

Rapid Software Development

Week 11

หัวข้อที่จะศึกษา

- Rapid software development
- วิธีการแบบอไจล์ (Agile methods)
- เทคนิคอไจล์ (Agile development techniques)
- การบริการโครงการอไจล์ (Agile project management)
- วิธีการอไจล์แบบปรับสเกล (Scaling agile methods)

Rapid software development

- ความสำคัญของการพัฒนาและส่งมอบซอฟต์แวร์ที่รวดเร็ว
 - การพัฒนาทางธุรกิจ ทำให้ความต้องการซอฟต์แวร์ที่พัฒนาอย่างรวดเร็ว
 - การรอให้ requirement อยู่ตัว อาจไม่ทันต่อการแข่งขันทางธุรกิจ
 - ซอฟต์แวร์ที่ดี ต้องตอบสนองต่อความต้องการที่เปลี่ยนแปลงไปอย่างรวดเร็วและไม่หยุดยั้ง
 - นักพัฒนาซอฟต์แวร์ ต้องยอมรับความจริงในข้อนี้ และ เตรียมตัวให้พร้อม สำหรับการเปลี่ยนแปลงทุกรูปแบบ
 - กระบวนการพัฒนาซอฟต์แวร์ที่ไม่รองรับการเปลี่ยนแปลงอย่างว่องไว ย่อมทำให้ซอฟต์แวร์ไม่เป็นที่
ต้องการของตลาด

Rapid software development

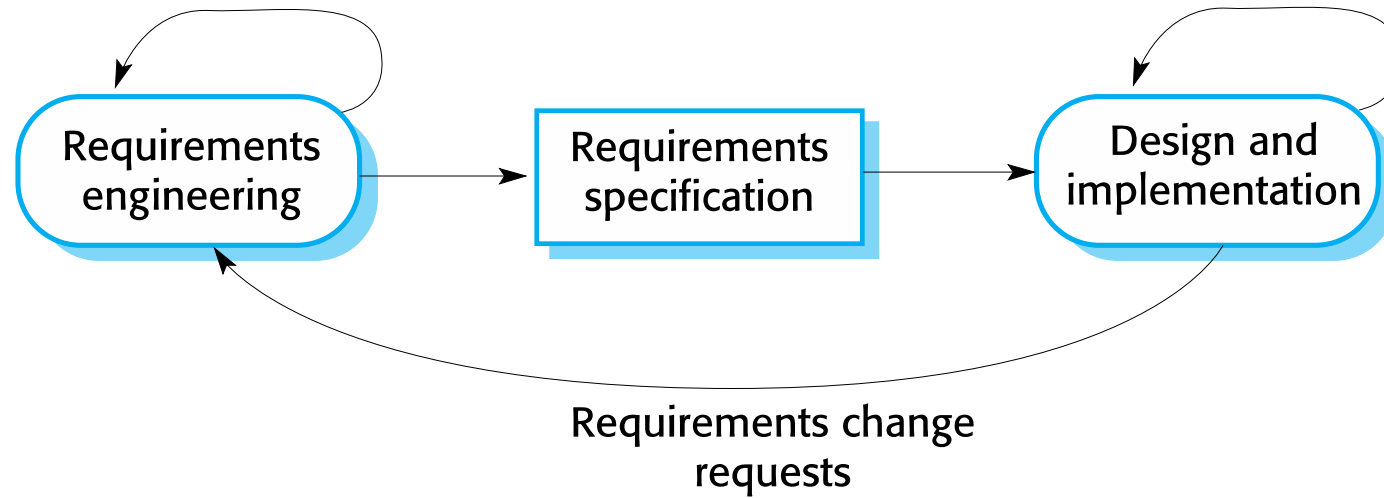
- การพัฒนาซอฟต์แวร์ ตามระเบียบแบบแผน (plan-driven) อาจช่วยในหลายเรื่อง แต่ไม่ยืดหยุ่นและไม่รองรับความต้องการที่เปลี่ยนแปลงอย่างรวดเร็ว
 - กว่าจะครบทุกขั้นตอนในการพัฒนา อาจใช้เวลาเป็นเดือนหรือปี ทำให้ไม่สามารถตอบสนองต่อความต้องการได้ทัน่วงที
- วิธีการพัฒนาแบบ **agile** กำเนิดขึ้นในยุค 1990
 - เพื่อลดระยะเวลาในการพัฒนาและส่งมอบซอฟต์แวร์ให้สั้นลง
 - วิธีการง่ายๆ คือ ปล่อยมาเป็นรุ่นย่อย ๆ (พัฒนาเป็น incremental)

ลักษณะเฉพาะของ Agile development

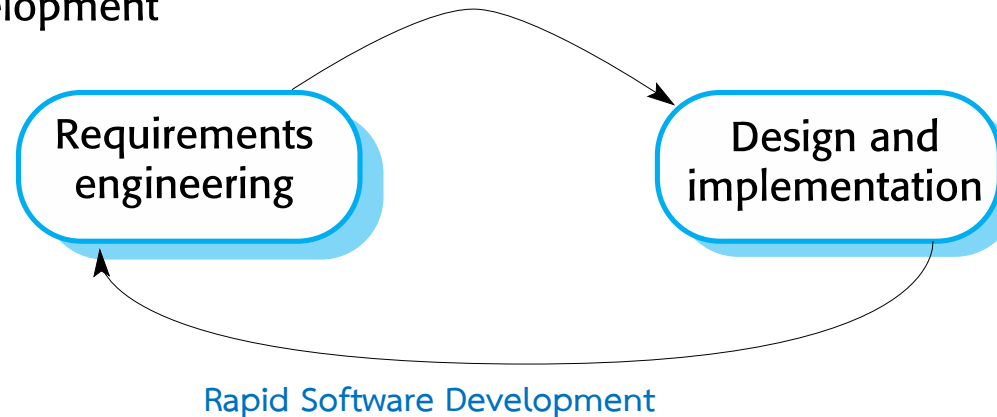
- การออกข้อกำหนดโปรแกรม (Program specification) การออกแบบและสร้าง (design and implementation) จะทำควบคู่กันไป
- การพัฒนาจะทำเป็นรุ่นๆ โดยปล่อยออกมาเป็น incremental
 - ในการพัฒนาแต่ละรุ่น จะมี stakeholders เข้ามาร่วมออก specification และทำการ evaluation
- ปล่อยรุ่นล่าสุดออกมา ให้บ่อยที่สุด เท่าที่จะเป็นไปได้
- ใช้เครื่องมือช่วยในการพัฒนาให้มากที่สุด (เช่น automated testing tools)
- ทำเอกสารน้อย ๆ
 - เน้นที่ working code (เขียนโค้ดให้มีความเป็น document ในตัว เช่นการตั้งชื่อตัวแปรที่สื่อความหมาย)

