เอกสารประกอบการสอน
วิชาวิศวกรรมซอฟต์แวร์

ชาญวิทย์ แก้วกสิ

สาขาวิชาวิศวกรรมคอมพิวเตอร์

สำนักวิชาวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีสุรนารี

คำนำ

วิศวกรรมซอฟต์วแวร์เป็นศาสตร์เชิงประยุกต์เพื่อพัฒนาซอฟต์แวร์เชิงอุตสาหกรรม โดยนำหลักทางวิศวกรรมมาปรับใช้ทำให้ซอฟต์แวร์มีคุณภาพสูง พัฒนาได้รวดเร็วและต้นทุนต่ำลง ความรู้ความเข้าใจด้านวิศวกรรมซอฟต์แวร์จึงเป็นสิ่งจำเป็นต่อการทำงานในภาคอุตสาหกรรม

เอกสารประกอบการสอนวิชาวิศวกรรมซอฟต์แวร์จัดทำขึ้นโดยประกอบไปด้วยเนื้อหาที่จำเป็นสำหรับการพัฒนาซอฟต์แวร์ที่ทันสมัย ใช้งานได้จริงในภาคอุตสาหกรรม เช่น พื้นฐานทางวิศวกรรมซอฟต์แวร์ กระบวนการพัฒนาซอฟต์แวร์ เครื่องมือทางด้านวิศวกรรมซอฟต์แวร์ เป็นต้น เพื่อให้นักศึกษามีความรู้ทั้งในเชิงทฤษฎีและปฏิบัติ รวมทั้งความพร้อมในการทำงานจริง ตอบสนองต่ออุตสาหกรรมซอฟต์แวร์ที่ปรับตัวอย่างรวดเร็ว

เอกสารประกอบการสอนฉบับนี้ใช้เวลาพัฒนากว่า 2 ปี และได้ใช้สอนจริงในรายวิชา 423306 วิศวกรรมซอฟต์แวร์เป็นเวลา 2 ปีการศึกษา รวมทั้งรายวิชา 204331 วิศวกรรมซอฟต์แวร์เป็นเวลา 1 ปีการศึกษา

สารบัญ

บทที่ 1
ความรู้เบื้องต้นเกี่ยวกับวิศวกรรมซอฟต์แวร์
Introduction to Software Engineering

ซอฟต์แวร์ (Software)

ซอฟต์แวร์ สามารถนิยามได้ดังนี้

1. ซอฟต์แวร์เป็นชุดคำสั่งที่เมื่อทำงานในระบบคอมพิวเตอร์แล้วสามารถให้คุณสมบัติ (Feature), ฟังก์ชั่น (Function) และ สมรรถนะ (Performance) ที่ผู้ใช้ต้องการ
2. ซอฟต์แวร์เป็นโครงสร้างข้อมูลซึ่งมีความเพียงพอที่จะจัดการสารสนเทศ (Information) ได้
3. ซอฟต์แวร์เป็นข้อมูลในลักษณะพรรณนาที่อาจอยู่ในสิ่งของที่จับต้องได้หรือรูปแบบเสมือนที่ใช้อธิบายกระบวนการหรือการทำงานของโปรแกรม

ซอฟต์แวร์เป็นสิ่งที่สร้างขึ้นในลักษณะที่ต่างกับวัตถุจริง โดยมีคุณลักษณะต่างจากฮาร์ดแวร์ (Hardware) ดังนี้

1. ซอฟต์แวร์จะใช้คำว่าพัฒนาซอฟต์แวร์แทนการผลิต
2. ซอฟต์แวร์ไม่มีการเสื่อมต่างกับฮาร์ดแวร์ที่มีการเสื่อมตามเวลา
3. ซอฟต์แวร์มักถูกสร้างแบบสั่งทำ แม้ว่าจะมีลักษณะที่เป็นเชิงคอมโพเน้นท์ (Component)
4. ในยุคปัจจุบันซอฟต์แวร์ถูกมองเป็นการบริการมากขึ้น

ซอฟต์แวร์มีความสำคัญในการเป็นเทคโนโลยีและเป็นองค์ประกอบของเทคโนโลยีอื่นที่เกี่ยวข้องกับธุรกิจ, วิทยาศาสตร์ และวิศวกรรมศาสตร์ อีกทั้งยังสามารถทำให้เกิดการสร้างเทคโนโลยีอื่นๆได้ เช่น พันธุวิศวกรรม, นาโนเทคโนโลยี เป็นต้น

เมื่อความสำคัญของซอฟต์แวร์เพิ่มขึ้น การพัฒนาซอฟต์แวร์ที่มีโครงสร้างซับซ้อนก็เกิดขึ้น กลไกการพัฒนาซอฟต์แวร์จึงมีความจำเป็นในการช่วยให้การพัฒนาง่ายขึ้น เร็วขึ้น ถูกลง ได้คุณภาพสูงและสามารถบำรุงรักษาโปรแกรมได้ง่าย ซึ่งกลไกดังกล่าวเรียกว่า วิศวกรรมซอฟต์แวร์ (Software Engineering)

ประเภทของซอฟต์แวร์

ซอฟต์แวร์ถูกสร้างขึ้นเพื่อตอบสนองความต้องการของผู้ใช้ โดยอาจแบ่งเป็นประเภทได้ดังต่อไปนี้

1. ซอฟต์แวร์ระบบ (System Software)

เป็นซอฟต์แวร์ที่ใช้ในการรองรับการทำงานและจัดการทรัพยากรพื้นฐานของคอมพิวเตอร์

1.1 ระบบปฏิบัติการ (Operating System)

เป็นซอฟต์แวร์ที่จัดการทรัพยากรของคอมพิวเตอร์และให้บริการแก่ซอฟต์แวร์ประยุกต์ (Application Software) โดยจะเป็นตัวกลางระหว่างฮาร์ดแวร์และซอฟต์แวร์ประยุกต์

สำหรับระบบปฏิบัติการบนคอมพิวเตอร์ที่ใช้กันมากในปัจจุบัน ได้แก่ Microsoft Windows, Linux และ Mac OS X (โอเอส เท็น) เป็นต้น ส่วนระบบปฏิบัติการบนอุปกรณ์เคลื่อนที่ส่วนบุคคล เช่น Android, iOS (สำหรับ iPhone), BlackBerry OS และ Nokia MeeGo เป็นต้น สำหรับแนวทางการพัฒนาระบบปฏิบัติการ

ในยุคถัดไปอาจมีการรวมเอา Web Browser เข้าเป็นส่วนหนึ่งของซอฟต์แวร์ระบบ ซึ่งปัจจุบันสามารถพบได้ใน Google Chrome OS

1.2 ดีไวซ์ไดรเวอร์ (Device Driver)

เป็นซอฟต์แวร์ที่ช่วยให้โปรแกรมคอมพิวเตอร์อื่นๆสามารถติดต่อกับอุปกรณ์ภายนอกได้ เช่น Device Driver สำหรับอุปกรณ์ USB ในระบบปฏิบัติการ จะทำให้สามารถติดต่อกับ Flash Drive ที่ผู้ใช้เสียบเข้ากับ USB Port เป็นต้น

1.3 ซอฟต์แวร์แม่ข่าย (Server Software)

เป็นซอฟต์แวร์ที่ช่วยให้เครือข่ายสามารถให้บริการต่างๆได้ เช่น Web Server ที่ให้บริการเมื่อลูกข่ายร้องขอ (Request) เพื่อที่จะเข้าถึงหน้าเว็บผ่านทาง Web Browser เป็นต้น

1.4 ระบบจัดการหน้าต่าง (Window System หรือ Window Manager)

เป็นซอฟต์แวร์ที่ใช้สำหรับติดต่อกับผู้ใช้ โดยปกติจะเป็นส่วนหนึ่งของสภาวะแวดล้อมเดสทอป (Desktop Environment) โปรแกรมกลุ่มนี้ทำหน้าที่รับการโต้ตอบจากผู้ใช้และทำการแสดงผลในรูปแบบของหน้าต่าง เมนูและชิ้นส่วนที่มองเห็นได้อื่นๆในระบบปฏิบัติการ Windows นั้น Window System จะไม่แยกออกมาชัดเจน แต่สำหรับระบบปฏิบัติการตระกูล Linux มีการแยกสถาปัตยกรรม ส่วนนี้ออกมาอย่างชัดเจน เช่น Gnome หรือ KDE เป็นต้น

1.5 ซอฟต์แวร์อรรถประโยชน์ (Utility Software)

เป็นซอฟต์แวร์ที่ออกแบบมาเพื่อช่วยในการวิเคราะห์ ปรับแต่ง และบำรุงรักษาคอมพิวเตอร์

2. ซอฟต์แวร์ประยุกต์ (Application Software)

เป็นซอฟต์แวร์ที่ให้บริการผู้ใช้ตามกลุ่มงานที่ซอฟต์แวร์นั้นๆถูกออกแบบมาให้รองรับ ซอฟต์แวร์กลุ่มนี้แตกต่างจากซอฟต์แวร์ระบบ เนื่องจากสามารถทำงานเฉพาะเจาะจงลงไปตามประเภทของงานที่ต้องการ เช่น

- ซอฟต์แวร์ธุรกิจ เช่น ซอฟต์แวร์กลุ่ม Enterprise Resource Planning (ERP), Customer Relationship Management (CRM) เป็นต้น
- ซอฟต์แวร์เกมส์
- ซอฟต์แวร์สำหรับใช้งานอินเตอร์เนต เช่น Web Browser, Instant Messenger เป็นต้น
- ซอฟต์แวร์ฐานข้อมูล
- ซอฟต์แวร์ชุดสำนักงานอัตโนมัติ เช่น Word Processing, Spreadsheet เป็นต้น

3. เว็บแอพพลิเคชั่น (Web Applications)

เป็นซอฟต์แวร์ที่ทำงานใน Web Browser

ซึ่งจะต่างจากแอพพลิเคชั่นทั่วๆไปที่ทำงานบนสภาพแวดล้อมเดสทอป เว็บแอพพลิเคชั่นนี้จะแตกต่างกับเว็บไซต์ธรรมดาที่นำเสนอเนื้อหาเป็นหลักในขณะที่ซอฟต์แวร์ในกลุ่มเว็บแอพพลิเคชั่นนี้มักถูกสร้างขึ้นมาเพื่อย้ายการทำงานที่เคยทำบนเดสทอปมาสู่เว็บ เช่น

- ซอฟต์แวร์ธุรกิจ เช่น Salesforce เป็นซอฟต์แวร์ธุรกิจประเภท CRM ที่เป็นเว็บแอพพลิเคชั่น
- ซอฟต์แวร์เกมส์ เช่น เกมส์ที่สร้างด้วย Adobe Flash

- ซอฟต์แวร์ชุดสำนักงานอัตโนมัติ เช่น Google Docs

4. แอพพลิเคชั่นบนอุปกรณ์มือถือ (Mobile Applications)

แอพพลิเคชั่นที่ทำงานบนมือถือหรืออุปกรณ์ประเภท Tablet จะมีการโต้ตอบกับผู้ใช้ในลักษณะเฉพาะ รวมถึงสามารถใช้ความสามารถบางอย่างของตัวอุปกรณ์นั้นๆเพื่อเพิ่มประสิทธิภาพในการใช้งาน เช่น การเปลี่ยนทิศทางการแสดงผล เมื่อผู้ใช้หมุนตัวอุปกรณ์ การระบุตำแหน่งของผู้ใช้ผ่านโมดูล GPS ที่มีอยู่ในตัวอุปกรณ์ หรือตอบโต้กับผู้ใช้ในลักษณะต่างๆกันผ่านจอชนิดสัมผัส เป็นต้น

5. ซอฟต์แวร์เชิงสายการผลิต (Product-line Software)

เป็นกลุ่มของซอฟต์แวร์ที่ถูกออกแบบมาให้มีความสามารถเฉพาะอย่างสำหรับลูกค้าที่มีความต้องการต่างๆ กัน ซอฟต์แวร์กลุ่มนี้มักมีตลาดในกลุ่มเฉพาะ เช่น โปรแกรมคุมคลังสินค้า โปรแกรมจัดการการผลิต โดยลักษณะเฉพาะของซอฟต์แวร์ประเภทนี้ คือ ในซอฟต์แวร์ตัวหนึ่งๆ
จะมีฟังก์ชั่นการทำงานร่วมกันอยู่และจะมีส่วนที่ต่างกันออกไปสำหรับลูกค้าแต่ละราย

6. ซอฟต์แวร์ประเภทฝังตัว (Embedded Software)

เป็นซอฟต์แวร์ที่อยู่ในอุปกรณ์ประเภทฝังตัวชนิดต่างๆ เช่น ไมโครคอนโทรลเลอร์ (Micro Controller) เพื่อสร้างแอพพลิเคชั่นที่ส่วนใหญ่ใช้ติดต่อกับอุปกรณ์ภายนอก โดยทั่วไปซอฟต์แวร์ประเภทนี้มักใช้รับข้อมูลจากเซนเซอร์ตรวจจับประเภทต่างๆ ซึ่งอาจเป็นการตรวจจับอุณหภูมิ ความชื้น หรือรับสัญญาณ GPS จากนั้นทำการประมวลผลเพื่อทำงานเฉพาะอย่าง ในทางกลับกันหากอุปกรณ์ฝังตัวนั้นมีสมรรถนะสูงมากๆ ซอฟต์แวร์ในอุปกรณ์ดังกล่าวอาจจะซับซ้อนได้เกือบเทียบเท่ากับซอฟต์แวร์ระบบที่ใช้กันในคอมพิวเตอร์ทั่วไปได้

เนื่องจากข้อจำกัดของทรัพยากรที่มีในอุปกรณ์ประเภทฝังตัวรวมไปถึงอายุการใช้งานของแบตเตอรี่ในอุปกรณ์เหล่านั้นทำให้การพัฒนาซอฟต์แวร์ประเภทฝังตัวมีลักษณะเฉพาะแตกต่างจากซอฟต์แวร์ประเภทอื่นๆอย่างชัดเจน เช่น ความสามารถในการประหยัดพลังงานเมื่อระบบว่าง เป็นต้น

7. ซอฟต์แวร์เชิงวิศวกรรมและวิทยาศาสตร์ (Engineering/Scientific Software)

เป็นซอฟต์แวร์ที่เน้นการสนับสนุนการทำงานและการคำนวณเชิงวิศวกรรมหรือวิทยาศาสตร์ โดยจะเน้นถึงความถูกต้องในการคำนวณ การสนับสนุนการแก้สมการประเภทต่างๆ การคำนวณสมรรถนะสูง การประมวลผลเชิงสัญลักษณ์ รวมถึงการคำนวณเชิงขนาน เป็นต้น ในงานบางประเภทอาจเป็นการประมวลผลข้อมูลขนาดมหาศาลที่ต้องการการจัดข้อมูลชนิดพิเศษ ซึ่งทำให้บางส่วนของซอฟต์แวร์ประเภทนี้เกี่ยวข้องกับซอฟต์แวร์ระบบ เช่น ระบบการเก็บข้อมูลของโครงการ CERN

8. ซอฟต์แวร์ปัญญาประดิษฐ์ (Artificial Intelligence Software)

เป็นซอฟต์แวร์ที่พัฒนาไว้เพื่อให้คอมพิวเตอร์มีความสามารถในการคิดเองได้ โดยอาจใช้เป็นส่วนประกอบในซอฟต์แวร์แอพพลิเคชั่นเชิงธุรกิจเพื่อช่วยในการตัดสินใจ ซอฟต์แวร์วิเคราะห์ตัวหนังสือเพื่ออ่านออกเสียง ซอฟต์แวร์ควบคุมการทำงานของหุ่นยนต์ เป็นต้น ซอฟต์แวร์ปัญญาประดิษฐ์มักจะ

เกี่ยวข้องกับโครงสร้างข้อมูลเฉพาะแบบและอาจต้องการการคำนวณสมรรถนะสูง ซึ่งคาบเกี่ยวกับซอฟต์แวร์ทางวิทยาศาสตร์ เป็นต้น

จากประเภทซอฟต์แวร์ที่กล่าวมา อาจสรุปได้ว่าซอฟต์แวร์แต่ละประเภทแม้จะมีลักษณะที่ต่างกันแต่มีความเกี่ยวข้องกันในหลายแง่มุม อีกทั้งซอฟต์แวร์บางกลุ่มมีลักษณะที่คาบเกี่ยวกันและเป็นที่น่าสนใจว่าความคาบเกี่ยวดังกล่าวอาจเพิ่มขึ้นหรือลดลงขึ้นกับการเปลี่ยนแปลงของเทคโนโลยีซอฟต์แวร์ที่ดำเนินต่อไปอย่างไม่หยุดนิ่ง ดังนั้นในบางช่วงการปรับใช้วิศวกรรมซอฟต์แวร์กับซอฟต์แวร์บางประเภทอาจนำมาใช้ได้กับซอฟต์แวร์อีกประเภทหนึ่งในอีกยุคหนึ่งได้

วิศวกรรมซอฟต์แวร์ (Software Engineering)

วิศวกรรมซอฟต์แวร์เป็นศาสตร์เชิงประยุกต์ที่เกี่ยวกับแนวทางที่เป็นระบบ มีระเบียบแบบแผนวัดได้เชิงปริมาณต่อการพัฒนาและการบำรุงรักษาซอฟต์แวร์ ซึ่งก็คือการประยุกต์หลักวิศวกรรมศาสตร์เข้ากับการพัฒนาซอฟต์แวร์ สาขาย่อยในวิศวกรรมซอฟต์แวร์มีดังนี้

