควรเป็นอย่างยิ่งที่จะมียูนิตเทสที่สามารถรันแบบอutoได้ตามปกติ
- เราไม่สามารถที่จะทดสอบได้ทุกๆ จุดของสิ่งที่เรากำลังพัฒนาอยู่ก็จริง แต่สิ่งเดือนใจดังกล่าวได้แสดงถึงแก่นของการที่แอจайлพัฒนามาให้มีคิดถึงการตรวจสอบในอีกมุมนึง ทดสอบให้เพียงพอ และพอเพียงในระดับที่ทำให้เรามั่นใจได้ว่า software มันทำงานได้ถูกต้อง ซึ่งต้องใช้วิจารณญาณของคุณเองเพื่อติใจทยว่าเราทำยูนิตเทสแค่ไหนถึงจะคุ้มค่า คุ้มกับเวลาที่ใช้ไปที่สุด
- ในบทที่ 14 ในหัวข้อเรื่อง Test-Driven Development เราจะได้เห็นว่า มันจะช่วยให้เราประมาณการได้ว่าจุดคุ้มทุนที่สุดในการที่เราจะมีจำนวนยูนิตเทสเท่าใด แค่ไหนนั้นได้อย่างไร และรวมไปถึงการหาจุดสมดุลย์ระหว่างการทดสอบทุกสิ่งทุกอย่าง กับเทสแบบพอดี
- ยูนิตเทสช่วยลดค่าใช้จ่ายในการทำ regression test ได้อย่างมากมาก
  - แทนที่จะต้องเสียเวลาไปกับการทำการทดสอบทุกอย่างด้วยมือเวลาเมื่อวานร้อนขึ้นใหม่ การทดสอบแบบอโต้จะช่วยคุณประหยัดเวลาต่อวันนี้ไปได้อย่างมาก ซึ่งจะทำให้คุณมีเวลาที่จะทดสอบที่ซับซ้อนมากกว่า
- ยูนิตเทสช่วยลดเวลาที่ใช้ในการ Debug
  - หากคุณพบว่ามียูนิตเทสที่ไม่ผ่าน คุณจะรู้ได้เลยว่าส่วนใดของโค้ดคุณที่มีปัญหา ไม่ต้องเปิด Debug เพื่อที่จะมองหาโค้ดเป็น พันๆ บรรทัด เพื่อหาจุดที่น่าจะทำให้เกิดปัญหา
- ยูนิตเทสช่วยให้คุณนำ้งานของคุณได้อย่างมั่นใจ
  - มันก็เป็นความรู้สึกที่ดี ที่ทุกครั้งที่คุณนำ้งานของฟีเจอร์ คุณรู้ว่ามีมันได้ผ่านยูนิตเทส แบบอัตโนมัติมาแล้วตั้งแต่ ก่อนนำ้งาน ถึงแม้ยูนิตเทสจะไม่ได้ช่วยได้ทุกอย่าง แต่อย่างน้อยมันก็ทำให้คุณมีเวลาไปทำการทดสอบที่ต้องต่อ กับระบบอื่น หรือจุดที่มีความซับซ้อน

ลองคิดถึงยูนิตเทสว่ามันเปรียบเสมือนกับเสื้อเกราะของอัศวินที่กำลังจะออกไปสู้รบสิครับ พอกำหนดเทสไป มันก็เหมือนกับเป็นสเปคที่สามารถที่จะรับได้ด้วย และจะอยู่เคียงข้างโค้ดของคุณชั่วขณะนั้นๆ และจะคอยช่วยป้องกันเราไม่ว่าจะเป็น จากรหัส แม้กระทั่งไฟ ตัวจริง หรือในเงื่อนไขทางการ และที่สำคัญตัวเราเอง



ข้อควรระวัง: คุณอาจจะทำญี่นิตเทสไปจนถึงช่วงที่คุณรู้สึกว่าการเขียนขอติเมตเทสนั้นยากเย็นเหลือเกิน ตัวอย่างเช่น เขียนเทสที่พื่อตรวจสอบว่าไฟของเรานั่นสามารถสักดได้ นั้นยาก ( เพราะผลลัพธ์ของการสักแต่ละครั้งนั้นก็จะไม่เหมือนกัน ) และการเขียนเทสในเรื่องที่มีการทำงานเกี่ยวกับ concurrency และ มัลติเพลท นั้นก็ท้าทายมากเช่นเดียวกัน

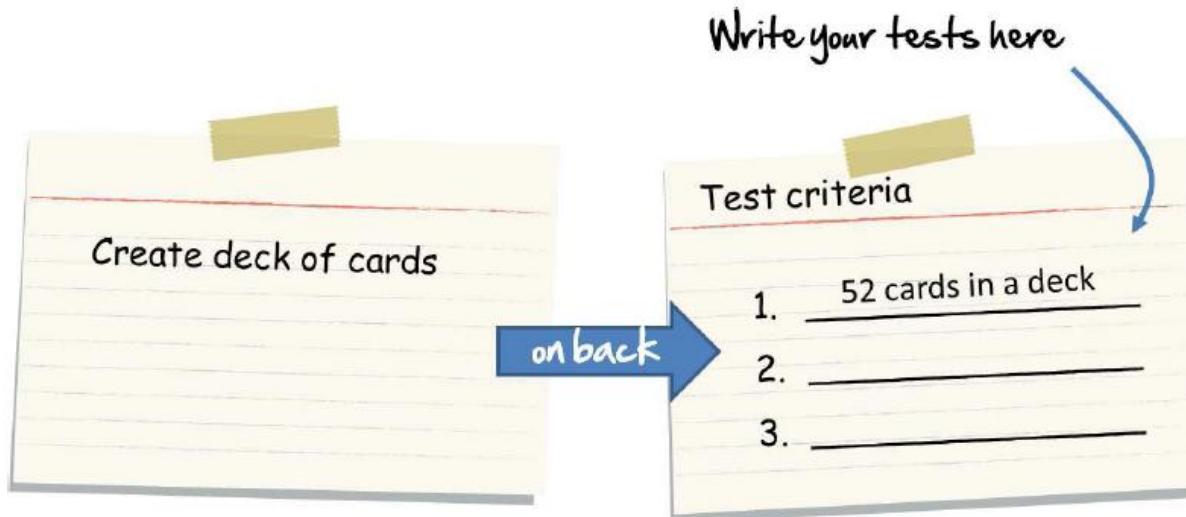
ถ้าคุณเจอกับสถานการณ์เหล่านี้ ก็อย่าเพิ่งถึงขั้นสิ้นหวังกันนะครับ สิ่งต่างๆเหล่านี้คือสิ่งที่ไม่ได้เจอกันทั่วๆ ไป และในหลาย ๆ กรณีจริงๆ แล้วคุณสามารถที่จะสร้างข้อมูลเบ็ดเตล็ด และจุดตรวจสอบขึ้นมา ใน method ที่คุณต้องการจะเขียน และนี่ก็เป็นจุดที่สามารถที่จะนำ mocking framework ที่มีกันอยู่ในปัจจุบันนี้มาช่วย

ส่วนในบางกรณีจริงๆ ที่คุณพบว่าไม่สามารถที่จะทำญี่นิตเทสได้เลย นั่นก็คือว่าจะมีสาเหตุมาจากเรื่องการออกแบบตั้งแต่แรก มากกว่า (ลองดูในบทที่ 14 เรื่อง Test-Driven Development) หรือคุณอาจจะตัดลอก ได้ดีเก่าซึ่งแยกต่อการทดสอบแต่ต้นอยู่ แล้วก็เป็นได้

ถ้าพบความจริงดังนั้น คุณเองก็ต้องยอมรับความจริงที่ว่า คุณเองก็ไม่สามารถทำการเขียนญี่นิตเทสให้ครอบคลุมทั้งหมด ทุกสิ่ง ทุกอย่างได้ ก็เพียงให้แน่ใจว่าคุณเองมีการทดสอบอย่างอื่นมารองรับแทน ไม่ว่าจะเป็นแม่นวนทดสอบ หรือ exploratory testing เพียงแค่อย่ายอมแพ้! ต้องระลึกเสมอว่าคุณต้องขอติเมตเทสเหล่านั้นให้ได้มากที่สุด เพราะมันจะเป็นเหมือนเกราะป้องกันที่จะมาช่วยคุณเอง จากบางครั้งหากมีความจำเป็นต้องแก้ Bug ฉุกเฉิน และจำเป็นต้องออก Release ในมืออย่างเร่งด่วน นอกจากนี้คุณสามารถอ่านหนังสือของ Michael Feathers ที่ชื่อว่า *Working Effectively with Legacy Code* ซึ่งมีคำแนะนำมากมายเกี่ยวกับการทำงานกับโค้ดเก่า และทำให้มันง่ายต่อการเปลี่ยนแปลงในอนาคต



เราสามารถคิดเห็นกันว่าเป็น Tester กันนะครับ ว่าเราจะเขียนยูนิตเทสเพื่อทดสอบ คลาสสำหรับไฟ อะไรมีปั้ง จากความต้องการของลูกค้าด้านล่างนี้? และเราจะสามารถหาวิธีป้องกันไม่ให้ Bug เกี่ยวกับไฟโจ๊กเกอร์ กลับมาได้อย่างไร?



ถ้า yunit เทสของคุณ ดูจะม่ายค้ายกับโค้ดข้างล่างนี้ แสดงว่าคุณกำลังมีปัญหาทางแล็ครับ เราต้องการจะมี yunit เทสเพื่อทดสอบให้เด้มากที่สุดครอบคลุมทุกจุดที่อาจทำให้เกิด Bug ได้ ดังนั้นหากคุณเกิดความสงสัยว่าจุดใด อาจจะทำให้เกิด Bug ได้ ก็ให้เขียน yunit เทสให้จุดนั้นได้เลย

```

[TestFixture]
public class DeckTest2
{
    [Test]
    public void Verify_deck_contains_52_cards()
    {
        var deck = new Deck();
        Assert.AreEqual(52, deck.Count());
    }

    [Test]
    public void Verify_deck_contains_thirteen_cards_for_each_suit()
    {
        var Deck = new Deck();
        Assert.AreEqual(13, Deck.NumberOfHearts());
        Assert.AreEqual(13, Deck.NumberOfClubs());
        Assert.AreEqual(13, Deck.NumberOfDiamonds());
        Assert.AreEqual(13, Deck.NumberOfSpades());
    }

    [Test]
    public void Verify_deck_contains_no_joker()
    {
        var Deck = new Deck();
        Assert.IsFalse(Deck.Contains(Card.JOKER));
    }

    [Test]
    public void Check_every_card_in_the_deck()
    {
        var Deck = new Deck();

        Assert.IsTrue(Deck.Contains(Card.TWO_OF_CLUBS));
        Assert.IsTrue(Deck.Contains(Card.TWO_OF_DIAMONDS));
        Assert.IsTrue(Deck.Contains(Card.TWO_OF_HEARTS));
        Assert.IsTrue(Deck.Contains(Card.TWO_OF_SPADES));

        Assert.IsTrue(Deck.Contains(Card.THREE_OF_CLUBS));
        Assert.IsTrue(Deck.Contains(Card.THREE_OF_DIAMONDS));
        Assert.IsTrue(Deck.Contains(Card.THREE_OF_HEARTS));
        Assert.IsTrue(Deck.Contains(Card.THREE_OF_SPADES));

        // the others
    }
}

```

คำอธิบายสำหรับบุคคลทั่วไป ได้ด้านบนนั้นคือยูนิตเทสที่ใช้ในการ:

- ทดสอบว่าไพ่นแต่ละชุดมี 13 ใบ
- ให้แน่ใจว่าไม่มีไพ่โจ๊กเกอร์อยู่ในสำรับ

- ตรวจไฟทุกใบในสำรับ (ทั้งหมด 52 ใบ)

## เราจะไปศึกษาเพิ่มเติมได้ที่ไหน?

ในบทนี้เราเพิ่งจะเริ่มได้สัมผัสถึงผิวๆ ของเรื่องยูนิตเทสเท่านั้น ซึ่งกล่าวได้ว่าymbém เรื่องราวอีกมากภายในหัวข้อนี้ โชคดีที่ยูนิตเทสในขณะนี้เป็นที่รู้จัก และใช้กันอย่างแพร่หลายมากขึ้นในการพัฒนาซอฟต์แวร์ ซึ่งในภาคคอมพิวเตอร์สมัยใหม่นี้จะมียูนิตเทส Framework ไว้ให้ใช้ (ฟรีและพร้อมให้ดาวน์โหลดรออยู่แล้ว) รวมถึงโปรแกรมการสอนใช้งานเบื้องต้น จุดเริ่มต้นสำหรับนักพัฒนาที่ต้องการเข้าถึงแก่นของการทำยูนิตเทสนั้นคือ บทความ<sup>1</sup> คลาสสิกของ Kent Beck



## Master Sensei and the aspiring warrior

**ศิษย์:** ท่านอาจารย์ เป็นไปได้อย่าไรที่การทำยูนิตเทสจะไม่ทำให้มีทำงานได้ช้าลง? หมายถึงว่าเรากำลังต้องเขียนโค้ดมากขึ้นถึงสองเท่าไม่ใช่หรือครับ?

**อาจารย์:** ถ้าการเขียนโปรแกรมคือการพิมพ์ดีด ประโยชน์นักก็อาจจะจริง ยูนิตเทสนั้นมีขึ้นเพื่อช่วยยืนยันว่าเมื่อไหร่ก็ตามที่เราเปลี่ยนแปลงได้ดี ทุกอย่างที่เคยทำงานได้ก็ยังทำงานได้เหมือนเดิม นี่ก็ช่วยประยัดเวลาของเรามากให้ต้องไปทำเมนวนะเทสทั้งระบบ ทุกครั้งที่เรามีการเปลี่ยนแปลงได้ดี

**ศิษย์:** นั่นก็ถูก ท่านอาจารย์ ว่าแต่ให้เจ้ายูนิตเทสเนี่ยมันจะไม่ทำให้ตัดขาดจากเราไปนานบ้างหรือ? แล้วเราจะมั่นใจได้อย่างไรว่าทุกครั้งที่เราแก้โค้ด ยูนิตเทสจะต้องผ่านเสมอ?

**อาจารย์:** ถึงแม้จะมีความเป็นไปได้ว่ายูนิตเทสของเรานั้นถูกเขียนได้ไม่ดี ซึ่งมีทั้ง ข้อบกพร่องที่ hard-coded ไว้, tightly coupled, มีการอุปกรณ์ที่ไม่ดี แต่คุณก็เริ่มชินกับการให้เทสนั้นเป็นตัวช่วยกำหนดการอุปกรณ์ (บทที่ 14 Test-Driven Development) คุณจะพบว่ายูนิตเทสของคุณนั้นจะไม่ break แต่กลับช่วยปรับปรุงการอุปกรณ์ให้ดีขึ้น ปัจจุบันนี้เครื่องมือพัฒนามีอยู่ในส่วน

<sup>1</sup><http://junit.sourceforge.net/doc/testinfected/testing.htm>

ในสูตรที่เราเปลี่ยนโค้ดของเรา หรือในyuninit เพื่อให้สามารถทำให้ Yuninit เพลิดเพลินได้สะดวกขึ้น คุณสามารถที่จะแก้ไขชื่อ method ในทุกๆ ที่ในโค้ดด้วยการกดเปลี่ยนพิมพ์ไม่เกิดซ้ำ ซึ่งจะช่วยให้เราสามารถทำให้ Yuninit เพลิดเพลินได้ และ โค้ดที่เราได้เขียนนั้นไปด้วยกัน

**ศิษย์:** จริงหรือไม่ที่เป้าหมายของทีมเราในการทำ Yuninit เพลิดเพลินนี้คือ ครอบคลุม 100 % ?

**อาจารย์:** ไม่ถูกต้อง เป้าหมายของ Yuninit เพลิดเพลินนี้ไม่ใช่เรื่องความครอบคลุม มันเพียงแค่ให้ความมั่นใจกับศิษย์ และทีม นั้นพร้อมที่จะให้ลูกค้าเข้าร่วมแล้ว

**ศิษย์:** ถ้าอย่างนั้น Yuninit เพลิดเพลินจะครอบคลุมมากน้อยแค่ไหน?

**อาจารย์:** นั่นก็แล้วแต่ตัวศิษย์และทีมจะต้องตัดสินใจ ในบาง Frameworks หรือภาษาที่นั้นช่วยให้การทำเทสให้ครอบคลุมนั้นง่าย แต่ในบางภาษาที่ต้องใช้ยาก ถ้าศิษย์กำลังเพิ่งจะเริ่มต้น ก็อย่าไปปั้งงงมากเรื่องความครอบคลุม แค่เพียงตั้งหน้าตั้งตาเขียน Yuninit เพลิดเพลินให้ได้มากที่สุดอย่าสุดฝีมือ ก็พอ

**What's Next?**

เก่งมาก ตอนนี้คุณก็ได้รู้จักกับ หลักการพื้นฐานสำคัญข้อหนึ่ง ของการพัฒนาซอฟต์แวร์แบบแยกไฟล์ ถ้าไม่มีขอโน้ต Yuninit เพลิดเพลินนี้ ลืมก็ไม่อาจเกิดขึ้นได้ เช่นกัน

ตัดไปเราจะไปดูกันว่าการสร้าง Yuninit เพลิดเพลินของเรางง และทำอะไรที่สำคัญที่บังคับร้องจากทำให้เราเข้ามาราทำ Yuninit เพลิดเพลินไปได้ โปรดดักษ์ของเรานั้น ก็จะมีราคาที่สูงเกินจริง ที่มีแต่โค้ดที่บำรุงรักษาลำบาก และไม่สะดวกต่อการเปลี่ยน

เราลองไปดูกันเลยครับ สำหรับ practice ที่สำคัญนี้ ก็คือ การรีเฟรชต่อร์

# ตอนที่ 13: รีไฟฟ์เดอร์: กลับมา

## บังคับความเมี้ยดงามเกิดนิดที่กึ่งไว้



ทำไ้มีช่างเบรียบเที่ยบได้อย่างใจร้ายใจจำว่าซอฟ์แวร์นั้นเหมือนกับบ้านที่ติดจำนำองที่เราต้องผ่อนดอกเบี้ยไปเรื่อยๆเพื่อ ลดจำนวนหนี้ลงไปนั่นบ้านแล้วซอฟ์แวร์ตู้จะผ่อนออกไว้อะง? บทนี้เราจะมาเรียนรู้ของการทำรีไฟฟ์เดอร์ว่า กระบวนการนี้ช่วยให้เราใช้หนี้ ไม่ใช่ ช่วยให้เราปรับปรุงและรักษาคุณภาพของซอฟ์แวร์เราให้ดีขึ้นเสมอได้อย่างไร มาดูกันว่าเราใช้หนี้ทางเทคนิคกันยังไง ???

