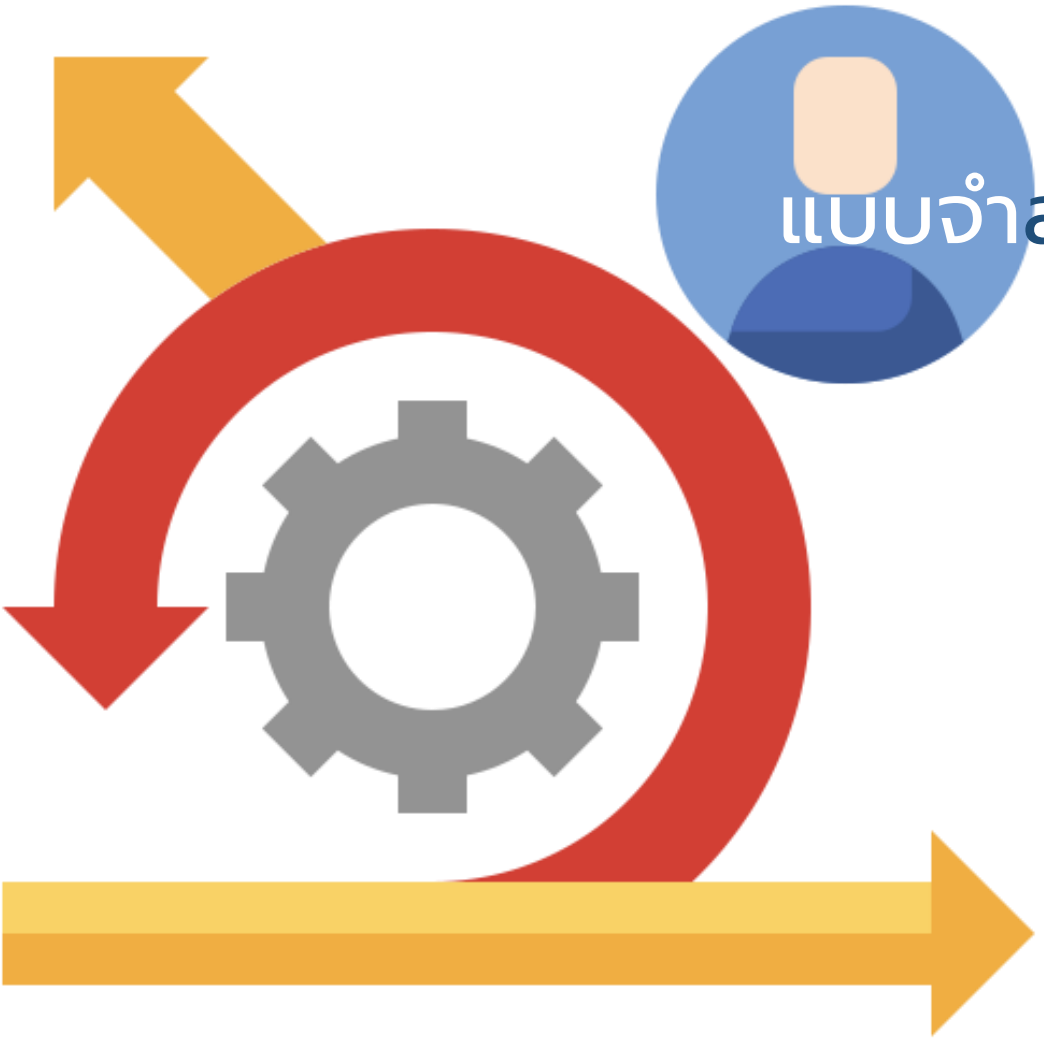
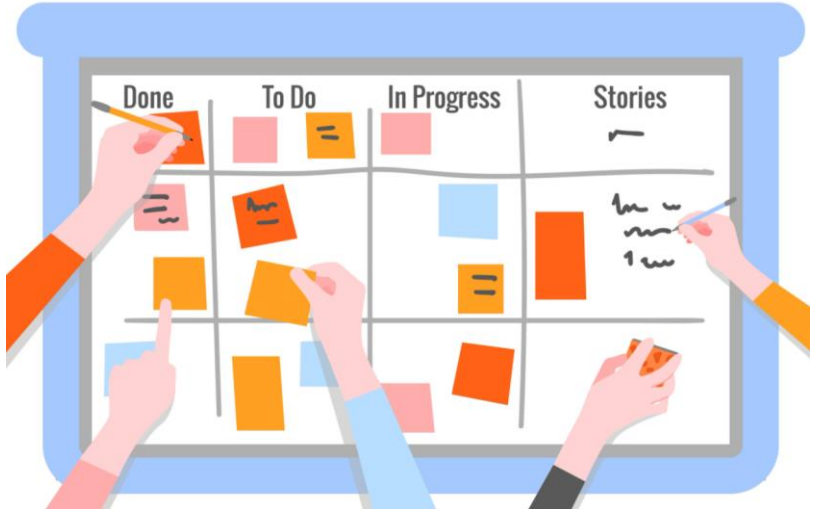


บทที่ 3
แบบจำลองการพัฒนาซอฟต์แวร์แบบเอจายล์ (Agile Software Process Model)

วิชา วิศวกรรมซอฟต์แวร์ (04-06-322)