1. ความต้องการเชิงซอฟต์แวร์ (Software Requirements)

   เป็นการศึกษาเกี่ยวกับความต้องการเชิงซอฟต์แวร์ รวมถึงแนวทางการสื่อสาร รวบรวมและจัดเก็บความต้องการเชิงซอฟต์แวร์จากกลุ่มเป้าหมาย

2. การออกแบบซอฟต์แวร์ (Software Design)

   เป็นการศึกษาเกี่ยวกับการวิเคราะห์และออกแบบความต้องการและข้อกำหนดทางซอฟต์แวร์ให้สามารถนำไปพัฒนาเป็นซอฟต์แวร์ที่สามารถทำงานได้

3. การพัฒนาซอฟต์แวร์ (Software Development)

   เป็นการศึกษากระบวนการการพัฒนาซอฟต์แวร์ โดยเกี่ยวข้องกับเทคนิคการสร้างซอฟต์แวร์อย่างเป็นระบบ มีการเลือกใช้ภาษาโปรแกรม เฟรมเวิร์ค (Framework) และ Design Pattern ที่เหมาะสม

4. การทดสอบซอฟต์แวร์ (Software Testing)

   เป็นการศึกษาเกี่ยวกับกลไกการทดสอบซอฟต์แวร์ทั้งในระดับที่ใกล้การพัฒนามากที่สุด ซึ่งสามารถเข้าถึงต้นรหัส (Source Code) ของตัวซอฟต์แวร์ได้ หรือในระดับที่ไกลที่สุด คือ การทดสอบซอฟต์แวร์ที่กำลังทำงานอยู่ก่อนนำไปใช้จริง รวมทั้งแง่มุมการทดสอบต่างๆ เช่น ประสิทธิภาพและการรับโหลด เป็นต้น

5. การบำรุงรักษาซอฟต์แวร์ (Software Maintenance)

   เป็นการศึกษาเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ การใช้เครื่องมือหรือภาษาในการแก้ไขบำรุงรักษาหลังจากระยะการนำซอฟต์แวร์ไปใช้งานแล้ว กลไกในการบำรุงรักษาอาจรวมถึงเทคโนโลยีการนำซอฟต์แวร์รุ่นใหม่ไปใช้แทนที่รุ่นเก่าโดยที่ไม่จำเป็นต้องหยุดการทำงานของระบบ

6. การจัดการปรับแต่งซอฟต์แวร์ (Software Configuration Management)

   เป็นการศึกษาเกี่ยวกับการจัดการรุ่นของต้นรหัสซอฟต์แวร์ ติดตามความเปลี่ยนแปลงของต้นรหัสและการบำรุงรักษาส่วนต่อยอดจากต้นรหัสที่มีอยู่แล้ว

7. การจัดการเชิงวิศวกรรมซอฟต์แวร์ (Software Engineering Management)

เป็นการศึกษาการจัดการระบบซอฟต์แวร์ในลักษณะเดียวกันกับการจัดการโครงการ โดยอาจจะมีการนำโปรแกรมสำหรับควบคุมจัดการโครงการมาใช้ดูแลการพัฒนาซอฟต์แวร์

8. กระบวนการพัฒนาซอฟต์แวร์ (Software Development Process)

เป็นการศึกษากระบวนการในการพัฒนาซอฟต์แวร์ครอบคลุมตั้งแต่การรวบรวมความต้องการไปจนถึงการดูแลรักษาซอฟต์แวร์ กระบวนการในการพัฒนาซอฟต์แวร์ เช่น Waterfall Model, Spiral Model และ Agile Development เป็นต้น

9. เครื่องมือทางวิศวกรรมซอฟต์แวร์ (Software Engineering Tools)

เป็นการศึกษาการสร้างและพัฒนาเครื่องมือขึ้นเพื่อใช้ช่วยในการพัฒนาซอฟต์แวร์ กลุ่มเครื่องมือดังกล่าวจะเรียกว่า Computer-Aided Software Engineering Tools (CASE Tools)

10. คุณภาพซอฟต์แวร์ (Software Quality)

เป็นการศึกษาเกี่ยวกับคุณภาพซอฟต์แวร์และเทคนิคการวัดคุณภาพทั้งในแง่มุมของคุณภาพในการออกแบบ คุณภาพของการพัฒนาซอฟต์แวร์ให้ได้ตามการออกแบบ คุณภาพของต้นรหัส และคุณภาพของผลิตภัณฑ์ เมื่อเสร็จสิ้นการพัฒนาในขั้นตอนสุดท้าย เป็นต้น

สายงานทางด้านวิศวกรรมซอฟต์แวร์

- นักวิเคราะห์ระบบ (System Analyst)

เป็นผู้ที่มีบทบาทสำคัญในการวิเคราะห์ปัญหา วางแผนการแก้ปัญหา รวมทั้งแนะนำซอฟต์แวร์ให้แก่ผู้ใช้ นักวิเคราะห์ระบบมีบทบาทที่สำคัญอีกอย่างหนึ่งก็คือ ประสานงานให้การพัฒนาซอฟต์แวร์เป็นไปตามความต้องการทั้งเชิงธุรกิจและอื่นๆ นักวิเคราะห์ระบบมักเกี่ยวข้องกับการแก้ปัญหาในหลายๆแง่มุม รวมทั้งจำเป็นต้องมีความรู้ในภาษาโปรแกรมหลายภาษา ระบบปฏิบัติการหลายประเภท และแพล็ตฟอร์มต่างๆของฮาร์ดแวร์ เนื่องจากนักวิเคราะห์ระบบมักจะเป็นผู้ที่เขียนความต้องการเชิงซอฟต์แวร์ให้อยู่ในรูปของข้อกำหนดเชิงเทคนิค เช่น เอกสารการวิเคราะห์และออกแบบต่างๆ ซึ่งอาจจะอยู่ในรูปแบบของ Flow Chart หรือ Use Case เป็นต้น

- สถาปนิกซอฟต์แวร์ (Software Architect)

เป็นผู้ที่รับผิดชอบเกี่ยวกับการกำหนดขอบเขตของการเลือกเทคโนโลยีและเฟรมเวิร์คระหว่างการพัฒนาซอฟต์แวร์ รวมทั้งการเลือกวิธีการที่ใช้เป็นมาตรฐานในการพัฒนา และอาจรวมถึงเป็นผู้สร้างหรือกำหนดเฟรมเวิร์คสำหรับแอพพลิเคชั่นที่กำลังพัฒนาอยู่ สถาปนิกซอฟต์แวร์มีความจำเป็นที่จะต้องรู้ว่าสามารถใช้ซ้ำสิ่งใดบ้างในองค์กรหรือในแอพพลิเคชั่นที่กำลังพัฒนา โดยต้องสังเกตและเข้าใจถึงภาพรวมสภาพแวดล้อมของระบบ และสามารถออกแบบชิ้นส่วนซอฟต์แวร์สำหรับระบบ รวมทั้งมีความรู้ครอบคลุมไปถึงแอพพลิเคชั่นอื่นๆ ในองค์กรที่แอพพลิเคชั่นซึ่งกำลังพัฒนาอยู่นั้นต้องติดต่อด้วย สถาปนิกซอฟต์แวร์ยังต้องสามารถแบ่งแอพพลิเคชั่นที่ซับซ้อนออกเป็นส่วนย่อยที่สามารถจัดการได้ง่าย มีความเข้าใจฟังก์ชั่นการทำงานของแต่ละคอมโพเน้นท์ในตัวแอพพลิเคชั่นเป็นอย่างดีและรวมถึงการติดต่อและการขึ้นต่อกันระหว่างคอมโพเน้นท์เหล่านั้น ที่สำคัญที่สุดคือ สามารถสื่อสารแนวคิดข้างต้นให้นักพัฒนาสามารถเข้าใจได้ เพื่อให้การทำงานเป็นไปอย่างมีประสิทธิภาพ สถาปนิกซอฟต์แวร์มักใช้ UML หรือ OOP เป็นเครื่องมือช่วยสื่อสารสิ่งเหล่านี้ไปยังนักพัฒนาซอฟต์แวร์

ประเภทของสถาปนิกซอฟต์แวร์ อาจแบ่งได้ดังนี้

1. Enterprise Architect ครอบคลุมความรับผิดชอบสถาปัตยกรรมระหว่างโครงการ
2. Solution Architect เน้นสถาปัตยกรรม solution ในระดับรายละเอียด
3. Application Architect ครอบคลุมการใช้ซ้ำคอมโพเน้นท์และการดูแลโครงการเดี่ยว

- โปรแกรมเมอร์ (Programmer)

เป็นผู้ที่ทำหน้าที่เขียนซอฟต์แวร์คอมพิวเตอร์ ในบางครั้งอาจเรียกว่า นักพัฒนาซอฟต์แวร์ (Software Developer) โดยทั่วไปโปรแกรมเมอร์จะมีหน้าที่ทั้งการเขียนโปรแกรม การทดสอบและการแก้ไขบั๊ก (Bug) รวมถึงการคิด ออกแบบและทดสอบตรรกะเพื่อแก้ปัญหาด้วยคอมพิวเตอร์

- ผู้ดูแลระบบ (System Administrator)

เป็นผู้มีหน้าที่รับผิดชอบในการดูแลและบำรุงรักษาระบบ ซึ่งอาจรวมทั้งระบบเครือข่ายในส่วนที่เกี่ยวข้องกับการพัฒนาโปรแกรม ผู้ดูแลระบบมักเขียนโปรแกรมในรูปแบบสคริปต์ เพื่อจัดการระบบในเบื้องต้น โดยใช้ภาษาโปรแกรม เช่น Perl หรือ Bash Script เป็นต้น โดยภาษาสคริปต์ดังกล่าว จะเรียกใช้ซอฟต์แวร์อรรถประโยชน์เพื่อทำงานระดับล่างต่อไป งานบำรุงรักษาระบบของผู้ดูแลระบบจะครอบคลุมถึงการปรับปรุงการติดตั้งโปรแกรมรุ่นใหม่ โดยเฉพาะอย่างยิ่งเมื่อซอฟต์แวร์บางตัวในระบบที่เป็นลักษณะแม่ข่ายเกิดข้อผิดพลาดที่อาจนำไปสู่ความไม่ปลอดภัยขององค์กร ผู้ดูแลระบบจำเป็นต้องมีความสามารถในการนำ Patch มาแก้ไขโปรแกรมรุ่นที่มีบั๊กให้เป็นรุ่นที่ปลอดภัยขึ้น ในกรณีของการใช้โปรแกรมประเภทเปิดต้นรหัส ผู้ดูแลระบบมักต้องมีความสามารถในการแก้ไขข้อผิดพลาดและคอมไพล์โปรแกรมรุ่นใหม่ได้เอง

- ผู้ดูแลเครือข่าย (Network Administrator)

เป็นผู้มีหน้าที่รับผิดชอบในการดูแลคอมพิวเตอร์ในส่วนที่เกี่ยวข้องกับเครือข่ายทั้งที่เป็นฮาร์ดแวร์และซอฟต์แวร์ และจะรวมไปถึงการปรับแต่ง ดูแลและติดตั้งอุปกรณ์เครือข่ายใหม่ ผู้ดูแลเครือข่ายจะต้องมีความรู้ทางเทคนิคเชิงลึกและความสามารถในการเรียนรู้อุปกรณ์เครือข่ายใหม่ๆและซอฟต์แวร์ที่เกี่ยวข้องได้อย่างรวดเร็ว และในหลายๆองค์กรอาจจะรวมถึงงานทางด้านการออกแบบเครือข่ายด้วย

- ผู้ดูแลฐานข้อมูล (Database Administrator)

เป็นผู้ที่มีหน้าที่ออกแบบ อิมพลีเมนต์ ดูแลรักษาและซ่อมบำรุงฐานข้อมูลขององค์กร โดยมีบทบาทที่อาจรวมถึงการกำหนดกลยุทธเพื่อการพัฒนาและออกแบบฐานข้อมูล เฝ้าระวังและปรับปรุงสมรรถนะและความจุของฐานข้อมูล และประเมินความต้องการในการขยายการใช้งานของระบบฐานข้อมูล ทักษะอื่นๆที่จำเป็นต่อตำแหน่งนี้ เช่น ความรู้ความเข้าใจเกี่ยวกับการย้ายโอนข้อมูล การทำซ้ำข้อมูล การสำรองและการกู้คืนข้อมูล เป็นต้น

- วิศวกรสนับสนุนลูกค้า (Customer Support Engineer)

เป็นวิศวกรที่ทำหน้าที่สนับสนุนภายในหน่วยงานหรือสนับสนุนสินค้าของบริษัทในทางเทคนิค โดยอาจรับแก้ไขปัญหาเกี่ยวกับระบบหรือซอฟต์แวร์ เพื่อให้แน่ใจว่าระบบหรือซอฟต์แวร์ดังกล่าวสามารถทำงานได้อย่างที่ควรจะเป็น ในระหว่างช่วงการพัฒนาหรือการออกแบบระบบใหม่นั้น การสนับสนุนการสร้างระบบมีความจำเป็น เพื่อทำให้การพัฒนาเป็นไปอย่างราบรื่น ในกรณีของซอฟต์แวร์วิศวกรสนับสนุนจะมีบทบาทในการปฏิบัติงานตลอดระยะเวลาการดำเนินการ เพื่อให้ระบบสามารถใช้งานได้อย่างมีประสิทธิภาพ

- ผู้ดูแลเว็บ (Web Master)

เป็นผู้ที่มีบทบาทในการพัฒนาและดูแลรักษาเว็บไซต์ทั้งในแง่มุมของระบบและเนื้อหา และอาจครอบคลุมไปถึงการสนับสนุนผู้ที่เข้ามาใช้งานเว็บไซต์ การจัดการกลั่นกรองการนำเสนอความเห็นและการปรับปรุงประสิทธิภาพการตอบสนองของเว็บไซต์ต่อผู้ใช้

- วิศวกรเครือข่าย (Security Engineer)

เป็นวิศวกรที่มีความรับผิดชอบในการตรวจสอบดูแลซอฟต์แวร์ เพื่อป้องกันการเกิดข้อผิดพลาดที่เกินกว่าซอฟต์แวร์ได้เตรียมรองรับไว้ รวมถึงช่องโหว่ของระบบ โดยวิศวกรเครือข่ายจะต้องมีความเข้าใจในการบ่งชี้อาการพื้นฐานและปรับความปลอดภัยให้กับเครือข่ายและซอฟต์แวร์ รวมถึงสามารถนำ Patch มาติดตั้งเพื่อทำให้ระบบมีความปลอดภัยได้

- ผู้จัดการโครงการ (Project Manager)

เป็นผู้ที่มีความรับผิดชอบในการจัดการโครงการ ซึ่งมีความจำเป็นอย่างมากที่จะต้องมีประสบการณ์ทางการพัฒนาซอฟต์แวร์สูง ผู้จัดการโครงการจำเป็นต้องมีความเข้าใจในรูปแบบวิธีการพัฒนาซอฟต์แวร์ที่นำมาใช้เป็นอย่างดี รวมถึงเข้าใจและคุ้นเคยกับวงจรการพัฒนาซอฟต์แวร์ (Software Development Life Cycle) ด้วย ผู้จัดการโครงการมักจะมีหน้าที่ต้องรับผิดชอบดังนี้

  - พัฒนาแผนสำหรับโครงการ
  - จัดการติดต่อประสานงานกับลูกค้า
  - จัดการประสานงานกับทีม
  - จัดการความเสี่ยงที่เกิดขึ้น
  - จัดการตารางเวลาการพัฒนา
  - จัดการงบประมาณ
  - จัดการและแก้ไขความขัดแย้ง

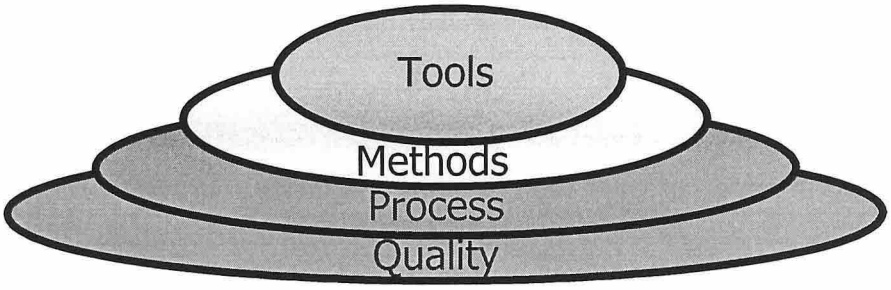
- ผู้จัดการการปรับแต่งซอฟต์แวร์ (Software Configuration Manager)

เป็นผู้ที่มีหน้าที่ดูแลและจัดการติดตามควบคุมซอฟต์แวร์ โดยเฉพาะอย่างยิ่งการจัดการรุ่นของต้นรหัสและการวาง Baseline ซึ่งเกี่ยวข้องกับการแยกสาขาของต้นรหัส และการจัดการการเปลี่ยนแปลงของต้นรหัสที่เกิดขึ้นมาปรับลงสู่ Baseline ที่มี เพื่อปรับปรุงคุณภาพซอฟต์แวร์ หรือลดข้อผิดพลาดในซอฟต์แวร์ลง