### 13.1 Turn on a Dime (มองผ่านๆตอนแรกอ่านเป็น Damn)

เราามาดูสถานการณ์ก่อนก่อนหน้านี้เรายังคงส่วนแบ่งตลาดทั้งหมดของคลิปโนทั้งหมดได้ไปเรื่อยๆเมื่อคู่แข่ง “ราชายส์” แต่หลังจากเราชำรุดได้ไม่นานคุณคู่แข่งของเราก็ถือกำเนิดและไปรักษาอย่างนั้นก็ขายดีอย่างกะครึ่ปีคิริม เราก็ไม่มีทางปล่อยให้ทางโน้นแซงไปได้จึงรับรวมพลังงานและเริ่มแก้ไขปรับปรุงซอฟ์แวร์ของเราให้ดีขึ้น (หวังว่า) และแล้วสิ่งที่เราคาดหวังก็ไม่เป็นดังผู้สิงที่เราคิดว่าจะดัง แจ้วแจ่มแหลมระเบิดกลับกลายเป็นงานยาก เพราะอะไรหรือยกตัวอย่างเช่นโค้ดของเรามีการจัดการได้เลยเรา copy n paste กันให้มันมีม้าวไปหมดใน code base ของเรา การทำแบบนี้ส่งผลร้ายกับเรามากเนื่องจากทุกครั้งที่เราต้องการเพิ่มความสามารถใหม่เข้าเราต้องลบกับงานทั้งคือเราต้องไปตามแก้ไขก้อนโค้ดที่เราได้ copy n paste ไปทั่วและรถกอกันมากกว่าันนี้โค้ดเก่าที่เราคิดว่าทำเสร็จไปแล้วเมื่อ base line ที่แล้วเกิดอาการเพี้ยนกลับมาหลอกหลอนทีมเราอีกและเราให้หนักเข้าไปอีก ไอ้คนที่เรียนโค้ดชุดที่แล้วเขาได้ลาออกจากทีมไปเรียบร้อย!!!! เราลองมาดูตัวอย่างโค้ดเทพกันดีกว่า

```
public bool DealerWins(Hand hand1) {
    var h1 = hand1; int sum1 = 0;
    foreach (var c in h1) {
        sum1 += Value(c.Value, h1);
    }
}
```

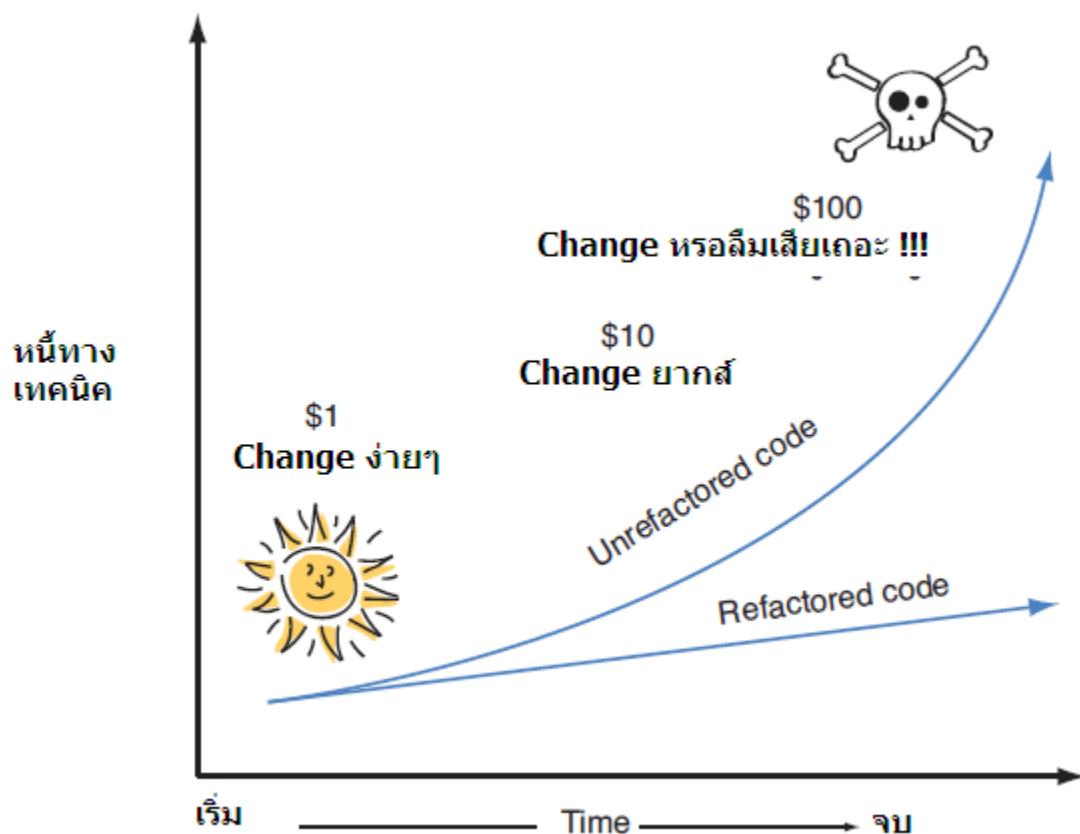
```

var h2 = DealerManager.Hand; int sum2 =0;
foreach (var c in h2) {
    sum2 += Value(c.Value, h2);
}
if (sum2>=sum1) {
    return true;
} else
return false;
return false;
}

```

อย่าตกใจถ้าอ่านไม่เข้าใจ (ถ้าอ่านเข้าใจสิประหลาด) เนี่ยແກລະຄວບหน້າຕາໂດດທີ່ເຮົາຕ້ອງຄູມນັດ່ອ ກາຣທີ່ເຮົາຕ້ອງມານັ່ງທຳ ທານກັບ ໂດດັບນັ້ນມັນເສີຍເວລາສິ້ນດີ ເລຍເດີດທຳໃຫ້ເສີຍເຈີນມາກື້ນໄປຄົກດັ່ງນັ້ນເພື່ອປັບປຸງໂດດເລັ່ນໃໝ່ເທິ່ງລັບມາເປັນສິ່ງທີ່ຖຸກຕ້ອງເຮົາຕ້ອງກາຣ ເວລາ 2 ອາທິດຍີແຕ່ເສີຍໃຈດ້ວຍຫຼືວິຈະຈິງມັນໄມ້ໄດ້ຂອງເວລາສອງອາທິດຍີຈ່າຍາກັນແບບນັ້ນ ມັນເກີດອະໄວ້ນັ້ນກັບແຄໂຄ້ດຊຸດເລັກາທີ່ເຄຍດູ ສ່ວຍງານອຸ່ດີ່ມັນກົກລາຍວ່າງເປັນສຸດວິປະລາດນ່າກລ້ວ ຄວບຄຸມໄດ້ຍາກ ນີ້ແກລະຄືອສິ່ງທີ່ເຮົາເຮີຍກວ່າ ພຶ້ທາງເທິນິກ ທີ່ເຈົ້າຈັກມັນ ໃນສຸານຕ່ອໄປ

## 13.2 ພຶ້ທາງເທິນິກ



ถ้าจะให้อธิบายง่ายๆ เจ้าก็สามารถเบรี่ยบเที่ยบให้เห็นภาพว่าหนึ่งทางเทคนิคคือโค้ดແຍ່ງໆที่โปรแกรมเมอร์ทำไม่ดีทิີ່ໃກ້ໃນ ไม่ว่าจะเป็นการแก้ปัญหาแบบทางลัด(shortcutss) การแยกสารพัดเพื่อให้ส่งงานได้(hack) การคัดลอกโค้ดไปวางในที่ต่างๆ ทำให้เกิดความซ้ำซ้อน(duplication) และบางครั้นก็มีความพยายามที่โปรแกรมเมอร์นักจะชอบทำตอนงานเร่งๆ ໄລ້ພຸດທິກວາມໄນດີເຫັນເນື່ອງເປັນສິ່ງທີ່ทำให้ເກີດหนึ่งทางเทคนิค(ถ้าໄມ່ມີໜີ້ເລີກຕູຈະແປລກເກີນໄປ ນັ້ນມັນເທົກກັບວ່າເວົາໄມ່ເຄຍຄົດທີ່ຈະທຳຂອງໄໝໄໝໆເລືຍ)

หนึ่งทางเทคนิค มັນໄຟໃຫ້ເຄີດນະຄົມ

หมายາຄົນກົດຕົວວ່າหนึ่งทางเทคนิคนີ້ມັນຄື້ອງໂຄ້ດ(ຫ່າຍ) ທີ່ເຮົາຫຼືໄວ້ໂຄ້ດໄຟເຫັນທີ່ໄວ້ແຕ່ໂຄ້ດເປັນແຕ່ອອກສ່ວນໃໝ່

ເພວະໜີ້ເວົາສ້າງໄດ້ໜາຍແບ່ມາກັບງານຄົງເວົາອີ່ນໄໝເຫັນເນື້ອງເປັນ config file, data ກີ່ມີມາໃຫ້ເວົາແກ້ເສມອ

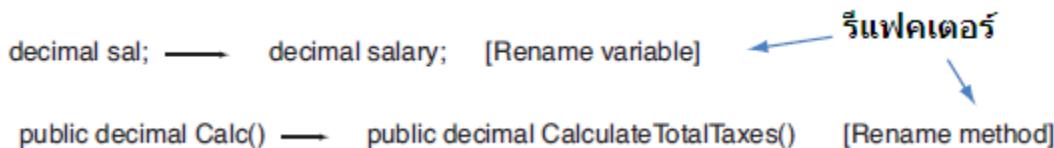
ຄົງໜີ້(ຄົນເຂົ້ານໄຟໃຫ້ເຄີນແປລນະ)ເຄຍເປັນໜີ້ໃນທີ່ມີມາຈົ່ວປະບຸນາດໃໝ່ທີ່ມີຄວາມຊັບຊົນນາກ ມີນັ້ນຄື້ອງເວົ່າງ  
ມາຕຽບຈຳການເຫົ່ານີ້ເມື່ອເມື່ອຕ່າງໆທີ່ "ໄມ່ມີ" ດັ່ງນັ້ນກະຫັນທີ່ຕ້ອງທຳມີເຄີຍເຫົ່ານີ້ມາເພື່ອຊ່າຍໃນການແປ່ງຄໍາແລ້ນໜັ້ນໃໝ່  
ຕອງກັນ ແກ່ນທີ່ຈະສັບຍັດກັບງານເລັກນ້ອຍແບ່ນີ້ກາລຍເປັນວ່າຕ້ອງຈ່າຍແພງແລະດັດຊູດນີ້ກີ່ຕ້ອງອູ້ໃນຮບປ່າໄປອຸກນານ ເພວະ  
ຢັ້ງດົກລົງກັນມີເຟີ່ຈະໃຫ້ຂະໃໝ່ໄວ້ເປັນຄ່າກລາງຕີ !!!

อย่างໄກກໍຕາມການໃຫ້ທາງລັດຫຼືການແກ້ປົງໝາເນີພວະໜີ້ນັ້ນກີ່ຕ້ອງມີກວບຫຼືເພດານກັນບ້າງໄຟເຫັນນັ້ນໄລ້ສົ່ງເລີກຕາງທີ່ເຮັດວຽກວ່າທາງລັດ  
ນັ້ນມັນຈະກັບມາທຳໃຫ້ເວົາເຈັບປັດແສນສາຫັສໃນກາຍຫລັງ

ໜີ້ທາງເທົນີ້ສາມາດພົບເຫັນໄດ້ໜາຍຮູປແບບຍກຕ້ວຍ່າງເຫັນ (spaghetti code, excessive com- plexity, duplication, ແລະ general sloppiness) ແລະທຸກຍ່າງກີ່ຈະມີພຸດທິກວາມໃນການແສດງຄວາມເທິພທີ່ແຕກຕ່າງກັນເມື່ອຍາມເກີດປົງໝາແຕ່ສິ່ງທີ່ມັນທຳ  
ເໜືອນກັນຄື້ອນຈະທຳໃຫ້ເວົາປັດກະໂຫລກນາກກົກກົກມີເວົາຕ້ອງມາຕາມແກ້ປົງໝາແລ່ລຳນັ້ນ ດັ່ງນັ້ນເວົາຕ້ອງກາຮະໄວສັກອ່າງທີ່  
ສາມາດເຂົ້າມາຈ່າຍເວົາດ້ວຍທີ່ເຮົາຫຼືຄົນອື່ນທີ່ໄວ້ເພື່ອໃຫ້ເວົາສາມາດສ້າງຂອົບທີ່ແວ່ງທີ່ມີຄຸນພາບແລະສາມາດຮອງຈັບຄວາມທ້າທາຍ  
ທີ່ຈະເກີດຂຶ້ນໃນອາຄືໂດຍທີ່ໄໝຕ້ອມມານັ້ນກັບລວມທີ່ເກົ່າຈະກັບມາຫລອນເຮົາອີກເມື່ອໄໝ່ ສໍາຫວັບວິດແໜ່ງແອຈໄຈດ້ວຍເວົາເຮົາເຈົ້າມື້ນີ້  
ຈ່າ Refactoring

### 13.3 ເຮົາແກ້ໄຂຄວາມຜິດພາດດ້ວຍການທຳ Refactoring ກັນ

Refactoring เปັນກະບວນການທຳການທີ່ເນັ້ນເວົ້ອງການເປັນເປົ້າໃຫ້ດ້ານໂຄ້ດຂອງໂທີ່ແລ້ວເລັກລະນ້ອຍເພື່ອເພີ່ມຄຸນພາບຂອງການໂຍຈະໄຟເປັນພຸດທິກວາມກາຍນອກຂອງໂທີ່ແວ່ງ  
ຫຼືອຸປະ່າງຍ່າດີເປັນມີຄົດເຮຍາມີເກົ່າໄໝເກົ່ານັ້ນໃຈາຫຼືສິ້ນ ເວັບປັນທີ່ໄດ້ກຳດົດຂຶ້ນ  
ອ່າຍ່າໃກ້ຕາມການປັບດ້ານຂອງເວັນນັ້ນຈະຫຍ່າໃຫ້ດ້ານຂອງເວັ້າໃຈໄດ້ຈ່າຍເກົ່າໄໝໄດ້ຈ່າຍ ແນ່ນອນວ່າເວົາສາມາດເປັນເປົ້າໃຫ້ຂຶ້ນໃນອາຄື ແລະໄຟກາປັບດ້ານແບບນີ້  
ເວົາເຮົາເຈົ້າມື້ນີ້ແກ້ໄຂການປັບດ້ານຂອງເວັນນັ້ນຈະຫຍ່າໃຫ້ດ້ານຂອງເວັ້າໃຈໄດ້ຈ່າຍເກົ່າໄໝໄດ້ຈ່າຍຂຶ້ນໃນອາຄື ແລະໄຟກາປັບດ້ານແບບນີ້  
ຈ່າຍເກົ່າໄໝໄດ້ຈ່າຍຂຶ້ນໃນອາຄື ແລະໄຟກາປັບດ້ານຂອງເວັນນັ້ນຈະຫຍ່າໃຫ້ດ້ານຂອງເວັ້າໃຈໄດ້ຈ່າຍເກົ່າໄໝໄດ້ຈ່າຍຂຶ້ນໃນອາຄື



