บทที่ 2
เอกสารและงานวิจัยที่เกี่ยวข้อง

ในการพัฒนาโปรแกรมระบบบุคลากรโดยใช้การโปรแกรมแบบเอกซ์ตรีม มีแนวทางและทฤษฎีด้านต่างๆที่ต้องนำมาประกอบการพัฒนาระบบ ดังนี้

2.1 อาไจลโมเดล (Agile Model)

2.2 การพัฒนาโปรแกรมแบบเอกซ์ตรีม

2.3 แนวทางปฏิบัติ 12 ข้อ ในการพัฒนาโปรแกรมแบบเอกซ์ตรีม

2.4 ขั้นตอนการพัฒนาโปรแกรมแบบเอกซ์ตรีม

2.5 ไอเอสโอ 12207 (ISO 12207) มาตรฐานสำหรับกระบวนการผลิตและพัฒนาซอฟต์แวร์

2.1 อาไจลโมเดล (Agile Model)

อาไจลโมเดล (Agile Model) [2] เป็นการพัฒนาจากโมเดลที่มีอยู่แล้วให้กระชับและทันต่อความเปลี่ยนแปลงในปัจจุบัน โดยออกแบบให้มีความรวดเร็ว ยืดหยุ่น พร้อมรับความเปลี่ยนแปลงเพื่อลดความเสี่ยงในการพัฒนาซอฟต์แวร์ โดยมีการแบ่งการพัฒนาออกเป็นไอเทอเรชั่น (iteration) เป็นการแบ่งเวลาออกเป็นช่วงๆ แต่ละช่วงไม่ควรเกิน 1 – 4 สัปดาห์ การพัฒนาจะดำเนินการอย่างต่อเนื่อง เมื่อมีความเปลี่ยนแปลงก็สามารถพัฒนาให้รองรับกับการเปลี่ยนแปลงนั้นได้อย่างไม่มีข้อจำกัด

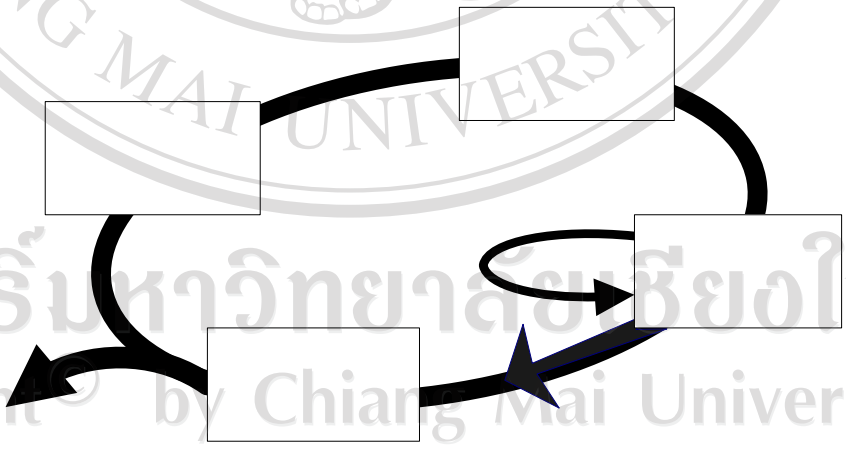
หลักการของอาไจลจะเน้นการสื่อสารภายในทีมงานและผู้ใช้มากกว่าการเน้นกระบวนการหรือเครื่องมือ การทำงานจะยึดที่ผลผลิตหรือตัวซอฟต์แวร์เป็นหลัก ไม่เน้นการทำเอกสาร แต่เน้นที่ความสัมพันธ์และการสื่อสารของทีมงานเป็นหลัก เพื่อให้ได้ความต้องการที่ครบถ้วนและพร้อมจะยอมรับการเพิ่มเติมความต้องการในภายหลัง

บางครั้งอาจมองว่าอาไจลเป็นส่วนขยายของกระบวนการพัฒนาซอฟต์แวร์แบบอื่นๆ ที่มีใช้เดิม โดยการเลือกเอากิจกรรมของอาไจลในส่วนที่สำคัญๆ แล้วนำมาปรับใช้ให้เหมาะสม สำหรับวิธีการของอาไจลในรูปแบบที่ควรรู้จัก ได้แก่ อาไจล ยูพี (Agile UP), เอกซ์พี (XP : Extreme Programming), เอฟดีดี (FDD : Feature Driven Development) และ สครัม (SCRUM)

2.2 การพัฒนาโปรแกรมแบบเอกซ์ตรีม

การพัฒนาโปรแกรมแบบเอกซ์ตรีม หรือ เอกซ์พี (Extreme Programming : XP) [3] เป็นกระบวนการพัฒนาซอฟต์แวร์ที่ได้กำหนดวิธีการที่จะทำให้ผู้ใช้และโปรแกรมเมอร์ทำงานร่วมกันในทีมเพื่อก่อให้เกิดการสื่อสารขึ้นภายในทีม การทำงานร่วมกันช่วยแก้ปัญหาจำนวนมากที่เกิดขึ้นระหว่างและหลังการสร้างซอฟต์แวร์ได้ เค็นต์ เบ็ก, วอร์ด คันนิงแฮม และรอน เจฟฟรีย์ ได้ร่วมพัฒนาเทคนิคนี้ขึ้นในปี 1996 โดยแนวคิดหลักจะเน้นไปที่การทำงานร่วมกัน (collaboration), การสื่อสาร (communication) และการปฏิบัติตามระเบียบ (discipline) เอกซ์พีเน้นที่การปฏิสัมพันธ์ จึงเหมาะสำหรับทีมงานที่มีขนาดเล็กจนถึงขนาดกลางที่มีสมาชิกไม่เกิน 15 คน แต่สำหรับโครงการที่มีขนาดใหญ่ก็สามารถใช้เทคนิคเอกซ์พีได้ โดยการแบ่งทีมพัฒนาออกเป็นทีมย่อย แล้วแต่ละทีมย่อยจึงใช้เทคนิคของเอกซ์พีเป็นการภายใน

เอก สุรเกียรติชานุกุล, เมสินี นาคมณี, สมนึก คีรีโต [4] ได้เสนองานวิจัยเรื่องการเปรียบเทียบกระบวนการซอฟต์แวร์แบบพีเอสพีและการโปรแกรมแบบเอกซ์ตรีม จากการทดสอบพบว่าเอกซ์พีจะเหมาะสมกับทีมขนาดเล็กที่ไม่ต้องการระเบียบแบบแผนในกระบวนการพัฒนามากนัก แต่ก็อาจจะมีปัญหาถ้านำไปใช้กับทีมที่มีขนาดใหญ่ ขณะที่การนำกระบวนการพัฒนาแบบพีเอสพีสามารถนำไปใช้กับทีมขนาดใหญ่หรือทีมขนาดเล็กที่ต้องการระเบียบแบบแผนในการทำงานมาก