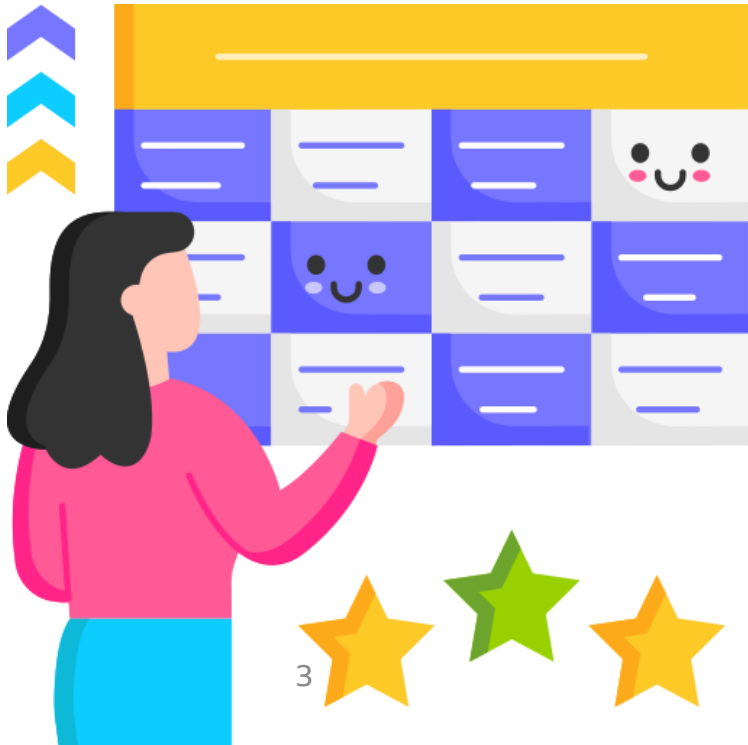
วัตถุประสงค์การเรียนรู้

- เพื่อให้ผู้เรียนมีความรู้ความเข้าใจเกี่ยวกับแนวคิดกระบวนการผลิตซอฟต์แวร์แบบเอจายล์
- เพื่อให้ผู้เรียนสามารถรู้และเข้าใจแนวทางการประยุกต์ใช้กระบวนการผลิตซอฟต์แวร์แบบเอจายล์ในการผลิตซอฟต์แวร์



หัวข้อ (Agenda)

- บทนำ (Overview)
- เอจายล์เมทธอด (Agile methods)
- แบบจำลองกระบวนการพัฒนาแบบเอจายล์
  - แบบจำลอง Extreme Programming (XP)
  - แบบจำลองสครัม (SCRUM)
- สรุป (Summary)



บทนำ (Overview)

ซอฟต์แวร์ (Software)

- ส่วนหนึ่งของการดำเนินการธุรกิจ
- ซอฟต์แวร์รุ่นใหม่ได้รับการพัฒนาอย่างรวดเร็ว ค้นพบโอกาสใหม่ ๆ ตลอดจนตอบสนองต่อแรงกดดันจากการแข่งขัน

กระบวนการพัฒนาซอฟต์แวร์อย่างรวดเร็ว (Rapid software development processes)

- สามารถรองรับความต้องการที่เปลี่ยนแปลงในช่วงเวลาใดๆ
- ออกแบบมาเพื่อผลิตซอฟต์แวร์ที่สามารถนำไปใช้งานได้อย่างรวดเร็ว

สิ่งแวดล้อมที่เปลี่ยนแปลง (Changing environment)

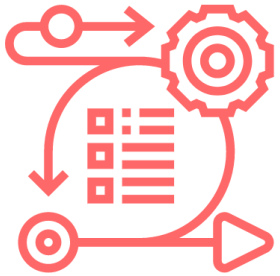
- ระบบจะส่งผลต่อการปฏิบัติงานอย่างไร
- การโต้ตอบกับระบบอื่นอย่างไร
- การดำเนินการใดของผู้ใช้งานควรเป็นแบบอัตโนมัติ

เอจายล์เมทรอด (Agile methods)

- วิธีการพัฒนาที่เพิ่มขึ้น (Incremental development methods)
- ลูกค้ามีส่วนร่วมในกระบวนการพัฒนา โดยรับข้อแนะนำที่เกี่ยวข้องกับความต้องการที่เปลี่ยนแปลงอย่างรวดเร็ว
- ลดเอกสาร โดยใช้การสื่อสารที่ไม่เป็นทางการมากกว่าการประชุมที่เป็นทางการและมีเอกสารประกอบ

เอจายล์เมทธอด (Agile methods)

ซอฟต์แวร์ถูกพัฒนาด้วยทีมขนาดใหญ่ที่ทำงานให้กับหน่วยงานต่าง ๆ **แนวทางการพัฒนาที่ขับเคลื่อนด้วยแผนงาน**



ความไม่พึงพอใจกับแนวทางด้านวิศวกรรมซอฟต์แวร์อย่างมาก นำไปสู่การเสนอวิธีการพัฒนาซอฟต์แวร์แบบใหม่ เรียกว่า "เอจายล์เมทธอด (Agile methods)" ในปี ค.ศ. 1990 โดยนักพัฒนาซอฟต์แวร์

ในปี ค.ศ. 1980 และช่วงต้น ๆ ปี ค.ศ. 1990

การวางแผนโครงการอย่างรอบคอบ การประกันคุณภาพที่มีแบบแผน การใช้วิธีการวิเคราะห์และการออกแบบที่สนับสนุนโดย CASE tools และการควบคุมที่เข้มงวดกับกระบวนการพัฒนาซอฟต์แวร์

ชุมชนด้านวิศวกรรมซอฟต์แวร์ (Software engineering community)

รับผิดชอบการพัฒนาระบบซอฟต์แวร์ขนาดใหญ่ที่ซึ่งมีอายุการใช้งานที่ยาวนาน เช่น ระบบการบินและอวกาศ ระบบสนับสนุนหน่วยงานภาครัฐและเอกชน