ການແກ້ໄຂໄດ້ແບບນີ້ຂ່າຍຈະດູດເລັກນ້ອຍໃຈ້ຕ່າງໃນຄັ້ງແກ່າແຕ່ມີອັນດັບພະຍາຍາວ່າໄໝເກົ່າໃຈທີ່ກຳມັນຍ່າງຕ່ອນເນື່ອງຍາວານກັບ code base ຂອງເວົາເຈົ້າມື້ນີ້ແລ້ກາແລ້ວຈະຫຍ່າ  
ທຳໃຫ້ເກີດຜົດດ້ານທີ່ເວົາໄດ້ໂດຍເຂົ້າໃຈເວົາໃຫ້ດ້ານຂອງເວັນນັ້ນຈະຫຍ່າໃຫ້ດ້ານຂອງເວັ້າໃຈໄດ້ມາກວ່າກັນ

```

if (Date.Before(SUMMER_START) || Date.After(SUMMER_END))
    charge = quantity * _winterRate + _winterServiceCharge;
else
    charge = quantity * _summerRate;

```

OR ...

```

if (NotSummer(date))
    charge = WinterCharge(quantity);
else
    charge = SummerCharge(quantity);

```

รีแฟคเตอร์ [Extract method]

จากตัวอย่างโค้ดด้านบนต่อให้ พอดล่า เทคโนโลยี มาอ่านก็ต้องเลือกแบบที่สอง เพราะอะไร เพราะการเขียนโค้ดนั้นสามารถเปลี่ยนไปกับการเปลี่ยนโครงสร้างของข้อมูลได้ยาก ดูสะอาดตาไม่ซับซ้อน มันน่าอ่านกว่ากันเยอะ และการได้มาซึ่งโค้ดแบบนี้นั้นหนีไม่พ้นที่จะได้มาจากการทำ รีแฟคเตอร์ นั้นเองโดยที่หลักการของการทำ รีแฟคเตอร์ ก็ไม่มีอะไรมากไปกว่า การใช้ชื่อที่เรียบง่าย ชื่อรายละเอียดที่ไม่จำเป็นไว้ แค่นี้โค้ดของเราก็จะอ่านสะดวกขึ้นเยอะมากแล้ว

## ธรรมชาติ

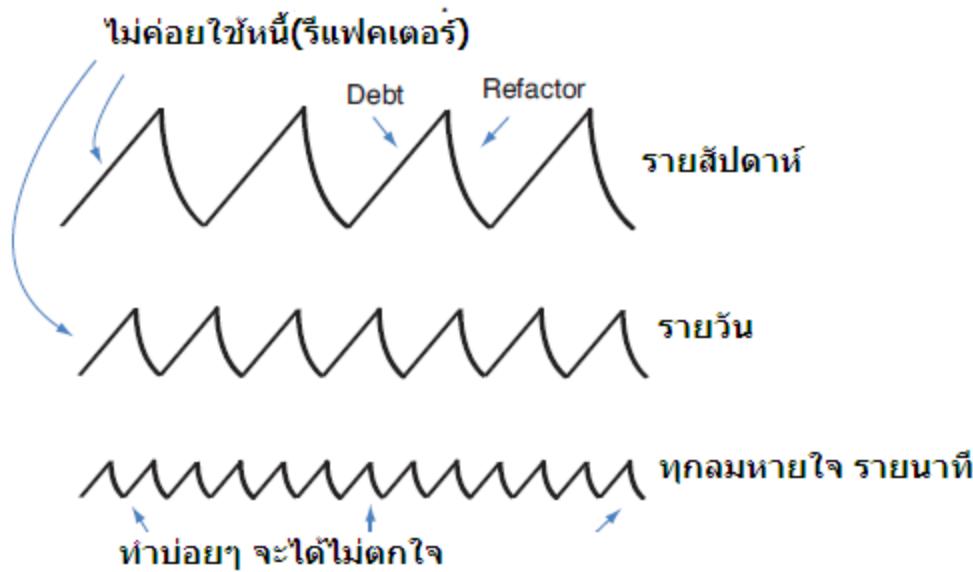


จะมีสติคิดถึงการทำงานทาง Technical ที่ดีอยู่เสมอ และการออกแบบที่ดี จะช่วยส่งเสริมความเป็นธรรมชาติ

ดังนั้นการทำรีแฟคเตอร์โดยเนื้อแท้แล้วไม่มีอะไรเลยนอกจากแนวคิดที่จะทำให้โค้ดของเรางาม เข้าใจง่าย เพราะเมื่อมันถูกสร้างขึ้นมาแล้วมันยังต้องถูกดูแลแก้ไขไปอีกสักระยะ (ไม่ว่าสักหนึ่งหรือยาวเท่าไหร่) ดังนั้นหน้าที่ของเราคือทำให้มันดูแลง่าย เพราะเราอาจจะสักสุกทุกครั้งที่มีการเปลี่ยนแปลง เราจะไม่กลัวการเปลี่ยนแปลงใดๆทั้งสิ้น

รีแฟคเตอร์ จัดให้หนัก—จัดให้ต่อเนื่อง

การทำรีแฟคเตอร์ที่ดีคือเราต้องเราให้ปอยๆทำทุกวันอย่าเว้นไว้ทำตอนจะจบโปรเจค ถ้าเราทำได้แบบนี้เราจะไม่มีปัญหา หรือข้ออ้างมาอ้างว่าการทำรีแฟคเตอร์ทำให้งานเราช้าลงในช่วงที่เร่งจะปิดงาน ดังนั้นจะทำมันทุกวันเอาให้เข้าเล่น และเมื่อเราทำมันจนชำนาญแล้วการทำไฟฟ้ารีแฟคเตอร์จะกลายเป็นส่วนหนึ่งของเรามันจะกลมกลืนไปกับการทำงานโดยที่บางครั้งเราเก็บแยกไม่ออก เพราะมันกลมกลืนมากๆถ้าถึงขั้นสูงสุดเราจะแยกไม่ออกเลยว่าการเพิ่มไฟเขียวใหม่เข้าไปใน โปรเจคกับการรีแฟคเตอร์ต่างกันอย่างไร (ดูสูงส่งมากครับแต่ทำได้จริง)



ได้เห็นทุกช่วงกันไปye กะ มาลองลงมือทำกันให้มันสมีค่า



เราต้องปรับอะไรบ้างกับสุดยอด black jack ของเรา

```
public bool DealerWins(Hand hand1)
{
    var h1 = hand1; int sum1 =0;
    foreach (var c in h1)
    {
        sum1 += Value(c.Value, h1);
    }
    var h2 = DealerManager.Hand; int sum2 =0;
    foreach (var c in h2)
    {
        sum2 += Value(c.Value, h2);
    }
    if (sum2>=sum1)
    {
        return true;      มีโค้ดอะไรเกินมาใหม่
    }
    else
        return false;
}
}                                มีตัวแปรไหนต้องเปลี่ยนชื่อใหม่?
```

มีอะไรข้ามในมุมมองของ functionalities

ที่แรกที่เราควรจะทำรีเฟคเตอร์คือการสำรวจนูกว่ามีตัวแปรหรือเมธอดไหนที่ซื้อไม่ สืบ บ้าง

ตั้งนั้นอย่างอื่นมาเปลี่ยนมันกันได้

```

public bool DealerWins(Hand playerHand) {
    int playerHandValue = 0;
    foreach (var card in playerHand)
    {
        playerHandValue += DetermineCardValue(card, playerHand);
    }

    var dealerHand = DealerManager.Hand;
    int dealerHandValue = 0;

    foreach (var card in dealerHand)
    {
        dealerHandValue += DetermineCardValue(card, dealerHand);
    }

    return dealerHandValue >= playerHandValue;
}

```

### รีแฟคเตอร์ ทำให้โค้ดดูสวยงามขึ้น

ดูดีขึ้นมาหน่อยครับ อ่านง่ายขึ้นมากินส แต่ยังงานเพิ่งเริ่มส่วนต่อไปคือหาสิ่งที่ซ้ำกันออกไปเรามากันหน่อยอย่าวลอกจิกบริเวณไหน มันซ้ำกันบ้างเอาจริงไปตั้งเป็นเมธอดใหม่จะ

```

public bool DealerWins(Hand playerHand)
{
    int playerHandValue = GetHandValue(playerHand);
    int dealerHandValue = GetHandValue(DealerManager.Hand);

    return dealerHandValue >= playerHandValue; รีแฟคเตอร์ แยกเมธอด ]
}

```

```

private int GetHandValue(Hand hand)
{
    int handValue = 0;

    foreach (var card in hand)
    {
        handValue += DetermineCardValue(card, hand);
    }
    return handValue;
}

```

ดูหล่อขึ้นมากมายครับหลังจากที่เราได้แยกจิกที่ซ้ำออกมารีบุกเป็นเมธอด GetPlayerHandValue ทำให้เมธอด DealerWins ของ เรายากดูง่าย多了 สำหรับทั้ง ส่งผลให้เราเข้าใจการทำงานของมันมากขึ้นยังไงไม่พอก็รับเรียบง่ายไปได้เช่นเดียวกัน

```

public bool DealerWins(Hand playerHand) {
{
    return GetHandValue(DealerManager.Hand) >= GetHandValue(playerHand);
}

```

รีเฟคเตอร์

[Inline variable]

สรุปกันหน่อยว่าที่ปานมาเราทำอะไรบ้าง เราทำงานง่ายๆไปสามอย่างคือ

- เปลี่ยนชื่อตัวแปร เมธอด
- เปลี่ยนมาใช้ inline variable
- แยกลอกจิกที่ข้ออกมาเป็นเมธอดใหม่

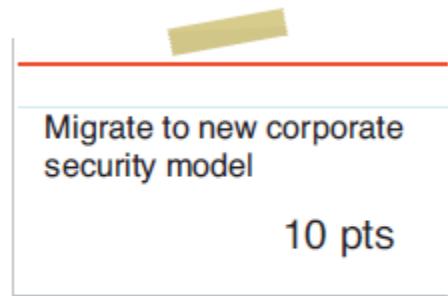
เมื่อเราสามารถย่อๆแล้วเรา ก็ได้โดยที่ savvy ขึ้น อ่านง่าย คนทั่วไปสามารถคาดเดาพฤติกรรมของ โค้ดชุดนี้ได้เมื่อถึงเวลา ที่ต้องการเปลี่ยนหรือเพิ่มอะไร ก็ทำได้ง่ายขึ้นใช้คนน้อย เวลาน้อย ใช้เงินน้อย เมื่อได้เห็นประโยชน์ที่เกิดขึ้นแบบนี้แล้วเราก็ควรจะตั้งใจ ผลักดันให้โปรแกรมเมอร์ทุกคน มีคำว่า “รีเฟคเตอร์” อยู่ในหัวใจ ทำให้บอยๆ จัดให้หนัก



เป็นคำถามที่ดีได้น่องเอย ในบางครั้งบางกรณีเรา ก็ต้องเจอกับการปรับเปลี่ยนในระดับใหญ่มากกว่าการเปลี่ยนทีละน้อย แบบที่เราเพิ่งว่ากันไป เช่นเราอาจจะต้องเปลี่ยน library หรือ framework ที่จำเป็นต้องเปลี่ยน ตลอดเปลี่ยน หรืออาจจะต้องการต่อเข้ากับเครื่องมือตัวอื่น หรือไม่ว่าจะด้วยเหตุผลใดการทำรีเฟคเตอร์ขนาดใหญ่ ก็ต้องเกิดขึ้นบ้าง แล้วเราจะรับมือมันยังไงดี ถ้าการทำรีเฟคเตอร์นี้มาจากคนนอกทีมเรา ก็จัดการมันให้เหมือนกับเป็น story หนึ่งๆแล้วก็นำกระบวนการฯลฯ ใจลุดเทพ ของเรามาเข้าไปจัดการ เช่น ประเมิน มัน, จัดความสำคัญ มัน, ตีราคा, และวิเคราะห์ผลกระทบที่จะเกิดกับโปรเจคโดยรวม

### Refactoring Gets a Dirty Name

Once, while building an energy-trading application, our team went off and did several large-scale refactorings in the code base and didn't add much in the way of new functionality for several weeks. Well, it didn't take long for management to come to despise the word refactoring (because it came to mean rework and not adding any new functionality), and soon the edicts came from above that thou shalt not refactor. Don't let this happen to you. Refactor continuously as you go. It's much harder to pay down the technical debt later, and the last thing you want to do is give refactoring a dirty name.



### รีแฟคเตอร์ ครั้งใหญ่

จริงๆแล้วการจัดการกับปัญหาเหล่านี้สามารถต้องใช้วิจารณญาณมากพอสมควร เนื่องจากถ้าเราเห็นว่าการทำรีแฟคเตอร์ขนาดใหญ่มีความสำคัญและเจาส์กองทัพไปแกรมเมอโนลิงไปทำงานนั้นเท่ากับว่ารายจ่ายของเราจะเพิ่มขึ้นอย่างเห็นได้ชัด ดังนั้นถ้าเกิดเหตุการณ์แบบนี้ก่อนตัดสินใจลงมือทำอะไรให้คิดถึงสองข้อคิดนี้ก่อน

1. ประโยชน์ใดๆบ้าง?
2. จริงๆแล้ววิธีรีแฟคเตอร์ใหญ่เมื่อมันทำให้เราเสียเวลา ดังนั้นเราจะไม่สามารถทำงานตามแผนที่วางไว้ได้ เพราะฉะนั้นแล้ว

เนื่องจากการทำรีแฟคเตอร์นานในภายมันทำให้เราเสียเวลา ดังนั้นเราจะไม่สามารถทำงานตามแผนที่วางไว้ได้ เพราะฉะนั้นแล้วถ้างานวางแผนมากรากลับส่งแล้ว ให้หยุดการทำรีแฟคเตอร์ไว้ก่อน แต่ถ้ามันจำเป็นต้องทำและงานยังอยู่ระหว่างกลางก็ให้แบ่งทำไปทีละเล็กๆน้อยๆเนื่องจาก เราจะสามารถทำพื้นที่ใหม่และค่อยซ้อมของเก่าไปเรื่อยๆ ถ้ามันต้องทำ ก็ต้องทำแต่จะดีมากถ้าเราสามารถทำให้ได้

เราจะไปศึกษาเพิ่มเติมได้ที่ไหน?

เนื้อหาเรื่องรีแฟคเตอร์ที่เราเพิ่งจะไปนั้นเป็นแค่ส่วนผิวนางมากๆเกี่ยวกับการทำรีแฟคเตอร์ ดังนั้นถ้าจะทำให้ถูกต้องไปอ่านเพิ่มเติม โดยหนังสือที่แนะนำเกี่ยวกับเรื่องนี้คือ

1. Martin Fowler's Refactoring: Improving the Design of Existing Code [FBB+ 99].
2. Michael Feathers' Working Effectively with Legacy Code [Fea04].