รูปที่ 2.1 วงจรการทำงานของกระบวนการเอกซ์พี

เอกซ์พีนำเสนอแนวคิดในการแบ่งการพัฒนาออกเป็นกลุ่มของงานย่อย โดยแต่ละงานย่อยควรทำเสร็จภายในเวลาสั้นๆ และควรมีการพบปะระหว่างทีมพัฒนาในแต่ละวัน เพื่อ

ทดสอบงานที่ทำเสร็จ ถ้าหากมีการเปลี่ยนแปลงกับข้อกำหนดของซอฟต์แวร์ (software specification) ต้องมีการจัดการอย่างตามขั้นตอนที่กำหนดไว้เท่านั้น สำหรับตัวโครงการจะมีการพัฒนาแบบก้าวหน้า โดยโค้ดที่เขียนต้องทำงานในระดับของฟังก์ชั่นได้ รวมทั้งรายละเอียดการออกแบบ คุณสมบัติและโค้ด ต้องได้รับการดูแลควบคู่กันไปอย่างต่อเนื่อง เพื่อให้ได้ผลลัพธ์ตรงกับความต้องการของลูกค้ามากที่สุด กฎเกณฑ์และกิจกรรมของการโปรแกรมแบบเอกซ์ตรีม มีการแบ่งส่วนการทำงานเป็น 4 ส่วน ดังนี้

1. การวางแผน (Planning)

เขียนยูสเซอร์สตอรี โดยให้ผู้ใช้เขียนสิ่งที่ต้องการให้มีในระบบ มีลักษณะเป็นข้อความสั้นๆ ประมาณ 3 ประโยค ในภาษาของผู้ใช้เอง โดยไม่มีข้อมูลทางด้านเทคนิคเข้ามาเกี่ยวข้อง นำไปใช้ในการประมาณเวลาที่จะนำไปสร้างแผนการส่งมอบ สามารถนำมาใช้แทนเอกสารความต้องการได้

การประมาณเวลาในยูสเซอร์สตอรีที่เหมาะสม คือ แต่ละยูสเซอร์สตอรี ควรใช้เวลาประมาณ 1 – 3 สัปดาห์ ถ้าพบว่ายูสเซอร์สตอรีใดใช้เวลามากกว่า 3 สัปดาห์ ให้แบ่งการทำงานออกเป็นงานย่อยๆ อีก และถ้าใช้เวลาน้อยกว่า 1 สัปดาห์ ให้นำไปรวมกับสตอรีอื่นๆ จำนวนยูสเซอร์สตอรีที่เหมาะสมในแต่ละโครงการ ควรมีประมาณ 80 ± 20 ยูสเซอร์สตอรี

ใช้การวางแผนการส่งมอบในการสร้างตารางการทำงาน การวางแผนการส่งมอบ ทำได้โดยการจัดประชุมเพื่อวางแผนการส่งมอบงาน ในภาพรวมของโครงการก่อน จากนั้นจึงนำแผนการส่งมอบงานที่ได้ไปสร้างเป็นแผนการพัฒนาในแต่ละรอบการพัฒนา ประโยชน์ของการจัดประชุมนี้เพื่อให้ทีมพัฒนาสามารถประมาณเวลาที่คาดว่าจะใช้ในแต่ละยูสเซอร์สตอรี เวลาที่คาดว่าจะใช้นี้หมายถึง เวลาที่คิดว่าจะใช้พัฒนาแต่ละยูสเซอร์สตอรี โดยไม่รวมงานภายนอก เช่น เวลารับโทรศัพท์ หรืองานพิเศษอื่นๆ นอกเหนือจากการพัฒนาโปรแกรม แต่เวลาที่ใช้ทดสอบโปรแกรมต้องนำไปรวมในเวลาที่คาดว่าจะใช้ด้วย แล้วจึงให้ผู้ใช้กำหนดระดับความสำคัญในแต่ละยูสเซอร์สตอรี และเลือกยูสเซอร์สตอรีที่จะนำไปพัฒนาในการส่งมอบแต่ละครั้ง

ทำการส่งมอบให้บ่อยครั้ง และทำการวัดผลของโปรเจควีโลซิตี ซึ่งโปรเจควีโลซิตี คือ ตัววัดความก้าวหน้าของโครงการ การวัดโปรเจควีโลซิตีทำได้ง่ายๆ โดยนับจำนวนของยูสเซอร์สตอรีหรือ หรือนับจำนวนของงานที่เสร็จไปในการพัฒนาไอเทอเรชัน โปรเจควีโลซิตีในสตอรีที่พัฒนาเสร็จแล้ว สามารถนำมาใช้ประมาณจำนวนสตอรีที่จะทำได้

ไม่ควรนำโปรเจควีโลซิตีของโครงการต่างๆ มาเปรียบเทียบกัน เนื่องจากแต่ละโครงการมีความแตกต่างกัน เช่น สตอรีที่คล้ายกัน บางโครงการอาจให้ความสำคัญมาก บางโครงการให้ความสำคัญน้อย เป็นต้น หากพบว่าต้องมีการเปลี่ยนแปลงโปรเจควีโลซิตีมากกว่า 1 ไอเทอเรชัน

ควรใช้การประชุมวางแผนการส่งมอบทำการประมาณการหรือต่อรองเพื่อให้ได้แผนการส่งมอบใหม่

แบ่งโครงการออกเป็นไอเทอเรชัน โดยแผนการทำไอเทอเรชันจะมีการเริ่มในแต่ละไอเทอเรชัน มีการปรับตำแหน่งงานของผู้พัฒนาไปเรื่อยๆ และเริ่มต้นการทำงานแต่ละวันด้วยสแตนอัพมีทติง (Stand – up Meeting) โดยที่สแตนอัพมีทติง มีจุดประสงค์เพื่อให้ทีมมีการติดต่อสื่อสารกันอย่างสม่ำเสมอ สามารถปรึกษาและหาวิธีแก้ไขปัญหาต่างๆที่เกิดขึ้นได้

สแตนอัพมีทติงควรทำเป็นประจำทุกวันก่อนเริ่มทำงาน มีลักษณะเป็นการพูดคุยโดยใช้เวลาสั้นๆ ในการเสนอความก้าวหน้าของงาน หรือปัญหาที่เกิดขึ้น เพื่อขอความคิดเห็นจากทีมพัฒนา เมื่อพบปัญหาควรหาทางแก้ไขปัญหานั้นทันที

2. การออกแบบ (Designing)

เน้นการออกแบบให้เข้าใจง่าย ซึ่งสามารถทำให้งานเสร็จเร็วกว่าการออกแบบที่ซับซ้อน และหากพบว่างานส่วนไหนที่มีความซับซ้อนให้พยายามแทนที่ด้วยการออกแบบให้เข้าใจง่ายที่สุดเท่าที่จะทำได้

เลือกใช้เมตาฟอร์ ซึ่งเป็นการกำหนดรูปแบบการตั้งชื่อคลาสและเมธอดให้สื่อความหมาย การตั้งชื่อมีความสำคัญในการทำความเข้าใจถึงระบบที่ได้ออกแบบไว้ และมีประโยชน์เมื่อต้องนำกลับมาใช้อีกครั้ง ดังนั้น ควรเลือกและกำหนดการตั้งชื่อที่คนในทีมยอมรับว่าสามารถเข้าใจได้ง่าย และชื่อที่ตั้งไว้ทำให้เข้าใจตรงกันทุกคน อาจใช้ซีอาร์ซีการ์ด (CRC cards) สำหรับช่วยในการออกแบบให้เข้าใจได้ง่ายขึ้น