Plan-driven vs agile development

Plan-based development



Agile development



Plan-driven and agile development

○ Plan-driven development

- การพัฒนาซอฟต์แวร์แบบ plan-driven จะมีการแบ่ง development stages อย่างชัดเจน
- outputs ที่ได้จากแต่ละ stages จะเป็นตัวขับเคลื่อน ให้การพัฒนาดำเนินต่อไปได้
- ไม่ใช่เพียงแต่ใน waterfall model เท่านั้น incremental development เอง ก็ทำให้เกิดรูปแบบ plan-driven ได้เช่นกัน

○ Agile development

- Specification, design, implementation and testing ของแต่ละรุ่นจะเกิดขึ้นขนานกันไป
- outputs ของการพัฒนา จะอยู่ที่การคุยกันระหว่างทีมพัฒนาและ stakeholder

ทำไมต้อง Agile process

- เป็นการยาก ที่จะพยากรณ์ว่า ความต้องการใดของซอฟต์แวร์ ที่จะคงอยู่ หรือ เปลี่ยนแปลงไป
- เป็นการยาก ที่จะพยากรณ์ว่า ในขณะที่การพัฒนากำลังดำเนินไป user จะเปลี่ยนความต้องการที่จุดใดบ้าง
- เป็นการยาก ที่จะบอกได้ว่าต้อง design มากแค่ไหน (ทั้งปริมาณและคุณภาพ) ถึงจะตอบโจทย์ requirement ได้อย่างเหมาะสม
 - บาง design จะให้ผลลัพธ์เมื่อสร้างเสร็จแล้วเท่านั้น
- เป็นการยากที่จะกำหนดกรอบเวลาหรือความสำเร็จของการ วิเคราะห์ ออกแบบ สร้าง และทดสอบซอฟต์แวร์ ตั้งแต่เริ่มโครงการ

Agile methods

- ข้อผิดพลาดในการพัฒนาซอฟต์แวร์ในยุค 80 และ 90 คือ การมี overhead มากเกินไปนำมาสู่การพัฒนาแบบ agile ซึ่งมีลักษณะเฉพาะคือ
 - เน้นที่การเขียน code มากกว่าที่จะเน้นที่การออกแบบ (design)
 - เน้นที่การพัฒนาซอฟต์แวร์แบบ iterative approach
- มุ่งเน้นในการส่งมอบซอฟต์แวร์ที่ใช้งานได้จริงและตรงตามความต้องการที่เปลี่ยนไปอย่างรวดเร็ว
- สิ่งที่เรียกว่า overhead ได้แก่ เอกสารจากขั้นตอนต่างๆ ที่เกิดขึ้นในกระบวนการพัฒนา
- เป้าหมายของ agile methods คือ
 - การลด overheads โดยการจำกัดปริมาณเอกสารที่ต้องทำ
 - มุ่งเน้นที่การตอบสนองต่อความต้องการของลูกค้ามากกว่าการทำงานที่ไม่จำเป็นและสิ้นเปลือง

คำประกาศของอไจล์ (Agile manifesto)

- พัฒนาซอฟต์แวร์ด้วยความเอื้อเฟื้อซึ่งกันและกัน

| สิ่งที่ต้องให้ความสำคัญ | สิ่งที่ไม่จำเป็นต้องทำ |
|-------------------------|---------------------------|
| บุคคลและการปฏิสัมพันธ์ | เครื่องมือและกระบวนการ |
| ซอฟต์แวร์ที่ใช้ได้จริง | เอกสาร |
| ความร่วมมือจากผู้ใช้ | การต่อรอง |
| พร้อมที่จะเปลี่ยนแปลง | ทำตามแผน (อย่างเคร่งครัด) |

The principles of agile methods (a)

| Principle | Description |
|--|---|
| การมีส่วนร่วมจากผู้ใช้ (Customer involvement) | <ul style="list-style-type: none">• Customers ควรมีส่วนร่วมอย่างใกล้ชิดตลอดระยะเวลาในการพัฒนา• หน้าที่ของ customer คือ การระบุและจัดลำดับความสำคัญของ requirements รวมทั้งต้องทำหน้าที่ทดสอบด้วย |
| การทยอยส่งมอบ (Incremental delivery) | <ul style="list-style-type: none">• ซอฟต์แวร์จะต้องทยอยพัฒนาและส่งมอบเป็นรุ่น ๆ ตามข้อกำหนดของผู้ใช้ |
| เน้นคน ไม่เน้นกระบวนการ (People not process) | <ul style="list-style-type: none">• ดึงทักษะของคนในทีมมาใช้อย่างเต็มศักยภาพ• ปล่อยให้สมาชิกในทีมทำงานอย่างอิสระ ไม่ต้องหากระบวนการใด ๆ มากำหนดการทำงาน |

The principles of agile methods (b)

| Principle | Description |
|---|---|
| น้อมรับการเปลี่ยนแปลง (Embrace change) | <ul style="list-style-type: none">• ให้ระลึกและตระหนักว่า system requirements จะมีการเปลี่ยนแปลงอยู่เสมอ• ออกแบบซอฟต์แวร์ให้ตอบโต้ต่อการเปลี่ยนแปลงที่จะเกิดขึ้น |
| รักษาความเรียบง่าย (Maintain simplicity) | <ul style="list-style-type: none">• มุ่งเน้นความเรียบง่าย ทั้งตัวซอฟต์แวร์ที่จะพัฒนา และกระบวนการที่จะใช้• ให้กำจัดความยุ่งยากซับซ้อนออกไปทันทีที่เป็นไปได้ |

ใช้งาน Agile method เมื่อใด?