ศิษย์: ท่านอาจารย์ มีสิ่งไหนบ้างที่เราควรหลีกเลี่ยงสำหรับการวิเคราะห์แก้ไข

อาจารย์: ก็ไม่ควรทำ วิเคราะห์ในญี่ปุ่น เราควรจะทำ วิเคราะห์ทุกครั้งที่เจ้าแก้ไขปรับปรุง ซอฟต์แวร์ของเรา

ศิษย์: จะผิดไหม ถ้าเกิดว่าทั้ง iteration ของเราไม่ทำอะไรเลยนอกจากการทำวิเคราะห์โค้ด

อาจารย์: มันก็เป็นไปได้呀ที่พ่อพิงให้นะศิษย์ แต่ทางที่ดีเราควรจะทำ วิเคราะห์บ่อยๆ ทำทีละน้อยๆ เพื่อจะสามารถทำ วิเคราะห์ขนาดใหญ่ๆ ได้เมื่อเวลาและเงินดังนั้นจะทำวิเคราะห์ใหญ่ๆ ก็ต่อเมื่อมันจำเป็นเท่านั้น

### What's Next?

ต่อไปเราจะได้เรียนรู้เรื่อง ยูนิตเทสท์ และการทำวิเคราะห์ไปพร้อมๆ กัน เมื่อเราเข้าสู่กระบวนการท่านนี้รวมกันแล้ว เราจะทำให้ design ห่วยๆ หมดไปจากระบบของเรา

เปิดตำราที่หน้าต่อไปโดยเร็วพั้นเพื่อศึกษาว่าเคล็ดลับ TDD นั้น เมพ ขนาดใหญ่เราจะเขียนซอฟต์แวร์ไม่ได้ถ้าไม่มี ยูนิตเทสท์ ก่อเท่าระเบิด

# บทที่ 14 ทีดีดี : เขียนยูนิตเทสต์ซะก่อน

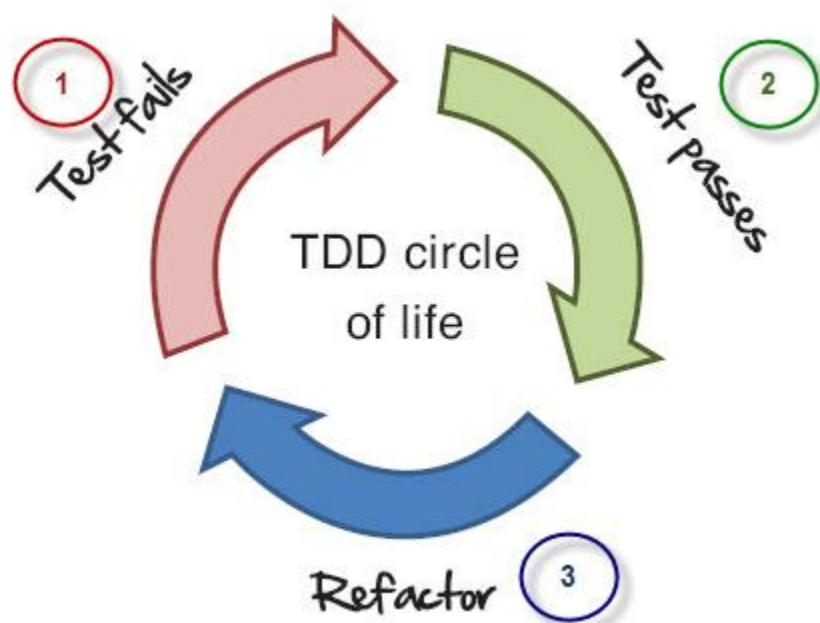
## ก่อน

นี่คุณ ติดอยู่ตรงนี้มานานแค่ไหนแล้ว ไปไม่เป็นเลยล่ะสิ ทั้งวันเขียนได้เท่านี้เองหรือ จะไปทางไหนต่อละที่นี่ ถ้าอธิบายเด็กเป็นตัวอย่างสิคุณไม่แพลงใจเลยหรือที่อธิบายเขียนโค้ดยังไงก็รู้ตลอดบางที่คุณก็ยิ่งโค้ดคำมาใช้บ้างมันก็ยังเวิร์กอยู่เลย ยังกะเด็กว่าคุณนะต้องการจะทำอะไร สิ่งที่อธิบายเด็กทำก็แค่มียูนิตเทสต์ครอบไว้ແນມยังรันเทสต์อัตโนมัติจะด้วย

อธิบายเด็กทำได้ยังไงนะแบบนั้น แล้วอยากรู้มั้ยว่าคุณทำไม่เหมือนเด็กตรงไหนເคลาลั่นถึงควรที่คุณจะต้องเลิกฉุนเนี่ยว รวมความกล้าแล้วเดินไปขอกความช่วยเหลือจากอธิบายเด็กได้แล้ว “อธิบาย ตามหน่อยนะ นายทำยังไง ให้ดูนายถึงดูสะอาด แล้วก็แบบนี้” “ง่ายจะตาย” อธิบายตอบ “ฉันก็แค่เขียนยูนิตเทสต์ก่อนเขียนโค้ดนะสิ”

### 14.1 เขียนยูนิตเทสต์ก่อน

ทีดีดี (Test-driven development) เป็นเทคนิคการพัฒนาซอฟต์แวร์ที่ช่วยทำให้ตัวดีไซน์เพิ่มขึ้นทีละนิดจากการใช้รูปแบบของวงจรการพัฒนาซอฟต์แวร์ (development cycle) ที่สั้น ยังไงนะหรือ มาว่าจักว่างๆ ทีดีดีก่อนเด็กกว่า



ข้อแรก: พัง – ก่อนจะเริ่มเขียนโค้ดของระบบ คุณต้องเริ่มเขียนยูนิตเทสต์ซะก่อน แหงล่ะมันก็รันไม่ผ่านอยู่แล้วแต่ที่ให้เขียนก็เพื่อให้รู้ว่าโค้ดของระบบมันจะทำอะไร นี่ถือว่าคุณกำลังออกแบบแบบระบบอยู่เลยนะ คิดให้ดีล่ะ

ข้อสอง: ผ่าน – ขั้นตอนนี้คุณเขียนโค้ดระบบได้เลย เขียนยังไงก็ได้ให้ยูนิตเทสต์มันรันผ่าน เอาแค่ให้มันผ่านพอนะ ถ้าคุณอยากรีเขียนโค้ดระบบเพิ่มมากกว่าyuนิตเทสต์ที่เขียนไว้ล่ะก็ กลับไปเพิ่มyuนิตเทสต์ก่อนเลย

ข้อสาม: ปรับ – เมื่อคุณรันเทสต์มันผ่านแล้ว ก็ได้โอกาสที่คุณจะกลับไปแก้โค้ดระบบส่วนที่มันเน่าๆ ก็ให้ส่วนที่คุณเอาไว้นี่มาไปเพื่อให้yuนิตเทสต์มันผ่านนะ ง่ายๆ ก็แค่เข้าไปแก้โค้ดที่มันชำรุดกันออก ให้ดีหน่อยไม่ใช้ก็ลบๆ ไปซะ ก็แค่ทำให้มันดูสะอาดหน่อยก็เท่านั้น

“แล้วเราจะเลิกทำสามอย่างนี้เมื่อไหร่ล่ะ” อธิบายก่าว่าเด็กๆ ทำสามอย่างนี้ไปเรื่อยๆ จนกว่าเด็กจะแนใจว่าโค้ดระบบันทำได้ทุกอย่างตามที่ทุก user story ต้องการ (ยังไงนะหรือ ก็ exit criteria ผ่านหมดทุกข้อแล้วนะสิ) นอกจากนี้ อธิบายตั้งกฎไว้ให้ด้วย เช่น “ให้ไม่หลุดวงจรที่ตีดีด้วย เดียวกันตัวอย่างมาให้ดู 2 กฎ

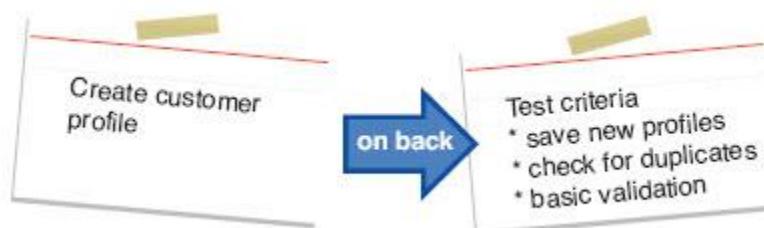
#### กฎข้อที่ 1: ไม่เขียนโค้ดระบบเพิ่มถ้าyuนิตเทสต์ยังผ่านอยู่

แต่อธิบายกับคนว่า “ไม่ต้องเคร่งกับกฎตลอดเวลาหรอก เพราะว่ามันก็มีโค้ดบางอย่าง จะให้มันทำyuนิตเทสต์ก่อนมันก็ยาก อย่างพวก interface แต่ที่ต้องเคร่งกับกฎข้อนี้มากๆ ก็คือ เราควรจะเขียนโค้ดระบบเท่าที่จำเป็นเท่านั้น การเขียนyuนิตเทสต์ขึ้นมาก่อนมันก็เป็นการบังคับไปในตัวว่าเราไม่ได้เขียนโค้ดเกินพร้าเพรื่อ หรือที่เรียกว่า over engineering นั่นแหล

#### กฎข้อที่ 2: เทสต์ดะ

เราต้องทดสอบทุกกรณีที่เราคิดว่าระบบมันจะพัง แต่ดูกฎข้อนี้ให้ดีๆ นะ มันไม่ได้หมายความว่าต้องทดสอบทุกอย่างนะ เพราะอะไรมันก็เราต้องให้เวลาทั้งชีวิตนะสิ ถ้าจะให้มันครบถ้วนอย่าง เขายังเป็นว่าเทสต์เฉพาะเคสที่มันน่าจะเกิดก็พอแล้ว หรือเคสอย่างเช่น ถ้าระบบเราอยู่ใต้ภาวะหนึ่งแล้วมันต้องพังแน่ๆ ก็ทำyuนิตเทสต์เพื่อเทสต์โค้ดของเราก็เท่านั้น

ดูตัวอย่าง



ขอขอบคุณที่ได้รับเก็บไปฟล์ของนักพนันในเวกัส สิ่งที่เจ้าของกาสินอย่างรู้สึกว่ากับนักพนันพากนี้ก็ เช่นอะไรที่คนพากนี้ชอบ ไม่ชอบ อาหาร เครื่องดื่ม หรืออะไรก็ได้ที่จะช่วยดึงให้นักพนันพากนี้กลับมาเป็นลูกค้าที่กาสินของเดา ถูก ได้ที่อธิบายเป็นคอมเม็คโปรดไฟล์ง่ายๆ และเดาก็ต้องเก็บโปรดไฟล์พากนี้ลงไว้ในฐานข้อมูลด้วย พอก็จะเริ่มเขียนโค้ด แน่นอน เดาก็ต้องเขียนยูนิตเทสต์ขึ้นมาก่อน สิ่งที่เดาทำก็คือ ใช้จินตนาการว่าหน้าตาใด้ระบบมันจะเป็นแบบไหน จากนั้นก็ออกแบบความผิดพลาดมีอยู่ยูนิตเทสต์ก่อน เพื่อพิสูจน์ว่าได้ที่เดาเห็นใจจินตนาการมันจะใช้จริงไหม หน้าตาของยูนิตเทสต์ที่อธิบายเป็นแบบไหนกันนะ

```
[Test]
public void Create_Customer_Profile()
{
    // setup
    var manager = new CustomerProfileManager();

    // create a new customer profile
    var profile = new CustomerProfile("Scotty McLaren", "Hagis");

    // confirm it does not exist in the database
    Assert.IsFalse(manager.Exists(profile.Id));
    // add it
    int uniqueId = manager.Add(profile); // get id from database
    profile.Id = uniqueId;

    // confirm it's been added
    Assert.IsTrue(manager.Exists(uniqueId));

    // clean up
    manager.Remove(uniqueId);
}
```

[Test]

[Test]

ต้องทำ assert เป็นจริง  
เทียบค่าเดียว

หลังจากนี้ยูนิตเทสต์ให้คุณใจว่าตอนที่เพิ่งโปรดเข้าไปในระบบมันจะไม่พังแล้ว ก็ได้เวลาเปลี่ยนเกียร์มาเขียนโค้ดระบบกันต่อเลย จะเขียนยังไงดีนะให้เจ้ายูนิตเทสต์มันผ่าน อะ! จากยูนิตเทสต์ก็เห็นอยู่แล้วว่าเมธอดที่ต้องการคือสร้างเมธอด รับพารามิเตอร์ เป็นคอมเม็คโปรดไฟล์ และก็ใส่ลงไว้ในฐานข้อมูล คิดได้แล้วก็มาเริ่มเขียนโค้ดระบบกันเถอะ

```

public class CustomerProfileManager
{
    public int Add(CustomerProfile profile)
    {
        // pretend this code stored the profile
        // in the database, and returned a real id
        return 0;
    }

    public bool Exists(int id)
    {
        // code to check if customer exists
    }

    public void Remove(int id)
    {
        // code to remove a customer from the database
    }
}

```

เขียนเสร็จแล้ว ก็รันตัวยูนิตเทสต์ อื้อ! ผ่านแล้ว ดีใจจังเลยพอยูนิตเทสต์ผ่านก็ต้องมาปรับปรุงได้ ก็ทำวิเฟคเตอร์นั่นแหล่ะ ก็ต้องไป่ลุ่งให้หมด ไม่ว่าจะเป็นโค้ดระบบ โค้ดในยูนิตเทสต์ คอนฟิกไฟล์ หรือไดอะกราฟแล้วแต่ที่อธิบายแก้เพื่อให้ยูนิตเทสต์รันผ่าน

พอทำวิเฟคเตอร์เสร็จแล้ว ก็มานั่งคิดใหม่ว่ามันจะมีเคสในอีกหนึ่งที่จะทำให้ระบบพังได้ อย่าง story นี้ยังจะต้องเทสต์ว่าโปรไฟล์ในระบบจะไม่ซ้ำกันด้วยคิดได้แล้วก็ใส่เกียร์โดยกลับไปเริ่มใหม่ เขียนยูนิตเทสต์ก่อน ใส่เกียร์อีกครั้งเพื่อเขียนโค้ดระบบ รันยูนิตเทสต์ พอรันผ่านหมดแล้ววิเฟคเตอร์อีกเรื่องการเขียนยูนิตเทสต์ก่อนนี้ มันก็เหมือนเรื่องไก่กะไก่ ใจเกิดก่อน เพราะบางทีมันก็ยกเหมือนกันที่จะไม่ได้ระบบมาก่อน เอาละไม่เป็นไร กลับไปเขียนเมธอดในโค้ดระบบก่อนก็ได้ แต่ก็เคยแคพอที่จะเขียนยูนิตเทสต์ต่อได้นะ แล้วก็มาเขียนยูนิตเทสต์ต่อให้เสร็จ ห้ามโงงจนน่าเกลียดละ เอาละ ที่นี่ก็พอจะเห็นแล้วใช่ไหมว่าที่ต้องเด็กากันยังไง ลองกลับไปทำกันเองได้แล้ว

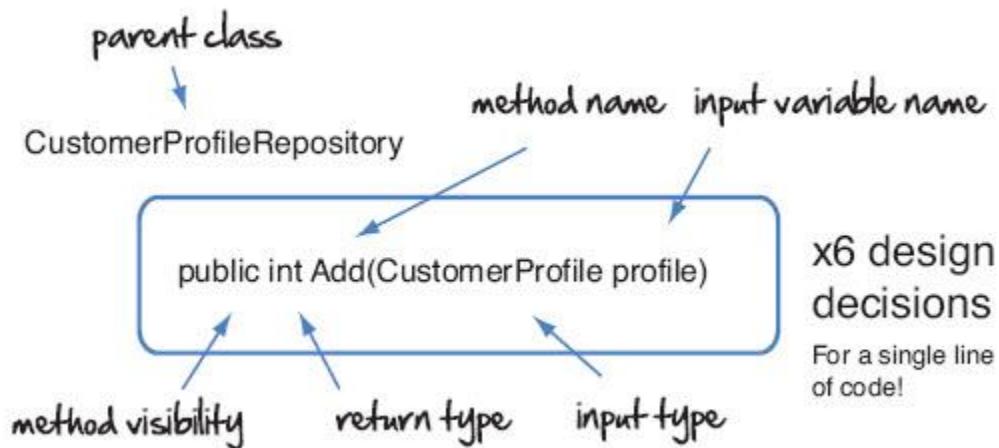
## ทวนกันสักนิด

เอาละ ก่อนจะไปต่อ เรามาทวนเรื่องที่ต้องเด็กากันหน่อยแล้วกัน ว่ามันคืออะไร แล้วทำไม่มันถึงสำคัญขนาดที่ต้องคือการเขียนยูนิตเทสต์ก่อน จากนั้นเขียนโค้ดระบบ รันเทสต์ให้ผ่าน เท่านี้เอง มันออกจะฝืนๆแล้วก็ไม่เห็นจะมีใครสอนเราให้ทำแบบนี้มาก่อนใช่ไหมแต่คิดดีๆอีกทีสิ ที่ต้องมันก็คือการออกแบบระบบนะ ดีกว่าหาดูในความผันแปรค่ายมาเขียนโค้ดระบบใช่ไหมล่ะที่ต้องยังช่วยให้เราเขียนโค้ดระบบแบบเพียง ไม่เขียนเพื่อเขียนเกินตัวยังไงรู้สึกว่าจะให้คุณกลับไปแล้วเริ่มทำที่ต้องเลียนนำมันยาก ไหนจะต้องรู้เรื่องเทคนิคการเขียนยูนิตเทสต์และการทำวิเฟคเตอร์อีก แน่นอนเลยว่าแรกๆคุณจะต้องงงงวยไปกับมันแน่นา ว่ามันทำ