สร้างสไปค์ขึ้นมาใช้เพื่อลดความเสี่ยง สไปค์ คือ โปรแกรมง่ายๆ ที่สร้างขึ้นมาทดสอบการทำงานที่อาจเกิดความเสี่ยงได้ จุดมุ่งหมายของสไปค์ คือ ช่วยลดความเสี่ยงที่อาจเกิดจากปัญหาด้านเทคนิค หรือเพิ่มความน่าเชื่อถือในการประมาณค่าในยูสเซอร์สตอรี ไม่ควรเพิ่มฟังก์ชันการทำงานเอาไว้ล่วงหน้า

ทำการรีแฟคเตอร์ (Refactor) ทุกครั้งที่จะทำได้ รีแฟคเตอร์ คือ การทำให้โค้ดอยู่ในรูปแบบที่อ่านและทำความเข้าใจง่าย หลีกเลี่ยงการเขียนโค้ดที่ซับซ้อนและไม่ได้นำมาใช้ประโยชน์

3. การเขียนโปรแกรม (Coding)

ต้องเขียนโค้ดตามมาตรฐานที่วางไว้ โค้ดทุกส่วนต้องมีมาตรฐานการเขียนที่ตกลงกันภายในทีมพัฒนา เพื่อให้การเขียนโค้ดเป็นไปในรูปแบบเดียวกันตลอดการพัฒนา และเพื่อให้ทุกคนภายในทีมสามารถอ่านเข้าใจและทำการรีแฟคเตอร์โค้ดได้

มีการทำยูนิตเทสต์ (Unit Test) ก่อนการเขียนโค้ด การทำยูนิตเทสต์ก่อน พบว่า มีส่วนช่วยให้การเขียนโค้ดง่ายขึ้นและเร็วขึ้น โดยกำหนดผลลัพธ์ที่ต้องการ จากนั้นเขียนโค้ดแบบง่ายๆ เพื่อ

สร้างโปรแกรมขึ้นมา และให้โปรแกรมนั้นสามารถผ่านการทดสอบที่ได้ออกแบบเอาไว้ โค้ดที่เขียนต้องมีลักษณะเป็นโค้ดที่อ่านง่ายและรัดกุม นำไปใช้ทดสอบเฉพาะงานที่ต้องการเท่านั้น

ใช้การโปรแกรมแบบคู่ในการเขียนโปรแกรม การโปรแกรมแบบคู่ คือ การที่โปรแกรมเมอร์ 2 คน พัฒนาโปรแกรมโดยใช้เครื่องคอมพิวเตอร์เครื่องเดียวกันและนั่งทำงานพร้อมกัน มีประโยชน์ในการเพิ่มคุณภาพของซอฟต์แวร์ เนื่องจากโปรแกรมเมอร์ 2 คนจะทำหน้าที่ตรวจสอบการเขียนโค้ดระหว่างกันเอง

ทำการรวมโค้ดให้บ่อยครั้ง ผู้พัฒนาควรทำการรวมโค้ดให้บ่อยที่สุดเท่าที่จะเป็นไปได้ และไม่ควรปล่อยทิ้งไว้นานเกิน 1 วัน การรวมโค้ดบ่อยๆ ช่วยหลีกเลี่ยงหรือป้องกันการเกิดปัญหาได้ตั้งแต่เนิ่นๆ และบุคคลในทีมพัฒนาสามารถทำงานกับชิ้นงานที่เป็นเวอร์ชั่นล่าสุดเสมอ

ใช้วิธีการคอลเลคทีฟโค้ดโอนเนอร์ชิพ (Collective Code Ownership)

คอลเลคทีฟโค้ดโอนเนอร์ชิพ

- ทำการออปติไมซ์เป็นขั้นสุดท้าย

ไม่ทำงานล่วงเวลา การทำงานล่วงเวลาจะทำให้ความกระตือรือร้นในการทำงานลดลง แทนที่จะทำงานล่วงเวลาเพื่อให้งานเสร็จตามกำหนด แต่ควรปรับขอบเขตของโครงการหรือปรับตารางเวลาในแผนการทำงานจะดีกว่า และการเพิ่มคนภายหลังจากที่พบว่างานเร็จช้า จะยิ่งเป็นการเพิ่มปัญหามากขึ้น

4. การทดสอบ (Testing)

ทุกโค้ดต้องทำยูนิตเทสต์ การทำยูนิตเทสต์ มี 3 ขั้นตอน ดังนี้

ขั้นตอนที่หนึ่ง สร้างหรือดาวน์โหลดยูนิตเทสต์เฟรมเวิร์ค ที่สามารถสร้างชุดทดสอบแบบอัตโนมัติได้ เพื่อประหยัดเวลาในการทดสอบ

ขั้นตอนที่สอง ควรทำการทดสอบทุกคลาสที่อยู่ในระบบ ส่วนเมธอดที่มีรายละเอียดเล็กๆน้อยๆ สามารถละไว้ก่อน ยังไม่ต้องทดสอบก็ได้

ขั้นตอนที่สาม ควรสร้างแบบทดสอบขึ้นมาก่อนที่จะทำการเขียนโค้ด

ทุกโค้ดต้องผ่านการทดสอบยูนิตเทสต์ก่อนทำงานส่งมอบ โค้ดส่วนใดที่ยังไม่ผ่านการทดสอบ จะไม่สามารถส่งมอบออกไปได้ และในขณะที่ทำยูนิตเทสต์แล้วพบว่ามีส่วนที่ยูนิตเทสต์นั้นไม่ครอบคลุมปัญหาที่เกิดขึ้น ให้ทำการสร้างยูนิตเทสต์เพื่อทดสอบปัญหาที่พบนั้นทันที

เมื่อพบบักส์ (Bugs) ต้องสร้างกรณีทดสอบทุกครั้ง เมื่อมีการพบบักส์ ต้องสร้างกรณีทดสอบขึ้นมาเพื่อป้องกันการเกิดซ้ำ การสร้างแอคเซปแตนซ์เทสต์ขึ้นก่อนการค้นหาบักส์ จะช่วยให้ผู้ใช้สามารถกำหนดแนวทางปัญหาที่พบได้ และสามารถเสนอปัญหานั้นให้กับโปรแกรมเมอร์อย่างถูกต้อง

มีการทำแอคเซปแตนซ์เทสต์ให้บ่อยครั้ง และนำเสนอคะแนนที่ได้จากการทดสอบให้ทีมพัฒนารับทราบ แอคเซปแตนซ์เทสต์ถูกสร้างมาจากยูสเซอร์สตอรีที่ได้การเลือกไว้ในขณะที่มีการประชุมวางแผนการทำไอเทอเรชัน ผู้ใช้ระบุซีนาริโอที่จะนำมาทดสอบเมื่อยูสเซอร์สตอรีพัฒนาได้ถูกต้อง หนึ่งสตอรีสามารถมีแอคเซปแตนซ์เทสต์ได้ 1 หรือหลายๆ แอคเซปแตนซ์เทสต์ได้