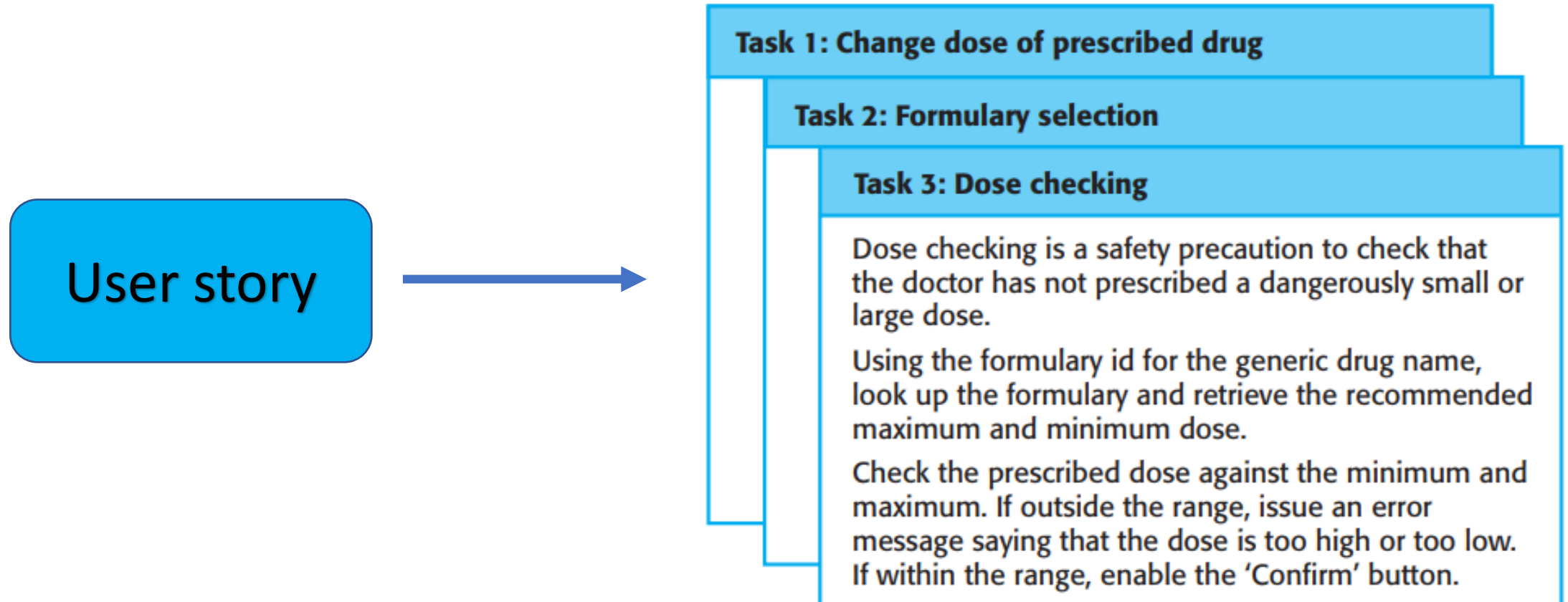
- ใช้ในการพัฒนาซอฟต์แวร์ขนาดเล็กถึงขนาดกลาง
 - แต่ดูเหมือนว่า ซอฟต์แวร์และแอปส์แทบทั้งหมดในปัจจุบัน จะพัฒนาโดยใช้ agile
- การพัฒนาซอฟต์แวร์ใช้งานภายในองค์กร
 - มี user ที่ตั้งใจจะเข้าร่วมทีมพัฒนา (ทำหน้าที่ตามคำประกาศของ agile)
 - ไม่มีการกำหนดกฎเกณฑ์จากภายนอก ที่จะกระทบต่อการพัฒนาซอฟต์แวร์นั้น
 - หรือถ้ามีก็ให้น้อยที่สุด

Agile development techniques

User stories

- เมื่อต้องการพัฒนาซอฟต์แวร์อย่างรวดเร็ว ย่อมไม่มีเวลาทำ requirement ที่สมบูรณ์
- Agile จึงพัฒนาแนวคิดที่เรียกว่า User stories ขึ้นมาทดแทนการทำ requirement
- ในทีม agile จะมี system customer คอยให้ scenario แก่ทีมนักพัฒนาซอฟต์แวร์
 - โดยใช้ story card
- เมื่อ user เขียน story card เสร็จแล้ว development team จะแตกออกเป็น task ซึ่งอาจจะระบุทรัพยากร (เช่นกำลังคน หรือเครื่องมือต่างๆ) แล้วมา refine กับ user ก่อนลงมือทำโปรแกรม

ตัวอย่าง story card tasks



User stories

- ในขณะที่การพัฒนากำลังดำเนินไป อาจจะมี requirement ใหม่ๆ เกิดขึ้น หรือมีการทิ้งบาง requirement ไปก็ได้
- System user จะเป็นผู้มีอำนาจในการตัดสินใจในเรื่องต่อไปนี้
 - เลือกที่จะเก็บ user story ไว้ในโปรแกรม
 - ทิ้ง user story
 - สร้าง user story ใหม่
- ถึงแม้จะใช้เทคนิคนี้ แต่ธรรมชาติของการพัฒนาซอฟต์แวร์คือ ไม่มีที่สิ้นสุด ซึ่งก็ไม่มีข้อยกเว้นสำหรับเทคนิค agile

Test-first development

- กระบวนการพัฒนาซอฟต์แวร์ จะมีขั้นตอนการทดสอบซอฟต์แวร์อยู่ด้วยเสมอ แต่กระบวนการทั่ว ๆ ไป มักทำการทดสอบเมื่อได้ซอฟต์แวร์ที่สมบูรณ์แล้ว
 - ถ้าเจอที่ผิด นั้นหมายความว่า อาจจะสายเกินไป
- ในกระบวนการพัฒนาโปรแกรมแบบ Rapid Software Development ใช้เทคนิคที่ต่างไปจากกระบวนการทั่วไป
 1. test-first development
 2. incremental test development จาก scenarios
 3. user มีส่วนร่วมในการ test ทั้งในขั้น development และ validation
 4. ใช้กระบวนการทดสอบอัตโนมัติ (automated testing frameworks)

Refactoring

- แนวคิดสำคัญมากอย่างหนึ่งในพัฒนา software คือ การออกแบบให้รองรับการเปลี่ยนแปลง “design for change”
 - แต่มีคำถามตามมาคือ จะเปลี่ยนแปลงได้มากน้อยแค่ไหน
- ในการทำ evolution นั้น ความจริงประการสำคัญคือ เมื่อมีการเปลี่ยนแปลงใดๆ กับ software ย่อมจะทำให้โครงสร้างของ software นั้นแย่งลงไปทุกครั้ง
- เพื่อให้เกิดข้อผิดพลาด (ซึ่งจะนำมาซึ่งการเปลี่ยนแปลง software) น้อยที่สุด ทีมพัฒนา ต้องมองหาจุดที่สามารถปรับปรุง code (ทำ refactor) อยู่เสมอ
 - เช่น code ที่ซ้ำๆ กัน ควรถูกนำไปไว้ใน function แล้วส่งค่าที่แตกต่างกันไปให้ function นั้น