ยังไงกัน ที่ค้ำเจ้าที่ติดเนี่ย แต่เชื่อเถอะว่า แค่คุณได้ลองเขียนยูนิตเทสต์เล็กๆ สักเคสหนึ่ง เวลาที่คุณรันเทสต์ แล้วเห็นมันผ่าน บางกับที่คุณได้มีเวลาามานั่งรีไฟค์เตอร์ติดด้วย คุณจะต้องภูมิใจในยูนิตเทสต์และได้ระบบของคุณได้อีกเยอะเลย

## 14.1 จัดการความซับซ้อนด้วยยูนิตเทสต์

ปกติเวลาเราเขียนโค้ด ก็ต้องเผชิญหน้ากับความซับซ้อนของโค้ดอยู่แล้ว ตัวอย่างง่ายๆ อย่าง อธิบาย API ขึ้นมาสักอันนึงตอนเดาเขียนโค้ดที่จะสร้างไฟล์ใหม่ขึ้นมาในระบบ เค้ามีเรื่องที่จะต้องตัดสินใจตั้ง 6 อย่าง



คิดดู แค่โค้ดบรรทัดเดียว ต้องตัดสินใจตั้ง 6 เรื่อง แล้วถ้าเอาโค้ดทั้งหมดมารวมกันล่ะ มันจะซักกี่เรื่อง แต่อย่างว่ามันก็ต้องมีตัดสินใจหลายบ้างอะแหลกคุณทำที่ติดล่ะ แน่นอนคุณจะไม่เพิ่มโค้ดระบบเข้าไปเยอะแยะมากmany เว่อร์กิنجริง ครบเท่าที่คุณยังทำยูนิตเทสต์ให้พังไม่ได้ ฉะนั้น ถ้าความซับซ้อนของโค้ดระบบของคุณมันก็จะลดลงไปโดยปริยาย

หนำซ้ำ ที่ติด ยังช่วยให้คุณออกแบบระบบได้อย่างมั่นใจไม่ปวดหัว เพราะคุณมีหน้าที่แค่จดจ่ออยู่กับการจะทำให้ยูนิตเทสต์แค่ 1 ข้อรันผ่านไปได้ยังไง ไม่ใช่ว่า เอาเวลาไปคิดถึงเรื่องร้อยพันอย่างในเวลาเดียวกัน อย่างนี้มันจะทำให้คุณแก้ปัญหาได้ไปทีละจุด แก้เสร็จแล้วก็ค่อยไปแก้ปัญหาตัวไป แบบนี้มันจะทำให้คุณแก้ปัญหาได้ถูกทางมากกว่าที่จะมองภาพกว้างๆ ใช้ใหม่ล่า

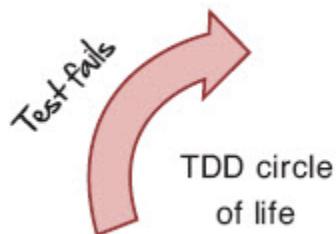
นอกจากนี้ยังมีอีก 6 เหตุผลที่คุณเค้ามักจะเอาที่ติดมาเขียนนะ



ทั้ง 6 เหตุผลนี้ ล้วนหมายถึงว่าโค้ดเราจะบำรุงรักษาง่าย จะแก้ไขง่าย ถึงจำนวนบรรทัดจะน้อย แต่นั่นก็ทำให้ความซับซ้อนน้อยลงไปด้วย ระบบก็ออกแบบไว้แล่นจะง่ายรองรับการเปลี่ยนและการแก้ไขได้ง่ายกว่า อีกข้อจุดเด่นของการอธิบายไว้แค่นี้ เพราะเดาคิดว่ามันถึงตาก็คงจะต้องมาลงมือเขียนยูนิตเทสต์แบบที่ได้ดูบ้างแล้ว



อีกคลองให้โจทย์คุณในการเขียนโค้ดเพื่อเปรียบเทียบค่าตัวเลขในไฟ 2 ใบ อีกคิดว่ามันน่าจะต้องมีการทำงานอยู่คลาสที่ชื่อว่า Card และไม่ต้องห่วง อีกจะช่วยคุณเริ่มเขียนตัวยูนิตเทสต์ด้วยที่นั่นลองมาเขียนยูนิตเทสต์ก่อน ขึ้นแรกลองมาเขียนเมrorod เอาจริงๆ ที่คุณคิดว่าสื่อความหมายสำหรับการเปรียบเทียบค่าในไฟ



```
public void Compare_value_of_two_cards() {
```

```
    Card twoOfClubs = Card.TWO_OF_CLUBS;
    Card threeOfDiamonds = Card.THREE_OF_DIAMONDS;
```

*Write your test here*

} *Imagine the code already exists - just type it out!*

เอกสาระ เห็นช่องว่างนั้นใหม่ อธิบายให้คุณลองเขียนโค้ดในตัวอยุนิตเทสต์กรณีที่ค่าในไฟล์เป็นมีค่ามากกว่าอีกใบนึง ให้ลองเขียน  
ดู

[Test]

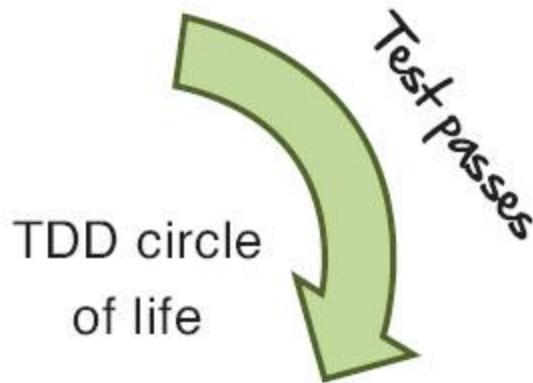
```
public void Compare_value_of_two_cards()
{
    Card twoOfClubs = Card.TWO_OF_CLUBS;
    Card threeOfDiamonds = Card.THREE_OF_DIAMONDS;

    Assert.IsTrue(twoOfClubs.IsLessThan(threeOfDiamonds));
}
```

เอกสาระ ที่นี่ก็ได้เวลารันเทสต์แล้ว

มันคอมไฟล์ไม่ผ่านใช่ไหม ที่นี่คุณลองเขียนโค้ดระบบ ทำยังไงก็ได้ให้ยูนิตเทสต์นี้ผ่าน ไหนๆ มาดูหน้าตาโค้ดของคุณซิ

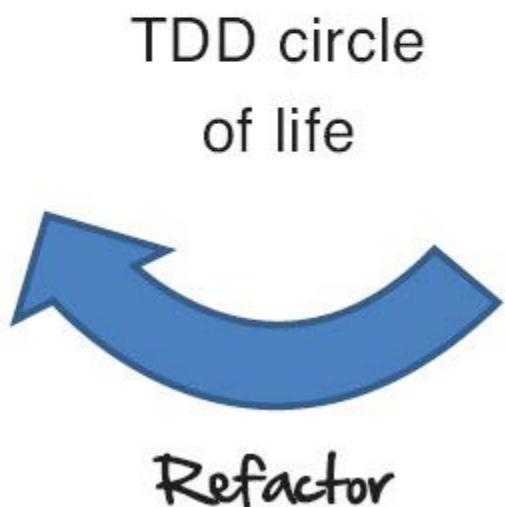
```
public bool IsLessThan(Card newCard)
{
    int thisCardValue = value;
    int newCardValue = newCard.value;
    return thisCardValue < newCardValue;
}
```



ลองรันเทสต์ใหม่ เมื่อรันเทสต์ผ่านแล้ว อธิคักรจะตามคุณว่าคุณอยากรู้ว่าไฟล์เดียวโค้ดตรงไหนไหม สมมติว่าคุณมีนั้น หน้าตาโค้ดของคุณหลังจากปรับโฉมแล้วก็จะเป็นแบบนี้

```
[Test]
public void Compare_value_of_two_card()
{
    Assert.IsTrue(Card.TWO_OF_CLUBS.IsLessThan(Card.THREE_OF_DIAMONDS));
}

public bool IsLessThan(Card newCard)
{
    return value < newCard.value;
}
```



ที่นี่คุณทำที่ดีคือครบถ้วนทั้งวงจรแล้ว คุณจะได้เห็นอธิบายมีความหวังแล้วบอกว่า “คุณทำได้” ที่นี่ก็อย่าลืมกลับไปเขียนยูนิตเทสต์ของคุณเองหลังจากกลับไปที่หน้าจอของคุณล่ะ

## ขั้นตอนไตรัศัย...

คุณอาจจะต้องได้อ่านหนังสือของ Kent Beck เรื่อง “Test Driven Development: By Example” ในหนังสือยังมี Tips and Tricks อีกมากมายที่จะช่วยให้คุณทำที่ดีได้คล่องแคล่วง่ายขึ้น

**ศิษย์:** อาจารย์ ข้างกับที่ดีเหลือเกิน จะเป็นไปได้อย่างไร ที่ข้าจะเขียนยูนิตเทสต์ก่อนจะมีโค้ดระบบเกิดขึ้นมา

**อาจารย์:** เจ้าจะใช้จินตนาการ ให้เหมือนว่ามันมีอยู่สิ

**ศิษย์:** แต่กระบวนการ ข้าก็ยังไม่รู้อยู่ดีว่าข้าจะเทสต์อะไร

**อาจารย์:** ศิษย์เคย อะไว้ก์ได้ที่เจ้าต้องการ

**ศิษย์:** อาจารย์หมายความว่า ให้ข้าเทสต์อะไว้ก์ให้ที่ข้าต้องการ แล้วก็สิ่งที่จะเกิดขึ้นกับโค้ดระบบในจินตนาการของข้าบ้างหรือ

**อาจารย์:** ถูกต้องแล้ว

**ศิษย์:** อาจารย์...ท่านช่วยบอกให้กระจุ่งอีกสักนิดถึงเรื่องมาๆ และจินตนาการเหล่านี้ ว่ามันจะประสบผลสำเร็จได้อย่างไร

**อาจารย์:** นี่นะ แท้จริงแล้ว มันไม่ใช่มาๆ หรือจินตนาการอะไรมันนั้น ที่เจ้าทำคือค้นหาจากยูนิตเทสต์ไปสู่การค้นพบว่าเจ้ากำลังจะทำอะไร ทำเยี่ยมนี้โค้ดระบบที่เจ้าได้จะตรงกับที่เจ้าต้องการมากที่สุด เยี่ยงนี้ยูนิตเทสต์จะทำให้เจ้ามั่นใจ ว่าเจ้าเดินทางถูกทางเหตุนี้ที่ดีจึงถูกกล่าวขานว่ามาช่วยในเรื่องของการออกแบบมากซะยิ่งกว่าการทำเทสต์อะไว้ก์

**ศิษย์:** นั้นที่ดี ก็เป็นเรื่องของการออกแบบ ไม่ใช่เรื่องของการทำเทสต์ อย่างนั้นหรือ

**อาจารย์:** จะว่าอย่างนั้น มันก็กล่าวข้างกันจริงไปหน่อย จนจำไว้ว่าหัวใจของที่ดีคือการทำเทสต์ เพราวยังไงเสีย ที่ดีก็เขามาใช้เทสต์โค้ดระบบของเรา เพื่อดูว่ามันทำงานได้จริงหรือไม่ แต่อย่างไรจะถูกทำที่ดี เจ้าก็เลี่ยงการออกแบบก่อนไม่ได้

**ศิษย์:** ขอบคุณท่านอาจารย์ ข้าคงจะต้องกลับไปพบทวนเรื่องนี้ให้มาก

## ขั้นตอนต่อไป

ที่ผ่านมาคงจะได้เห็นแล้วว่า ยูนิตเทสต์ช่วยในการทำเทสต์ได้อย่างไร รีไฟคเตอร์ช่วยให้คัดสรรหาด้วยความแม่นยำได้ แต่ในเรื่องของการออกแบบและจัดความซับซ้อนได้อย่างไรคิดดีๆ ทุกอย่างถูกรวมมาไว้ในที่เดียวหมดแล้ว

ต่อไปปั๊บถึงคราวจะต้องรู้เกี่ยวกับพลังของ continuous integration แล้ว!

# ตอน 15 Continuous Integration: ทำให้มันพร้อมจะเป็น

## โปรดักซ์ชันเลย



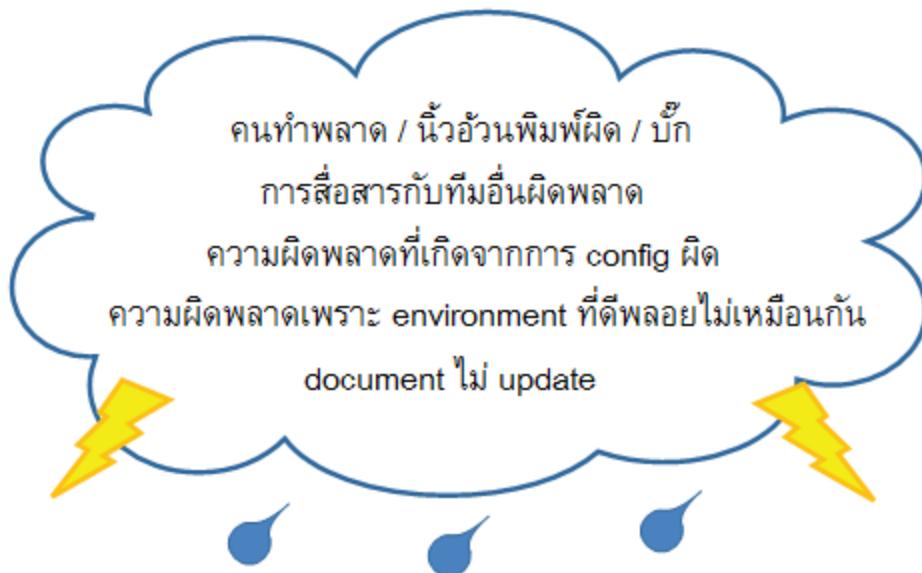
เติร์ยมพร้อมที่จะพบกับสิ่งดีๆ ที่จะมา กับการพัฒนาขั้นไป โปรดักชัน เสมอ กับ เดอะ คุณได้ เรียนรู้ว่า คุณ ควร นำ ส่วน ต่างๆ ของ ซอฟต์แวร์ ของ คุณ มา รวม กัน อย่าง ต่อเนื่อง ได้ อย่าง ไร ถึง จะ ทำ ให้ คุณ ฝ่า บึก ได้ แต่ เนินๆ เพื่อลด ค่าใช้จ่าย ของการ แก้ไข ซอฟต์แวร์ ลง และ สามารถ ดีเพลย์ งาน ได้ อย่าง มั่นใจ และ เมื่อ นัก วิเคราะห์ คุณ คง จะ ต้อง ทำ มัน ได้ ยิ่ง นี้!

### 15.1 ได้เวลา โซฟต์

ข่าวดี ก่อน ได้ เร็ว เดอร์ ของ คุณ กำลัง จำ พาน กอง ทุน ผู้ มี อิทธิพล ใน การ มา ดู ผลิตภัณฑ์ ตัว สำคัญ แบล็ค แจ็ค เวอร์ชัน ล่าสุด ของ คุณ ช่วง วาระ ที่ คือ เค้า กำลัง จจะ มาก ภายใน หนึ่ง ชั่ว โมง! นั่น ก็ หมาย ความ ว่า คุณ มี เวลา น้อย กว่า 60 นาที ที่ จะ ทำ ตัว อย่าง ที่ ทำงาน ได้ ถูก ต้อง และ เอา ชื่น test server เพื่อ เตรียม พร้อม สำหรับ เดโม่ คุณ จะ ทำ ยัง ไง ดี? (แล้ว บ้าน เรียก ล่า)

ก่อน คุณ จะ ตอบ คุณ ควร จะ ใช้ เวลา ชั้ก 2 นาที คิด ถึง สิ่ง นิด พลัด ที่ อาจ เกิด ขึ้น ได้ ใน การ ดี พล อย ซอฟต์แวร์

## สิ่งผิดพลาดที่เกิดขึ้นได้ในการติดตั้งอย่างอัตโนมัติ



ข้อผิดพลาดด้านบนคือสิ่งที่เราต้องการจะจัดทิ้งไป หรืออย่างน้อยก็ควบคุมมัน ด้วยเครื่องมือที่เรียกว่า Continuous Integration (aka CI อย่างว่าซีไอ) เป็นสิ่งที่เราสามารถใช้เมื่อเวลาต้องการจะสร้างวัฒนธรรมการทำงานที่ส่งเสริมให้เกิดการพัฒนาและรับการ build release ขึ้นไปเรื่อยๆ แล้วสามารถตรวจสอบผลิตภัณฑ์เราให้ครบรอบ เวลาไหนก็ได้ ที่ไหนก็ได้ ที่ไหนก็ได้ (เมพ) เพื่อให้เห็นภาพมากขึ้น

มาตรฐานการทดสอบเรื่องต่อไปนี้กันเถอะ

เหตุการณ์ที่ 1: งานเข้าอย่างแรง

หนึ่งชั่วโมง! คุณไม่มีเวลาทำงานนัก คุณตื่นตุ่ม คุณเรียกคนในทีมเข้ามา แล้วก็เริ่มยิงคำถามอย่างกระปือกล:

ใครมี build ล่าสุด?