แอคเซปแตนซ์เทสต์ เป็นการทดสอบแบบแบล็คบ็อกซ์ (Black box testing) คือ การทดสอบที่สนใจแต่ผลลัพธ์ที่ได้ แต่ไม่สนใจการทำงานที่อยู่ภายใน ผู้ใช้จะเป็นผู้ทำแอคเซปแตนซ์เทสต์เอง เพื่อตรวจสอบความถูกต้อง และให้คะแนนเพื่อประเมินผลการทดสอบ

2.3 แนวทางปฏิบัติ 12 ข้อ ในการพัฒนาโปรแกรมแบบเอกซ์ตรีม

การพัฒนาโปรแกรมแบบเอกซ์ตรีม มีข้อปฏิบัติที่สามารถนำมาใช้เป็นแนวทางในการพัฒนาซอฟต์แวร์ 12 ข้อ ดังนี้ [5]

1. การทดสอบ (Test Driven Development)

ให้เขียนการทดสอบหน่วยย่อย (unit tests) ก่อนและระหว่างที่มีการเขียนโปรแกรม เพื่อให้สามารถแก้ไขได้ทันที เมื่อมีการเปลี่ยนแปลงเกิดขึ้น

2. การส่งมอบงานขนาดเล็ก (Small Release)

การส่งมอบงานขนาดเล็กมีส่วนช่วยให้มั่นใจได้ว่าผู้พัฒนาและผู้ใช้งานสามารถทำงานร่วมกันได้ และเปิดรับความคิดเห็นระหว่างกันได้อย่างต่อเนื่อง

3. รีแฟคเตอริ่ง (Refactoring)

รีแฟคเตอริ่งเป็นการยอมให้แก้ไขรูปแบบของโปรแกรมตลอดเวลา เพื่อให้มีความต่อเนื่องและเข้าใจง่าย เพื่อลดความซ้ำซ้อนและความซับซ้อนที่เกิดขึ้นในการเขียนโปรแกรม

4. ออกแบบให้เข้าใจง่าย (Simple Design)

เป็นการลดความซับซ้อน ซึ่งในส่วนนี้มีความสำคัญมากเท่ากับการเขียนโปรแกรม

5. การวางแผน (Planning Game)

สร้างแผนการทำงานให้สามารถเริ่มพัฒนาได้ และให้ทำการแก้ไขแผนการทำงานทุกๆ ครั้งหลังจากส่งมอบงานย่อย

6. การเขียนโปรแกรมคู่ (Pair Programming)

ในการเขียนโค้ดทุกส่วนต้องใช้ผู้พัฒนาสองคน โดยใช้เครื่องคอมพิวเตอร์เครื่องเดียวกัน ซึ่งทุกคนสามารถเป็นเจ้าของโค้ดร่วมกัน และยังช่วยให้พบข้อผิดพลาดตั้งแต่เนิ่นๆ ด้วย

7. ออนไซต์คัสตอมเมอร์ (Onsite Customer)

เป็นการเพิ่มความมั่นใจว่าปัญหาหรือข้อสงสัยจะได้รับการตอบในทันที เนื่องจากมีผู้ให้ข้อมูลร่วมทำงานกับผู้พัฒนา

8. ซิสเต็มเมตาฟอร์ (System Metaphor)

ซิสเต็มเมตาฟอร์เป็นรูปแบบที่ใช้ในการพัฒนา ซึ่งทุกคนในทีมพัฒนาทั้งผู้พัฒนา ผู้ใช้งาน และผู้จัดการโครงการต้องเข้าใจตรงกัน และนำมาใช้เป็นข้อมูลพื้นฐานในโครงการ

9. คอลเลคทีฟโค้ดโอนเนอร์ชิพ (Collective Code Ownership)

คอลเลคทีฟโค้ดโอนเนอร์ชิพช่วยให้มั่นใจได้ว่าจะไม่มีสมาชิกในทีมคนไหนที่พัฒนาในจุดที่เสี่ยง หรือมีการเก็บความรู้ไว้คนเดียว ทุกส่วนการทำงานต้องมีการใช้งานร่วมกัน ดังนั้นจึงถือว่าทุกคนเป็นเจ้าของงานร่วมกัน

10. การรวบรวมอย่างต่อเนื่อง (Continuous Integration)

งานส่วนที่มีการเปลี่ยนแปลงจะถูกนำไปแทนที่งานเก่าทันที และให้ทำการอัพเดตงานให้บ่อยครั้งเท่าที่จะทำได้ เพื่อให้ผู้พัฒนาได้ทำงานกับงานที่เป็นเวอร์ชั่นล่าสุดเสมอ การทำเช่นนี้จะช่วยให้พบปัญหาได้เร็วขึ้น

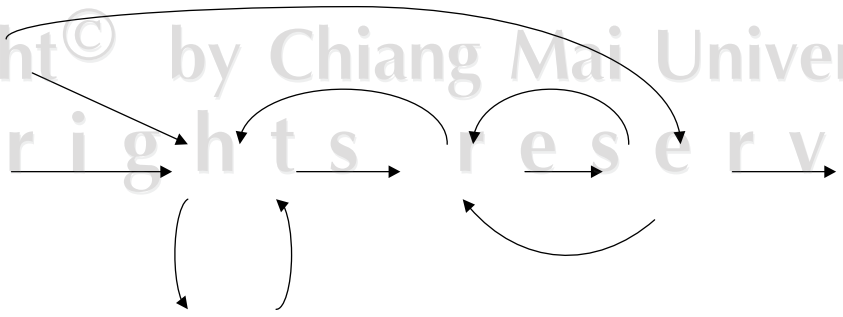
11. แบบแผนการเขียนโค้ด (Coding Conventions)

การตรวจสอบโค้ดให้มีแบบแผนที่ตรงกันเป็นการเพิ่มการสื่อสารภายในทีม และช่วยลดเวลาการอ่านโค้ดในส่วนอื่นๆ

12. ซัสเทนเนเบิลเพส (Sustainable Pace)

ควรใช้เวลาทำงานเพียง 40 ชั่วโมงต่อสัปดาห์ เพื่อให้มีความสมดุลกันระหว่างเวลาในการทำงานและเวลาพักผ่อน เป็นการช่วยลดความเครียดและกระตุ้นให้มีพลังในการทำงานในสัปดาห์ต่อไป

2.4 ขั้นตอนการพัฒนาโปรแกรมแบบเอกซ์ตรีม [6]



รูปที่ 2.2 ภาพรวมการพัฒนาโปรแกรมแบบเอกซ์ตรีม

ขั้นตอนที่ 1 จัดทำยูสเซอร์สตอรี (User Stories)