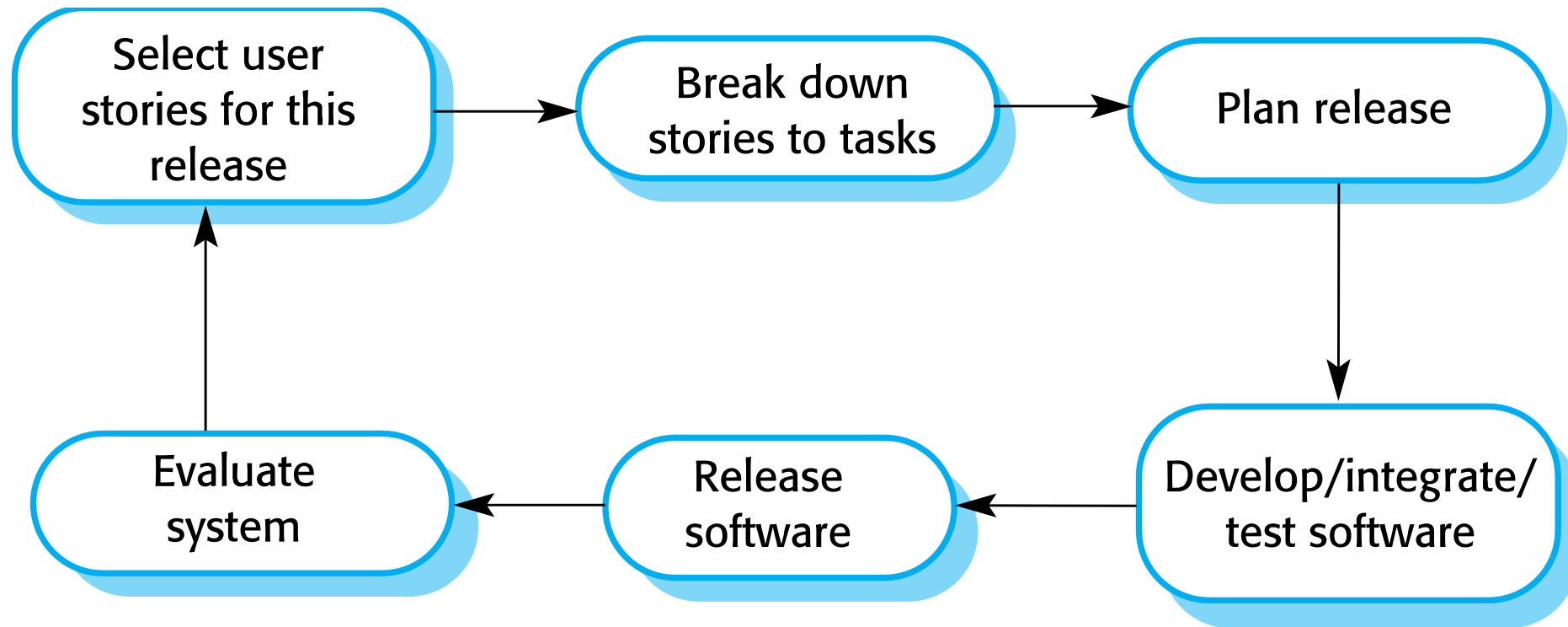
Pair programming

- Programmer มีความรู้สึกในการเป็นเจ้าของร่วมและความรับผิดชอบต่อ code และระบบ
- เป็นการ review code ไปในตัว เนื่องจากแต่ละบรรทัดของ code ถูกอ่านโดยมากกว่า 1 คน
- มีโอกาสที่ code จะถูก refactor จาก programmer ที่มีประสบการณ์สูง
- มีโอกาสถ่ายทอดองค์ความรู้ระหว่าง programmer
- ถ้า programmer คนใดคนหนึ่งลาออก โครงการจะสามารถดำเนินต่อไปได้
- “สองหัวดีกว่าหัวเดียว”

Extreme programming (XP)

- การพัฒนาแบบ agile ได้รับความนิยมในช่วงปลายยุค 1990s และมีการคิดค้นรูปแบบออกมาอย่างมากมาย
- แต่มีรูปแบบหนึ่งที่สำคัญ คิดค้นโดย **Kent Beck** ถูกเรียกว่า **Extreme Programming (XP)** ใช้การพัฒนาแบบ iterative เหตุที่เรียก extreme เนื่องจาก
 - มีการออกรุ่นใหม่บ่อยมาก อาจจะหลายรุ่นในวันเดียว
 - แต่ละรุ่น จะส่งไปยังผู้ใช้ทุก ๆ 2 สัปดาห์
 - มีการทดสอบ (tests) กับทุก build
 - แต่ละ build จะถูกยอมรับ (accepted) ถ้าผ่านการรัน tests ได้อย่างสมบูรณ์เท่านั้น

The extreme programming release cycle



Extreme programming practices (a)

| Principle or practice | Description |
|--|--|
| (การวางแผนแบบ Incremental) Incremental planning | <ul style="list-style-type: none">• Requirements จะถูกเขียนลงบน story cards• Story ที่จะถูกเพิ่มใน release จะถูกเลือกจากระยะเวลาที่ต้องใช้และลำดับความสำคัญของ story• Developers จะกระจาย stories เหล่านั้นออกเป็น development 'Tasks' |
| ปล่อยรุ่นเล็ก ๆ (Small releases) | <ul style="list-style-type: none">• เริ่มต้น จะพัฒนาส่วนเล็ก ๆ ที่มีผลตอบแทนสูงสุดก่อน• ปล่อยรุ่นถัดมาอย่างสม่ำเสมอและเพิ่มความสามารถจาก release แรก ๆ |
| ออกแบบให้ง่ายเข้าใจ (Simple design) | <ul style="list-style-type: none">• ออกแบบ (design) แคพอให้ตอบสนองต่อ requirement ปัจจุบันเท่านั้น• อย่าเยอะ |

Extreme programming practices (b)

| Principle or practice | Description |
|--|--|
| พัฒนาแบบ ทดสอบก่อนเสมอ (Test-first development) | <ul style="list-style-type: none">• ใช้ automated unit test framework เพื่อทดสอบฟังก์ชันใหม่ ๆ ที่พัฒนาขึ้น• เขียน test ก่อนเขียน code ที่จะใช้งานจริงเสมอ |
| Refactoring | <ul style="list-style-type: none">• นักพัฒนาทุกคน ต้องทำ refactoring อย่างสม่ำเสมอ และทำทันทีที่พบว่าสามารถทำได้• จะช่วยให้ทำความเข้าใจ code และบำรุงรักษาได้ง่าย |
| มีบัดดี้ในการเขียนโปรแกรม (Pair programming) | <ul style="list-style-type: none">• ทำงานเป็นคู่• ตรวจสอบซึ่งกันและกัน• ช่วยกันพัฒนาความสามารถของบัดดี้ |