- ผู้จัดการคุณภาพซอฟต์แวร์ (Software Quality Manager)

เป็นผู้ที่มีหน้าที่รับผิดชอบในการจัดการคุณภาพของกระบวนการพัฒนาซอฟต์แวร์และตัวผลงานที่ได้ โดยผลงานที่มีคุณภาพจะต้องเป็นซอฟต์แวร์ที่ตรงต่อความต้องการที่กำหนดไว้โดยผู้ใช้และผู้ใช้พึงพอใจ ในมุมมองของการปรับปรุงคุณภาพของกระบวนการพัฒนาซอฟต์แวร์ ผู้จัดการคุณภาพซอฟต์แวร์จะต้องมีบทบาทในการผลักดันให้บุคคลากรในองค์กรมีทัศนคติที่เป็นบวกต่อการปรับปรุงคุณภาพ โดยให้เป็นหน้าที่ของทุกคนในองค์กรหรือในทีมพัฒนาซอฟต์แวร์ ผู้จัดการคุณภาพซอฟต์แวร์อาจรับหน้าที่ตรวจสอบคุณภาพด้วยการรันระบบงาน ก่อนนำไปให้ลูกค้าใช้งานจริง

ลำดับชั้นของวิศวกรรมซอฟต์แวร์ (Software Engineering Layers)



วิศวกรรมซอฟต์แวร์เป็นเทคโนโลยีซึ่งแยกออกเป็นลำดับชั้น แนวทางของวิศวกรรมซอฟต์แวร์ อยู่บนพื้นฐานของความต้องการที่จะนำไปสู่คุณภาพซึ่งมาจากความพยายามที่จะปรับปรุงคุณภาพการพัฒนาอย่างต่อเนื่อง และนำไปสู่การพัฒนาที่มีประสิทธิภาพที่สูงขึ้น นั่นคือ พื้นฐานสำคัญที่สนับสนุนวิศวกรรมซอฟต์แวร์คือ "การเน้นคุณภาพ" สำหรับชั้น "กระบวนการ"เป็นตัวประสานชั้นที่เป็นเทคโนโลยีต่างๆเข้าด้วยกัน เพื่อให้การพัฒนาซอฟต์แวร์เป็นไปได้อย่างราบรื่นและอยู่ในขอบเขตของเวลานำส่ง กระบวนการจะเป็นทั้งตัวกำหนดเฟรมเวิร์คที่ต้องเตรียมขึ้นเพื่อให้เทคโนโลยีทางวิศวกรรมซอฟต์แวร์สามารถนำไปใช้เพื่อการพัฒนาซอฟต์แวร์ได้อย่างมีประสิทธิภาพ โดยกระบวนการพัฒนาซอฟต์แวร์จะเตรียมพื้นฐานของการควบคุมโครงการนั้น และเตรียมข้อมูลแวดล้อมเพื่อที่จะนำ "วิธีการ" พัฒนามาประยุกต์ อีกทั้งระบุวิธีการสร้างผลิตภัณฑ์ บอกถึงการเตรียมไมล์สโตน (Milestone) เพื่อให้ควบคุมคุณภาพได้เป็นระยะๆ รวมทั้งสามารถจัดการการเปลี่ยนแปลงได้อย่างเหมาะสม วิธีการในวิศวกรรมซอฟต์แวร์จะเป็นกลไกในเชิงเทคนิคที่จะบอกว่า จะสร้างซอฟต์แวร์อย่างไร วิธีการนั้นรวมไปถึงงานต่างๆ ตั้งแต่การติดต่อ, การวิเคราะห์ความต้องการ, การออกแบบ, การสร้างโปรแกรม, การทดสอบไปจนถึงการสนับสนุนและดูแลรักษา วิธีการทางวิศวกรรมซอฟต์แวร์จะอยู่บนพื้นฐานของแต่ละกลุ่มเทคโนโลยีและรวมถึงเทคนิคการโมเดลและการอธิบายต่างๆ สำหรับ "เครื่องมือ" ในวิศวกรรมซอฟต์แวร์จะให้การสนับสนุนทั้งในลักษณะอัตโนมัติหรือกึ่งอัตโนมัติแก่กระบวนและวิธีการเพื่อให้ทำงานร่วมกัน ซึ่งเครื่องมือดังกล่าวเรียกว่า CASE Tool

บทที่ 2
วงจรการพัฒนาซอฟต์แวร์
Software Development Life Cycle

วงจรการพัฒนาซอฟต์แวร์ (Software Development Life Cycle – SDLC)

หรือที่เรียกว่ากระบวนการทางซอฟต์แวร์ (Software Process) เป็นกลุ่มของกิจกรรม การกระทำ หรืองานที่ต้องปฏิบัติเมื่อต้องการสร้างผลิตภัณฑ์หนึ่งๆ ซึ่งไม่ใช่เฉพาะการสร้างซอฟต์แวร์เท่านั้น กระบวนการพัฒนาซอฟต์แวร์นั้นครอบคลุมตั้งแต่การเก็บข้อมูล การวิเคราะห์ความต้องการของผู้ใช้ การทวนสอบข้อกำหนด ไปจนถึงการฝึกอบรมผู้ใช้

ระยะในการพัฒนาซอฟต์แวร์ (Software Development Phase)

การพัฒนาซอฟต์แวร์ทั่วไปสามารถแบ่งเป็นระยะ ดังนี้

1. ระยะวางแผนโครงการ (Project Planning)

   เป็นระยะการวางแผนกำหนดขอบเขตของโครงการ เตรียมเอกสารและแนวทางสำหรับขั้นตอนอื่นๆที่เกี่ยวข้อง

2. ระยะการติดต่อ การเก็บรวบรวมความต้องการ (Requirements)

   เป็นระยะที่ต้องทำการติดต่อประสานงานกับลูกค้าหรือเจ้าของกิจการเพื่อเก็บรวบรวมข้อมูลที่เกี่ยวข้องกับโครงการ ในบางกระบวนการโดยเฉพาะกระบวนการแบบ Agile ระยะการติดต่อจะกระจายตัวอยู่ในตลอดกระบวนการ

3. ระยะการวิเคราะห์ระบบ (Analysis)

   เป็นการนำความต้องการของลูกค้าหรือเจ้าของกิจการมาวิเคราะห์และสร้างแบบจำลอง เพื่อนำไปใช้สร้างในระยะอื่นๆ โดยสกัดสิ่งที่จำเป็นในการพัฒนาซอฟต์แวร์ออกมาอยู่ในรูปแบบจำลองหรือต้นแบบ

4. ระยะการออกแบบ (Design)

   เป็นระยะที่นำเอาแบบจำลองในระยะการวิเคราะห์มาออกแบบในรายละเอียดเกี่ยวกับวิธีการสร้างเพื่อใช้ในระยะการสร้างต่อไป

5. ระยะการสร้าง (Implementation)

   เป็นระยะที่นำแบบจำลองที่วิเคราะห์และออกแบบแล้วมาแปลงเป็นต้นรหัสโปรแกรมที่ทำงานได้

6. ระยะทดสอบ (Testing)

   เป็นระยะที่สร้างการทดสอบขึ้นมาเพื่อทดสอบโปรแกรมในระดับต่างๆ เช่น การทดสอบระดับหน่วย (Unit Testing) การทดสอบระดับบูรณาการ (Integration Testing) หรือการทดสอบเพื่อยอมรับ (Acceptance Testing)

7. ระยะการนำไปใช้ (Deployment)

   เป็นระยะที่นำซอฟต์แวร์ไปติดตั้งให้ลูกค้าเพื่อใช้งานหรือเพื่อทดสอบและรวบรวมผลตอบรับ

8. ระยะบำรุงรักษา (Maintenance)

   เป็นระยะหลังการนำไปใช้โดยผู้ใช้จะรายงานข้อผิดพลาดที่เกิดขึ้นจากตัวระบบเพื่อให้ทีมพัฒนารวบรวมและปรับปรุงโปรแกรมต่อไป

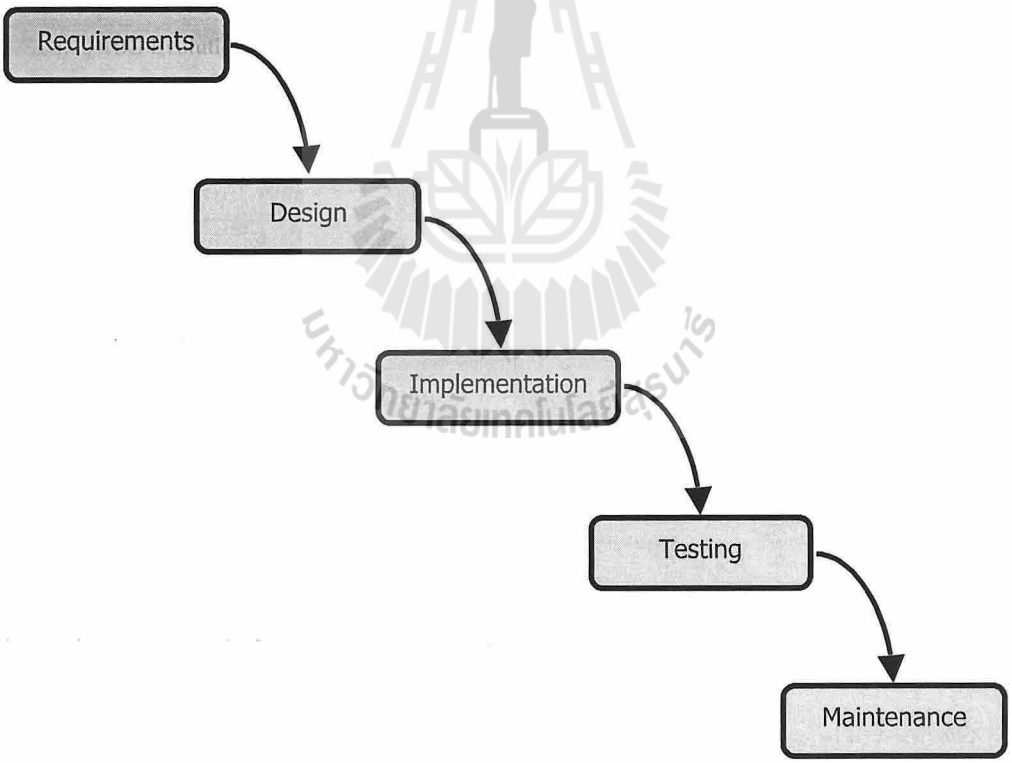
แบบจำลองกระบวนการทางซอฟต์แวร์ (Software Process Model)

กระบวนการทางซอฟต์แวร์ในปัจจุบันมีดังต่อไปนี้

1. แบบ Waterfall
2. แบบ Incremental Process
3. แบบ Iterative
4. แบบ Evolutionary
5. แบบ Spiral
6. แบบ Unified Process
7. การพัฒนาแบบ Agile

1. แบบ Waterfall

แบบ Waterfall นี้บางครั้งอาจเรียกว่า Classic Life Cycle เป็นแนวทางการพัฒนาซอฟต์แวร์ที่มีความเป็นระบบสูงและเป็นลำดับ โดยจะเริ่มจากความต้องการซอฟต์แวร์ของลูกค้า ต่อเนื่องไปถึงการวางแผน การออกแบบ การสร้าง และการนำไปใช้ ซึ่งเหมาะกับสถานการณ์ที่มีการเพิ่มความสามารถให้กับระบบที่มีอยู่แล้ว หรือถ้าเป็นการสร้างฟังก์ชันการทำงานใหม่ก็จะเป็นการพัฒนาเพิ่มแบบไม่มากนัก มีความต้องการที่ชัดเจนและไม่เปลี่ยนแปลง



มีแบบจำลองอีกรูปแบบหนึ่งที่ปรับปรุงมาจากกระบวนการแบบ Waterfall เรียกว่า V-model ที่นำเอาการทดสอบปริมาณเชิงคุณภาพมาโยงไว้กับแต่ละกิจกรรมในแบบ Waterfall

แบบ Waterfall นี้เป็นแบบจำลองกระบวนการที่เก่าแก่ที่สุดในวิศวกรรมซอฟต์แวร์ โดยมีการวิพากษ์วิจารณ์ข้อจำกัดของกระบวนการมาตลอด 40 ปี ซึ่งปัญหาที่พบในแบบ Waterfall มีดังนี้

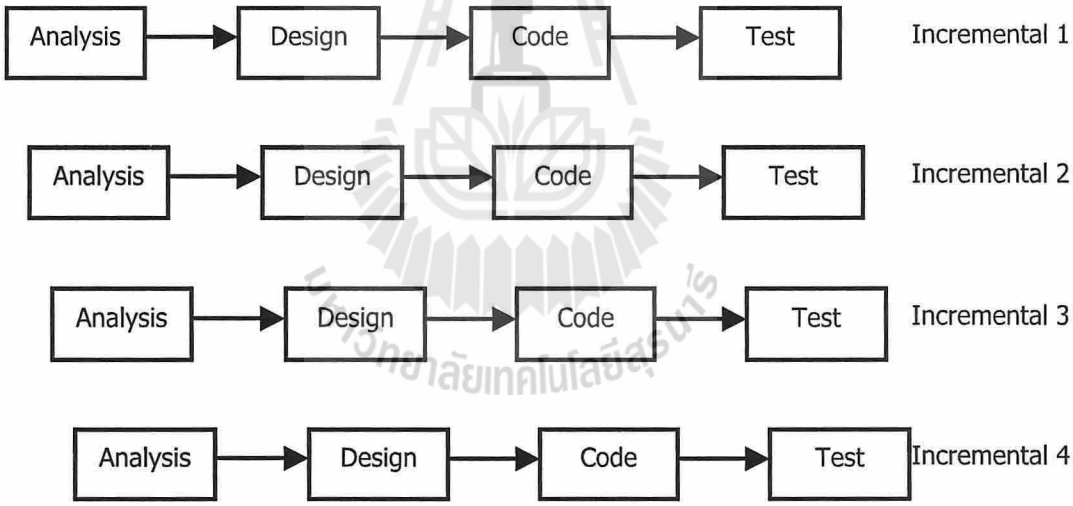
1. โครงการในความเป็นจริงจะไม่ค่อยมีกระบวนการที่เป็นลำดับต่อเนื่องกันเป็นเส้นตรงอย่างที่เสนอในแบบจำลอง
2. เป็นการยากที่ลูกค้าจะสามารถบอกความต้องการทางซอฟต์แวร์ได้อย่างถูกต้องครบถ้วนทั้งหมดตั้งแต่เริ่มโครงการ ซึ่งเป็นสิ่งที่จำเป็นสำหรับกระบวนการแบบ Waterfall
3. ลูกค้าต้องคอยซอฟต์แวร์เป็นเวลานาน

เนื่องจากซอฟต์แวร์ที่ทำงานได้จะไม่ปรากฏจนกระทั่งช่วงสุดท้ายของโครงการ ทำให้ความผิดพลาดบางอย่างจะไม่ได้ตรวจพบจนกว่าซอฟต์แวร์จะได้รับการตรวจสอบในช่วงท้ายนั้น ซึ่งอาจทำให้เกิดความเสียหายกับโครงการได้

ในการวิเคราะห์กระบวนการนี้จากโครงการจริงพบว่า กระบวนการที่เป็นลำดับต่อเนื่องกันเป็นเส้นตรงอาจทำให้เกิด "การรอ" เพราะสมาชิกในทีมต้องคอยสมาชิกอื่นทำงานบางอย่างให้เสร็จก่อน

2. แบบ Incremental Process

แบบ Incremental Process จะรวมเอาลักษณะแบบจำลองที่เป็นลำดับต่อกันเนื่องกันเป็นเส้นตรงเข้ากับขั้นตอนการทำงานเชิงขนาน โดยที่ปลายกระบวนการของแต่ละรอบ (Incremental) จะผลิตซอฟต์แวร์ที่สามารถนำส่งได้

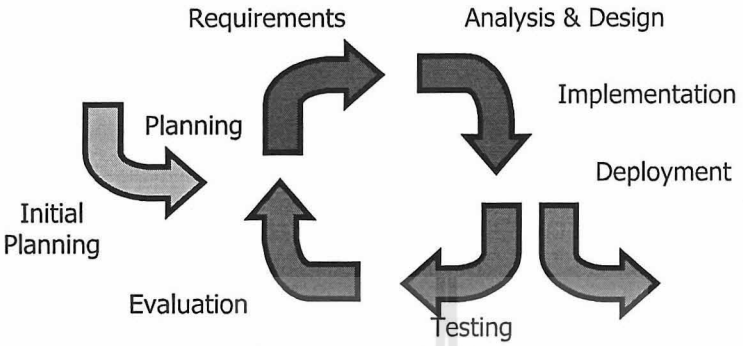


ในแบบจำลองนี้จะเรียกรอบแรกว่าการพัฒนาผลิตภัณฑ์หลัก นั่นคือที่ปลายกระบวนการในรอบแรกนั้นจะมีการนำส่งซอฟต์แวร์ซึ่งมีคุณสมบัติ (Feature) ที่จำเป็น ในขณะที่ความต้องการเพิ่มเติมนั้นจะนำไปพัฒนาในรอบถัดไป