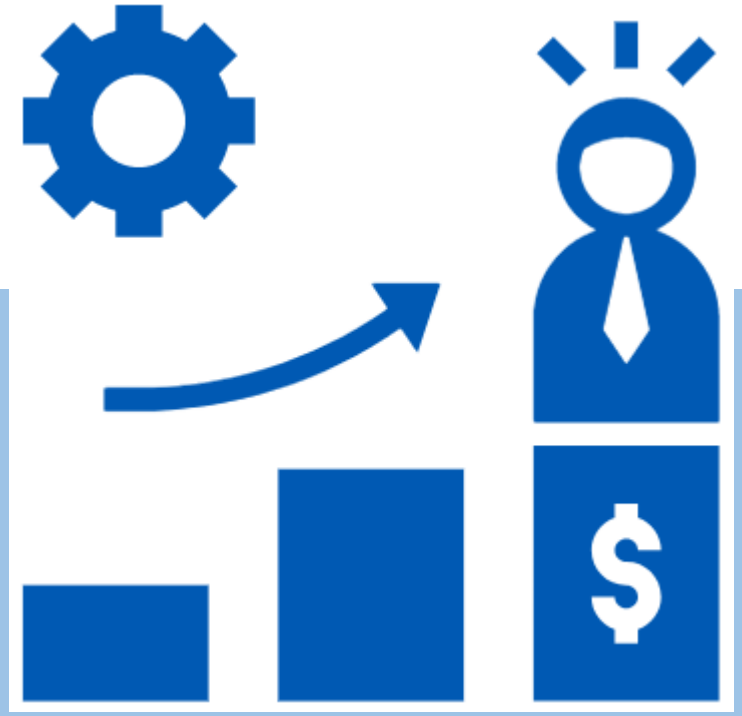
แนวปฏิบัติของเอจายล์เมทรอด (The principles of agile methods)

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

ที่มา: Lan Sommerville, Software Engineering Ninth Edition, 2011.

เอจายล์เมทธอด (ต่อ)

- เอจายล์เมทธอด ถือว่าประสบความสำเร็จอย่างมากกับการพัฒนาซอฟต์แวร์บางประเภท ดังนี้



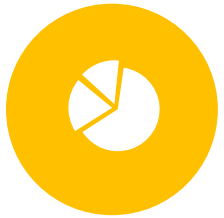
การพัฒนาผลิตภัณฑ์ สำหรับบริษัทผลิตซอฟต์แวร์เพื่อจำหน่ายขนาดเล็กไปจนถึงขนาดกลาง

การพัฒนาซอฟต์แวร์แบบปรับแต่งเพื่อใช้งานภายในองค์กร ที่ซึ่งลูกค้ามีเป้าหมายที่ชัดเจนในการส่วนร่วมในกระบวนการพัฒนา และอาจจะไม่มีกฎและข้อบังคับจากภายนอก ที่ส่งผลกระทบต่อซอฟต์แวร์ค่อนข้างมาก

เอจายล์เมทธอด (ต่อ)

- ในทางปฏิบัติ บางครั้งหลักการที่อยู่ภายใต้วิธีการแบบเอจายล์เมทธอดนั้นอาจจะไม่ง่ายต่อการทำความเข้าใจ

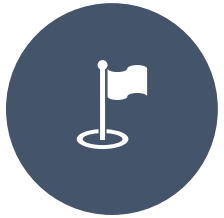
แนวคิดการมีส่วนร่วมของลูกค้าในกระบวนการพัฒนาซอฟต์แวร์เป็นสิ่งที่น่าสนใจ แต่การที่จะประสบความสำเร็จนั้นขึ้นอยู่กับการมีลูกค้าที่เต็มใจและสามารถใช้เวลากับทีมผู้พัฒนาและกลุ่มบุคคลที่มีส่วนได้ส่วนเสียของระบบทั้งหมด



การบำรุงรักษาที่เรียบง่ายอาจต้องมีขั้นตอนการทำงานพิเศษเพิ่มเติม

การจัดลำดับความสำคัญการเปลี่ยนแปลงที่เกิดขึ้นอาจเป็นเรื่องที่ค่อนข้างยาก โดยเฉพาะในระบบที่มีผู้มีส่วนได้ส่วนเสียจำนวนมาก

สมาชิกในทีมยางคนอาจจะไม่มีบุคลิกที่เหมาะสมกับการมีส่วนร่วมที่เข้มงวดมากนักซึ่งเป็นเรื่องปกติสำหรับแนวทางเอจายล์ ดังนั้นจึงไม่สามารถโต้ตอบกับสมาชิกในทีมคนอื่น ๆ ได้ดีเท่าที่ควร



หลายองค์กร โดยเฉพาะบริษัทขนาดใหญ่อาจต้องใช้เวลาค่อนข้างนานในการเปลี่ยนแปลงวัฒนธรรมในองค์กร เพื่อให้มีการกำหนดและปฏิบัติตามกระบวนการต่าง ๆ

เอจายล์เมทธอด (ต่อ)

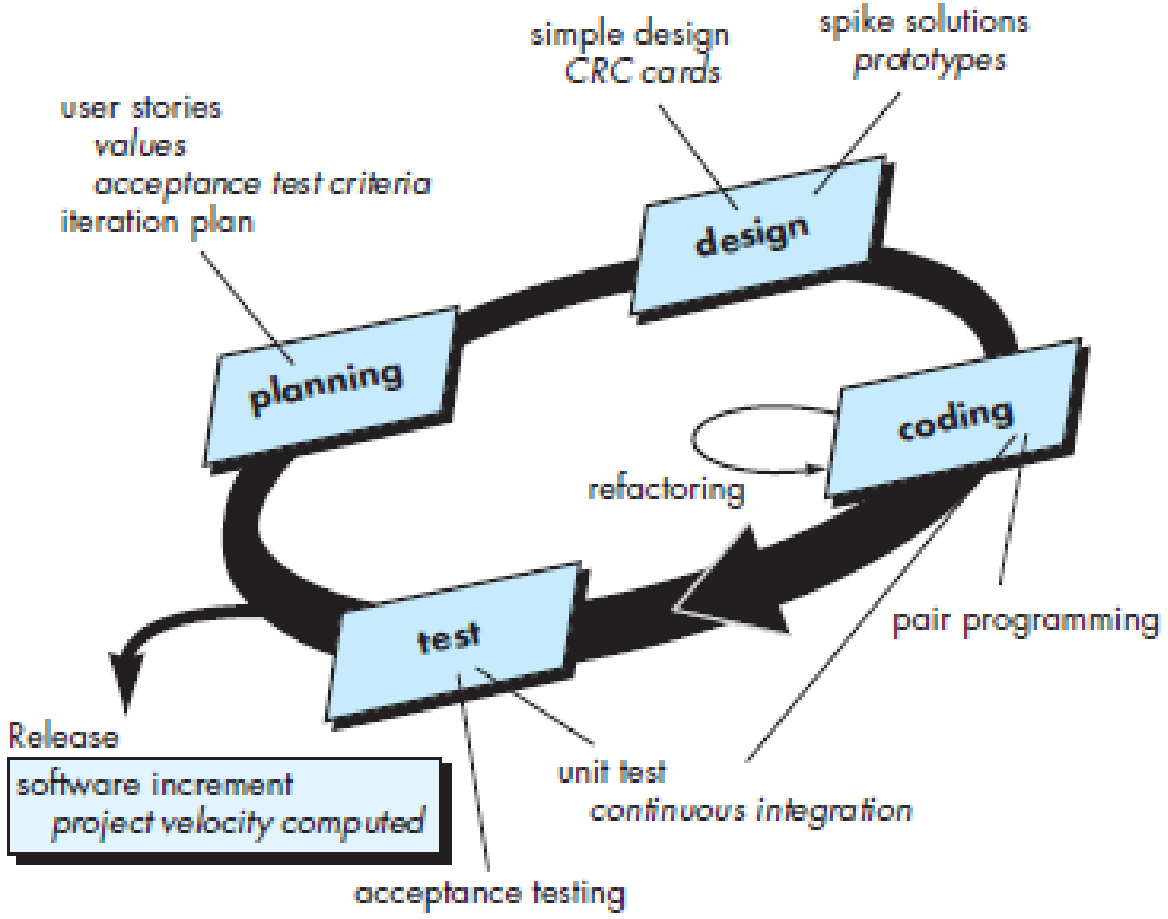
- คำถาม 2 ข้อที่ควรนำมาพิจารณาเมื่อพิจารณาถึงเอจายล์เมทธอดและการบำรุงรักษา