Extreme programming practices (c)

| Principle or practice | Description |
|--|---|
| มีความเป็นเจ้าของร่วมกัน (Collective ownership) | <ul style="list-style-type: none">• นักพัฒนาทำงานเป็นคู่ ในทุกส่วนของระบบ ไม่มีใครเก่งกว่าใคร• ทุกคนในทีมมีความเป็นเจ้าของ code ร่วมกัน• ทุกคนสามารถแก้ไข เปลี่ยนแปลงได้ ทุกส่วนของระบบ |
| บูรณาการอย่างต่อเนื่อง (Continuous integration) | <ul style="list-style-type: none">• ทันทีที่ work ตาม task เสร็จสมบูรณ์ จะถูกนำไปรวม (บูรณาการ) เข้ากับส่วนอื่น ๆ ของระบบ• หลังจากบูรณาการสำเร็จ จะต้องทำการทดสอบทั้งระบบจนผ่าน |
| ก้าวอย่างที่ยั่งยืน (Sustainable pace) | ถึงจะมีความเร่งรีบในการพัฒนา แต่การทำงานมากเกินไปจนเกินกำลัง ไม่ส่งผลดีนัก เนื่องจากจะทำให้ code ที่ได้มีความด้อยคุณภาพลง |

Extreme programming practices (d)

| Principle or practice | Description |
|--|---|
| ลูกค้าร่วมเป็นทีมพัฒนา (On-site customer) | <ul style="list-style-type: none">• ลูกค้าหรือ end-user ของระบบควรเข้าร่วมทีมอย่างเต็มเวลา (full time) กับ XP team.• ในกระบวนการพัฒนาแบบ extreme programming นั้น customer ถือเป็นบุคลากรหนึ่งในทีมพัฒนา มีหน้าที่นำ system requirements มาให้ทีมทำการสร้างซอฟต์แวร์ |

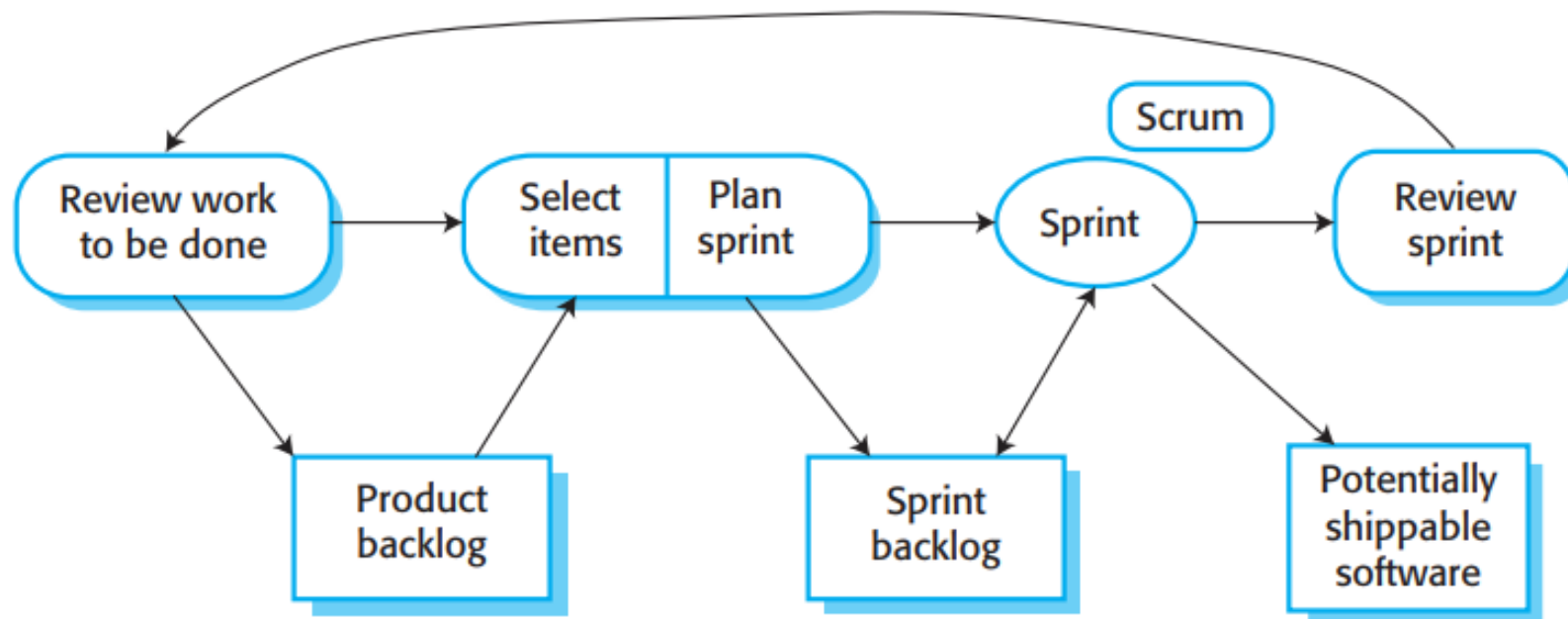
Agile project management

Agile project management

- ในทุกโครงการ ผู้บริหารโครงการจำเป็นต้องรู้ความเป็นไปของโครงการ เพื่อการบริหารจัดการและ
การประเมินโครงการ
 - โครงการเป็นไปตามแผนการหรือไม่ แต่ละขั้นตอนหรือส่วนประกอบได้รับการจัดการที่ดีหรือไม่
 - เมื่อใดที่จะสามารถส่งมอบ software ไปยังผู้ใช้
 - ฯลฯ
- แต่ในโครงการ agile มีสิ่งที่แตกต่าง
 - รูปแบบองค์กรแบบ self-organization
 - ไม่เน้นเอกสาร
 - วงรอบในการพัฒนาที่สั้นมาก

Scrum agile method

- เป็นแนวคิดหรือวิธีการที่ถูกออกแบบมาเพื่อบริหารโครงการแบบ agile โดย Schwaber และ Beedle 2001; Rubin 2013