กระบวนการแบบ Incremental Process เหมาะสมกับสถานการณ์ที่จุดเริ่มต้นของโครงการนั้นมีการเตรียมการไว้ค่อนข้างดี แต่มีกระบวนการพัฒนาที่ไม่ได้ต่อเนื่องเป็นเส้นตรง และอาจมีความต้องการให้ผู้ใช้สามารถเข้าถึงฟังก์ชั่นการทำงานของซอฟต์แวร์เพียงจำนวนหนึ่งก่อนแล้วจึงค่อยขยายความสามารถต่างๆเพิ่มเติมภายหลัง

3. แบบ Iterative

เป็นแบบจำลองการพัฒนาซอฟต์แวร์ที่อนุญาตให้เกิดการพัฒนาแบบวนรอบได้ เพื่อแก้จุดอ่อนสำคัญของกระบวนการแบบ Waterfall แบบจำลองนี้จะเริ่มด้วยการวางแผนและจบด้วยการนำซอฟต์แวร์ไปใช้ในแต่ละรอบ โดยมีกิจกรรมของการจัดการความต้องการ, การวิเคราะห์และออกแบบระบบ, การสร้าง, การทดสอบ และการประเมินผลอยู่ในแต่ละรอบ

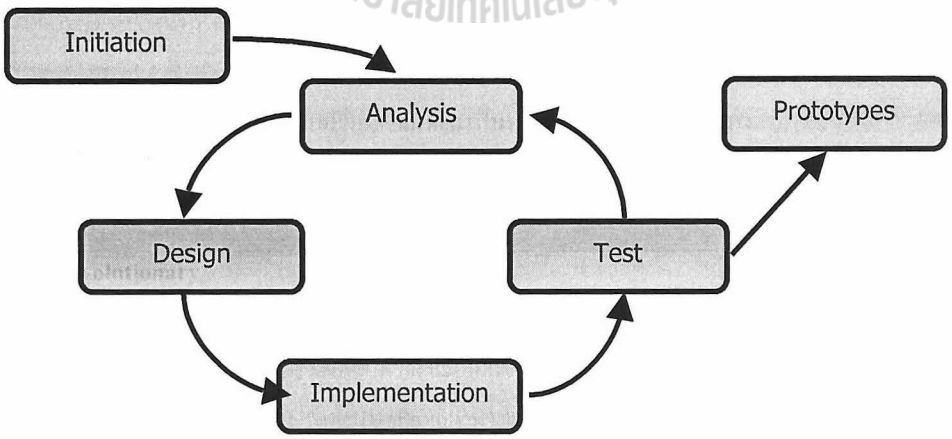


การวนรอบของแบบจำลองนี้มีข้อดีคือ ทำให้ทีมพัฒนาสามารถเรียนรู้จากการสร้างซอฟต์แวร์ในรอบก่อนหน้านี้ เพื่อประเมินและปรับปรุงการทำงานสำหรับรอบต่อไปได้

ทั้งกระบวนการแบบ Iterative และ Incremental เป็นหัวใจสำคัญในแบบจำลองอื่นๆ เช่น แบบ Unified Process และการพัฒนาแบบ Agile

4. แบบ Evolutionary

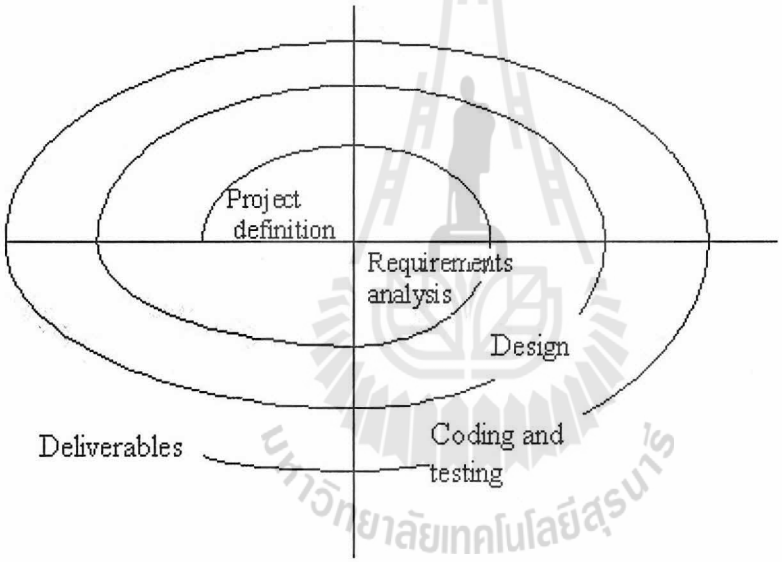
เป็นแบบจำลองที่ใช้เพื่อสร้างให้ซอฟต์แวร์ค่อยๆสมบูรณ์ขึ้นในแต่ละรอบของการพัฒนา ในสถานการณ์ที่ความต้องการซอฟต์แวร์หลักชัดเจนแล้วแต่รายละเอียดของตัวซอฟต์แวร์ที่จะสร้างยังไม่ครบถ้วน กระบวนการในแบบจำลองนี้จะเริ่มจากการสร้างต้นแบบ (Prototype) ขึ้นมาจากความต้องการที่มีก่อน จากนั้นจึงทำการนำต้นแบบไปให้ผู้ใช้ลองใช้งานแล้วนำผลตอบรับที่ได้มาปรับปรุงต้นแบบให้สมบูรณ์ขึ้นจนกลายเป็นผลิตภัณฑ์จริงที่ทำงานได้ครบถ้วน



การสร้างต้นแบบมี 2 แนวคิดใหญ่ คือ การสร้างต้นแบบแล้วทิ้ง และการสร้างต้นแบบแล้วค่อยๆปรับเป็นระบบจริง ในแบบจำลองนี้ คือ กระบวนการสนับสนุนการสร้างต้นแบบขึ้นเพื่อสามารถปรับปรุงให้เป็นระบบจริงที่ใช้งานได้ต่อไป

5. แบบ Spiral

เป็นการรวมเอากระบวนการแบบ Iterative เข้ากับความเป็นระบบของกระบวนการแบบ Waterfall แบบจำลองนี้เสนอโดย Bory Boehm ในปี ค.ศ. 1988 โดยแต่ละรอบของการพัฒนาเรียกว่า Spiral จะมีช่วงกว้างประมาณ 6 เดือนถึง 2 ปี ซึ่งผลลัพธ์จากแต่ละรอบจะเป็นซอฟต์แวร์ที่สมบูรณ์ขึ้นเรื่อยๆ ในลักษณะเดียวกับผลที่ได้ตามแบบ Evolutionary หรืออาจเป็นสิ่งที่สามารถส่งมอบได้ เช่น ข้อกำหนดของซอฟต์แวร์ (Software Specification) หรือแบบจำลองจากการออกแบบ ในรอบแรกของ Spiral มักจะเป็นการกำหนดข้อกำหนดของซอฟต์แวร์ จากนั้นใน Spiral ที่สองและถัดมาจะเป็นซอฟต์แวร์ที่สมบูรณ์ขึ้นหรือซับซ้อนขึ้น ในแต่ละรอบจะมีการผ่านระยะการวางแผน ซึ่งในจุดนี้จะเป็นการวิเคราะห์ต้นทุน, ตารางเวลา และปรับให้เหมาะสมตามการตอบรับที่ได้จากลูกค้า ซึ่งทดลองใช้ระบบที่ส่งมอบจาก Spiral ก่อนหน้านี้



6. แบบ Unified Process

เป็นกระบวนการพัฒนาซอฟต์แวร์ที่พยายามดึงเอาคุณสมบัติและลักษณะเด่นของหลายๆแบบจำลองออกมาไว้ด้วยกัน กระบวนการแบบ Unified Process ให้ความสำคัญกับการสื่อสารกับลูกค้าและอธิบายความต้องการของซอฟต์แวร์ให้อยู่ในรูปแบบ Use Case (จากมุมมองของลูกค้า) อีกทั้งกระบวนการแบบ Unified Process ยังเน้นความสำคัญของสถาปัตยกรรมซอฟต์แวร์และใช้กระบวนการงานทำงานเป็นแบบวนรอบ ซึ่งแต่ละรอบจะได้ผลลัพธ์เป็น Increment ของซอฟต์แวร์ รอบหนึ่งรอบในระยะต่างๆกันของกระบวนการแบบ Unified Process จะทำกิจกรรมในกระบวนการไม่เท่ากัน

<u>ระยะในกระบวนการแบบ Unified Process</u>

กระบวนการแบบ Unified Process แบ่งออกเป็น 4 ระยะ ประกอบไปด้วย

<u>ระยะ Interception</u>

ในระยะนี้จะเน้นการสื่อสารกับลูกค้า และการวางแผน มีการเริ่มวางสถาปัตยกรรม และจะมีการออกแบบการสร้าง และการทดสอบบ้างเล็กน้อย นอกจากนี้ยังมีการเก็บความต้องการให้อยู่ในรูปแบบของ Use Case เบื้องต้น

ระยะ Elaboration

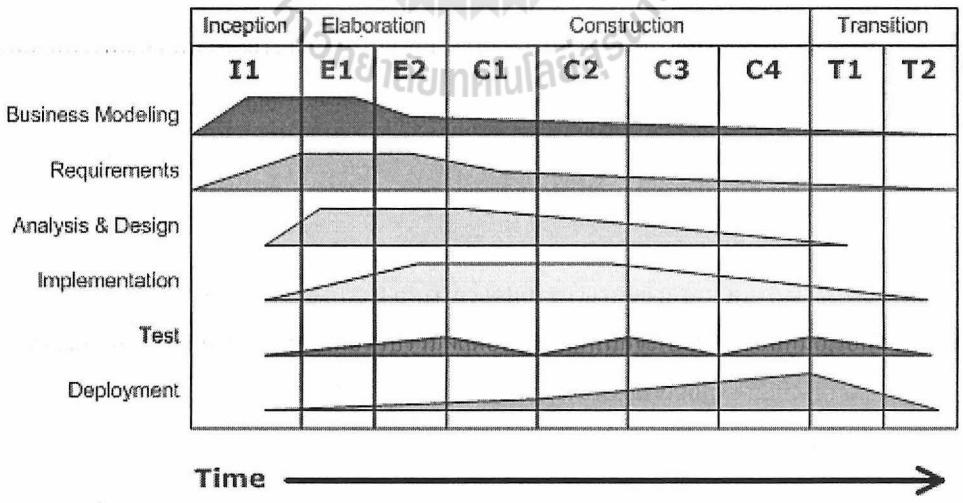
จะเน้นการสื่อสารมากที่สุด โดยทำการลงรายละเอียด Use Case ที่ได้จากระยะ Interception ให้มากขึ้น มีการวางสถาปัตยกรรมที่ครบถ้วน และสร้างแบบจำลอง Use Case, แบบจำลองการวิเคราะห์, แบบจำลองการออกแบบ, แบบจำลองการสร้าง และแบบจำลองการนำไปใช้ ในช่วงปลายของระยะนี้อาจจะมีการสร้างซอฟต์แวร์ที่ทำงานได้ตัวแรกออกมาให้กับผู้ใช้ รวมทั้งแผนงานจะถูกตรวจทานเพื่อให้มั่นใจว่าระยะเวลาสำหรับการส่งงานยังสมเหตุสมผลอยู่ ซึ่งการเปลี่ยนแปลงแผนมักจะกระทำในระยะนี้

ระยะ Construction

เป็นระยะที่เน้นไปยังการพัฒนา โดยใช้แบบจำลองเชิงสถาปัตยกรรมเป็นตัวป้อนเข้า ในระยะนี้จะเป็นการสร้างหรือหาชิ้นส่วนซอฟต์แวร์ (Software Component) มาใช้เพื่อทำให้ Use Case สามารถทำงานได้จริง และเพื่อให้การพัฒนาเป็นไปอย่างลุล่วง แบบจำลองต่างๆ ควรมีความสมบูรณ์มาจะระยะ Elaboration ทุกๆคุณสมบัติที่จำเป็นสำหรับ Increment นี้ จะถูกสร้างออกมาเป็นซอฟต์แวร์ในรูปแบบของต้นรหัส และในทุกๆชิ้นส่วนที่พัฒนาก็จะมีการทดสอบระดับหน่วยประกอบไปด้วย ในส่วนปลายระยะอาจจะมีการประกอบชิ้นส่วนซอฟต์แวร์เข้าด้วยกัน และจะมีการทดสอบเพื่อยอมรับสำหรับแต่ละ Use Case ก่อนจะเข้าสู่ระยะต่อไป

ระยะ Transition

เป็นระยะที่รวมเอาช่วงสุดท้ายของกระบวนการสร้าง และช่วงแรกของกระบวนการนำไปใช้เข้าด้วยกัน โดยจะนำซอฟต์แวร์ไปให้ผู้ใช้ทดสอบเพื่อรวบรวมผลตอบรับ จากนั้นก็จะเป็นการเขียนคู่มือ, เอกสารการแก้ปัญหา และวิธีการติดตั้ง เป็นต้น ในปลายระยะนี้ Increment ของซอฟต์แวร์จะกลายเป็นการปล่อย (Release) ซอฟต์แวร์ที่ใช้งานได้



จากภาพเป็นการพัฒนาแบบ Iterative
ที่มีการส่งมอบงานที่เกิดขึ้นอย่างต่อเนื่องตามรอบที่อยู่ในกรอบเวลาและมีการทำงานหลายกิจกรรม

7. การพัฒนาแบบ Agile

กระบวนการพัฒนาแบบ Agile เป็นการรวมเอาปรัชญาและแนวทางแบบ Agile เข้าไว้ด้วยกัน โดยปรัชญาของการพัฒนาแบบ Agile จะเน้นการทำให้ลูกค้าพึงพอใจ, การส่งมอบซอฟต์แวร์ในระยะสั้นๆ อย่างต่อเนื่อง, การมีทีมขนาดเล็กที่พร้อมและคล่องตัว, การใช้หลักวิศวกรรมซอฟต์แวร์อย่างพอดี และการพัฒนาที่มีกระบวนการเรียบง่ายและมีประสิทธิภาพ สำหรับแนวทางการพัฒนาแบบ Agile เน้นการวิเคราะห์และออกแบบให้พอเหมาะ เพื่อให้ซอฟต์แวร์ส่งมอบได้ทันเวลา และการสื่อสารกันอย่างต่อเนื่องระหว่างลูกค้าและนักพัฒนา

การพัฒนาแบบ Agile ถูกสร้างขึ้นเพื่อตอบสนองกับการเปลี่ยนแปลงที่เกิดขึ้นอย่างรวดเร็วตามสถานการณ์การพัฒนาซอฟต์แวร์ในปัจจุบัน ซึ่งมีผลทำให้ต้นทุนในการพัฒนาซอฟต์แวร์เพิ่มขึ้นมากด้วย การเปลี่ยนแปลงจึงเป็นสิ่งที่อันตรายหากไม่มีการจัดการที่เหมาะสม การพัฒนาแบบ Agile มีข้อดีที่จะสามารถช่วยลดต้นทุนของการเปลี่ยนแปลงในระหว่างการพัฒนา โดยต้นทุนของการเปลี่ยนแปลงจะเพิ่มมากขึ้นอย่างไม่เป็นเส้นตรงระหว่างเวลาที่โครงการกำลังดำเนินไป การเปลี่ยนแปลงนั้นสามารถทำได้ง่ายในช่วงต้นของโครงการ แต่จะทำได้ยากมากเมื่อโครงการดำเนินไปแล้วหลายเดือน และถ้าการเปลี่ยนแปลงสามารถกระทบกับตัวสถาปัตยกรรมได้ ต้นทุนของการเปลี่ยนดังกล่าวจะเพิ่มมากขึ้นไปอีกหลายเท่าตัว จากการศึกษาพบว่า การพัฒนาแบบ Agile ช่วยให้กราฟต้นทุนของการเปลี่ยนแปลงดังกล่าวแบนลง นั่นคือยอมให้การเปลี่ยนแปลงเกิดขึ้นในช่วงปลายของการพัฒนาได้โดยไม่เกิดผลกระทบมากนัก และหนึ่งในหลักการของ Agile คือ การยินดีที่จะเปลี่ยนแปลงความต้องการของซอฟต์แวร์ แม้จะอยู่ในช่วงปลายของการพัฒนาก็ตาม

อย่างไรก็ดีการพัฒนาแบบ Agile เป็นแนวคิดที่แฝงอยู่ในกระบวนการพัฒนาซอฟต์แวร์มาตั้งแต่อดีตแล้ว แต่ก็เพิ่งมีแนวทางชัดเจนเมื่อไม่นานมานี้เอง โดยสรุปแล้วการพัฒนาแบบ Agile สร้างขึ้นมาเพื่อแก้ไขจุดอ่อนของวิศวกรรมซอฟต์แวร์แบบดั้งเดิม ซึ่งแม้ว่าวิธีการแบบ Agile นี้จะมีประโยชน์ชัดเจน แต่ก็ไม่สามารถใช้ได้กับทุกโครงการ หรือทุกองค์กร หรือทุกบุคคลเสมอไป