เครื่อง desktop ใครอยู่ที่มันใจว่าจะไม่เพียง พั้นที่จะใช้งานไม่มีปัญหา?

ใครสามารถรันมันขึ้นมาได้เร็วที่สุด?

เนื่องจากคุณไม่เชื่อใจใคร นอกจากตัวคุณเอง คุณบอกทุกคนว่า เครื่องคุณจะเป็นเครื่องที่ใช้ในการรวมโค้ดสำหรับเดโมนี้ และทุกคนมีเวลา 15 นาทีในการ merge code มาที่ branch ของคุณ พอกคนอื่นๆ เริ่ม merge code เข้ามา ก็เริ่มมีปัญหาเกิดขึ้น Interface ใน class หลักถูกเปลี่ยนไปแล้ว ไฟล์ config ก็ไม่เหมือนเดิม ไฟล์จากระบบเก่าก็โดน refactor ออกไป ไม่มีอยู่แล้ว การรวมการเปลี่ยนแปลงทุกอย่างในทีเดียวเริ่มกลายเป็นฝันร้าย

คุณก็ได้แต่ด่าได้เรตเตอร์คุณในใจ ว่าทำไม่ไว้เวลาเยอะกว่านี้(พระ) แล้วก็บอกคนอื่นๆ ว่าให้ comment ผู้นี้ออก และใส่ hack เข้าไปเพื่อจะได้ยังไม่ต้องรวม code ใหม่ๆเข้ามา

ยังเหลือเวลาอีก 5 นาที คุณก็เริ่มเห็นแสงสว่างริบหรี่—compile ผ่านแล้ว!

แต่แล้ว บุญมีแต่กรรมบัง—นักลงทุนมาถึงก่อนเวลา 5 นาที ไม่เหลือเวลาจะ test แล้ว  
นั่งคาดมันต์ภาร重任ไปพร้อมดีเพลย์ซอฟต์แวร์ไป เปิด app ขึ้นมาจะเดไม่ แล้วมันก็... crash คุณแก้ปัญหาอย่างรวดเร็วแล้วก็เปิด app ขึ้นมาใหม่ เพียงเพื่อได้เห็น splash screen แล้วมันก็ crash อีก  
หลังจากหน้าแรกเลิกน้อย และเห็นว่าเดโมคงไม่เกิดแน่ ไดร์คเตอร์ก็เลยขอๆ app จำลอง (mock-ups) แทน จบช้า

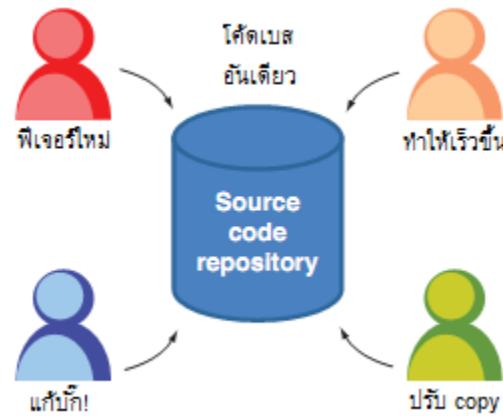
#### เหตุการณ์ที่ 2: งานเข้าสายๆ

เนื่องจากคุณรู้ว่าคุณมีเวลาอีกหนึ่งชั่วโมงเต็มก่อนจะเดไม่ คุณกับอกทีมของคุณไว้ล่วงหน้าว่ากำลังจะมีเดไม่ แล้วคุณก็จะตีใจมาก ถ้าทุกคนจะทำสิ่งที่ทำอยู่ให้เสร็จและเช็คอินโค้ด หลังจากที่งานของทุกคนได้ถูกเช็คอินแล้ว คุณก็เช็คเข้าที่โค้ดล่าสุดของมาร์ค เทส และหลังจากเห็นว่าเทสผ่านหมดแล้ว ก็ส่งต่อไปเทส ซึ่งเป็นการเทสแบบอัตโนมัติ ใช้เวลาแค่ 5 นาทีเท่านั้น  
นักลงทุนมาถึงก่อนเวลา เดไม่ก็ไปได้สวย และหัวหน้าคุณก็ขอบคุณคุณที่คุณสามารถเตรียมการนำเสนอได้ในเวลาอันสั้น  
แล้วก็ส่งของบางอย่างที่คุณต้องการมาตลอดให้คุณ—กุญแจห้องน้ำของผู้บริหาร (ฝ่าย...)  
โอดี้ โอดี้ คุณไม่อยากได้กุญแจห้องน้ำก็ได้อะไร แต่คุณเข้าใจใช่มั้ยว่าผมกำลังพยายามจะบอกอะไร การเตรียมตัวเดไม่หรือส่ง code ขึ้นไปตั้งชั่นไม่จำเป็นต้องเป็นเหตุการณ์ตึงเครียด หนักหนาสาหัส เสมอไป คุณควรต้องการให้ การ build การรวมโค้ด และการ deploy ซอฟต์แวร์ไม่เป็นเหตุการณ์อะไรที่ใหญ่โต และการที่คุณจะทำอย่างนั้นได้ คุณก็ต้องมีกระบวนการ continuous integration ที่ราบรื่น และวัฒนธรรมการพร้อมจะขึ้นโปรดักชันอยู่เสมอ

## 15.2 วัฒนธรรมการพร้อมจะขึ้นโปรดักชันอยู่เสมอ

มีคำพูดนึงใน Extreme Programming ที่ว่า โปรดักชันเริ่มตั้งแต่วันแรกของโปรเจค ตั้งแต่คุณเริ่มเขียนโค้ดบรรทัดแรก คุณก็จะต้องปฏิบัติกับมันเหมือนกับมันอยู่บนโปรดักชัน และหลังจากนั้น ก็ให้เหมือนกับคุณกำลังแก้ไขระบบที่มีคนใช้อยู่จริงแล้ว มันเป็นมุมมองต่อโค้ดที่แตกต่างออกไปอย่างสุดซึ้ง แทนที่จะมองโปรดักชันและการ deploy เป็นเหตุการณ์ไกลๆ ในอนาคต คุณจินตนาการว่าคุณและทีมของคุณอยู่ในโปรดักชันทุกวัน และก็เปลี่ยนแปลงพฤติกรรมให้เหมาะสม ชาวแอจайл์ชอบแนวความคิดของการพร้อมจะขึ้นโปรดักชันอยู่เสมอ เพราะมันเน้นให้เห็นว่า ซอฟต์แวร์จะใช้เวลาอยู่บนโปรดักชันมากกว่าในช่วง development และมันทำให้ทีมเคยชินกับการทำการเปลี่ยนแปลงระบบที่พร้อมจะขึ้นโปรดักชันเสมอ การรักษาวัฒนธรรมการพร้อมจะขึ้นโปรดักชันอยู่เสมอไม่ได้ง่ายหรือได้มาฟรีๆ ถึงมันจะต้องใช้วินัยขั้นสุดยอด และมันก็มีสิ่งล่อใจให้เลื่อนการลงทุนกับโค้ด เพื่อให้มีคุณภาพโปรดักชันออกไป เพราะตารางเวลาอยู่เยอะมาก แต่คนที่ลงทุนกับสิ่งเหล่านี้แต่เนิ่นๆ สามารถทำการเปลี่ยนแปลงโปรดเจคได้อย่างรวดเร็ว พากເคำสามารถ deploy ได้อย่างง่ายดาย เปลี่ยนแปลงระบบได้ปอยๆ และอย่างมีความมั่นใจ และตอบรับความต้องการของลูกค้าได้เร็วกว่าคู่แข่ง และสิ่งที่ทำให้เราทำสิ่งเหล่านี้ได้ ก็คือ continuous integration

### 15.3 อะไรคือ Continuous Integration



Continuous integration คือ การทำการเปลี่ยนแปลงที่ dev ทำกับซอฟต์แวร์ สามารถกันอยู่ตลอดเวลา ตลอดทั้งวัน ลองเปรียบเทียบกับการเขียนหนังสือ จินตนาการว่าคุณและคนเขียนหนังสือร่วมกับคุณกำลังเขียนบทหนึ่งอยู่ด้วยกัน และคุณจะต้องการการเปลี่ยนแปลงของคุณกับเค้า การรวมการเปลี่ยนแปลงไม่เกี่ยวกับคนนั้นไม่ยาก

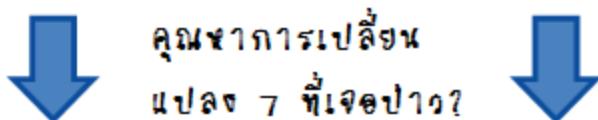
*The brown fox jumps over the lazy dog.*

↓                          ↓  
จ่างนิจเดี้ยง...

*The brown fox jumps over the lazy black dog.*

มันเป็นเฉพาะตอนที่เราไม่เข้ากันมาร่วมกันเป็นระยะเวลานานเท่านั้นแหละ ที่มันจะเป็นปัญหาหนัก

The brown fox jumps over the lazy dog. But then the dog did something utterly amazing! He baked a batch of chocolate chip cookies and proceeded to hand deliver them to everyone he passed on the street. The cats of course saw this, got angry, and decided to counter the dog's good will cookie campaign with a campaign of their own—chocolate cheesecake!



The brown fox jumps over the lazy dog. But then the dog did something utterly amazing! He made a batch of chocolate chip muffins and proceeded to deliver them to everyone he passed on the avenue. The cats, of course saw this, got angry and decided to counter the dog's good will muffin campaign with a campaign of their own—vanilla cheesecake!

การเขียนซอฟต์แวร์เหมือนกัน คุณไม่รู้ว่าการเปลี่ยนแปลงของคุณกับเพื่อนๆ ในทีมของคุณนานเท่าไร การอาดัดมาร่วมกันก็ยกเพิ่มขึ้นเท่านั้น เรามาดูกันว่า เราจะเอาสิ่งนี้ไปใช้จริงได้ยังไง

“รักษาความเร็วไว้”

ครั้งนึง ผมอยู่ใน gerade ซึ่งมีเครื่องมือทดสอบแบบอัตโนมัติที่สุดยอด ที่สามารถบันทึกการทำงาน แล้วนำมา play ในมือถือ มันดีมากจนทุกคนเริ่มใช้มันในการบันทึกทดสอบทั้งหมด และก็เลิกเขียน unit tests ที่รันเร็วกว่าและทดสอบใน level ที่เล็กกว่า มันก็โดยอยู่ระดับนึง แต่หลังจากที่มีทดสอบบันทึกเพิ่มขึ้นเรื่อยๆ build อัตโนมัติของเราระยะๆ ใช้เวลา 10 นาทีน่ารักๆ กล้ายมาเป็นมากกว่า 3 ชั่วโมง สิ่งนี้มันม่าเราในที่สุด หลายๆ คนลิอกจะรัน build เด้าเริ่มจะไม่ค่อยเช็คอินได้ดี และ build พังก็เป็นเรื่องปกติของโปรเจคอย่างได้ทำพลาดเหมือนอย่างพวกเรา ที่ปล่อยให้ build ใช้เวลานานเกินไป คอยสังเกตเวลา build ของคุณไว้ตลอด ให้มันน้อยกว่า 10 นาทีน่าจะเป็นกฎที่ดี โปรเจคเด็กๆ ส่วนใหญ่สามารถทำให้น้อยกว่า 5 นาทีได้

## 15.4 มันทำยังไง?

การที่จะเข็มกระบวนการ continuous integration คุณต้องการสิ่งเหล่านี้:

- repository ของ source code

- กระบวนการในการเช็คอินโดยตัว
- build ที่รันอัตโนมัติ
- ความตื่นใจที่จะทำงานแบบทีละน้อยๆ

repository ของ source code เป็นที่เก็บและทำเวอร์ชันให้กับโค้ด ที่นี่คือที่ที่ทีม dev ของคุณ “เช็ค” ได้เข้าไป มันเป็นที่ที่ได้มารวมกัน และเป็นต้นฉบับของโค้ดของคุณ repository แบบ open source อย่าง Git หรือ Subversion จะเป็นเพื่อนช่วยคุณได้ตรงนี้ อย่างไรก็ตามกรุณาอย่าไปใช้ repository ที่มีการ lock ไฟล์แบบมองโลกในแง่ร้าย (หมายถึงขันที่ยอมให้ dev เพียงคนเดียวแก้ไฟล์ใดๆ ในช่วงเวลาหนึ่งๆ) มันจะทำให้ dev คุณหงุดหงิด ทำให้ทีมทำงานช้าลง และยังไม่ทำให้เกิดการที่คนในทีมจะรู้สึกเป็นเจ้าของโค้ดร่วมกันด้วย กระบวนการเช็คอินที่ดีนั้นน่าสนใจกว่า เรามาดูว่าทีมแอลจีล็อตทัวไปเด็กทำกันยังไง

## 15.5 สร้างกระบวนการเช็คอิน

กระบวนการเช็คอินทัวไปสำหรับ dev ที่ทำงานในทีมที่เป็นแอลจีล็อตมักจะเป็นอย่างนี้:



- ເອົາໂຄດສ່າສຸດ จาก repository ก่อนที่คุณจะเริ่มงานขึ้นใหม่ คุณจะต้องมั่นใจว่าคุณมีโค้ดใหม่ล่าสุดจาก repository ตรงนี้คือตรงที่คุณเช็คเอาท์โค้ดล่าสุด และเริ่มต้นแบบคลีนๆ
- ທຳການປັບປຸງແປງ แล้วคุณก็ทำงานของคุณ คุณเพิ่มความสามารถใหม่ แก้บັບ หรือทำອະໄກດາตามที่จำเป็น
- ຮັນເທສ เพื่อให้มั่นใจว่า สิ่งที่คุณປັບປຸງແປງไป จะไม่ทำให้ลิสต์อื่นพังในโค้ดทุกๆ ส่วนคุณรันເທສเพื่อให้เห็นว่าทุกอย่างยังผ่านอยู่
- ເຊື້ອນມີການປັບປຸງແປງເພີ່ມເລື່ອນີ້ เมื่อคุณมั่นใจว่า การປັບປຸງແປງของคุณจะทำงานได้ແລ້ວ คุณก็ update มาจาก repository อີກຮັງ เพื่อว่าມີຄວາມທຳອະໄໄປ ในระหว่างที่คุณทำการປັບປຸງແປງของคุณอยู่
- ຮັນເທສອີກຮັງ แล้วคุณก็ຮັນເທສອີກຮັງนึง เพื่อทำให้มั่นใจว่าມີງານຍັງເວົ້າມີກັບການປັບປຸງແປງໃໝ່ ที่ເຂົ້າມາໃນระหว่างคุณทำงานอยู่

6. เช็คอิน ระบบทำงานได้ ทุกอย่าง build ผ่าน เทศรัตน์ผ่าน เจ้มีโค้ดล่าสุด ก็เช็คอินได้อย่างปลอดภัยแล้ว  
นอกเหนือจากการกระบวนการเช็คอินแล้ว มีสิ่ง ควรทำ และ ไม่ควรทำสองสามอย่าง เกี่ยวกับความประพฤติต่อ build ที่ดี