Scrum definition

| Scrum term | Definition |
|---|---|
| Development team | <ul style="list-style-type: none">• กลุ่มนักพัฒนาซอฟต์แวร์ที่จัดกลุ่มเอง (self-organization)• ไม่ควรมากกว่า 7 คน• มีหน้าที่รับผิดชอบในการพัฒนาซอฟต์แวร์และเอกสารโครงการที่จำเป็น |
| Potentially shippable product increment | <ul style="list-style-type: none">• Incremental ของซอฟต์แวร์ที่ได้มาจากขั้นตอนการ sprint• เป็นสิ่งที่สามารถส่งมอบให้ลูกค้าได้ ควรเป็นสิ่งที่เสร็จสิ้นแล้วและไม่ต้องดำเนินการใดๆ เพิ่มเติมอีก (เช่นการทดสอบ) |
| Product backlog | <ul style="list-style-type: none">• รายการ "สิ่งที่ต้องทำ" (to do list) ที่ทีม Scrum ต้องจัดการ• อาจเป็น feature definitions , software requirements, user stories, รายละเอียดงานที่ต้องทำหรือจำเป็นต้องใช้เช่น architecture definition และ user documentation |

Scrum definition

| Scrum term | Definition |
|---------------|--|
| Product owner | <ul style="list-style-type: none">• บุคคล (หรืออาจเป็นกลุ่มคนจำนวนน้อยๆ)• มีหน้าที่ในการระบุคุณลักษณะหรือข้อกำหนดของผลิตภัณฑ์ จัดลำดับความสำคัญเหล่านี้เพื่อการพัฒนา และ ตรวจสอบงานอย่างต่อเนื่องเพื่อให้แน่ใจว่าโครงการ ยังคงตอบสนองความต้องการทางธุรกิจที่สำคัญ• อาจจะเป็นลูกค้า ผู้จัดการผลิตภัณฑ์ในบริษัทซอฟต์แวร์ หรือตัวแทนผู้มีส่วนได้ส่วนเสียอื่นๆ |
| Scrum | <ul style="list-style-type: none">• การประชุมประจำวันของทีม Scrum เพื่อทบทวนความคืบหน้าและจัดลำดับความสำคัญของงานที่ต้องทำในวันนั้น• แนวทางที่เหมาะสมคือบุคคลากรทั้งทีมมาพบกันหมดและพูดคุยเป็นเวลาสั้นๆ ให้ทุกคนรับรู้ทุกอย่างที่เป็นไปในการพัฒนา |

Scrum definition

| Scrum term | Definition |
|-------------|---|
| ScrumMaster | <ul style="list-style-type: none">• ScrumMaster มีหน้าที่ดูแลให้กระบวนการ Scrum เป็นไปตามแนวทางและแนะนำทีมในการใช้ Scrum อย่างมีประสิทธิภาพ• มีหน้าที่ติดต่อกับส่วนที่เหลือของบริษัท• ตรวจสอบให้แน่ใจว่าทีม Scrum ไม่ถูกรบกวนจากการแทรกแซงจากภายนอก• ScrumMaster ไม่ควรถูกมองว่าเป็นผู้จัดการโครงการ |
| Sprint | <ul style="list-style-type: none">• วงรอบในการพัฒนา โดยทั่วไปมีระยะเวลา 2 – 4 สัปดาห์ |
| Velocity | <ul style="list-style-type: none">• การประมาณการว่าทีมงานสามารถดำเนินการกับ backlog ที่ค้างอยู่ได้มากน้อยเพียงใดในแต่ละ sprint• การเข้าใจความเร็วของทีมงานจะช่วยให้สามารถประเมินได้ว่าทีมงานสามารถรองรับงานในแต่ละ sprint ได้มากน้อยเท่าใด และทำอย่างไรที่จะสามารถปรับปรุงประสิทธิภาพได้อีก |

Scaling agile methods

Scaling agile methods

- วิธีการแบบ Agile ได้รับการพัฒนาเพื่อใช้โดยทีมเขียนโปรแกรมขนาดเล็ก
 - มักจะรวมกันอยู่ในห้องเดียวกันและสื่อสารกันอย่างไม่เป็นทางการ
 - เดิมใช้สำหรับการพัฒนาระบบและผลิตภัณฑ์ซอฟต์แวร์ขนาดเล็กและขนาดกลาง
 - ใช้ในบริษัทขนาดเล็กที่ไม่มีกระบวนการหรือระบบราชการที่ซับซ้อน
- ในช่วงไม่กี่ปีที่ผ่านมา มีความจำเป็นในการส่งมอบซอฟต์แวร์ที่รวดเร็วขึ้น
 - ส่งผลกระทบต่อระบบขนาดใหญ่และบริษัทขนาดใหญ่ด้วยเช่นกัน
- ระบบขนาดใหญ่และบริษัทขนาดใหญ่จึงต้องปรับใช้เทคนิค agile ด้วยเช่นกัน

แนวทางในการ Scaling agile methods

1. Scaling Up

ขยายขนาดของวิธีการ agile เพื่อรองรับการพัฒนาระบบขนาดใหญ่เกินกว่าจะพัฒนาโดยทีมเล็กๆ ทีมเดียว

2. Scaling Out

ขยายวิธีการ agile จากทีมพัฒนาเฉพาะกิจ ไปใช้กับทีมอื่นๆ ที่อาจจะมีประสพการณ์สูงกว่าแต่ยังคงพัฒนาตามแนวทางเดิม ๆ

หน่วยงานที่ปรับเปลี่ยนมาใช้ในการพัฒนาแบบ agile จะมี productivity improvement ประมาณ 15 % ที่ระยะเวลา 3 ปี

[Ambler, S. W. 2010. “Scaling Agile: A Executive Guide.” http://www.ibm.com/developerworks/community/blogs/ambler/entry/scaling_agile_an_executive_guide10/]