การพัฒนาแบบ Agile ที่เป็นที่นิยมในปัจจุบันมีดังต่อไปนี้

1. Scrum
2. Agile Modeling
3. Agile Unified Process
4. Extreme Programming
5. OpenUP

บทที่ 3

การพัฒนาแบบ Agile

Agile Development

การพัฒนาแบบ Agile เป็นคำเรียกของกลุ่มกระบวนการพัฒนาซอฟต์แวร์ที่มีลักษณะบางอย่างร่วมกัน ซึ่งแสดงให้เห็นถึง "ความว่องไว" (Agility) ที่เกี่ยวกับการเปลี่ยนแปลง, การวางแผน, การสื่อสาร และการเรียนรู้ โดยลักษณะของกระบวนการพัฒนามักอยู่ในรูปแบบตามแบบ Iterative และแบบ Incremental ในปี ค.ศ. 2001 ผู้คิดค้นกระบวนการพัฒนาซอฟต์แวร์จำนวน 17 คนได้พูดคุยเพื่อร่วมกันหาจุดร่วมจากแต่ละกระบวนการ โดยสรุปออกมาเป็นคำแถลงการณ์ของการพัฒนาซอฟต์แวร์แบบ Agile (Manifesto for Agile Software Development) ดังนี้

"เรากำลังเปิดเผยวิธีการที่ดีกว่าในการพัฒนาซอฟต์แวร์โดยลงมือทำและช่วยผู้อื่นทำ
ซึ่งเราให้คุณค่าของ:

บุคคลและการติดต่อสื่อสาร	มากกว่า	กระบวนการและเครื่องมือ
ซอฟต์แวร์ที่ทำงานได้	มากกว่า	เอกสารที่ละเอียด
ความร่วมมือของลูกค้า	มากกว่า	การเจรจาด้วยสัญญา
การตอบสนองต่อการเปลี่ยนแปลง	มากกว่า	การทำตามแผนงาน

นั่นคือ ในขณะที่สิ่งทางขวามือมีคุณค่า เราให้คุณค่าสิ่งทางซ้ายมือมากกว่า"

ในกลุ่มการพัฒนาซอฟต์แวร์แบบ Agile นั้นมีการตั้งหลักการของ Agile (Agile Principles) ไว้จำนวน 12 ข้อ เพื่ออธิบายคุณค่าที่กล่าวไว้ในคำแถลงการณ์ให้ละเอียดและเป็นรูปธรรมมากขึ้น ดังนี้

1. Agile ให้ความสำคัญสูงสุดในการทำให้ลูกค้าพึงพอใจผ่านทางการนำส่งซอฟต์แวร์ที่ใช้งานได้อย่างต่อเนื่องตั้งแต่ช่วงแรกของการพัฒนา โดยหลังจากมีการเตรียมร่างของโครงการเรียบร้อยแล้ว การพัฒนาแบบ Agile จะหาวิธีนำส่งซอฟต์แวร์ให้ได้เร็วที่สุดเท่าที่จะทำได้ โดยวิธีการเช่นนี้สามารถเข้าไปแทนที่การจัดการความต้องการและระยะการออกแบบที่พบได้ในกระบวนการเดิมๆ วิธีการทาง Agile จะเหมือนการทำต้นแบบในกระบวนการอื่นๆ แต่ต่างออกไปตรงจะเป็นการสร้างซอฟต์แวร์ที่ใช้จริงได้แทนการสร้างต้นแบบแล้วทิ้ง

   การพัฒนาซอฟต์แวร์แบบ Agile จะใช้กลยุทธ์การพัฒนาแบบ Increment และลดความเสี่ยงของโครงการโดยการส่งมอบอย่างรวดเร็วและต่อเนื่อง เพื่อลูกค้าจะได้สามารถตรวจทานและทวนสอบการตัดสินใจและสมมติฐานของโครงการว่าเป็นไปตามที่ลูกค้าต้องการหรือไม่ รวมทั้งสามารถช่วยตรวจสอบประโยชน์การใช้งานจากซอฟต์แวร์ที่ส่งมอบในแต่ละครั้งได้อย่างต่อเนื่อง

2. กระบวนการแบบ Agile ยินดีที่จะเปลี่ยนแปลงความต้องการแม้ในช่วงหลังการพัฒนาซอฟต์แวร์เพื่อให้ได้ประโยชน์สูงสุดแก่ลูกค้า หลักการนี้เป็นหัวใจสำคัญของ Agile ที่สะท้อนจากชื่อของกระบวนการ ซึ่งแสดงถึงความคล่องตัวโดยยอมรับว่าการเปลี่ยนแปลงเป็นสิ่งที่ต้องเกิดขึ้นและหลีกเลี่ยงไม่ได้ Agile จึงตั้งปรัชญาที่ยอมให้เกิดความเปลี่ยนแปลงตลอดกระบวนการพัฒนาแทนความพยายามในการปิดกั้นหรือควบคุมไม่ให้เปลี่ยนแปลงเพื่อให้เกิดประโยชน์ต่อลูกค้ามากที่สุด

3. ส่งมอบซอฟต์แวร์ที่ใช้งานได้อย่างสม่ำเสมอโดยในช่วงเวลาระดับไม่กี่สัปดาห์หรือไม่กี่เดือน โดยให้ช่วงเวลาสั้นที่สุดเท่าที่จะเป็นไปได้ หลักการข้อนี้เป็นการขยายความการส่งมอบซอฟต์แวร์อย่างต่อเนื่อง

จากหลักการแรก โดยเน้นว่าแต่ละ Increment ในโครงการที่ใช้ Agile ควรจะสั้นที่สุดเท่าที่จะสั้นได้ และเน้นต่อไปอีกว่าระยะเวลาหนึ่งหรือสองสัปดาห์ไม่ได้สั้นจนเกินไปสำหรับ Increment หลักการนี้ยังตั้งเงื่อนไขเวลามากสุดไว้ที่ 2-3 เดือน ซึ่งเมื่อเทียบกับระยะในการพัฒนาซอฟต์แวร์โดยทั่วไปมักจะไม่ต่ำกว่า 3 เดือน

4. บุคคลากรทางธุรกิจและนักพัฒนาต้องทำงานด้วยกันทุกวันตลอดโครงการ เพราะวิธีการประเภท Agile ต้องการการติดต่อสื่อสารระหว่างทีมพัฒนาและลูกค้าตลอดเวลา โดยบทบาทที่สำคัญที่สุดในโครงการนั้นอยู่ที่ลูกค้า รวมทั้งผู้มีส่วนในโครงการในบทบาทอื่น ก็มีความจำเป็นที่จะต้องพูดคุยกับทีมพัฒนาเป็นประจำ

5. สร้างโครงการรอบๆบุคคลที่ตั้งใจ รวมทั้งสามารถเตรียมสภาพแวดล้อมที่สนับสนุนบุคคลในทีมและมีความเชื่อมั่นว่าทีมจะทำงานได้สำเร็จ หลักการนี้ของวิธีการแบบ Agile สร้างอยู่บนสมมติฐานที่ว่าบุคคลากรในทีมพัฒนามีความทุ่มเท ตั้งใจและได้รับการสนับสนุนจากองค์กรเป็นอย่างดีเพื่อส่งเสริมให้ทำงานได้อย่างมีประสิทธิภาพสูงสุด ในขณะเดียวกันกระบวนการแบบ Agile จะเน้นการสร้างสภาพแวดล้อมการทำงานที่กระตุ้นให้ทีมพัฒนาแต่ละคนเรียนรู้และจัดการด้วยตัวเองอย่างเป็นธรรมชาติที่สุดเท่าที่จะทำได้

6. วิธีการที่มีประสิทธิภาพและประสิทธิผลมากที่สุดในการได้มาซึ่งข้อมูลสำหรับการพัฒนา คือ การพูดคุยกันตัวต่อตัว หลักการนี้ของ Agile แสดงเหตุผลถึงการลดการทำงานเอกสารลง ในกระบวนการแบบ Agile จะใช้การสื่อสารแบบตัวต่อตัวเป็นกลไกหลักในการแลกเปลี่ยนข้อมูลกัน โดยมีเครื่องมือ เช่น กระดานไวท์บอร์ด ช่วยในการบันทึก จากนั้นข้อมูลที่ได้อาจจะบันทึกลงเป็นเอกสารในรูปแบบที่เรียกว่า "ตัวแผ่ข้อมูล" (Information Radiation) ซึ่งผู้เกี่ยวข้องกับข้อมูลนั้นๆต้องสามารถเข้าถึงได้ทันที ตัวอย่างของตัวแผ่ข้อมูลที่ใช้กัน เช่น การบันทึกข้อลงในกระดาษโน๊ตที่แยกเป็นสีๆแล้วแปะไว้บนกระดานไวท์บอร์ด เป็นต้น

7. ใช้ซอฟต์แวร์ที่ทำงานได้เป็นตัวหลักในการวัดความก้าวหน้าของโครงการ หลักการนี้เป็นอีกหลักการหนึ่งของ Agile ที่ระบุไว้เพื่อเน้นว่า ซอฟต์แวร์สำคัญกว่าเอกสารประกอบ เนื่องจากโครงการส่วนใหญ่เน้นการทำงานเอกสารข้อกำหนดความต้องการซอฟต์แวร์และระบุให้เป็นไมล์สโตน (Milestone) เพื่อแสดงถึงความก้าวหน้าของโครงการ แต่ในกระบวนการแบบ Agile นั้นจะให้คุณค่ากับซอฟต์แวร์ที่เป็นผลลัพธ์มากกว่าเอกสารประกอบ และเนื่องจากมีการส่งมอบซอฟต์แวร์ที่ทำงานอย่างต่อเนื่องตั้งแต่ช่วงต้นของโครงการ ซอฟต์แวร์จึงสามารถเป็นตัววัดความก้าวหน้าที่สมเหตุสมผล

8. กระบวนการแบบ Agile สนับสนุนการพัฒนาซอฟต์แวร์ที่ยั่งยืน โดยผู้สนับสนุนโครงการ, ผู้พัฒนาและลูกค้าควรจะรักษาอัตราการทำงานให้คงที่ต่อเนื่อง ในหลายๆองค์กรทำงานนอกเวลาจนเป็นเรื่องธรรมดา และอีกหลายองค์กรต้องการการทำงานล่วงเวลาอยู่บ่อยครั้ง แต่สำหรับกระบวนการแบบ Agile จะเน้นให้ตารางเวลาคงที่และยั่งยืน นั่นคือ การวางแผนงานไว้ที่ 40 ชั่วโมงต่อสัปดาห์ ก็จะต้องใช้เวลา 40 ชั่วโมงให้ได้ประสิทธิภาพที่สุดและไม่ทำเกินแผนที่วางไว้ ตามหลักการนี้กระบวนการแบบ Agile จะเน้นให้รักษาการทำงานให้คงที่สม่ำเสมออย่างเป็นธรรมชาติ

9. ความสนใจในการพัฒนาความเป็นเลิศทั้งทางเทคนิคและการออกแบบอย่างต่อเนื่องเพื่อเพิ่มความคล่องตัวในการทำงานให้ดียิ่งขึ้น หลักการนี้เน้นให้สมาชิกในทีมพัฒนาแบบ Agile ปรับปรุงคุณภาพและทักษะให้สูงขึ้นเรื่อยๆ เพื่อให้แน่ใจได้ว่างานที่พัฒนาออกมาจะมีคุณภาพอยู่ในระดับสูง โดยงานของนักพัฒนาไม่ใช่เพียงแต่เขียนโปรแกรมแล้วมีผู้อื่นมาเป็นคนตรวจสอบเพียงอย่างเดียว

จากหลักการแรก โดยเน้นว่าแต่ละ Increment ในโครงการที่ใช้ Agile ควรจะสั้นที่สุดเท่าที่จะสั้นได้ และเน้นต่อไปอีกว่าระยะเวลาหนึ่งหรือสองสัปดาห์ไม่ได้สั้นจนเกินไปสำหรับ Increment หลักการนี้ยังตั้งเงื่อนไขเวลามากสุดไว้ที่ 2-3 เดือน ซึ่งเมื่อเทียบกับระยะในการพัฒนาซอฟต์แวร์โดยทั่วไปมักจะไม่ต่ำกว่า 3 เดือน

4. บุคคลากรทางธุรกิจและนักพัฒนาต้องทำงานด้วยกันทุกวันตลอดโครงการ เพราะวิธีการประเภท Agile ต้องการการติดต่อสื่อสารระหว่างทีมพัฒนาและลูกค้าตลอดเวลา โดยบทบาทที่สำคัญที่สุดในโครงการนั้นอยู่ที่ลูกค้า รวมทั้งผู้มีส่วนในโครงการในบทบาทอื่น ก็มีความจำเป็นที่จะต้องพูดคุยกับทีมพัฒนาเป็นประจำ

5. สร้างโครงการรอบๆบุคคลที่ตั้งใจ รวมทั้งสามารถเตรียมสภาพแวดล้อมที่สนับสนุนบุคคลในทีมและมีความเชื่อมั่นว่าทีมจะทำงานได้สำเร็จ หลักการนี้ของวิธีการแบบ Agile สร้างอยู่บนสมมติฐานที่ว่าบุคคลากรในทีมพัฒนามีความทุ่มเท ตั้งใจและได้รับการสนับสนุนจากองค์กรเป็นอย่างดีเพื่อส่งเสริมให้ทำงานได้อย่างมีประสิทธิภาพสูงสุด ในขณะเดียวกันกระบวนการแบบ Agile จะเน้นการสร้างสภาพแวดล้อมการทำงานที่กระตุ้นให้ทีมพัฒนาแต่ละคนเรียนรู้และจัดการด้วยตัวเองอย่างเป็นธรรมชาติที่สุดเท่าที่จะทำได้

6. วิธีการที่มีประสิทธิภาพและประสิทธิผลมากที่สุดในการได้มาซึ่งข้อมูลสำหรับการพัฒนา คือ การพูดคุยกันตัวต่อตัว หลักการนี้ของ Agile แสดงเหตุผลถึงการลดการทำงานเอกสารลง ในกระบวนการแบบ Agile จะใช้การสื่อสารแบบตัวต่อตัวเป็นกลไกหลักในการแลกเปลี่ยนข้อมูลกัน โดยมีเครื่องมือ เช่น กระดานไวท์บอร์ด ช่วยในการบันทึก จากนั้นข้อมูลที่ได้อาจจะบันทึกลงเป็นเอกสารในรูปแบบที่เรียกว่า "ตัวแผ่ข้อมูล" (Information Radiation) ซึ่งผู้เกี่ยวข้องกับข้อมูลนั้นๆต้องสามารถเข้าถึงได้ทันที ตัวอย่างของตัวแผ่ข้อมูลที่ใช้กัน เช่น การบันทึกข้อลงในกระดาษโน๊ตที่แยกเป็นสีๆแล้วแปะไว้บนกระดานไวท์บอร์ด เป็นต้น

7. ใช้ซอฟต์แวร์ที่ทำงานได้เป็นตัวหลักในการวัดความก้าวหน้าของโครงการ หลักการนี้เป็นอีกหลักการหนึ่งของ Agile ที่ระบุไว้เพื่อเน้นว่า ซอฟต์แวร์สำคัญกว่าเอกสารประกอบ เนื่องจากโครงการส่วนใหญ่เน้นการทำงานเอกสารข้อกำหนดความต้องการซอฟต์แวร์และระบุให้เป็นไมล์สโตน (Milestone) เพื่อแสดงถึงความก้าวหน้าของโครงการ แต่ในกระบวนการแบบ Agile นั้นจะให้คุณค่ากับซอฟต์แวร์ที่เป็นผลลัพธ์มากกว่าเอกสารประกอบ และเนื่องจากมีการส่งมอบซอฟต์แวร์ที่ทำงานอย่างต่อเนื่องตั้งแต่ช่วงต้นของโครงการ ซอฟต์แวร์จึงสามารถเป็นตัววัดความก้าวหน้าที่สมเหตุสมผล

8. กระบวนการแบบ Agile สนับสนุนการพัฒนาซอฟต์แวร์ที่ยั่งยืน โดยผู้สนับสนุนโครงการ, ผู้พัฒนาและลูกค้าควรจะรักษาอัตราการทำงานให้คงที่ต่อเนื่อง ในหลายๆองค์กรทำงานนอกเวลาจนเป็นเรื่องธรรมดา และอีกหลายองค์กรต้องการการทำงานล่วงเวลาอยู่บ่อยครั้ง แต่สำหรับกระบวนการแบบ Agile จะเน้นให้ตารางเวลาคงที่และยั่งยืน นั่นคือ การวางแผนงานไว้ที่ 40 ชั่วโมงต่อสัปดาห์ ก็จะต้องใช้เวลา 40 ชั่วโมงให้ได้ประสิทธิภาพที่สุดและไม่ทำเกินแผนที่วางไว้ ตามหลักการนี้กระบวนการแบบ Agile จะเน้นให้รักษาการทำงานให้คงที่สม่ำเสมออย่างเป็นธรรมชาติ

9. ความสนใจในการพัฒนาความเป็นเลิศทั้งทางเทคนิคและการออกแบบอย่างต่อเนื่องเพื่อเพิ่มความคล่องตัวในการทำงานให้ดียิ่งขึ้น หลักการนี้เน้นให้สมาชิกในทีมพัฒนาแบบ Agile ปรับปรุงคุณภาพและทักษะให้สูงขึ้นเรื่อยๆ เพื่อให้แน่ใจได้ว่างานที่พัฒนาออกมาจะมีคุณภาพอยู่ในระดับสูง โดยงานของนักพัฒนาไม่ใช่เพียงแต่เขียนโปรแกรมแล้วมีผู้อื่นมาเป็นคนตรวจสอบเพียงอย่างเดียว