ระบบที่พัฒนาขึ้นมา สามารถใช้แนวทางเอจายล์บำรุงรักษาได้หรือไม่ ซึ่งต้องให้ความสำคัญกับกระบวนการพัฒนา สำหรับลดเอกสารในแนวทางที่เป็นทางการ

สามารถใช้งานเอจายล์เมทธอดอย่างมีประสิทธิภาพสำหรับการพัฒนาระบบเพื่อตอบสนองต่อคำขอเปลี่ยนแปลงของลูกค้าได้หรือไม่

แบบจำลอง Extreme programming (XP)

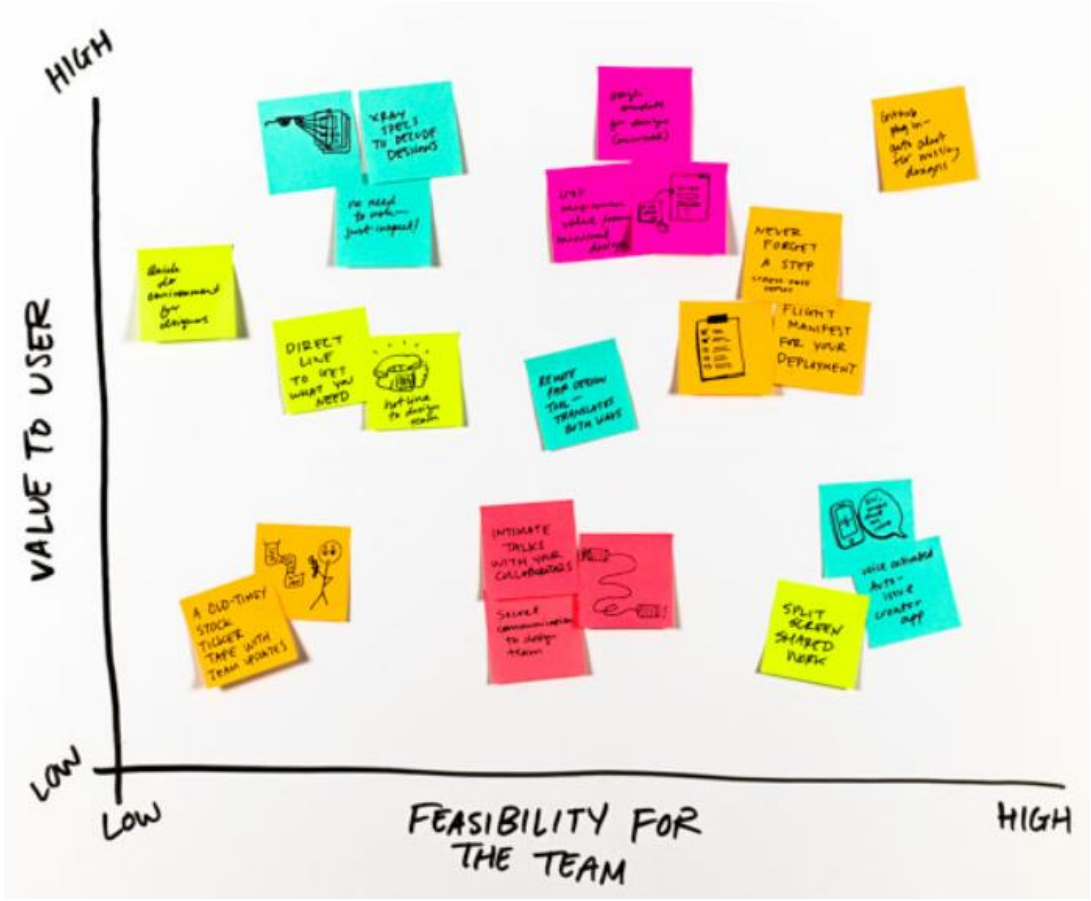
- แบบจำลอง Extreme programming
  - ใช้แนวคิดการออกแบบเชิงวัตถุ
  - คิดค้นโดย Kent Back ในปี ค.ศ. 1999
  - มีการดำเนินงาน 4 ขั้นตอน ดังนี้
    - การวางแผน (Planning)
    - การออกแบบ (Design)
    - การเขียนโปรแกรม (Coding)
    - การทดสอบ (Test)



Source: Roger S. Pressman and Bruce R. Maxim, Software Engineering: A Practitioner's Approach, Eighth Edition, 2015.