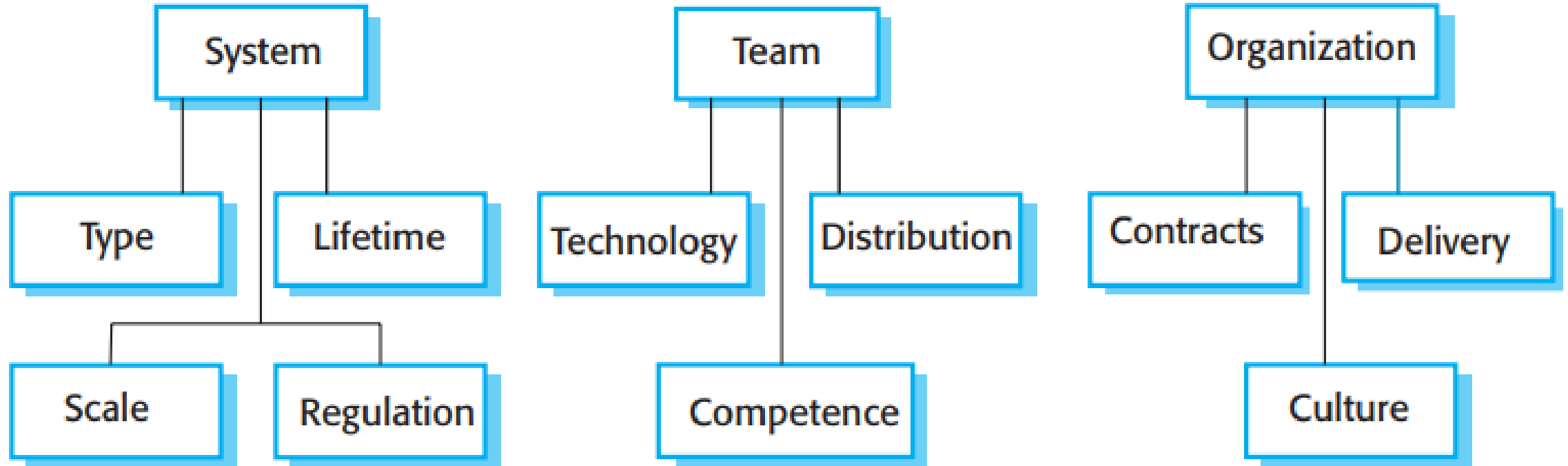
ปัญหาในทางปฏิบัติที่เกิดขึ้นกับ agile methods

- ความไม่เป็นทางการของ agile อาจจะไม่สอดคล้องกับแนวทางทางกฎหมายในบริษัทขนาดใหญ่
 - ส่วนใหญ่ข้อสัญญาของโครงการต้องมี requirement, specification และวิธีการตรวจรับที่ชัดเจนก่อนลงมือทำ
- วิธีการแบบ Agile เหมาะสมสำหรับการพัฒนาซอฟต์แวร์ใหม่มากกว่าสำหรับการบำรุงรักษาซอฟต์แวร์
 - ต้นทุนซอฟต์แวร์ส่วนใหญ่ในบริษัทขนาดใหญ่ มาจากการบำรุงรักษาระบบซอฟต์แวร์ที่มีอยู่
- วิธีการแบบ Agile ออกแบบมาสำหรับทีมขนาดเล็กที่อยู่ร่วมกัน แต่มีซอฟต์แวร์จำนวนมาก
 - การพัฒนาในปัจจุบันมีการร่วมงานของทีมที่กระจายไปทั่วโลก

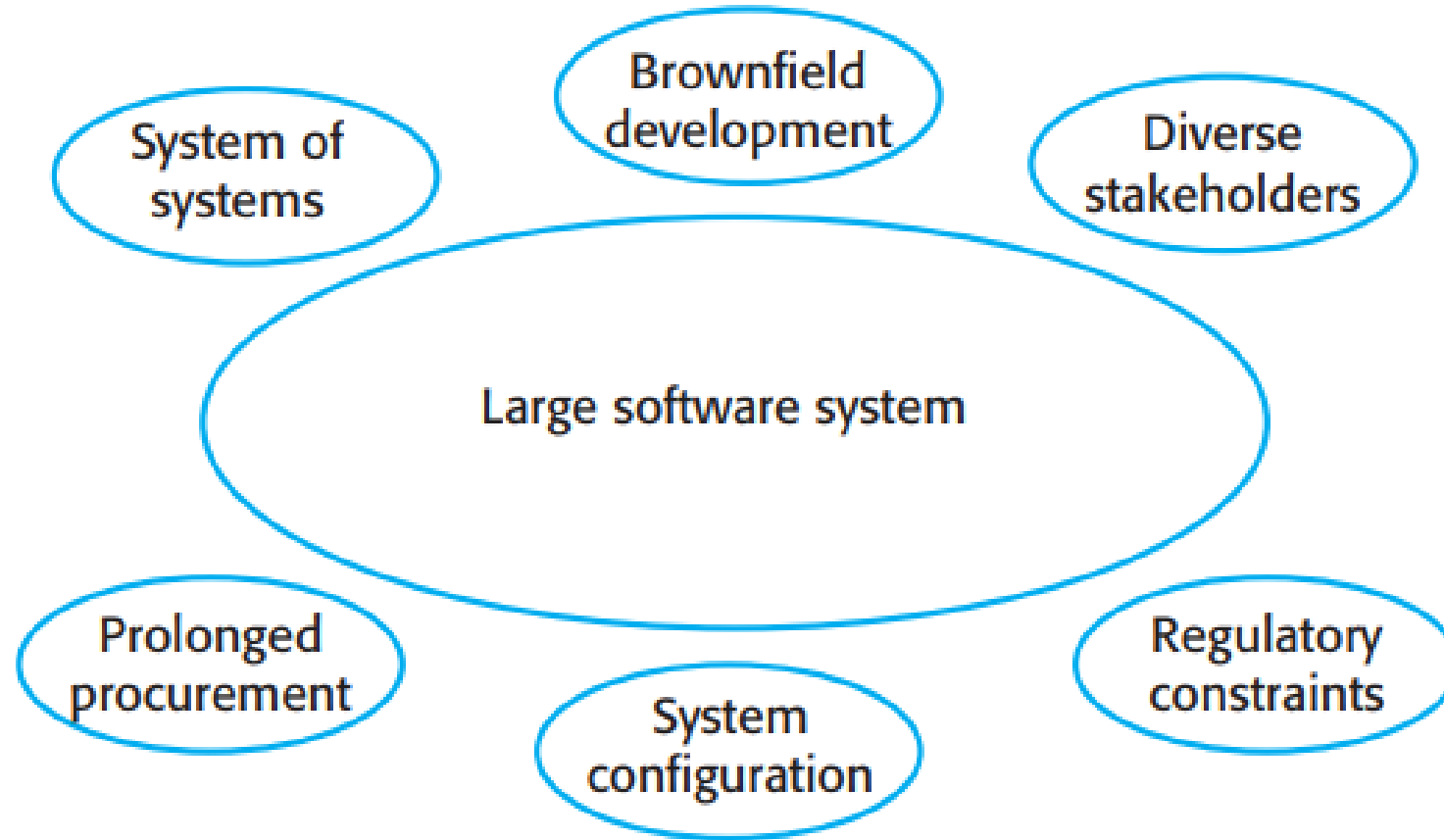
Agile and plan-driven methods

- Agile มีต้นกำเนิดมาในยุคต้นศตวรรษที่ 21 และมีคำประกาศที่ชัดเจน ในแนวทางที่ไม่ยอมรับ plan-driven software development
- หลายๆ หน่วยงาน เห็นประโยชน์ของ agile จึงได้พยายามปรับมาใช้ โดยให้เข้ากันได้กับวัฒนธรรมและแนวทางการดำเนินการแบบเดิมขององค์กร
 - บางที่ก็ใช้ agile ที่ปรับสเกลให้ใหญ่ขึ้น
 - บางที่ก็ดำเนินการแบบ plan-driven แต่มี process ย่อยเป็น agile