แต่นักพัฒนาจะต้องเป็นกลุ่มคนที่รับผิดชอบหลักในการส่งมอบสิ่งที่ถูกต้องและทดสอบแล้วว่าตรงตามความต้องการของลูกค้า อย่างไรก็ตามการทวนสอบโดยบุคคลอื่นก็ยังเป็นสิ่งจำเป็นเช่นกัน

10. ความเรียบง่ายเป็นสิ่งจำเป็นยิ่งยวด กระบวนการแบบ Agile เน้นการใช้กระบวนการพัฒนาซอฟต์แวร์ที่เบาที่สุดเท่าที่จะเป็นไปได้ อย่างไรก็ตามความเรียบง่ายไม่ได้หมายถึงความมักง่าย

11. ทั้งสถาปัตยกรรม, ข้อกำหนดความต้องการและการออกแบบ จะปรากฏออกมาจากทีมพัฒนาเอง การจัดการด้วยตนเองในทีมเป็นจุดเด่นของกระบวนการแบบ Agile ซึ่งตรงข้ามกับการควบคุมจากภายนอกที่พบเห็นในกระบวนการอื่นๆ อย่างไรก็ตามการสร้างทีมที่สามารถจัดการตัวเองได้นั้นจำเป็นต้องมีการเปลี่ยนแปลงจากระดับองค์กร การเปลี่ยนแปลงในลักษณะนี้ทำได้ยากและจำเป็นให้บุคคลากรเรียนรู้ทักษะและพฤติกรรมใหม่ แต่หากทำได้แล้วจะได้ผลลัพธ์ในเชิงบวกอย่างมีนัยสำคัญ

12. ในแต่ละช่วงเวลาทีมพัฒนาต้องสามารถประเมินได้ว่าจะเพิ่มประสิทธิภาพในการทำงานได้อย่างไร จากนั้นจึงปรับพฤติกรรมให้เป็นไปตามการประเมิน กระบวนการแบบ Agile มีการปรับใช้กิจกรรมตามหลักการนี้เพื่อกระตุ้นให้เกิดการปรับปรุงกระบวนการพัฒนา กระบวนการแบบ Agile บางแบบมีการบังคับให้ทำการประเมินในทุกๆระยะของการพัฒนาส่งผลให้เกิดการเรียนรู้และปรับวิธีการทำงานหลายๆครั้งภายใน 1 ปี

แบบจำลองกระบวนการแบบ Agile (Agile Process Model)

กระบวนการพัฒนาแบบ Agile ที่ได้รับความนิยมมีด้วยกันหลายกระบวนการดังนี้

1. Scrum

เป็นกระบวนการที่คิดค้นโดย Jeff Sutherland และทีมในช่วงต้นของทศวรรษ 1990 ในยุคหลัง Scrum พัฒนาต่อโดย Schawaber และ Beedle หลักการของ Scrum เป็นไปตามคำแถลงการณ์ของ Agile และหลักการเหล่านี้ถูกใช้เพื่อเป็นแนวทางให้กับกิจกรรมต่างๆในการพัฒนา โดยอยู่ในรูปแบบของกระบวนการเป็นรอบ เรียกว่า Sprint โดยงานที่นำเข้าไปทำในแต่ Sprint นั้นจะนำมาจาก Product backlog และจะมีการคุยกันทุกๆวันประมาณ 15 นาทีเรียกว่า Scrum Meeting

2. Agile Modeling

เป็นวิธีการสำหรับสร้างแบบจำลองและเอกสารของระบบซอฟต์แวร์โดยใช้แนวทางของ Agile ซึ่งสามารถใช้เป็นตัวเสริมให้กับกระบวนการอื่นๆ เช่น Extreme Programming, Agile Unified Process หรือ Scrum ได้ และสามารถแทนการใช้ UML ในกระบวนการนั้นๆได้

3. Agile Unified Process

เป็นการประยุกต์ใช้ระยะการพัฒนาที่พบใน Unified Process ได้แก่ Inception, Elaboration, Construction และ Transition เข้ากับปรัชญาแบบ Agile โดยแต่ละรอบจะมีกิจกรรมดังต่อไปนี้

- Modeling เป็นการสร้างแบบจำลองธุรกิจและโดเมนปัญหาด้วย UML โดยแบบจำลองเหล่านี้สร้างเพียงแค่ให้ “พอใช้ได้” เพื่อให้ทีมสามารถทำส่วนถัดไปได้อย่างต่อเนื่อง
- Implementation เป็นการแปลงแบบจำลองเป็นต้นรหัส (Source Code) ของโปรแกรม
- Testing เป็นการทดสอบเพื่อให้แน่ใจว่าต้นรหัสเป็นไปตามข้อกำหนด
- Deployment เป็นการนำซอฟต์แวร์ในส่วนที่พัฒนาไปส่งมอบแบบ Increment เพื่อรวบรวมผลตอบรับจากผู้ใช้

- Configuration และ Project Management ในบริบทของ Agile Unified Process เป็นการจัดการความเปลี่ยนแปลงและความเสี่ยง รวมทั้งการติดตามงานและควบคุมดูแลความก้าวหน้าของโครงการ
- Environment Management
  เป็นการจัดการเพื่ออำนวยความสะดวกให้กับกระบวนการรวมถึงการใช้มาตรฐาน, เครื่องมือและเทคโนโลยีสนับสนุนอื่นๆ

4. Extreme Programming (XP)

เป็นวิธีการพัฒนาซอฟต์แวร์แบบ Agile ที่ใช้กันมากระดับหนึ่ง ซึ่ง XP มีการนิยามคุณค่า (value) 5 ประการขึ้นมาเป็นรากฐาน โดยคุณค่าทั้ง 5 ประการนี้เป็นตัวขับเคลื่อนกิจกรรม, การกระทำ และงานใน XP ซึ่งได้แก่

การสื่อสาร communication

ความเรียบง่าย simplicity

การตอบรับ feedback

ความมีวินัย discipline

ความเคารพ respect

5. OpenUP

เป็นกระบวนการพัฒนาซอฟต์แวร์ที่สร้างขึ้นโดยสกัดส่วนหลักของ Unified Process ออกมา และปรับใช้ร่วมกับปรัชญาทาง Agile ซึ่งเน้นการพัฒนาซอฟต์แวร์เชิงความร่วมมือ อย่างไรก็ตาม OpenUP ยังคงลักษณะสำคัญของ Unified Process ซึ่งเป็นการพัฒนาแบบ Incremental และยังมีการใช้ Use Case ในการขับเคลื่อนการพัฒนา นอกจากนี้ยังคงการจัดการความเสี่ยงแบบ Unified Process ไว้ รวมทั้งยังคงใช้แนวทางที่มีสถาปัตยกรรมเป็นศูนย์กลาง

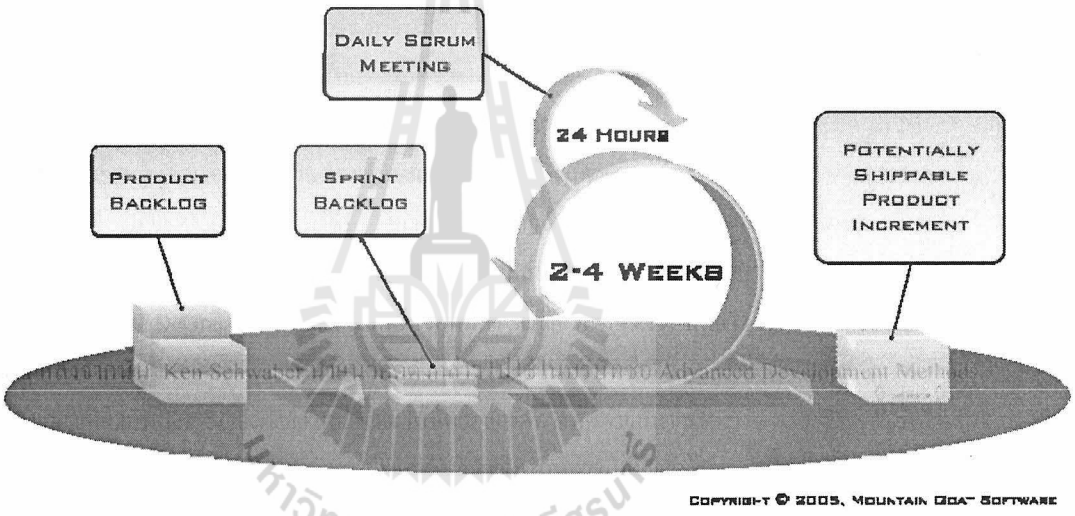
บทที่ 4

การพัฒนาแบบ Scrum

Scrum Development

เป็นการกระบวนการพัฒนาซอฟต์แวร์แบบ Agile ที่ใช้การวนรอบ (Iterative) และการเพิ่มส่วนขึ้น (Incremental) สำหรับจัดการโครงการ โดยจุดเริ่มต้นของ Scrum เกิดขึ้นในปี ค.ศ. 1986 โดย Takeuchi และ Nonaka อธิบายถึงแนวทางแบบใหม่ที่ใช้ทีมเดียวซึ่งมีความสามารถหลายอย่างทำงานข้ามหน้าที่ได้ มีระยะการพัฒนาหลายระยะ โดยมีชื่อเหมือนกีฬารักบี้ แนวคิดดังกล่าวนี้นำมาจากกรณีศึกษาในหลายๆอุตสาหกรรม

ในปี ค.ศ. 1991 DeGrace และ Stahl เรียกแนวคิดนี้ว่า Scrum ตามคำเรียกที่กล่าวไว้โดย Takeuchi และ Nonaka หลังจากนั้น Ken Schwaber นำแนวคิดดังกล่าวไปใช้ในบริษัทชื่อ Advanced Development Methods และช่วงเวลาเดียวกัน Jeff Sutherland และคณะได้พัฒนาแนวทางที่คล้ายกันขึ้นในบริษัท Easel และในปี ค.ศ. 1995 Schwaber และ Sutherland ร่วมกันนำเสนองานวิจัยเกี่ยวกับการอธิบาย Scrum ในการประชุมวิชาการ OOPSLA'95 ซึ่งเป็นการนำแนวคิดดังกล่าวมาสู่สาธารณะเป็นครั้งแรก



ใน Scrum จะมี Scrum team ซึ่งถูกออกแบบให้มีความยืดหยุ่น, มีการทำงานข้ามหน้าที่และการทำงานเป็นรอบ และมีแนวคิดที่เกี่ยวข้องกับ Scrum Team คือ

- กรอบเวลา (Timebox)
- ชิ้นงาน (Artifacts)
- บทบาท (Roles)

บทบาทใน Scrum Team มีดังต่อไปนี้

1. Scrum Master คือ ผู้ดูแลกระบวนการทำงาน
2. Product Owner คือ ตัวแทนของผู้ประกอบการ, ตัวแทนของลูกค้า
3. Team คือ กลุ่มผู้พัฒนาซึ่งมีจำนวนประมาณ 5-9 คน ซึ่งทุกคนในกลุ่มจะมีความสามารถในการวิเคราะห์ออกแบบ เขียนโปรแกรม และทดสอบ (ทำงานข้ามหน้าที่ได้)

Scrum ใช้เทคนิคการวางแผนที่เรียกว่า กรอบเวลา เพื่อสร้างระเบียบในการทำงาน โดยองค์ประกอบของ Scrum ที่อยู่ในกรอบเวลามีดังนี้

1. การประชุมวางแผน Release (Release Planning)
2. การทำ Sprint (Sprint)
3. การประชุมวางแผน sprint (Sprint Planning)
4. การประชุม Scrum รายวัน (Daily Scrum)
5. การตรวจทาน Sprint (Sprint Review)
6. การทบทวน Sprint ย้อนหลัง (Sprint Retrospective)

โดยหัวใจของ Scrum คือ Sprint ซึ่งเป็นรอบของการพัฒนาที่มีช่วงประมาณ 1 เดือนหรือน้อยกว่า และจะต้องเท่าๆกันตลอดทั้งกระบวนการพัฒนา โดยทุกๆรอบของ Sprint จะใช้แนวคิดจากกรอบงาน Scrum เดียวกัน และทุกๆ Sprint จะส่งมอบส่วนเพิ่มของผลิตภัณฑ์ที่ทำงานได้จริง โดยจะทำการ Sprint ต่อเนื่องไปตลอดจนกว่าจะหมดกระบวนการพัฒนา

ใน Scrum จะมีการใช้ชิ้นงานหลัก 4 ประเภท คือ

- Product Backlog เป็นงานรวมของทั้งระบบ
- Sprint Backlog เป็นงานสำหรับแต่ละรอบของ Sprint
- Release Burndown Chart เป็นกราฟวัดงานที่เหลือในแต่ละช่วงแผนการ Release
- Sprint Burndown Chart เป็นกราฟวัดงานที่เหลือในแต่ละช่วง Sprint

สำหรับบทบาทของบุคคลใน Scrum Team ที่ประกอบไปด้วย ScrumMaster, Product Owner และ Team นั้น จะมีบทบาทอีก 2 บทบาท คือ

บทบาทของ "ผู้ทำ" และ "ผู้มีส่วนร่วม" เช่น

- Product Owner จะเป็น "ผู้ทำ" Product Backlog และจะเป็น "ผู้มีส่วนร่วม" ในส่วนอื่น
- Team จะเป็น "ผู้ทำ" งานใน Sprint
- ScrumMaster จะเป็น "ผู้ทำ" ในกระบวนการ Scrum

ส่วนบทบาท "ผู้มีส่วนร่วม" จะไม่ใช่ส่วนหนึ่งของกระบวนการแบบ Scrum จริงๆ แต่จะต้องมีบทบาทในการเข้ามาร่วมพิจารณา

บทบาทของ ScrumMaster

มีหน้าที่รับผิดชอบกระบวนการทั้งหมดเพื่อให้แน่ใจว่า Scrum Team สามารถทำตามกฎเกณฑ์ที่กำหนดไว้ โดย ScrumMaster จะช่วยให้ Scrum Team และองค์กรปรับใช้ Scrum รวมทั้งช่วยแนะนำทีมให้สามารถพัฒนาซอฟต์แวร์ให้มีประสิทธิภาพสูงขึ้น อีกทั้ง ScrumMaster จะมีหน้าที่เสริมความเข้าใจให้กับทีมในการจัดรูปแบบภายในทีมด้วยตนเอง และให้สมาชิกในทีมทำงานแทนกันได้ บทบาทของ ScrumMaster รวมถึงการช่วยปรับปรุงกระบวนการโดยการปรับเปลี่ยนวัฒนาธรรมบางอย่างในองค์กร และเมื่อเกิดการเปลี่ยนแปลงในทางที่ดีขึ้นเราจะเรียกว่า การกำจัดอุปสรรค (Removing Impediments)

บทบาทของ Product Owner

เป็นผู้รับผิดชอบเพียงผู้เดียวในการจัดการ Product Backlog มีบทบาทในการสร้างและดูแล Product Backlog คอยติดตามงานของทีมพัฒนาและทำให้มั่นใจได้ว่า ทุกคนในทีมสามารถรับรู้และเข้าใจถึงรายการของ Backlog ได้ รวมทั้งทุกคนในทีมต้องรู้ว่าจะทำ Backlog ไหนก่อนหลัง และรู้ว่าใครจะทำงานอะไร Product owner จะต้องเป็นบุคคลคนเดียวไม่ใช่กลุ่มคน โดยในความเป็นจริงแล้วกลุ่มคนอาจจะมีอยู่ได้เพื่อให้คำปรึกษาแก่ Product Owner แต่ไม่ว่าใครก็ตามจะไม่สามารถเปลี่ยนแปลงแก้ไขลำดับความสำคัญหรือรายละเอียดของ Backlog ได้หากไม่ผ่าน Product Owner และเพื่อให้บทบาทนี้ทำงานได้อย่างมีประสิทธิภาพ องค์กรจะต้องเคารพการตัดสินใจของผู้ที่เป็น Product Owner ซึ่งการตัดสินใจของ Product Owner จะสะท้อนออกมาในรูปของเนื้อหาและลำดับความสำคัญของ Product Backlog นั่นคือ Product Owner จะต้องเป็นบุคคลที่มีความสามารถในการอธิบายความต้องการเชิงซอฟต์แวร์ได้ดีที่สุด เพื่อเตรียมข้อมูลดังกล่าวเป็น Backlog

บทบาทของ Team

ทีมพัฒนามีหน้าที่แปลง Product Backlog ให้กลายเป็นส่วนที่เพิ่มขึ้นที่สามารถทำงานได้และส่งมอบได้ในทุกๆรอบของการ Sprint และทีมจะต้องสามารถทำงานข้ามหน้าที่กันได้ นั่นคือ สมาชิกทุกๆคนในทีมจะต้องมีทุกทักษะที่จำเป็นในการสร้างส่วนที่เพิ่มขึ้นของงาน โดยเฉพาะอย่างยิ่งทักษะในการทำความเข้าใจความต้องการเชิงซอฟต์แวร์แล้วเปลี่ยนให้กลายเป็นซอฟต์แวร์ที่ใช้งานได้ บุคคลที่ปฏิเสธการเขียนโปรแกรมเพราะเป็นสถาปนิคซอฟต์แวร์, นักออกแบบหรือนักวิเคราะห์เพียงอย่างเดียวนั้นไม่ควรนำมาร่วมทีม ทุกคนในทีมต้องให้ความร่วมมือถึงแม้จะต้องเรียนรู้ทักษะใหม่ๆก็ตาม ในทีมจะไม่มีชื่อตำแหน่ง ทุกๆคนจะเป็นนักพัฒนา ทีมจะต้องจัดการตัวทีมเอง จะไม่มีใครมาบอกแม้แต่ ScrumMaster ว่า นักพัฒนาในทีมควรจะเปลี่ยน Product Backlog ให้กลายเป็นซอฟต์แวร์ได้อย่างไร การทำงานร่วมกันจะเป็นตัวช่วยในการปรับปรุงประสิทธิภาพของทีมขึ้นเอง และทีมควรจะมีจำนวนสมาชิก 5-9 คน