ยูสเซอร์สตอรีใช้ในการประมาณเวลาสำหรับการประชุมจัดทำแผนการส่งมอบ ซึ่งยูสเซอร์สตอรีต้องเขียนโดยลูกค้าถึงสิ่งที่ต้องการให้มีในระบบ คล้ายกับซีนาริโอ แต่ไม่มีข้อจำกัดในด้านการอธิบายส่วนติดต่อผู้ใช้งาน ยูสเซอร์สตอรีมีลักษณะเป็นข้อความประมาณ 3 ข้อความที่เขียนด้วยภาษาของผู้ใช้ โดยไม่มีเรื่องเกี่ยวกับเทคโนโลยีเข้ามาเกี่ยวข้อง

ยูสเซอร์สตอรียังใช้สร้าง แอคเซปแตนซ์เทสต์ (Acceptance Test) เพื่อนำไปใช้ทวนสอบกับสตอรีที่พัฒนาเสร็จแล้ว

รายละเอียดของยูสเซอร์สตอรีใช้เพื่อกำหนดระยะเวลาในการพัฒนาแต่ละสตอรีเท่านั้น เมื่อถึงเวลาผู้พัฒนาจะพูดคุยกับผู้ใช้เพื่อซักถามถึงรายละเอียดเพิ่มเติม

ผู้พัฒนาประมาณเวลาที่ใช้พัฒนาในแต่ละสตอรี แต่ละสตอรีต้องใช้เวลา 1 – 3 สัปดาห์ เวลาที่ประมาณขึ้นนี้ เป็นเวลาที่คาดว่าจะใช้ในการสร้างแต่ละสตอรี โดยไม่มีเหตุการณ์ใดๆ มารบกวนการทำงาน หากพบว่าสตอรีใดใช้เวลามากกว่า 3 สัปดาห์ ให้แบ่งออกเป็นงานย่อยอีก และหากสตอรีใดใช้เวลาน้อยกว่า 1 สัปดาห์ให้นำไปรวมกับสตอรีอื่น จำนวนสตอรีที่เหมาะสมในการสร้างเป็นแผนการส่งมอบ คือ 80±20

สิ่งที่แตกต่างระหว่างสตอรีและเอกสารความต้องการ คือ ความสนใจในความต้องการของผู้ใช้ การเขียนยูสเซอร์สตอรีควรหลีกเลี่ยงการลงรายละเอียดเกี่ยวกับเทคโนโลยีที่ใช้ รูปแบบของฐานข้อมูล และอัลกอริทึม ควรเก็บเฉพาะสตอรีที่เน้นไปที่ความต้องการของผู้ใช้เท่านั้น

ขั้นตอนที่ 2 วางโครงสร้างของสไปค์ (Architectural Spike)

การสร้างสไปค์ ทำเพื่อหาคำตอบของปัญหาทางด้านเทคนิคและด้านการออกแบบ สไปค์มีลักษณะเป็นโปรแกรมง่ายๆ สร้างขึ้นมาเพื่อใช้ระบุปัญหาที่เกิดขึ้นภายใต้การทดสอบ และสามารถนำผลที่ได้ทิ้งไป ส่วนใหญ่แล้วเราไม่จำเป็นต้องเก็บสไปค์เอาไว้ จุดประสงค์ของการทำสไปค์ คือ ลดความเสี่ยงที่เกิดจากปัญหาด้านเทคนิค และเพิ่มความน่าเชื่อถือในการประมาณค่าในยูสเซอร์สตอรี

เมื่อเกิดปัญหาที่แก้ไขได้ยาก ควรใช้ผู้พัฒนาเป็นคู่ในการแก้ไขปัญหานั้น โดยใช้เวลาประมาณ 1 – 2 สัปดาห์ และยังลดความเสี่ยงที่จะเกิดขึ้นด้วย

ขั้นตอนที่ 3 วางแผนการส่งมอบ (Release Planning)

การประชุมวางแผนการส่งมอบใช้สำหรับการสร้างแผนการส่งมอบในภาพรวมของระบบ ซึ่งแผนการส่งมอบนี้จะนำไปใช้สร้างเป็นแผนการทำไอเทอเรชัน สำหรับแต่ละไอเทอเรชัน

การวางแผนการส่งมอบเป็นกลุ่มของกฎเกณฑ์ที่อนุญาตให้ทุกคนที่มีความเกี่ยวข้องกับการพัฒนาใช้เพื่อทำการตัดสินใจต่างๆ กฎเกณฑ์นี้จะกำหนดขั้นตอนการต่อรองตารางเวลาเพื่อให้ทุกคนสามารถยอมรับได้

ความสำคัญของการประชุมวางแผนการส่งมอบสำหรับทีมพัฒนา คือ การประมาณเวลาที่คาดว่าจะใช้ในแต่ละยูสเซอร์สตอรี ซึ่งเวลาที่คาดว่าจะใช้นี้เป็นการประมาณเวลาที่ใช้ในการพัฒนาโดยคิดว่าจะไม่มีงานอื่น หรือปัจจัยอื่นๆ ภายนอกมารบกวนการทำงาน แต่เวลาที่คาดว่าจะใช้ต้องรวมเวลาที่ใช้ในการทดสอบเข้าไปด้วย จากนั้นผู้ใช้จะตัดสินใจว่าสตอรีไหนมีความสำคัญมากที่สุดในการนำมาพัฒนาให้แล้วเสร็จ

ยูสเซอร์สตอรีที่พิมพ์หรือเขียนลงบนการ์ดแล้ว นำมาวางบนโต๊ะ ให้ผู้ใช้และผู้พัฒนาเลือกกลุ่มของสตอรีที่ต้องการพัฒนาในการส่งมอบครั้งแรก (หรือครั้งต่อไป) สตอรีที่ทำงานรือผ่านการทดสอบตามที่ต้องการแล้วต้องทำการส่งออกไป (delivery) ก่อน

เราสามารถวางแผนโดยใช้เวลาหรือขอบเขตได้ เมื่อนำโปรเจควีโลซิตี (Project Velocity) ซึ่งใช้ในการตัดสินจำนวนยูสเซอร์สตอรีที่สามารถพัฒนาได้ก่อนวันที่กำหนด (เวลา) หรือ ใช้เวลาในการพัฒนาแค่ไหนจึงจะแล้วเสร็จ (ขอบเขต) เมื่อต้องการการวางแผนโดยใช้เวลา นำจำนวนรอบของการพัฒนา (iterations) คูณด้วยโปรเจควีโลซิตี เพื่อกำหนดจำนวนยูสเซอร์สตอรีที่สามารถพัฒนาเสร็จได้ และเมื่อต้องการวางแผนโดยใช้ขอบเขต ให้นำผลรวมของเวลาที่กำหนดในยูสเซอร์สตอรีมาหารด้วยโปรเจควีโลซิตี เพื่อกำหนดจำนวนรอบที่จะใช้พัฒนา

รายละเอียดของแต่ละไอเทอเรชัน จะกำหนดก่อนการเริ่มพัฒนาแต่ละไอเทอเรชัน การประชุมวางแผนการส่งมอบ จะถูกเรียกว่า แพลนนิ่งเกม (Planning Game)