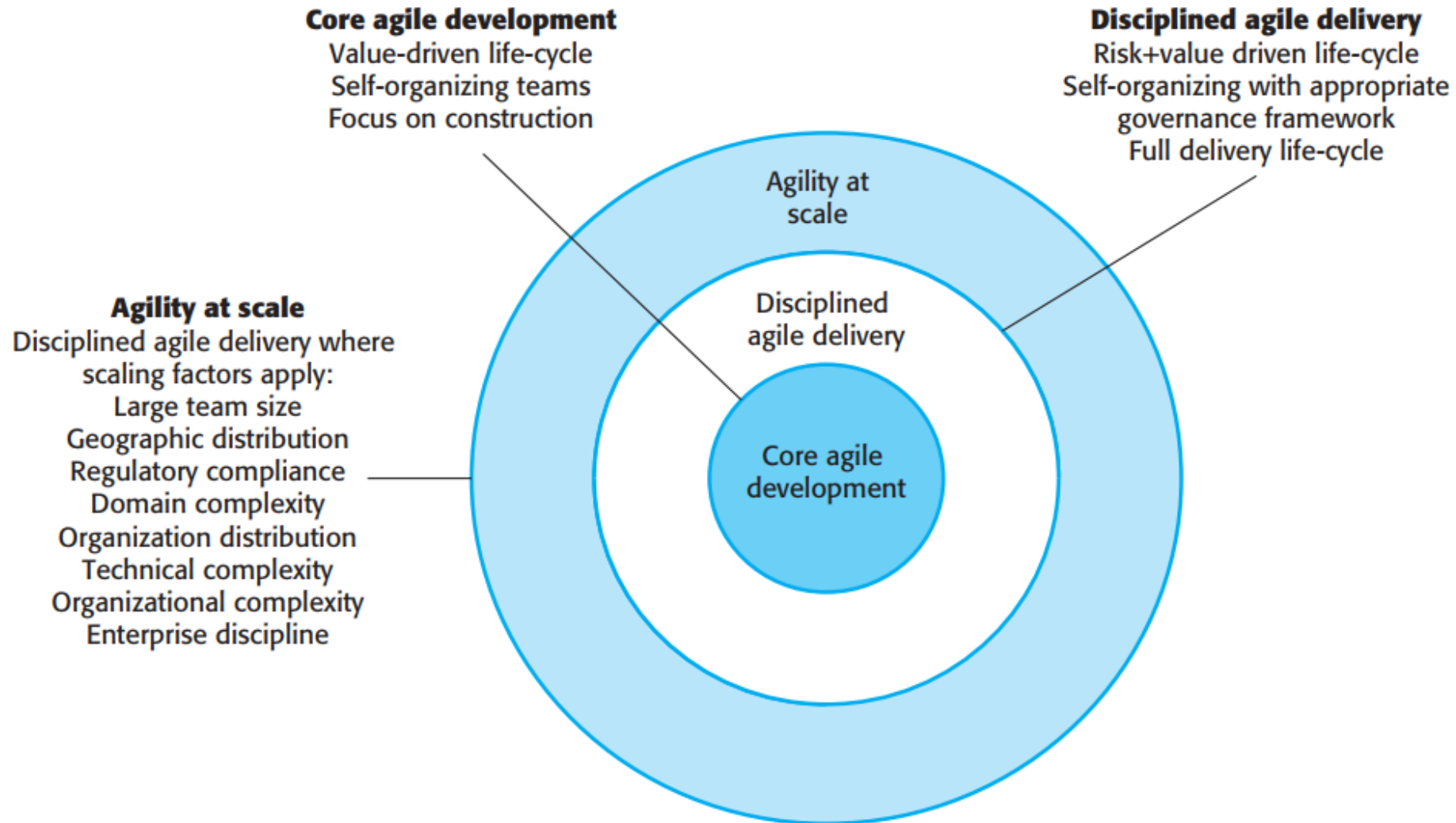
ปัจจัยที่ส่งผลต่อการตัดสินใจ plan-based vs agile



ลักษณะของโครงการซอฟต์แวร์ขนาดใหญ่



IBM's Agility at Scale model (© IBM 2010)



Key points

- วิธีการแบบ Agile คือวิธีการพัฒนาแบบ iteration ที่เน้นการลด overhead ในกระบวนการ เอกสารประกอบและการส่งมอบซอฟต์แวร์ โดยจะมีตัวแทนลูกค้ามาร่วมในกระบวนการพัฒนา
- การตัดสินใจว่าจะใช้แนวทาง agile หรือ plan-based ขึ้นอยู่กับประเภทของซอฟต์แวร์ที่กำลังพัฒนา ความสามารถของทีมพัฒนา และวัฒนธรรมของบริษัทผู้พัฒนาระบบ ในทางปฏิบัติ อาจใช้เทคนิค Agile และ Plan-based ผสมกัน

Key points

○แนวทางการพัฒนา Agile ประกอบด้วย

○ข้อกำหนดอยู่ในรูปแบบ user story

○การเขียนโปรแกรมแบบคู่ (pair programming)

○การปรับโครงสร้างใหม่ (refactor)

○การบูรณาการอย่างต่อเนื่อง (continuous integration)

○การพัฒนาแบบทดสอบก่อน (test-first development)

Key points

- Scrum เป็น framework ที่ใช้จัดระเบียบการพัฒนาซอฟต์แวร์แบบ agile
 - มี sprint เป็นศูนย์กลาง
 - มีเวลาสำหรับวงรอบการพัฒนาที่คงที่
 - แผนการทำงานจะขึ้นอยู่กับ backlog และมีการเลือกงานที่สำคัญที่สุดมาทำก่อน
- ในการปรับขนาด agile มักจะมีการนำ plan-driven มาผสมเข้ากับ agile ซึ่งมีงานที่เพิ่มขึ้นมาคือ
 - มี user เข้าร่วมทีมพัฒนาในจำนวนที่มากขึ้น เนื่องจากกระจายส่วนในการพัฒนา
 - มีเอกสารต่างๆ มากขึ้น
 - มีการแบ่งปันทรัพยากรและเครื่องมือระหว่างทีม
 - มีการกำหนดและจัดแบ่ง release ระหว่างทีม