กิจกรรมใน Scrum จะเป็นกิจกรรมที่ระบุขอบเขตของเวลาชัดเจนด้วยกลไกของกรอบเวลา โดยมีดังต่อไปนี้

1. Release Planning Meeting

   เป็นการประชุมเพื่อวางแผนการ Release และเป้าหมายที่ Scrum Team และองค์กรสามารถเข้าใจร่วมกันได้ การวางแผน Release มีเพื่อตอบคำถามต่อไปนี้

- เราสามารถเปลี่ยนวิสัยทัศน์ให้กลายเป็นผลิตภัณฑ์ที่ดีได้อย่างไร
- เราสามารถตอบสนองความพึงพอใจของลูกค้า ให้ผลตอบแทนการลงทุนที่ดีหรือเกินความคาดหวังได้อย่างไร

  แผนการ Release จะทำการกำหนด

  1. เป้าหมายของการ Release
  2. Product Backlog เรียงความสำคัญจากมากไปหาน้อย
  3. ความเสี่ยงที่สำคัญ
  4. คุณสมบัติและความสามารถที่จะส่งมอบใน Release
  5. วันส่งมอบเบื้องต้น

6. ประเมินต้นทุนบนสมมติฐานที่ว่าถ้าไม่มีอะไรเปลี่ยนแปลง องค์กรจะทำการตรวจสอบและประเมินความก้าวหน้าเพื่อปรับเปลี่ยนแผนการ Release นี้ได้ในทุกๆรอบของการ Sprint

2. Sprint เป็นรอบการพัฒนาที่กำหนดเวลาชัดเจน ScrumMaster จะต้องดูแลไม่ให้เกิดการเปลี่ยนแปลงที่ส่งผลกระทบต่อเป้าหมายของ Sprint ตลอดระยะเวลาการ Sprint สมาชิกในทีมและเป้าหมายเชิงคุณภาพจะไม่เปลี่ยนแปลงตลอด Sprint และในแต่ละ Sprint จะประกอบไปด้วย

    1. การประชุมวางแผน Sprint
    2. งานที่พัฒนาใน Sprint
    3. การตรวจทาน Sprint
    4. การทบทวน Sprint ย้อนหลัง

    ใน Scrum ช่วงเวลา Sprint ถูกจำกัดให้อยู่ที่ประมาณ 2 สัปดาห์ - 1 เดือน ซึ่งเป็นเทคนิคเพื่อลดความเสี่ยง นั่นคือทำให้สามารถประเมินและควบคุมสถานการณ์ได้อย่างน้อยทุกๆ 1 เดือน หรือน้อยกว่า

3. Sprint Planning Meeting

    เป็นการประชุมเพื่อกำหนดรอบ Sprint โดยจะใช้เวลาประมาณ 8 ชั่วโมงสำหรับ Sprint ที่ยาว 1 เดือน และถ้าเป็น Sprint ที่สั้นกว่านั้นจะสามารถลดเวลาลงเป็นสัดส่วนได้ เช่น 4 ชั่วโมงสำหรับ Sprint ยาว 2 สัปดาห์ เป็นต้น การประชุมวางแผน Sprint จะแบ่งออกเป็น 2 ช่วง โดยช่วงแรกจะเป็นการตอบคำถามว่า จะทำ "อะไร" ใน Sprint นี้ โดย Product Owner จะนำ Product Backlog ที่เรียงตามลำดับความสำคัญมาเสนอต่อทีมพัฒนา จากนั้นจะเป็นหน้าที่ของทีมพัฒนาที่จะร่วมกันคิดว่าความสามารถอะไรบ้างของระบบจะถูกพัฒนาขึ้น รวมทั้งปริมาณงานที่จะทำใน Sprint นี้ ปัจจัยสำหรับการประชุมวางแผน Sprint มีดังต่อไปนี้

    1. Product Backlog
    2. ส่วนเพิ่มล่าสุดของระบบ
    3. ความสามารถของทีม
    4. ประสิทธิภาพของทีมย้อนหลัง

    หลักจากเลือก Product Backlog ที่จะทำใน Sprint นี้แล้ว ก็จะเป็นการกำหนดเป้าหมายให้ทีมมีความชัดเจนว่าทำ Sprint นี้เพื่ออะไร ถ้าเป้าหมายยากเกินไปสำหรับ Sprint นี้ก็จะมีการปรึกษากับ Product Owner เพื่อพัฒนาเฉพาะบางส่วนของความสามารถทั้งหมด

    สำหรับช่วงที่ 2 ของการประชุมจะเป็นการตอบคำถามว่า จะทำ "อย่างไร" โดยการนำ Product Backlog มาออกแบบเป็นงานที่จะกลายเป็นส่วนเพิ่มที่ทำงานได้ระหว่างการออกแบบก็จะเป็นการระบุว่าจาก Product Backlog 1 ชิ้น จะมีงานอะไรบ้างโดยงานควรจะแยกย่อยลงไปในระดับที่ 1 งานสามารถทำเสร็จได้ภายใน 1 วัน และงานทั้งหมดจะเรียกรวมกันว่า Sprint Backlog ทีมก็จะจัดการปรับตัวเพื่อรับมือกับงานใน Spint Backlog ที่สร้างขึ้น การปรับตัวอาจเกิดขึ้นทั้งระหว่างการประชุมและระหว่างการ Sprint และ Product Owner จะร่วมประชุมในช่วงที่ 2 นี้เพื่อช่วยอธิบาย Product Backlog ในกรณีที่ทีมยังไม่เข้าใจ และทีมจะปรึกษาเรื่องปริมาณงานในช่วงนี้กับ Product Owner เช่นกัน

4. Daily Scrum

    เป็นการประชุมทุกวันโดยใช้เวลาประมาณ 15 นาที เพื่อตอบคำถามต่อไปนี้

1. อะไรที่เสร็จไปแล้วบ้างหลังจากการประชุมครั้งก่อน
2. อะไรที่จะทำก่อนการประชุมครั้งหน้า
3. และอะไรที่เป็นอุปสรรค

ประโยชน์ของ Daily Scrum คือ สามารถพัฒนาการสื่อสารระหว่างบุคคลในทีม และระบุปัญหาและอุปสรรคที่เกิดขึ้น รวมทั้งเป็นสิ่งที่เกี่ยวข้องต่อการดำเนินงานไปสู่เป้าหมายของ Sprint

5. Sprint Review

เมื่อจบการ Sprint จะมีการประชุมเพื่อทบทวน Sprint นั้นๆ ระหว่างทีมและผู้เกี่ยวข้อง ซึ่งเป็นการประชุมที่ใช้เวลา 4 ชั่วโมงสำหรับ Sprint ความยาว 1 เดือน นั่นคือ 2 ชั่วโมงสำหรับ Sprint ความยาว 2 สัปดาห์ เป็นต้น ในระหว่างการประชุมนี้จะเป็นการทบทวนงานที่เพิ่งทำเสร็จไป และอะไรควรจะเป็นสิ่งที่จะเสร็จต่อไป โดยการประชุมควรจะมีองค์ประกอบเหล่านี้

1. Product Owner ระบุว่าอะไรที่เสร็จแล้วและอะไรยังไม่เสร็จ
2. ทีมอภิปรายว่าอะไรที่ทำได้เป็นอย่างมีประสิทธิภาพและอะไรที่เป็นปัญหา
3. ทีมทำการสาธิตส่วนที่เพิ่มใหม่ของระบบให้ Product Owner ดูและตอบคำถาม
4. Product Owner อภิปราย Product Backlog ที่เหลืออยู่และวันที่ควรจะเสร็จด้วยการประเมินจากความเร็วในการทำงาน
5. ทั้งสองฝั่งทบทวนการประชุมเพื่อให้ทราบถึงสิ่งที่ต้องทำต่อ ข้อมูลที่ได้จากการทบทวน Sprint จะใช้เป็นปัจจัยสำหรับการประชุมวางแผน Sprint ในครั้งต่อไป

6. Sprint Retrospective

การทบทวน Sprint ย้อนหลังเป็นการประชุมหลังจากการตรวจทาน Sprint และก่อนการประชุมวางแผน Sprint ถัดไป โดยการทบทวน Sprint ย้อนหลังจะเป็นการประชุมขนาด 3 ชั่วโมงสำหรับ Sprint ความยาว 1 เดือน นั่นคือ 1.5 ชั่วโมงสำหรับ Sprint ความยาว 2 สัปดาห์ ซึ่งในการประชุมนี้จะเป็นการอภิปรายกันระหว่าง ScrumMaster กับทีมเพื่อปรับปรุงกระบวนการพัฒนาให้มีประสิทธิภาพมากขึ้น โดยตรวจสอบว่า บุคคล กระบวนการและเครื่องมือมีปัญหาอะไรบ้าง รวมทั้งเป็นการรวมทีมใหม่ และการเตรียมการประชุมครั้งต่อๆไป เป็นต้น

ชิ้นงานใน Scrum (Scrum Artifacts) ประกอบไปด้วย

1. Product Backlog

เป็นความต้องการเชิงซอฟต์แวร์ที่ทีมพัฒนานำมาแปลงเป็นส่วนเพิ่มของซอฟต์แวร์ที่สามารถทำงานได้ โดย Product Owner จะเป็นผู้รับผิดชอบต่อรายการใน Product Backlog ทั้งในเรื่องของตัวเนื้อหา และการจัดลำดับความสำคัญ ตัว Product Backlog นั้นไม่จำเป็นต้องเสร็จสิ้นสมบูรณ์ เพราะซอฟต์แวร์สามารถพัฒนาได้เรื่อยๆ ส่วนที่หยิบมาพัฒนาจะเป็นความต้องการที่ระบุได้ในช่วงแรกและสามารถเข้าใจได้ดีที่สุดในขณะนั้น ตัว Product Backlog จะค่อยๆเปลี่ยนแปลงตามซอฟต์แวร์และสภาพของการพัฒนา โดยรายการของ Product Backlog จะประกอบไปด้วย ทุกอย่างที่จำเป็นต่อการพัฒนาซอฟต์แวร์ แต่ละรายการของ Product Backlog อาจจะเป็นคุณสมบัติ, ฟังก์ชั่นการทำงาน, เทคโนโลยี, การเพิ่มความสามารถ, การแก้ไขข้อผิดพลาด ที่ทำให้เกิดการเปลี่ยนแปลงต่อตัวซอฟต์แวร์ และ อาจจะมี Attribute หรือคุณสมบัติกำกับเพื่อบอกรายละเอียด เช่น

- คำอธิบาย (Description)
- ลำดับความสำคัญ (Priority)
- การประมาณค่า (Estimation)

เป็นต้น

ในกระบวนการพัฒนาซอฟต์แวร์ที่ใช้ Scrum โดยปกตินั้นจะใช้ User Story ในการอธิบาย Product Backlog แต่ละรายการ อย่างไรก็ตาม Use Case ก็สามารถใช้ได้เช่นเดียวกัน ซึ่ง Use Case จะเหมาะสำหรับระบบที่มีขั้นตอนชัดเจนมากกว่า User Story

2. Release Burndown Chart

เป็นกราฟสำหรับบันทึกงานที่เหลือทั้งหมดของทั้งโครงการ แกนตั้งจะเป็นการประมาณตามหน่วยที่ใช้ใน Scrum Team เช่น Function Point หรือ Use Case Point เป็นต้น และโดยทั่วไปแกนเวลาจะเป็นจำนวนครั้งของ Sprint โดยการประมาณแต้มแต่ละรายการของ Product Backlog จะคำนวณในช่วงการประชุมวางแผน Release และสามารถทบทวนและแก้ไขได้ตลอดเวลา โดยทีมพัฒนาจะเป็นผู้รับผิดชอบการประมาณ ซึ่ง Product Owner สามารถช่วยทำความเข้าใจและเจรจากับทีมพัฒนาในการประมาณแต้มของราย Product Backlog ได้ แต่การตัดสินใจจะเป็นที่สิ้นสุดโดยทีมพัฒนาเท่านั้น

3. Sprint Backlog

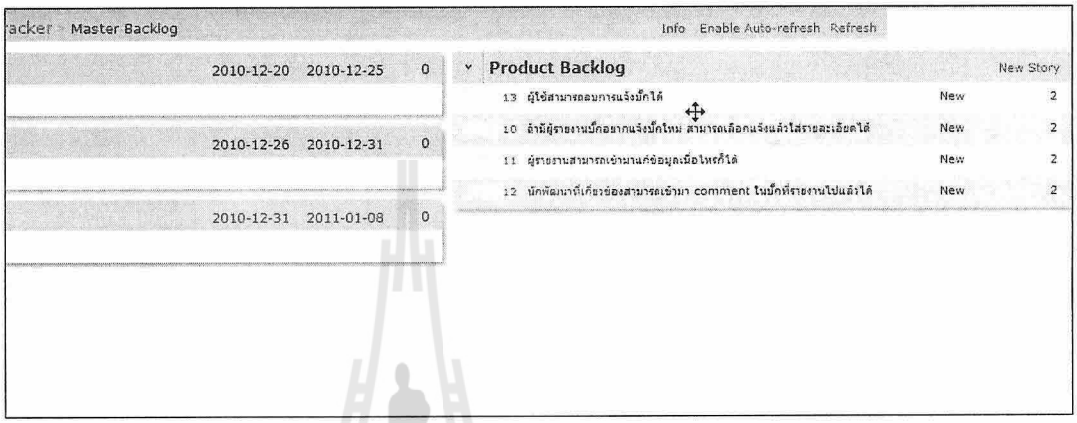
เป็นรายการของงาน (Task) ที่ทีมพัฒนาทำการเปลี่ยนรายการของ Product Backlog ให้กลายเป็นส่วนเพิ่มของซอฟต์แวร์ที่ "เสร็จ" และนำส่งได้ โดยทั่วไปรายการ Sprint Backlog จะถูกสร้างในขณะการประชุมวางแผน Sprint ซึ่งรายการเหล่านี้จะเป็นงานที่จำเป็นต่อการดำเนินไปยังเป้าหมายของ Sprint นั้นๆ อย่างที่ได้กล่าวไว้แล้ว แต่ละรายการของ Sprint Backlog จะต้องถูกแยกเป็นงานย่อยที่สามารถทำเสร็จได้ภายใน 1 วัน

4. Sprint Backlog Burndown Chart

เป็นกราฟที่แสดงจำนวนของงานซึ่งเป็น Sprint Backlog ตลอดระยะเวลาในระหว่างการ Sprint แกนตั้งเป็นจำนวนงานที่ค้างอยู่จากแต่ละวันรวมกัน และแกนนอนจะเป็นหน่วยจำนวนวัน ทีมพัฒนามีหน้าที่ติดตามความก้าวหน้าของงานใน Sprint ด้วยกราฟนี้

ลำดับขั้นตอนในการพัฒนาซอฟต์แวร์โดยใช้ Scrum

1. กำหนด Product Owner
2. กำหนดทีมพัฒนา โดยควรจะมีจำนวน 5-9 คน
3. กำหนด ScrumMaster
4. Product Owner ทำการเตรียม Backlog



5. ประชุมวางแผน Release โดยมีลำดับขั้นตอนดังนี้

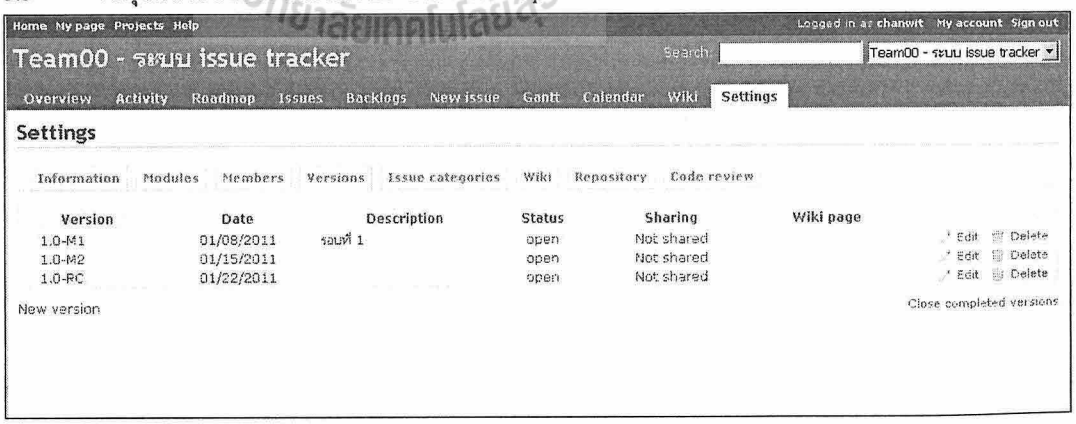
5.1 กำหนดเป้าหมายของการ Release เช่น “พัฒนาระบบพื้นฐานของตัว Issue Tracker ให้สำเร็จ”