เมื่อสร้างแผนการส่งมอบสุดท้ายแล้ว และพบว่าเกิดความไม่พอใจขึ้น ควรมีการต่อรองเพื่อปรับเปลี่ยนแผนการส่งมอบจนกว่าทุกฝ่ายจะพอใจ

หลักในการวางแผนการส่งมอบ คือ การกำหนดปริมาณของโครงการด้วย 4 ตัวแปร และผู้จัดการโครงการสามารถเลือกได้เพียง 3 ใน 4 เท่านั้น ดังนี้

- ขอบเขต คือ ต้องทำอะไรบ้าง
- ทรัพยากร คือ มีใครทำงานบ้าง
- เวลา คือ เมื่อไหร่ที่จะเริ่มงาน
- คุณภาพ คือ ความสามารถของโปรแกรมที่ผ่านการทดสอบแล้ว

ขั้นตอนที่ 4 การสร้างสไปค์ (Spike)

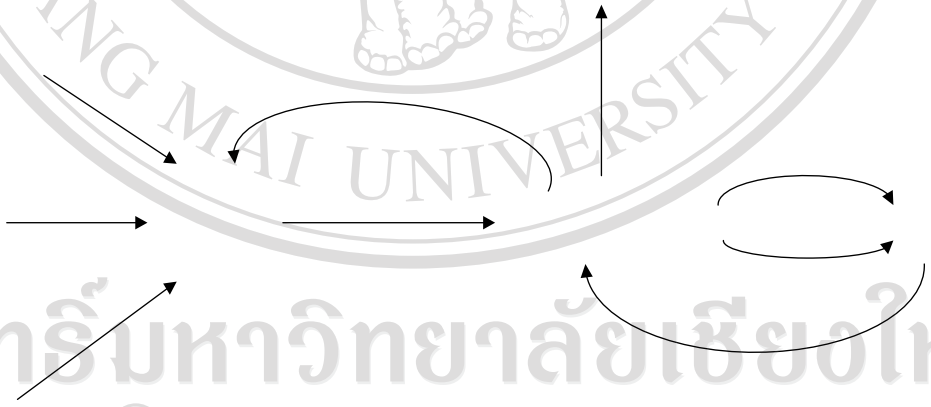
เป็นการดำเนินการระหว่างการวางแผนการส่งมอบงาน อาจมีการสร้างสไปค์เพิ่ม หรือแก้ไข เมื่อมีการประมาณการณ์บางสิ่งคลาดเคลื่อนไป หรือเกิดปัญหาที่นอกเหนือความคาดหมาย ก็สามารถกลับมาแก้ไขสไปค์ได้

ขั้นตอนที่ 5 การพัฒนาไอเทอเรชัน (Iterative Development)

การพัฒนาด้วยไอเทอเรชันเป็นการเพิ่มความสามารถให้กับขั้นตอนในการพัฒนา โดยแบ่งตารางการพัฒนาออกเป็นไอเทอเรชันย่อยๆ ที่ใช้เวลาในแต่ละไอเทอเรชัน ประมาณ 1 – 3 สัปดาห์ ไม่ควรกำหนดเวลาล่วงหน้าในส่วนการเขียนโปรแกรม ควรวางแผนการทำไอเทอเรชันตอนเริ่มในแต่ละไอเทอเรชัน เพื่อกำหนดสิ่งที่ต้องทำ

มีกฎว่าให้มองข้ามและอย่าพยายามพัฒนาในสิ่งที่ยังไม่ได้กำหนดในแต่ไอเทอเรชันนั้นๆ ยังพอมีเวลาที่จะพัฒนาส่วนนั้นๆได้ถ้ามันมีความสำคัญในแผนการส่งมอบ ถ้าไม่เพิ่มการทำงานหรือคิดการทำงานไปล่วงหน้าก่อน เราจะสามารถเปลี่ยนความต้องการของผู้ใช้ได้ง่ายขึ้น

ในขั้นตอนนี้ มีขั้นตอนที่แยกย่อยออกไปอีก ดังรูปต่อไปนี้



รูปที่ 2.3 การพัฒนาไอเทอเรชั่น

ขั้นตอนที่ 5.1 การวางแผนการพัฒนา (Iteration Planning)

ในขั้นตอนนี้จะมีการพิจารณาข้อมูลส่วนต่างๆ ดังนี้

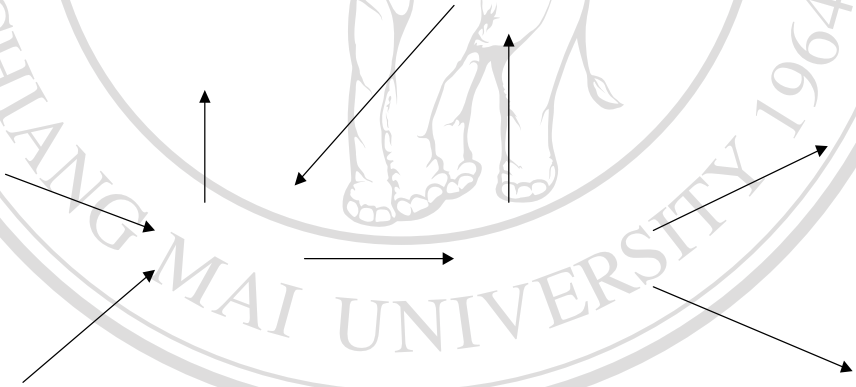
1. จากขั้นตอนการวางแผนการส่งมอบ จะเป็นตัวกำหนดยูสเซอร์สตอรีว่าสตอรีใดบ้างที่จะเข้าสู่การพัฒนาในแต่ละรอบ

2. ในรอบถัดไปของการพัฒนาแต่ละรอบ ต้องกำหนดโปรเจควีโลซิตี้ให้กับการวางแผนการพัฒนาด้วย
3. ข้อมูลข้อผิดพลาดหรือบักส์ที่พบระหว่างการทดสอบโปรแกรม จะใช้เป็นข้อมูลในการวางแผนการพัฒนาแต่ละรอบด้วย

แผนการพัฒนาในแต่ละไอเทอเรชัน (Iteration Plan) ที่ได้ออกมา จะนำไปเข้าสู่ขั้นตอนการพัฒนาโปรแกรม (Development) (อธิบายรายละเอียดเพิ่มเติมที่ขั้นตอนที่ 5.2) ซึ่งส่วนของการเรียนรู้และสื่อสารระหว่างกันในขั้นตอนนี้อาจทำให้พบกับยูสเซอร์สตอรีใหม่ๆ ทำให้ต้องกำหนดโปรเจควีโลซิตี้ใหม่ ในส่วนของงานที่ยังไม่เสร็จในขั้นตอนการพัฒนาโปรแกรมจะย้อนกลับมาสู่ขั้นตอนการวางแผนการพัฒนาอีกครั้ง

ขั้นตอนที่ 5.2 การพัฒนาโปรแกรม (Development)

ในขั้นตอนนี้ จะมีขั้นตอนย่อย ดังรูปต่อไปนี้



รูปที่ 2.4 การพัฒนาโปรแกรม

ขั้นตอนที่ 5.2.1 การทำสแตนอัพมีทติ้ง (Stand Up Meeting)