แบบจำลอง Extreme programming: การวางแผน

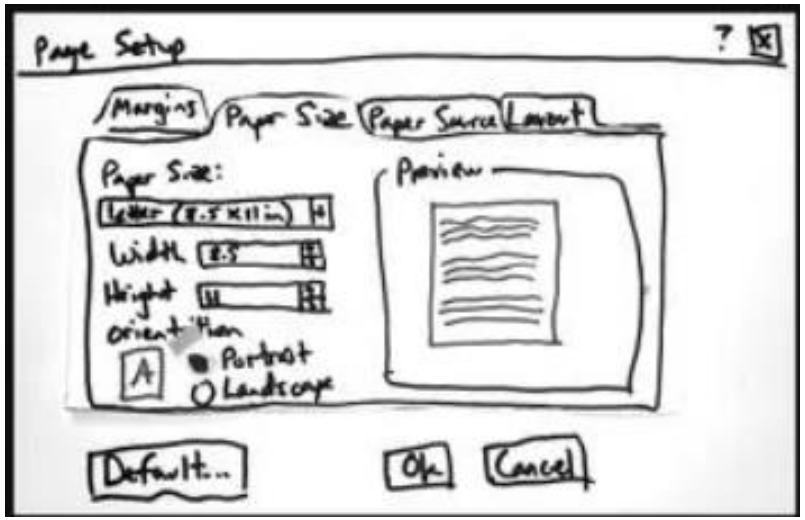
- กำหนดเรื่องราวในมุมของลูกค้าหรือผู้ใช้
  - User Story
  - ดำเนินการโดยนักวิเคราะห์ระบบในกิจกรรมต่าง ๆ
- วิเคราะห์ต้นทุนและผลตอบแทนในเวลาที่กำหนด
- เกณฑ์ที่ใช้ในการยอมรับสำหรับการทดสอบต่าง ๆ
- แผนการทำซ้ำ (Iteration plan)

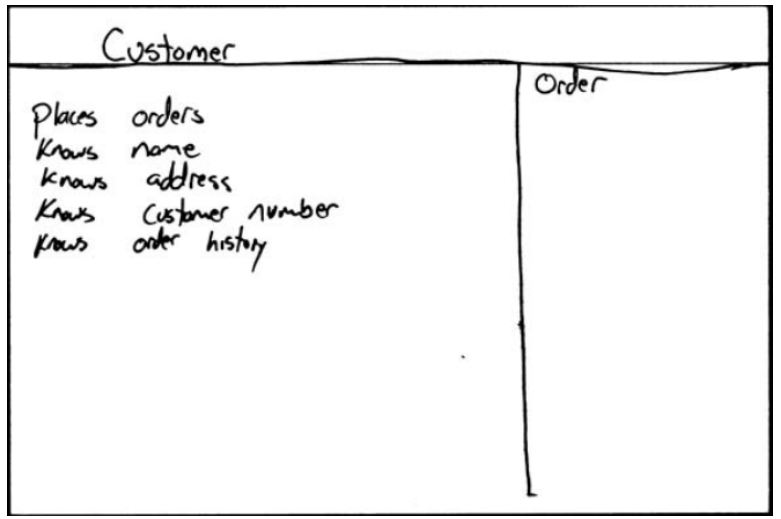


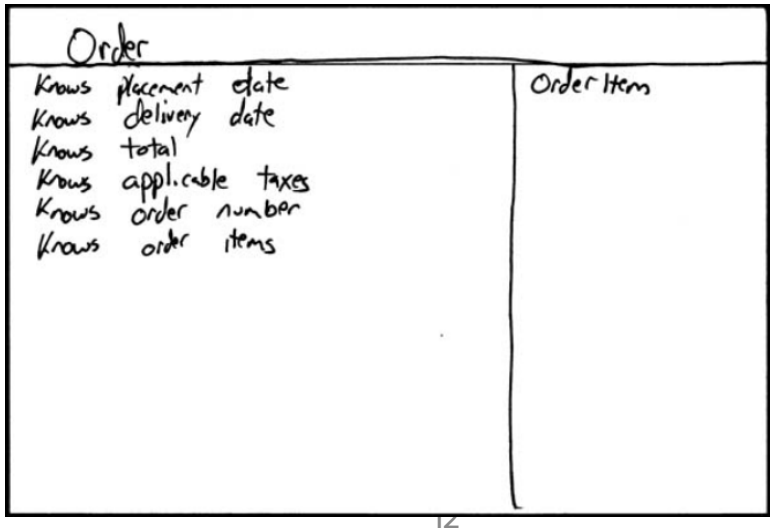
ที่มา: https://www.ibm.com/design/thinking/static/prioritization-3-7e06178844d46170f0f51b15e40a478c.jpg

แบบจำลอง Extreme programming: การออกแบบ

- สไปค์โซลูชั่น (Spike solutions)
- เน้นดำเนินการให้เรียบง่าย (Keep it Simple: KIS)
- ตัวอย่าง:
  - โปรโตไทป์ (Prototypes)
  - บัตรซีอาร์ซี (CRC Cards)







ที่มา: http://agilemodeling.com/images/models/crcCardExample.jpg

แบบจำลอง Extreme programming: การเขียนโปรแกรม

- การเขียนโปรแกรมแบบคู่ (Pair programming)
- การบูรณาการอย่างต่อเนื่อง (Continuous integration)

ที่มา: https://en.wikipedia.org/wiki/Pair_programming

แบบจำลอง Extreme programming: การทดสอบ

- การทดสอบระดับหน่วย (Unit testing)
- การทดสอบการยอมรับของผู้ใช้ (Acceptance testing)

Release:
- การเพิ่มขึ้นของซอฟต์แวร์ (Software increment)
- การคำนวณความเร็วของโครงการ (Project velocity computed)

แนวทางปฏิบัติสำหรับ XP (Extreme programming practices)

Principle or practice	Description
Incremental planning	Requirements are recorded on Story Cards and the Stories to be included in a release are determined by the time available and their relative priority. The developers break these Stories into development 'Tasks'. See Figures 3.5 and 3.6.
Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple design	Enough design is carried out to meet the current requirements and no more.
Test-first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.
Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site customer	A representative of the end-user of the system (the Customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

Source: Lan Sommerville, Software Engineering Ninth Edition, 2011.