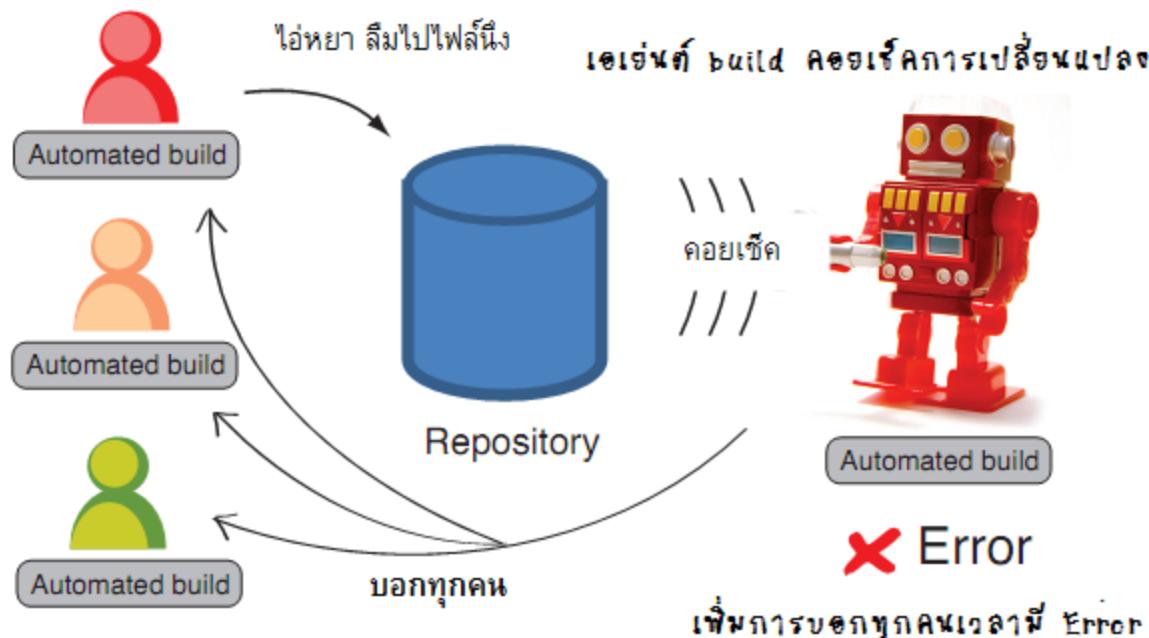


ท้ายที่สุดแล้ว มันคือการให้ความเคารพกับ build โดยดูว่ามันยังรันดีอยู่ และช่วยเหลือกันเวลาที่มันพัง (ซึ่งเกิดขึ้นได้บ้าง)

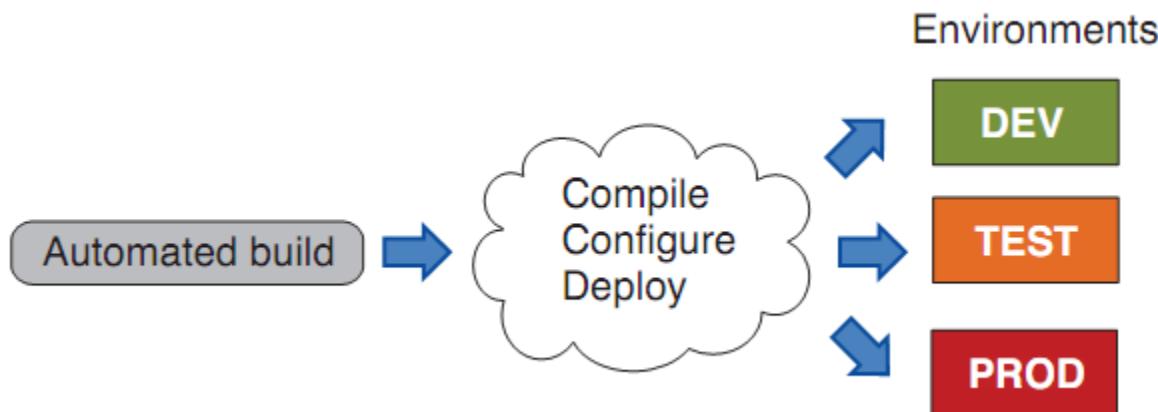
## 15.6 สร้าง build อัตโนมัติ

สเต็ปด้านไป คือ การสร้าง build อัตโนมัติ นี้เป็นแกนหลักของกระบวนการ continuous integration ของทีมคุณเลย build อัตโนมัติที่ดี จะ compile ได้ดี รันเทส พูดง่ายๆ ก็คือทำอะไรก็ตามที่จะต้องถูกทำเป็นประจำ ในกระบวนการ build ของโปรดเจคคุณ dev วันมันตลอดเวลา โดยเป็นส่วนหนึ่งของชีวิตประจำวันของ TDD และเอเย่นต์ build (อย่าง CruiseControl) ก็ใช้มันเพื่อรัน build เมื่อไหร่ก็ตามที่มันสามารถตรวจจับได้ว่ามีการเปลี่ยนแปลงเกิดขึ้นใน repository

## Developers



build ອັດໃນມົດຍັງສາມາດຖືຈະ deploy ທຸອົພົວໄປຢັງໂປຣດັກຂັ້ນຍ່າງອັດໃນມົດໄດ້ດ້ວຍ ຊື່ຈະສາມາດຕັດຄວາມຜິດພາດທີ່ເກີດຈາກມຸນຫຼົງໄປໄດ້ອີກ

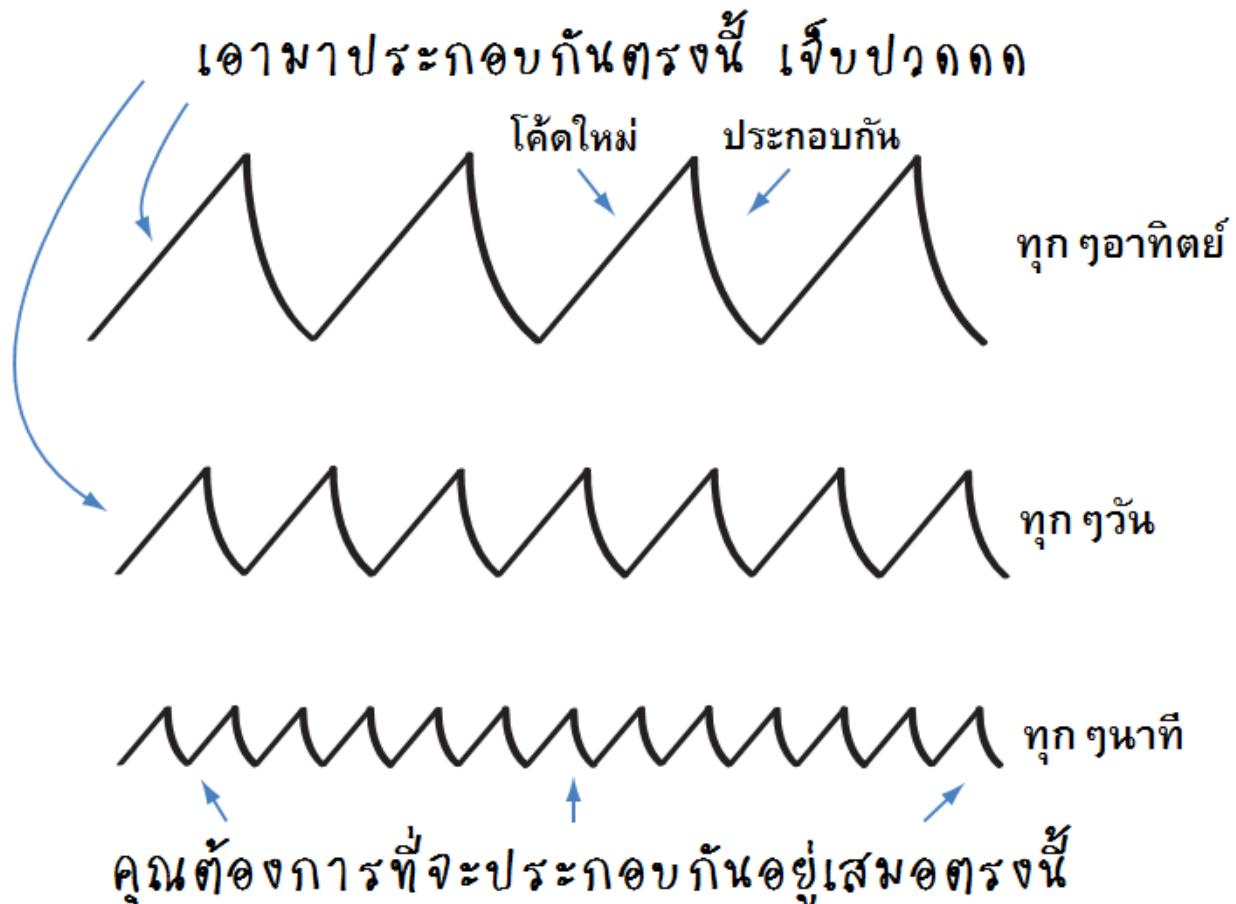


ຫວ່າໃຈລັກຂອງ build ໄດ້ກົດຕືອງ ຄວາມອັດໃນມົດ—ນີ້ຄູນເກີຍຂັ້ອນນ້ອຍເທິ່ງໄວ້ຢຶ່ງດີ ຄູນເຢັງຄວາມຕ້ອງຮັກຊາວເລາ build ຂອງຄູນໃໝ່ເງົາວູ້ເສັມອ ເພວະຄູນແລະທີມຂອງຄູນຈະວັນມັນອຸ່ນປ່ອຍໆ ພາຍາຄວັ້ງຕ່ວນວັນ (ນ້ອຍກວ່າ 10 ນາທີ ນໍາຈະເປັນກູກທີ່ດີ)

ການຊາຍໄປແກ່ນໃໝ່ ແລ້ວໃນເປົ້ານີ້ໄປແມ່ນ framework ສໍາຮັບ build ອັດໃນມົດເປັນຂອງຕ້ວເອງ (Ant ສໍາຮັບ Java, NAnt ຮີ້ອ MS-Build ສໍາຮັບ .NET, ແລ້ວ rake ສໍາຮັບ Rails) ດ້ວຍພາກພາກທີ່ຄູນໃໝ່ມີ ຄູນກີສາມາດສ້າງເຂົ້າມາເອງໄດ້ໂດຍໃ້ໄຟຟິ້ນ bat ຂອງ DOS ຮີ້ອ ສະລິບປົກທີ່ຂອງ Unix ແຕ່ລື່ງແນ່ຈະມີກະບວນກາເຫັນວັນທີ່ build ອັດໃນມົດທີ່ດີຍິງໄລ ສິ່ງທີ່ທໍາໄໝມັນເວົ້າຈົງກົງທີ່ຄູນ ຄວາມເຕັມໃຈທີ່ຈະທຳມະນຸດນັ້ນ

## 15.7 ทำงานทีลับน้อยๆ

เขียนเดี่ยวกับ TDD การเข้าโค้ดมาร่วมกันทีลับน้อยๆจะง่ายกว่า



บ่อยมาก ที่ทีมมักจะไม่เอางานมาร่วมกันเป็นวันๆหรือเป็นอาทิตย์ฯ—มันนานเกินไป คุณต้องการที่จะรวมโค้ดของคุณทุกๆ 10 ถึง 15 นาทีหรืออะไรประมาณนี้ (อย่างน้อยที่สุดทุกชั่วโมง) ไม่ต้องเครียดไป ถ้าคุณไม่สามารถจะเข้าร่วมได้บ่อยขนาดนั้น แค่คุณต้องเข้าใจว่า ยิ่งคุณทำบ่อยแค่ไหน มันก็จะง่ายมากขึ้นเท่านั้น ดังนั้น merge โค้ดของคุณเร็วๆและบ่อยๆ เพื่อเลี้ยงความเจ็บปวดของกระบวนการโค้ดเยอะฯแล้วจะไปเรียนรู้เพิ่มเติมได้ที่ไหน?

Continuous integration ได้กลายมาเป็นวิธีปฏิบัติที่ใช้กันแพร่หลาย ที่คุณสามารถหาทุกสิ่งที่คุณต้องการผ่านเว็บได้ Wikipedia มีบทสรุปที่ดี เกี่ยวกับวิธีปฏิบัตินี้ และคุณก็ยังสามารถหาบทความแรกๆเกี่ยวกับ continuous integration ได้ที่เก็บของ Martin Fowler



## ปรมาจารย์ กับ นักบุญแห่งเยอรมانيا

**ลูกศิษย์:** ท่านอาจารย์ เราไม่สามารถทำให้ทุกอย่างพร้อมจะขึ้นโปรดตั้งขึ้นได้ใน iteration แรกແນ່ງฯ ท่านหมายความว่าอย่างไรกัน  
ແນ່ เมื่อท่านพูดถึง การพร้อมจะขึ้นโปรดตั้งขึ้น

**อาจารย์:** การพร้อมจะขึ้นโปรดตั้งขึ้นมันเป็นทัศนคติ เมื่อเจ้าเขียนโค้ดที่พร้อมจะขึ้นโปรดตั้งขึ้น เจ้าเทส และรวมซอฟต์แวร์ของเจ้า  
เข้าด้วยกันวนนี้ เมื่อไรที่เจ้าเห็นบีก เจ้าก็จะแก้มันเดียวันนี้ เจ้าไม่กวนมันเข้าไปชื่อนี้ให้พรอม และจนตอนการว่าจะไปตัดการกับ  
มันในซักวันข้างหน้า เจ้ามีทัศนคติที่ว่า ซอฟต์แวร์ต้องทำงานได้วันนี้ ไม่ใช้ซักวันในอนาคต ใช่ เจ้าอาจจะไม่ได้มีไฟเซอร์เจําๆ ทุก  
อย่างที่เจ้าอยากได้ และใช่ เจ้าอาจจะเลือกที่จะไม่ deploy จนกว่าจะมีไฟเซอร์มากกว่านี้ แต่การที่เจ้ามีตัวเลือกว่าเจ้า deploy ได้  
และการที่เจ้ารู้ว่าซอฟต์แวร์เจ้าทำงานได้อย่างถูกต้อง คือการที่เจ้ายอมรับว่า ซอฟต์แวร์ของเจ้าจะใช้ชีวิตส่วนมากอยู่ในโปรดตั้งขึ้น  
(ไม่ใช่ในการพัฒนา) และทำให้เจ้าเคยขึ้นกับการเปลี่ยนแปลงสิ่งที่อยู่ในระบบโปรดตั้งขึ้นแล้ว

**ลูกศิษย์:** แล้วถ้าข้าไม่สามารถ build ระบบทั้งหมดได้ เพราะโปรดตั้งของข้าเป็นเพียงส่วนเล็กๆ ของระบบที่ใหญ่กว่าเท่านั้นละ  
ท่าน

**อาจารย์:** เจ้าก็จะ build เทส และ deploy ในส่วนที่เจ้าทำได้สิ สักวันนึง เจ้าก็จะต้องรวมส่วนของเจ้าเข้ากับส่วนอื่นๆ เจ้าจะทำให้  
ตัวเจ้ามันใจว่า ส่วนของเจ้านั้นพร้อมอยู่เสมอ เพื่อที่เจ้าจะได้ทำการแก้ไขได้ทุกเมื่อที่จำเป็น อย่าให้ความเป็นจริงที่ส่วนของเจ้า  
เป็นเพียงส่วนเล็กๆ ส่วนหนึ่ง หมายความเจ้าจากการทำ build อัตโนมัติ และรวมซอฟต์แวร์ของเจ้าอยู่เสมอ

จบแล้ว!

คุณได้เห็นมันไปหมดแล้ว ความงดงามของวิธีปฏิบัติของวิศวกรรมซอฟต์แวร์ในแบบแอจайл์:

- Unit testing—เพื่อพิสูจน์ว่าสิ่งที่เจ้าสร้างมาทำงานได้อย่างถูกต้อง
- Refactoring—ศิลปะของการทำให้โค้ดชัดชื่อน้อยลง สะอาด และอ่านง่าย
- Test-driven development (TDD)—สำหรับการออกแบบ และจัดการกับความซับซ้อน
- Continuous integration—นำทุกสิ่งทุกอย่างมารวมกันอยู่เสมอ และรักษาสภาพของความพร้อมที่จะขึ้นโปรดตั้งขึ้น

ถ้าไม่มีสิ่งเหล่านี้แล้ว โอกาสที่ไปรเเจคแอจайл์ของเรายังคงประสบความสำเร็จคงมีน้อยมาก และเราอาจจะกลับไปอยู่ในยุคหินที่ “ได้  
แล้วก็มาแก้”

## 15.8 จะไปไหนต่อดี?

ยินดีด้วย! ตอนนี้ คุณมีความรู้และทักษะเป็นอาชญาติดตัวที่จะเริ่มต้น วางแผน และเริ่มทำไปเจ้าจ้าวของคุณเองได้แล้ว คุณจะไปไหนต่อดีขึ้นอยู่กับคุณ ถ้าคุณกำลังจะเริ่มไปเจ้าจ้าว คุณอาจจะอยากรีเมิร์ตันด้วยชั้นเริ่มต้น (ตอนที่ 3 จะช่วยทุกคนมาขึ้นรถได้อย่างไร ที่หน้า 48) ชวนทุกคนขึ้นรถไปด้วยกัน และไปยังทางที่ถูกต้อง ด้วยการตามคำダメยากๆ ตั้งแต่เริ่มไปเจ้าจ้าวหรือ ถ้าคุณกำลังอยู่กลางไปเจ้าจ้าว (และแผนของคุณก็พลาดมากอย่างเห็นได้ชัด) บางที่คุณอาจจะเริ่มต้นใหม่ ด้วยการจัดไวร์คซ์ ข่าวบล๊อก หรือ จัดไวร์คซ์ข่าวบล๊อก (ที่หัวข้อ 6.4 จะจัดไวร์คซ์ข่าวบล๊อกที่ได้อย่างไร ที่หน้า 108) เลือกสตอรี่ที่สำคัญมากๆ มาไว้ขั้น และดูว่า คุณจะสามารถทำมันให้เสร็จที่ลับนิดที่ลับหน่อยในทุกๆ อาทิตย์ได้หรือไม่ แล้วก็สร้างแผนใหม่จากตรงนั้น หรือ ถ้าคุณมีปัญหาทางด้านวิศวกรรม คุณอาจจะเริ่มด้วยการดูวิธีปฏิทัติทางด้านวิศวกรรมของคุณ ดูว่าคุณไม่ได้ทำเทสแบบ กะหรรษา และคุณจ่ายหนี้ทางเทคนิคอยู่เสมอ เราไม่ได้มีแผนที่ คุณจะต้องค้นหาเขาเองว่าอะไรที่ได้ที่สุดสำหรับคุณและไปเจ้าจ้าวของคุณ แต่คุณต้องเข้าใจว่า คุณมีเครื่องมือ ต่างๆ และคอมพิวเตอร์ที่ให้ผลลัพธ์ที่ดีที่สุด แล้วด้วยซ้ำว่าจะต้องทำอะไร อย่างไรที่หยุดคุณอยู่ล่ะ?