การทำสแตนอัพมีทติ้ง จะถูกกำหนดจากสิ่งต่อไปนี้ คือ ซอฟต์แวร์รุ่นล่าสุดที่ได้มาวันต่อวัน ซึ่งไม่ผ่านการทดสอบโปรแกรม และงานที่ถูกกำหนดไว้ในการวางแผนการพัฒนา การทำสแตนอัพมีทติ้งสามารถช่วยในการแยกแยะงานที่ยังทำไม่สำเร็จ เนื่องจากงานมีขนาดใหญ่เกินไปได้

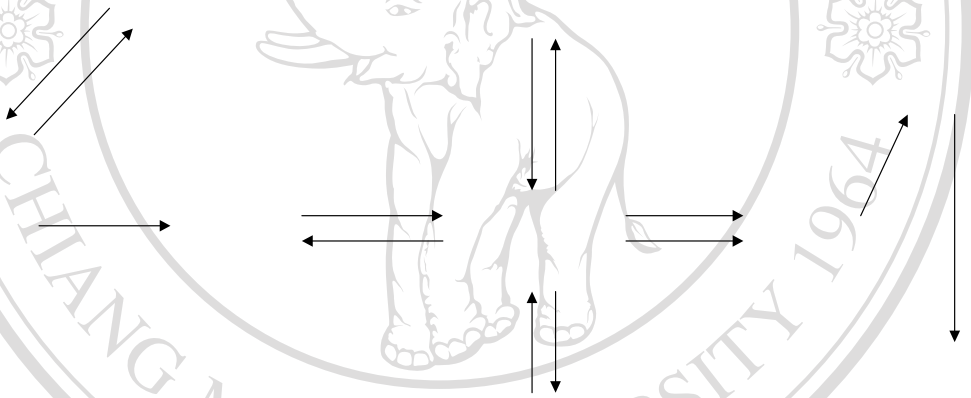
งานต่อไป หรืองานที่ไม่ผ่านการทดสอบจะนำเข้าสู่กระบวนการคอลเลคทีฟโค้ดโอนเนอร์ชิพ (Collective Code Ownership) (อธิบายเพิ่มเติมที่ขั้นตอนที่

5.2.2) ในส่วนนี้จะมีการจะมีการเรียนรู้และสื่อสารระหว่างกัน เพื่อให้ได้ข้อมูลที่จะนำมาใช้ร่วมกันภายในทีมพัฒนา

ซอฟต์แวร์ที่ผ่านขั้นตอนการทำคอลเลคทีฟโค้ดโอนเนอร์ชิพแล้ว จะได้ซอฟต์แวร์ที่มีฟังก์ชั่นใหม่เพิ่มเข้ามา และต้องผ่านการทดสอบการทำงานแล้วอย่างครบถ้วน หรือเป็นซอฟต์แวร์ที่ผ่านการแก้ไขและทำการทดสอบเรียบร้อยแล้ว

ขั้นตอนที่ 5.2.2 การทำคอลเลคทีฟโค้ดโอนเนอร์ชิพ (Collective Code Ownership)

มีการทำงาน ดังรูปต่อไปนี้



รูปที่ 2.5 คอลเลคทีฟโค้ดโอนเนอร์ชิพ

มีการใช้ซีอาร์ซีการ์ด (CRC Cards) ช่วยในการออกแบบให้ง่ายขึ้น เพื่อให้ได้งานต่อไป ส่วนงานที่ไม่ผ่านการทดสอบเนื่องจากงานมีความซับซ้อนเกินไป ก็นำไปออกแบบใหม่โดยใช้ซีอาร์ซีการ์ด จากนั้นส่งงานที่ได้ไปยังการสร้างแบบทดสอบการทำงาน โดยให้มีการจับคู่กันของผู้พัฒนาก่อนเริ่มทำการพัฒนา

จากผลลัพธ์ที่ได้จะเริ่มเข้าสู่การพัฒนาโปรแกรมแบบคู่ (Pair Programming) หากพบว่าโค้ดที่เขียนซับซ้อนเกินไปให้ทำการรีแฟคเตอร์ (Refactor) เพื่อให้ได้โค้ดที่อ่านง่ายขึ้น หากการพัฒนายังไม่ก้าวหน้าเท่าที่ควรและต้องการความช่วยเหลือ ให้ทำการเปลี่ยนคู่หรือจับคู่ใหม่ของผู้พัฒนาซอฟต์แวร์

CRC Cards

Single Design

Complex Problem

Next Task Pair

Failed Unit Test

ซอฟต์แวร์ที่ได้จากการพัฒนาโปรแกรมแบบคู่ จะเป็นซอฟต์แวร์ที่ใช้แบบทดสอบการทำงานใหม่ หรือมีฟังก์ชั่นใหม่เพิ่มเข้ามา เพื่อนำไปเข้าสู่กระบวนการรวบรวมอย่างต่อเนื่องต่อไป

จากนั้นนำซอฟต์แวร์ที่ได้ไปทดสอบกับแบบทดสอบการทำงาน ถ้าผ่านก็จะได้ซอฟต์แวร์ที่ผ่านการทดสอบการทำงานอย่างสมบูรณ์ เพื่อรอไปทำการทดสอบแบบแอคเซปแตนซ์เทสต์ต่อไป

ขั้นตอนที่ 6 แอคเซปแตนซ์เทสต์ (Acceptance Test)

แอคเซปแตนซ์เทสต์สร้างมาจากยูสเซอร์สตอรี ที่ได้เลือกไว้ในการประชุมวางแผนการทำไอเทอเรชัน แล้วนำไปแปลงเป็นแอคเซปแตนซ์เทสต์ ผู้ใช้เลือกซีนาริโอเพื่อทดสอบยูสเซอร์สตอรีที่พัฒนาเสร็จแล้ว สตอรีสามารถมีได้ 1 หรือหลายแอคเซปแตนซ์เทสต์ เพื่อให้มั่นใจในการทดสอบว่าสามารถทำงานได้จริง

แอคเซปแตนซ์เทสต์มีลักษณะเป็นแบล็คบอกซ์ (Black Box) แต่ละแอคเซปแตนซ์เทสต์จะเสนอผลที่คาดว่าจะได้จากระบบ ผู้ใช้สามารถตรวจสอบความถูกต้องและให้คะแนนเพื่อตัดสินผลการทดสอบได้

ยูสเซอร์สตอรีจะไม่สมบูรณ์หากไม่ได้ทำแอคเซปแตนซ์เทสต์ หมายถึงว่าจะมีการสร้างแอคเซปแตนซ์เทสต์ใหม่ขึ้นในแต่ละไอเทอเรชัน

การประกันคุณภาพ (Quality Assurance) เป็นส่วนสำคัญในการโปรแกรมแบบเอกซ์ตรีม ในบางโครงการการประกันคุณภาพจะทำโดยทีมงานที่แยกต่างหาก ในขณะที่การประกันคุณภาพอื่นจะทำโดยทีมพัฒนาเอง