การมีส่วนร่วมของผู้ที่เกี่ยวข้องในการพัฒนาซอฟต์แวร์

การสื่อสาร (Communication)

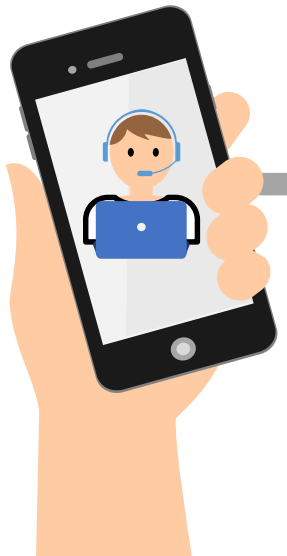
ความเรียบง่าย (Simplicity)

Lightweight Methodology

ความกล้า (Courage)

คำติชม (Feedback)

แนวทางปฏิบัติของแบบจำลอง XP



มีการวางแผนที่ดี

ส่งมอบงานบ่อยขึ้น

นิยามคำเฉพาะ/ใช้เป็นระเบียบแบบแผนในการออกแบบซอฟต์แวร์

ออกแบบซอฟต์แวร์ให้ง่าย ปรับปรุงเรื่อย ๆ และจำกัดความซับซ้อน

การทดสอบซอฟต์แวร์

ปรับปรุงและแก้ไขซอฟต์แวร์

Standard

On-site customer

Sustainable Pace

พัฒนา module เสร็จให้รวม module อื่นทันที

นักเขียนโปรแกรมเป็นเจ้าของโปรแกรมร่วมกัน

Pair programming

ข้อดี/ข้อเสียแบบจำลอง XP

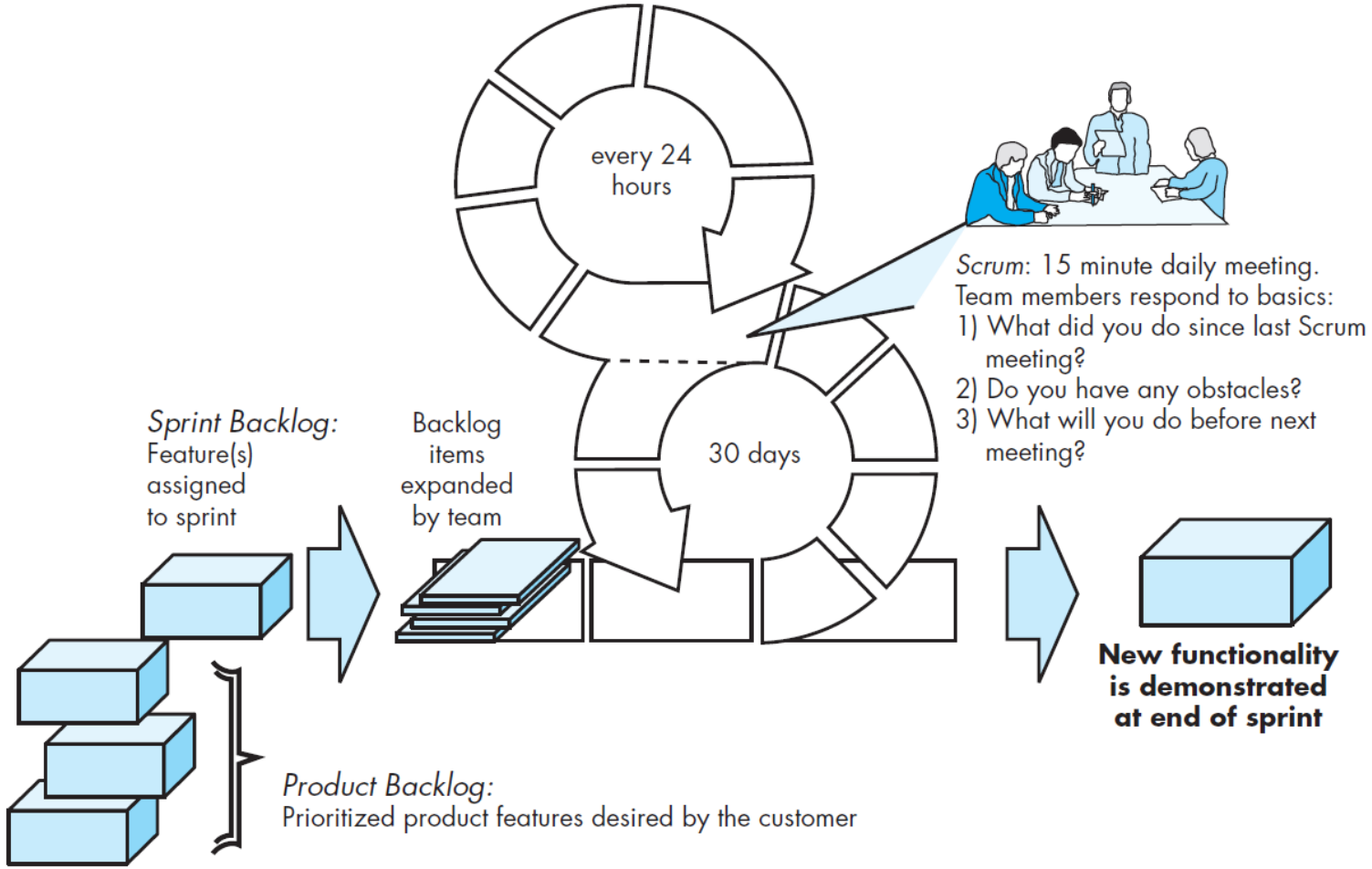
Advantage

- เหมาะสำหรับโครงการขนาด Small และ medium
- ออกแบบซอฟต์แวร์ให้ง่าย เน้นความคล่องตัว
- สามารถออกแบบซอฟต์แวร์ใหม่ได้บ่อยครั้ง
- นักพัฒนาและผู้ใช้มีความสัมพันธ์ที่ดีต่อกัน
- การเปลี่ยนแปลงโปรแกรมเป็นไปตามความต้องการของลูกค้า
- ประหยัดเวลาและค่าใช้จ่าย