ออกไปเริ่มต้นทำมันได้แล้ว!

คำสั่งเสียสุดท้าย



ทุกอย่างขึ้นอยู่กับการเลือก

ไม่มีใครหยุดคุณจากการทำซอฟต์แวร์คุณภาพดีได้ ไม่มีใครหยุดคุณจากการพูดอย่างจริงใจกับลูกค้า เกี่ยวกับสถานภาพของไปเจ้าจ้าวและสิ่งที่ต้องทำได้

อย่าเข้าใจผิดคิด—สิ่งเหล่านี้ไม่ง่าย เรามีดีตีเป็นทศวรรษและสิ่งต่างๆ ที่จะต่อต้านคุณ แต่ท้ายที่สุดแล้ว คุณต้องเข้าใจว่า การที่คุณจะทำงานยังไง และคุณภาพของงานของคุณจะเป็นยังไง ขึ้นอยู่กับคุณเท่านั้น

คุณไม่ต้องซักชวน

คุณไม่ต้องบอกคนอื่นว่าควรทำยังไง

แต่คุณนำด้วยการทำให้ดู ยอมรับว่าคนอื่นๆ อาจจะไม่ร่วมด้วยในบางที่ และทำในสิ่งที่จำเป็นต้องทำ ข้าว และอีกอย่างนึง

ไม่ต้องกังวลกับการเป็นแอจใจล์

คำถามนี้ที่ได้ยินตลอดเวลาที่มีทีมเริ่มทำแอจใจล์ใหม่ๆ ก็คือ “เราถึงรึยัง? เราเป็นแอจใจล์รึยัง?”

และมันก็เป็นคำถามที่คุณมีสิทธิจะถาม เมื่อกันกับเวลาคุณเริ่มทำอะไรใหม่ๆ ครั้งแรก คุณก็คงอยากรู้ว่าคุณเป็นยังไงบ้าง และคุณกำลังทำตามในหนังสืออย่างถูกต้องหรือเปล่า และนั่นมันก็โโค แค่คุณต้องเข้าใจว่า มันไม่ได้มีหนังสืออะไรหรอก—ไม่ใช่เล่นนี้ หรือเล่นไหนๆ มันไม่ได้มีรายการที่ต้องถูกเข้าค่าว่าทำแล้วในแอจใจล์ ที่ผ่านหรือใครสามารถจะให้คุณได้ ที่จะบอกคุณได้ว่าคุณเป็นแอจใจล์รึยังมันเป็นการเดินทาง ไม่ใช่จุดจบ คุณจะไม่มีวันถึงและอย่าลืม มันไม่ใช่เรื่องของการ “เป็น” แอจใจล์ มันเป็นเรื่องของการทำผลภัยที่ยอดเยี่ยม และให้บริการระดับโลกกับลูกค้าของคุณ ผู้พูดได้อย่างเดียวว่า ถ้าคุณคิดว่าคุณทำมันสำเร็จแล้ว และคุณมีคำตอบทุกอย่าง คุณได้หยุดเป็นแอจใจล์แล้ว ดังนั้น อย่ามีดิติกับวิธีปฏิบัติ นำสิ่งต่างๆ จากหนังสือเล่นนี้ไป แล้วทำให้มันเหมาะสมกับสถานการณ์และบริบทของคุณ และเมื่อไหร่ก็ตามที่คุณสงสัยว่า คุณกำลังทำสิ่งต่างๆ ใน “แบบแอจใจล์” อญรีเปล่า ให้ถามคำถามสองคำถามนี้แทน:

- เรากำลังลงงานที่มีคุณค่าอยู่ทุกๆ อาชีวกรรมรึเปล่า?
- เรายพยายามอย่างหนักที่จะพัฒนาตัวเองให้ดีขึ้นอยู่รึเปล่า?

ถ้าคำตอบของคุณคือ ใช่ ทั้งสองคำถามแล้วล่ะก็ คุณเป็นแอจใจล์แล้ว

# Appedix A วิธีสื่อการของเรา

## มาแปล Agile Samurai กันนะ

ริเริ่มโดย @roofimon บ.ก. เกรียงเพชร

### แบบงาน

Part	Name	p	By	Note
1	Introducing Agile	32p	@roofimon รูป	@roofimon อีกสองวันน่าจะเสร็จทัน Barcamp Bangkok พอดี :) @roofimon ลองตัวเสร็จแล้ว
2	Agile Project Inception	46p	@sinapam แป้ม	@sinapam เริ่มแผ่แล้ว - -" @roofimon ฟิตมากผิดแคบเห็น มีพระเอกใหม่มาช่วยบท 5 ครับ <u>byrdbird</u> จัดหนัก
3	Agile Project Planning	67p	@kluak110 ป้อม	@kluak110 เริ่มแล้ว อย่างช้าๆ @roofimon เสร็จแล้วเดี้ยวนะไปช่วยครับ ตั้งหกสิบหน้า <u>@giornos</u> ป้อม ผนขอตัดบท 7 ไปให้ เก้นะคนเค้าแรงส์
4	Agile Project Execution	42p	@chonnikan นิ ผู้สึกน้องเก๊ จะสน ของน้องนิน เพราะ ได้ข่าวว่า นิ มาอยู่ ==” @juacompe ผู้สร้างภาพ	@juacompe: สร้างรูปให้ 3 รูป เนื้ออยแล้ว พรุ่งนี้มาทำต่อ ไป เล่น angry bird ก่อ...
				@roofimon jua ติด angry bird วะ ทำรูปเสร็จหมดแล้ว รอ คนแปลเนี่ย :)
				@natty อีก 5 หน้าเท่านั้นๆๆๆ บทที่ 9 ก็จะเสร็จแล้วววว (แบบไม่มีรูปนะ แต่ไม่ต้องห่วง หาแรงงานมาทำรูปได้แน่นะ) @chonnikan มาแล้วๆ เดี้ยวนนี้จะเริ่มแปลบทที่ 10 ค่ะ เดียวลองดู speed ก่อน ภาษาไทยไม่แข็งแรง อิอิ @nattyait เสร็จหมดแล้ว บท 9 และ 11 แต่ไม่มีรูปก็หาๆ รอ

				คนทำรุปเนี่ย :) @เดียวไปทำให้ จ้าไปไหนเนี่ย
5a	Creating Agile Software (c12-13)	22p	@siriwat ช้อ	@siriwat เพิงเริ่มครับบทที่ 12 ยกเหมือนกันนะเนี่ย ที่จะให้อ่านแล้วเรื่องสื่อความได้เหมือนกับที่ผู้แต่งต้องการ
5b	Creating Agile Software (c14-15)	22p	@chompoonut	@chompoonutเริ่มแล้วครับ มี deadline มั้ยเนี่ย หรือว่าสายไปจะแล้ว @roofimon ไม่มีครับ จดไปเรื่อยๆ

### หลักการ

1. จงใส่ศัพท์ เกรียง และ DRAMATIC ลงไปบ้างเพื่อความบันเทิง
2. แยกไฟล์กันแปลเพื่อความสะดวก ไม่งั้นจะยากไปใหญ่ ตั้งชื่อประมาณว่า “Part 1 - Introducing Agile”

# Appendix B ทีม

**AeY Chompoonut** สาว(แก)ลูกน้ำเด็ม เกิดที่ตราด เล็กๆเรียนที่จันทบุรี มาโตอิกที่ชลบุรี ได้ทำงานดีๆที่กรุงเทพ แต่ไม่ชอบเที่ยวทะเลแแกบ้าน ทำงานบริษัทซอฟต์แวร์เต็มไม่ได้เขียนโค้ด โดนให้ใจใหญ่มาแปลงสีอ...เอย.. พี่จะเอาสาระจริงจังหรือเปล่าค่ะ เช่นไได้แต่งให้ใหม่ :)

**@byrd\_bird** ทำงานพนักงานบริษัทเอกชน เป็น Corporate Ladder ที่ไม่ได้มีเป้าหมายที่จะได้บันไดไปให้สูง แต่เรื่องในประชญาของการดำเนินธุรกิจและแบ่งคิดที่ว่าผลย่อมเป็นไปตามเหตุ ไม่ว่าเมัวจะสีอะไรก็ตามวัจบหูได้ก็คือจับหูได้ สรุสุดของศิลปะการต่อสู้คือทำยังไงให้คู่ต่อสู้ของเราง่ายแพ้ย่างปลดภัยและไม่บาดเจ็บ ถ้าเรามองหา Win-Win Solution ได้ก่อนย่อมเจอทางออกที่ดีที่สุด แนะนำว่ามั่นคงไม่มาทางออกที่ดีที่สุดเสมอไป แต่จะมีที่ดีที่สุดด้วยเงื่อนไข ณ เวลานั้นๆเสมอ คือจริงๆแล้วมองอย่างจะสื่อว่าตัดจาก agile มันอาจจะมีอะไรที่ดีกว่า ตอนนี้เรารอยู่ในยุค Empirical ที่เราไม่สามารถหาทฤษฎีมาอ้างอิงการทำ Software Industry ได้ดี Process เป็นเพียงสิ่งที่เป็นกรอบให้คนทำงานได้สะવากวดเร็วและทำซ้ำได้ วันนึงที่ Maturity เราดีกว่าวนี้อาจจะมีอะไรที่ไม่ต้องคิดมากเหมือนเราผลิตรถยนต์ได้ครับ

**@chonnikan** อกหักมาจากบริษัท software ยักษ์ใหญ่ ที่รักฯก็ได้แต่ค่อยหลังลงน้ำตก เลยมาซบอกสำนักชานุรักษ์เพื่อฝึกวิทยาหยทธให้แก่รุ่นก้าว มีความไฟแรงว่าจะ apply และเผยแพร่ Agile methodology ไปสู่การอื่นนอกจาก IT เพราะเห็นว่ามีดีเลยอย่างแพร่

**@giornos** Giorno(Ita.)=วัน Add(Eng.)=เพิ่ม GiornoAdd=ขอวันเพิ่มนหน่อยครับ ทำงานไม่ทัน สิ่งที่ Programmer ต้องการกันถูกคน Programmer เป็นงานที่เป็นทั้งศาสตร์และศิลป์ ถ้างานไม่ถึง Deadline สมองไม่แหล่น เขายาก็ทำให้เรา Art เหมือนเดิมแต่เป็น Art ที่หล่อมาจากการทำงาน ให้ไปในแนวทางแยกใจลีที่ลับนิดๆ เพื่อที่จะได้กลับบ้านเร็วขึ้นๆอิกที่ลับหน่อยๆ....

**@gomora77** ไม่เคยคิดว่าจะได้มาทำงานสายคอม แต่ก็เบลอๆทั้งเรียน ทั้งทำงานได้หลายปี ในส่วน Software Development ของบริษัทผู้ให้บริการมือถือแสน"ดี" ได้พบว่าหนังสือแอร์ใจลีค่อนข้างกว่าที่คิด ในตอนที่มาแปลง Agile Samurai นี่เอง ขึ้นต่อไปจะเป็นการลองปรับการทำงาน ให้ไปในแนวทางแยกใจลีที่ลับนิดๆ เพื่อที่จะได้กลับบ้านเร็วขึ้นๆอิกที่ลับหน่อยๆ....

**@juacompe** เดิมที่เขาเคยเป็นนินจา(ว่า)อยู่หมู่บ้านนาริสา ได้เด้งจากนินจาฝึกหัดจนขึ้นมาเป็นเจ้า top gun ก่อนจะได้รับแต่งตั้งจากโยค่าจะให้เป็น site admin 2 ปีก่อนเด็กได้สมัผัสกับ Hyper-Productivity (H-P) ในทำงานบนเส้นทาง dynamic language แบบเว็บฯ เขียนมท...วิชานินจาทั้งหมด ผันตัวมาเป็นชามูโรเนจรอร์นเรื่อตามหา H-P แม้หลายคนจะหาว่าเข้าบ้าเข้ากึ้งตั้งหน้าตั้งตามนั่นต่อไปด้วยหัวใจบุชิโด

**@kluak110** ทำ software development มาหลายปี หลายอย่าง หลายตำแหน่ง หันหลังให้ waterfall มาพักใหญ่ ไม่คิดจะกลับไปอีกแล้วชาตินี้ สงสารทุกคนที่ยังคงปลักอยู่กับ waterfall และอยากทำให้พากเข้ารู้ว่าชีวิตเข้าจะดีขึ้นได้ด้วย agile ปัจจุบันเป็นชานมไว้พ่อลูกก่อนอยู่บริษัท software ข้ามชาติขนาดใหญ่ที่ agile พอกเป็นที่รู้จักในวงแคบแต่ไม่ถึงกับยอมรับในวงกว้าง แบบใช้ตำแหน่งหน้าที่การงานคงอยู่แล้วแต่ agile ให้กับรอบข้างทุกครั้งที่มีโอกาส

**@nattyait** ชื่อเล่นว่า เก เซี่ยฟรังว่า nattyตามหาได้ແລວ່າ สถานที่ที่มี Agile หรือ Roofimon ได้รับความรู้ Agile อย่างเป็นทางการ มาจาก AIT อดีตเคยทำงานในแผนก IT ของบริษัทน้ำดำแห่งหนึ่งและพยายามเอา Agile ไปใช้แต่ไม่สำเร็จปัจจุบันจึงออกมาราดตามหาผู้สนใจองค์กรที่ Agile และพัฒนาซอฟต์แวร์ด้วย Python และรับงานสอนเป็นอาจารย์พิเศษด้านคอมพิวเตอร์ (เท่าที่มีคนด้าง -\_-) มีความฝันอยากอยู่ในองค์กร Agile ระดับโลกร..ก \*0\*

มันใจว่า เป็นหนึ่งใน Agile evangelists หญิง ที่มีเพียง 2 คนในประเทศไทย :D ใช้เวลาว่างได้มีสาระเป็นครั้งแรกในชีวิต โดยการแปลงหนังสือ Agile Samurai และจะยังคงดำเนินชีวิตเกรียนๆ อยู่ใน เกรียนเพรสต่อไปเรื่อยๆ แล้วเจอกันนะ ชาวเกรียน

**@roofimon** เกรียนลูกสอง ที่ยังเชื่อเสมอว่างานพัฒนาซอฟต์แวร์เป็นงานที่มีเสน่ห์ต้องใช้คนที่มีพื้นฐานทางความคิดที่ดีเข้ามาทำถึงจะได้งานที่ดีและมีคุณภาพ

**@sinapam** แอจใจล้ำชามไว้ขันอยู่ในสายเลือด คิดว่าแอจใจล์ จริงๆคือ ตอนเข้าปีในการใช้ชีวิต ไม่ใช่แค่ process ในการพัฒนาซอฟต์แวร์ ยึดหลัก ทุกอย่างในโลกมี เป้าหมาย กับ วิธีการ เป้าหมายต้องชัดเจน วิธีการมาเลือกเอา เช่นว่า ถ้าแยกสองอย่างนี้ไม่ออกชีวิตจะคร่าม่ามว่ากัน

**Siriwat Jithunsa** ชามไว้พ่อลูกสอง ทำ software development มานานอยู่ ได้รู้จักแอจล์มาประมาณ 4 ปี เริ่มจาก XP ตอนนี้ทำงานอยู่บริษัท Software ข้ามชาติ ทำไปทำมา ก็คิดว่า ก็อย่าไปติดยึดกับวิธีการงานนัก อะไรมีคิดว่า work กับเรา กับทีม น่าลอง ก็ทำเลย