แอคเซปแตนซ์เทสต์ควรเป็นการทดสอบแบบอัตโนมัติ เพื่อที่จะสามารถทดสอบได้บ่อยครั้ง คะแนนในการทดสอบต้องนำมารายงานให้ทีมพัฒนารับทราบ เพื่อทำการเปลี่ยนแปลงตารางเวลา ในกรณีที่พบปัญหาในการทดสอบและต้องทำการแก้ไข

แอคเซปแตนซ์เทสต์ได้เปลี่ยนชื่อมาจากฟังก์ชันนอลเทสต์ (Functional Test) ซึ่งเป็นการรับประกันว่าสามารถเก็บความต้องการจากผู้ใช้ได้ทั้งหมด และระบบได้รับการยอมรับ

ขั้นตอนที่ 7 การส่งมอบ (Small Release)

ผู้พัฒนาต้องการส่งมอบงานแต่ละไอเทอเรชันให้กับผู้ใช้ได้บ่อยครั้งที่สุด การประชุมวางแผนการส่งมอบใช้เพื่อให้ครอบคลุมหน่วยการทำงานเล็กๆ ทุกหน่วยและสามารถส่งมอบให้แก่ผู้ใช้ได้ในขั้นต้นของโครงการ เป็นการยากที่จะได้ผลตอบรับภายในเวลาเพื่อนำมาปรับการพัฒนา ดังนั้นจึงเป็นการดีกว่าหากนำเสนองานแก่ผู้ใช้โดยตรง จะสามารถรับผลตอบรับเพื่อนำมาปรับปรุงได้ทันที

2.5 ไอเอสโอ 12207 (ISO 12207) มาตรฐานสำหรับกระบวนการผลิตและพัฒนาซอฟต์แวร์

มาตรฐาน ISO 12207 Software Life Cycle Process เป็นกระบวนการควบคุมมาตรฐานระบบบสารสนเทศ ที่ใช้ในการควบคุมวงจรชีวิตของระบบสารสนเทศทั้งวงจร มีการแบ่งการทำงานทั้งหมดเป็น 17 กระบวนการ (Process) 74 กิจกรรม (Activity) 232 งาน (Task) 154 เอกสาร (Artifact) มีการจัดกระบวนการที่เกิดขึ้นเป็น 3 กลุ่ม ดังนี้

1. กระบวนการของวัฏจักรหลัก (Primary Life Cycle Process) มี 5 กระบวนการ
2. กระบวนการของวัฏจักรสนับสนุน (Supporting Life Cycle Process) มี 8 กระบวนการ
3. กระบวนการของวัฏจักรการจัดระบบ (Organizational Life Cycle Process) มี 4 กระบวนการ

แต่ในการพัฒนาระบบบุคลากรของนี้ จะเลือกใช้กิจกรรมหลัก 15 กิจกรรม [7] ดังต่อไปนี้

กระบวนการได้มา (Acquisition process group)

1. กิจกรรมการจัดซื้อจัดจ้าง จัดทำแผนการจัดซื้อจัดจ้าง โดยมีการกำหนดรายละเอียดและงบประมาณในการซื้ออุปกรณ์ที่ใช้ในโครงการ

กระบวนการวิศวกรรม (Engineering process group)

2. กิจกรรมการสำรวจความต้องการ (Requirement Elicitation) จัดทำแผนการสำรวจความต้องการ กำหนดขั้นตอนและวิธีการที่ใช้สำรวจความต้องการ รวมถึงเป้าหมายและขอบเขตของการสำรวจความต้องการด้วย
3. กิจกรรมการวิเคราะห์ความต้องการของระบบ (System Requirement Analysis) จัดทำแผนการวิเคราะห์ความต้องการของระบบเลือกรูปแบบและกำหนดขั้นตอนที่จะใช้วิเคราะห์ระบบ อาจมีการเลือกเครื่องมือหรือมาตรฐานที่ใช้ในการช่วยวิเคราะห์ระบบก็ได้
4. กิจกรรมการออกแบบสถาปัตยกรรมของระบบ (System Architectural Design) จัดทำแผนการออกแบบสถาปัตยกรรม กำหนดขั้นตอนและแผนการตรวจสอบความถูกต้องระหว่างสถาปัตยกรรมของระบบและความต้องการของผู้ใช้ กำหนดงบประมาณและรายละเอียดของเครื่องมือที่นำมาใช้ในการออกแบบ
5. กิจกรรมการวิเคราะห์ความต้องการของซอฟต์แวร์ (Software Requirement Analysis) จัดทำแผนการวิเคราะห์ เลือกรูปแบบและกำหนดขั้นตอนการวิเคราะห์ความต้องการของซอฟต์แวร์

6. กิจกรรมการออกแบบซอฟต์แวร์ (Software Design) จัดทำแผนการออกแบบซอฟต์แวร์ เลือกรูปแบบการออกแบบซอฟต์แวร์ (Software Model) กำหนดขั้นตอนการตรวจสอบความถูกต้องกับความต้องการของผู้ใช้ ระบุรายละเอียดของการทำงานและรายละเอียดของระบบด้วย
7. กิจกรรมการสร้างซอฟต์แวร์ (Software Construction) จัดทำแผนการสร้างซอฟต์แวร์ กำหนดขั้นตอนการสร้างและการทดสอบการสร้างซอฟต์แวร์
8. กิจกรรมการประกอบซอฟต์แวร์ (Software Integration) จัดทำแผนการประกอบซอฟต์แวร์ กำหนดขั้นตอนการประกอบและการทดสอบการประกอบซอฟต์แวร์
9. กิจกรรมการทดสอบซอฟต์แวร์ (Software Testing) จัดทำแผนการทดสอบซอฟต์แวร์ และกำหนดขั้นตอนการทดสอบซอฟต์แวร์
10. กิจกรรมการติดตั้งซอฟต์แวร์ (Software Installation) จัดทำแผนการติดตั้งซอฟต์แวร์ กำหนดขั้นตอนการทดสอบการติดตั้ง และมีการกำหนดกำลังคน และงบประมาณที่ใช้ด้วย
11. กิจกรรมการบำรุงรักษาซอฟต์แวร์และระบบ (Software & System Maintenance) จัดทำแผนการบำรุงรักษาซอฟต์แวร์และระบบ และกำหนดขั้นตอนการบำรุงรักษา

กระบวนการจัดการ (Management process group)

12. กิจกรรมการบริหารโครงการ (Project Management) จัดทำแผน และกำหนดขั้นตอนของการบริหารโครงการ

กระบวนการสนับสนุน (Supporting Life Cycle Process)

13. กิจกรรมการประกันคุณภาพ (Quality Assurance) จัดทำแผน และกำหนดขั้นตอนการประกันคุณภาพ
14. กิจกรรมการบริหารโครงร่างซอฟต์แวร์ (Configuration Management) จัดทำแผนและกำหนดขั้นตอนการบริหารโครงร่างซอฟต์แวร์
15. กิจกรรมการบริหารการเปลี่ยนแปลง (Change Request Management) จัดทำแผนการบริหารการเปลี่ยนแปลง กำหนดขั้นตอนและบุคคลผู้มีสิทธิอนุมัติให้มีการเปลี่ยนแปลง