- ขาดการวางแผนและการจัดการคุณภาพ
- ผู้ใช้มีการเปลี่ยนแปลงความต้องการบ่อย
- เน้น Software coding มากกว่า software design
- Pair programming อาจทำให้เกิดความขัดแย้ง
- การสร้าง Test cases อาจทำได้ยาก
- ขาดการสนับสนุนที่ดี

Disadvantage

แบบจำลองสครัม (SCRUM)

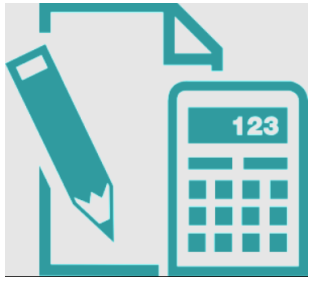


- Scrum Model
  - Jeff Sutherland, ในปี ค.ศ. 1990
  - Ken Schwaber และ Mike Beedie ในปี ค.ศ. 2001
  - รูปแบบการทำงานแต่ละคนผลักดันซึ่งกันและกัน
- กระบวนการทำงาน
  - การจัดลำดับความสำคัญในคุณลักษณะของผลิตภัณฑ์ที่ลูกค้าต้องการ (Product Backlog)
  - การกำหนดกรอบเวลาในการพัฒนาผลิตภัณฑ์ แบบวนซ้ำ (Sprint)
- องค์ประกอบ
  - ทีมงาน (Teams), กระบวนการ (Process), การนำเสนอผลงาน (Demo)

Source: Roger S. Pressman and Bruce R. Maxim, Software Engineering: A Practitioner's Approach, Eighth Edition, 2015.

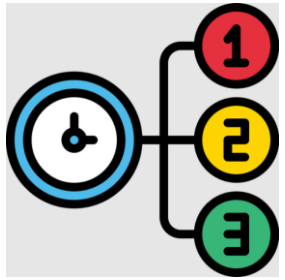
แบบจำลองสครัม: ทีมงาน (Team)

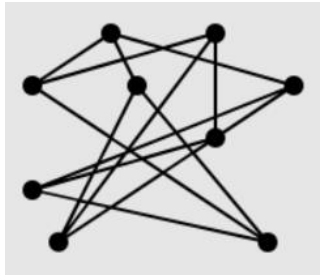
เจ้าของงาน
(Product Owner)



ทีมงานสครัม
(Scrum Team)

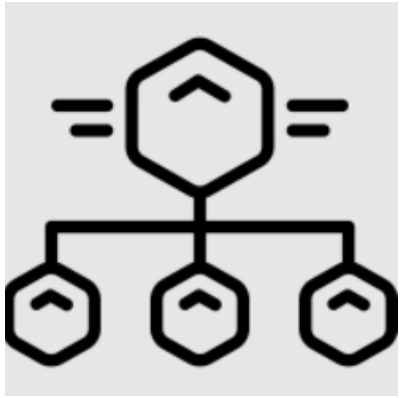
หัวหน้าสครัม
(Scrum Master)





แบบจำลองสครัม: กระบวนการ (Process)

ความต้องการ (Backlog / Wishlist)

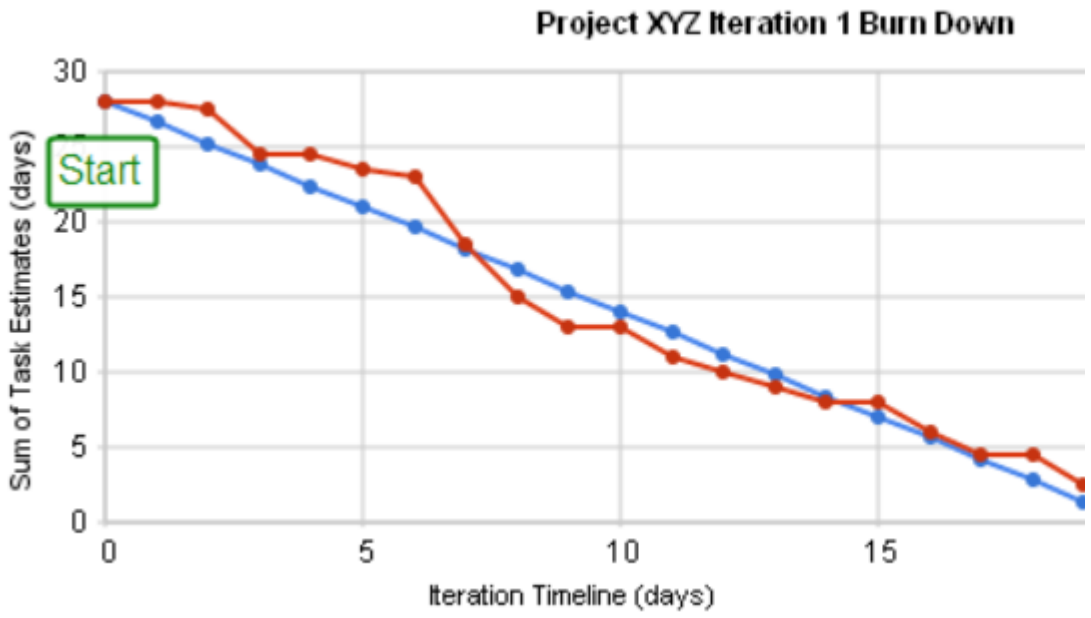


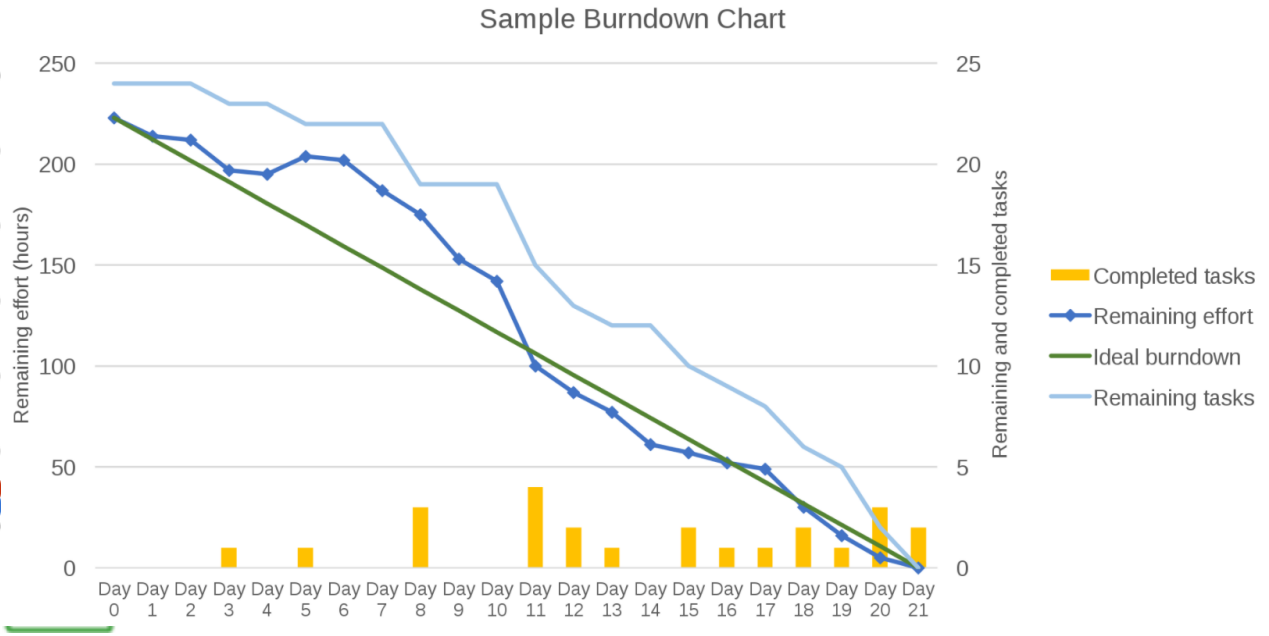
ขั้นตอนการวนซ้ำ (Sprint Phase)

การสครัมประจำวัน (Daily Scrum)

แบบจำลองสครัม: การนำเสนอผลงาน (Demo)

- แผนภูมิเบิร์นดาวน์ (Burndown Chart)
  - เส้นประมาณการ (Estimate Burndown)
  - เส้นที่เกิดขึ้นจริง (Actual Burndown)





ที่มา: https://en.wikipedia.org/wiki/Burn_down_chart

ข้อดี/ข้อเสียแบบจำลองสครัม

Advantage

- ทีมงานขนาดเล็กใช้เทคนิคฯ แบบไม่เป็นทางการ
- ปรับกระบวนการให้สอดคล้องกับการเปลี่ยนแปลง
- แบ่งงานออกเป็นกลุ่ม ให้ขึ้นต่อกันน้อยที่สุด
- ผลิตซอฟต์แวร์ให้มีหลายเวอร์ชัน
- นักพัฒนาฯ จัดทำเอกสารที่สำคัญ / ทำงานสำเร็จ
- ส่งมอบซอฟต์แวร์ให้กับลูกค้าได้เร็ว

- การเปลี่ยนแปลงขอบเขตงานอาจจะดำเนินการลำบาก
- ไม่มีความยืดหยุ่นในการทำงานจากการกำหนดบทบาทที่แน่นอน
- อาจทำให้เกิดปัญหาจากการ Cross-Functional Team
- การเปลี่ยนแปลงกระบวนการวนซ้ำอาจเปลี่ยนแปลงบ่อยไม่ได้
- ขาดความต่อเนื่องและมองเห็นภาพรวมได้ยากของความต้องการขนาดใหญ่

Disadvantage

แบบจำลองกระบวนการพัฒนาแบบเอจายล์เพิ่มเติม

- Adaptive Software Development (ASD)
- Crystal Model
- Agile Unified Process (AUP)
- Microsoft Solutions Framework (MSF)

สรุป (Summary)

- เอจายล์เมทธอด (Agile methods)
- แบบจำลองกระบวนการพัฒนาแบบเอจายล์
  - แบบจำลอง Extreme Programming (XP)
  - แบบจำลองสครัม (SCRUM)

กิจกรรมท้ายบท

- จงอธิบายเกี่ยวกับหลักการและความสำเร็จของการนำกระบวนการพัฒนาซอฟต์แวร์แบบเอจายล์ในโครงการพัฒนาซอฟต์แวร์
- จงยกตัวอย่างกระบวนการพัฒนาซอฟต์แวร์แบบเอจายล์ (เลือกอธิบายอย่างน้อย 1 กระบวนการ) พร้อมวิเคราะห์เกี่ยวกับความสำคัญ การประยุกต์ใช้งาน ตลอดจนข้อดีและข้อเสียของกระบวนการดังกล่าว

เอกสารอ้างอิง

- กิตติ ภักดีวัฒนะกุล, วิศวกรรมซอฟต์แวร์ (Software Engineering), กรุงเทพฯ: เคทีพี คอมพ์ แอนด์ คอนซัลท์, 2552
- น้ำฝน อัศวเมฆิน, หลักการพื้นฐานของวิศวกรรมซอฟต์แวร์ (Fundamentals of Software Engineering), กรุงเทพฯ: ซีเอ็ดยูเคชั่น, 2560.
- Lan Sommerville, Software Engineering Ninth Edition, Pearson Education, Inc., publishing as Addison-Wesley, 2011.
- Roger S. Pressman and Bruce R. Maxim, Software Engineering: A Practitioner's Approach, Eighth Edition, McGraw-Hill Education, 2015.