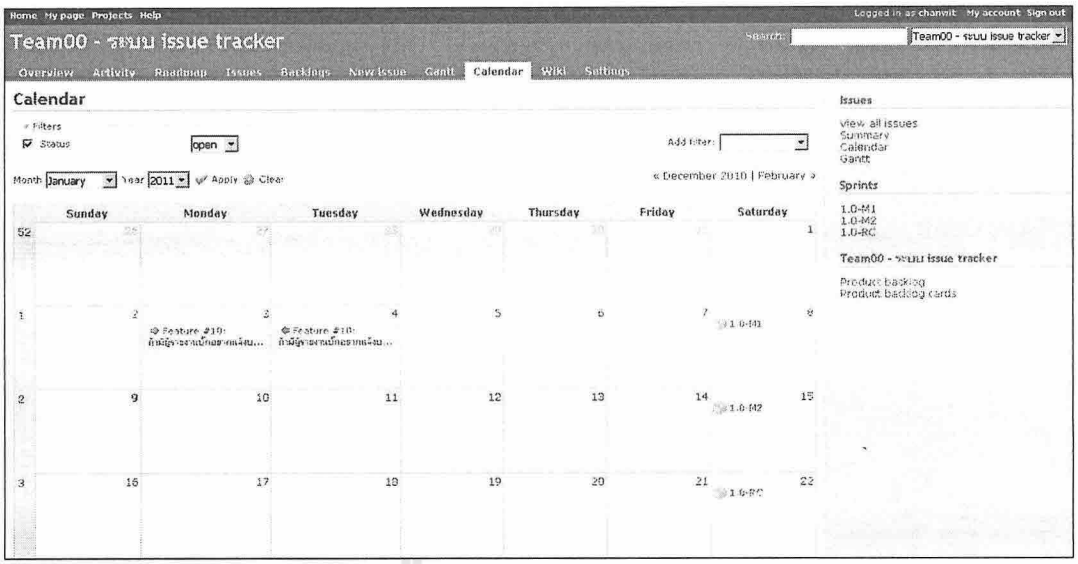
5.2 พิจารณา Product Backlog

5.3 ระบุความเสี่ยงที่สำคัญ เช่น ทักษะของสมาชิกในทีมไม่สูง และไม่มีประสบการณ์พัฒนาซอฟต์แวร์

5.4 ระบุคุณสมบัติและความสามารถของซอฟต์แวร์ที่จะส่งมอบใน Release นี้

5.5 ระบุวันส่งมอบเบื้องต้น และประเมินจำนวน Sprint ที่จะเกิดขึ้นทั้งหมดก่อนการ Release

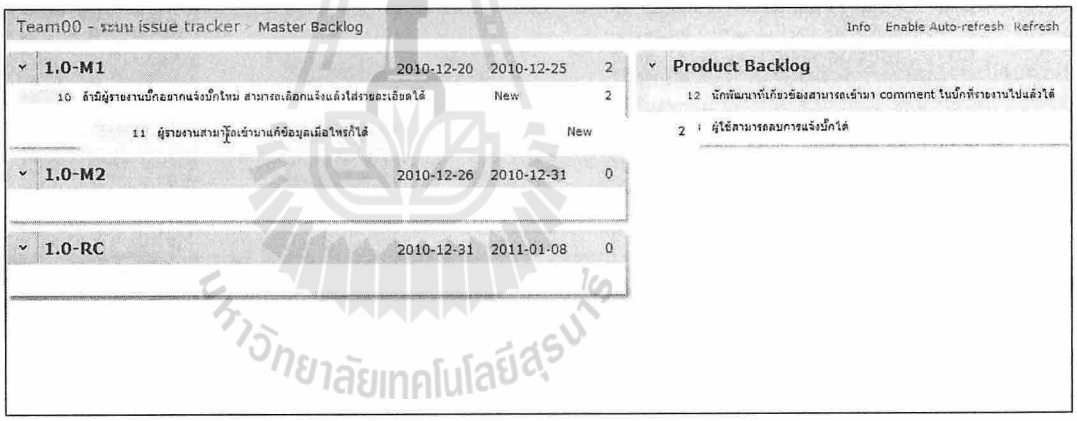




5.6 ประมาณต้นทุน

6. ประชุมวางแผน Sprint โดยใช้เวลาไม่เกิน 2 ชั่วโมง

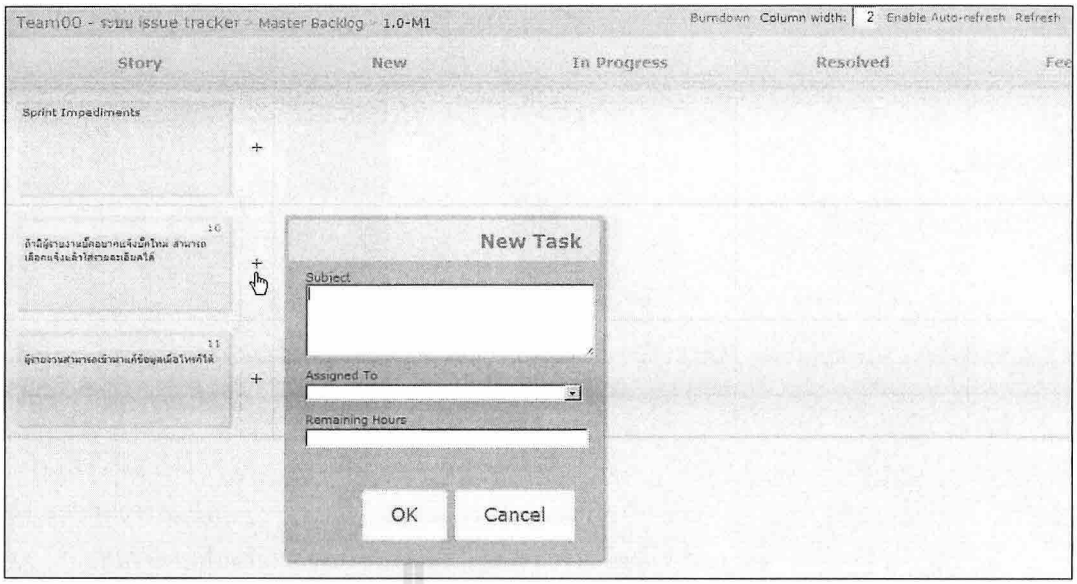
6.1 สมาชิกในทีมเลือก Product Backlog มาเป็น Sprint Backlog และกำหนดเป้าหมายให้ Sprint ปัจจุบัน

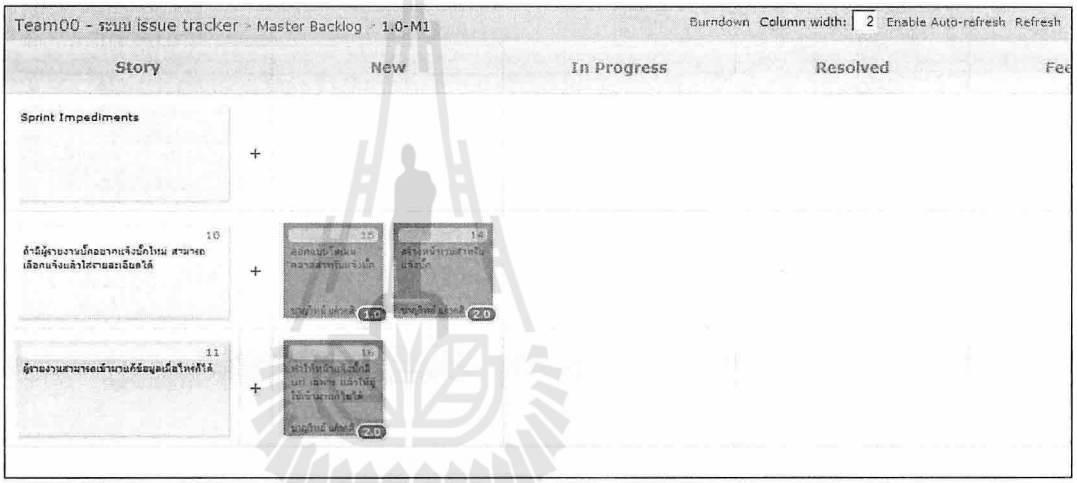


6.2 สร้าง System Use Case ขนาดย่อม เพื่อประเมินความยากง่ายของแต่ละ Backlog Item (User Story)

6.3 สร้าง Activity Diagram ให้แต่ละ System Use Case

6.4 แยกงานออกมาจาก Activity Diagram ที่ออกแบบให้เป็นงาน (Task) ย่อยที่สามารถทำเสร็จได้ใน 1 วัน

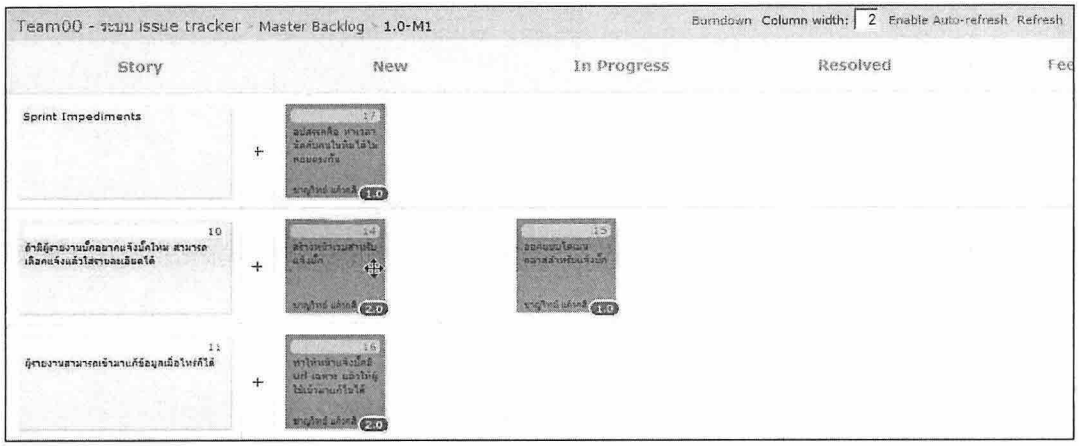




6.5 ประมาณโครงการโดยใช้หลัก Use Case Point ให้แต่ละ Use Case

6.6 ทีมพัฒนาปรึกษากับ Product Owner เพื่ออธิบาย User Story เพิ่มเติมหากมีข้อสงสัย

ถ้าเป็น Sprint รอบอื่นๆ ที่ไม่ใช่รอบแรก จะมีการประเมินความสามารถของทีมและอุปสรรคประกอบกันไปด้วย



7. ทำกระบวนการ Sprint

7.1 สมาชิกในทีมสร้าง Communication Diagram ต่อจาก Activity Diagram ที่ได้จากขั้นตอนที่แล้ว

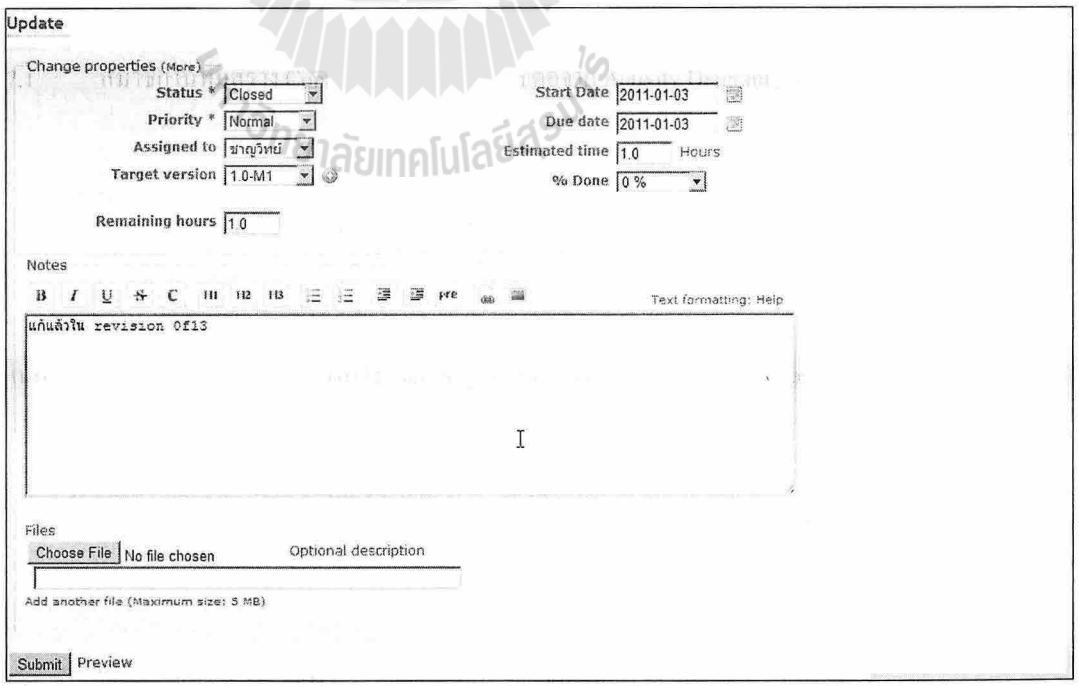
7.2 สร้าง Class Diagram

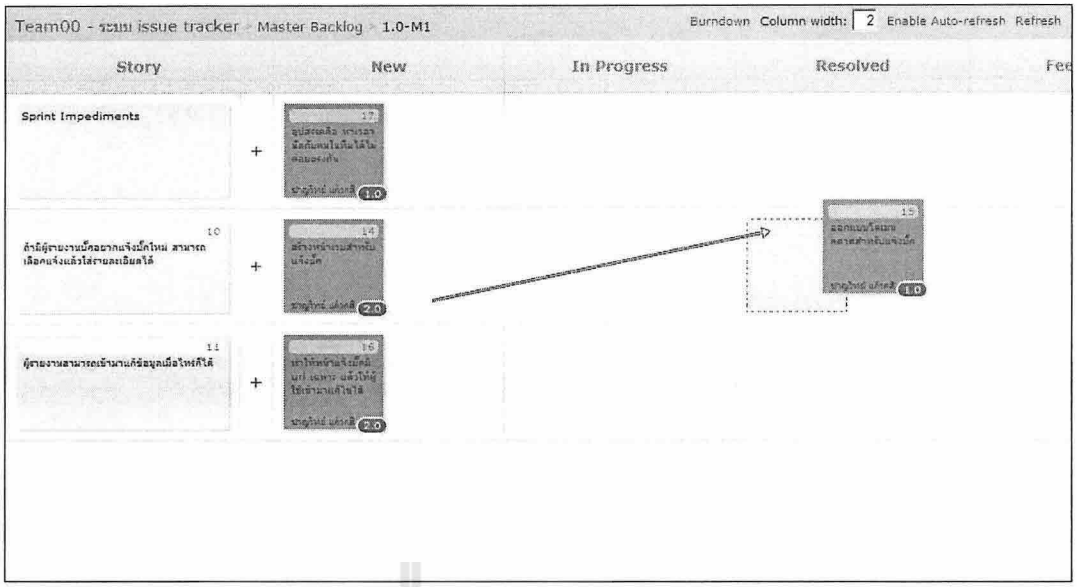
7.3 พัฒนาโปรแกรมโดยสร้างคลาสหลักก่อน (Domain-Driven) ตามด้วยคลาสสนับสนุน (Controller/GUI)

7.4 สร้าง Unit Test เพื่อทดสอบระดับหน่วย

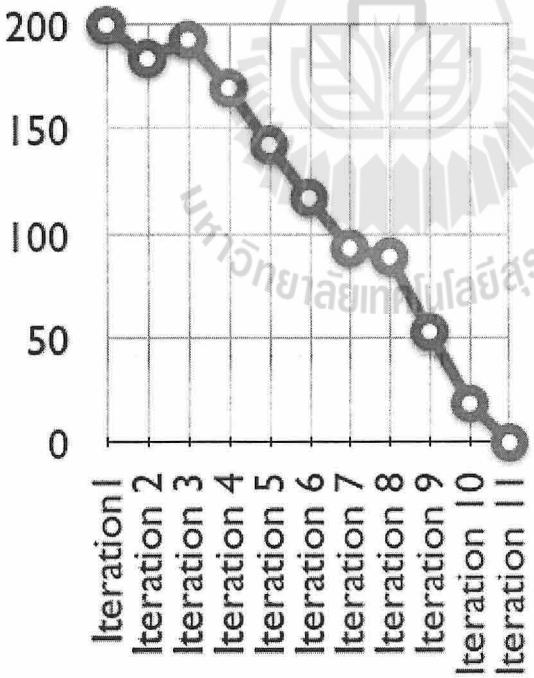
ในกระบวนการ 7.3 และ 7.4 สามารถใช้ Pair Programming ของ Extreme Programming มาช่วยได้ วิธีการคือให้มีนักพัฒนา 2 คน ทำงานพร้อมกันโดยคนหนึ่งเป็นผู้เขียนโปรแกรม และอีกคนหนึ่งเป็นผู้ให้คำแนะนำเท่านั้น

7.5 ปรับสถานะงานแต่ละงานใน Sprint





8. ทุกๆ วันของการทำงานมีการประชุม 15 นาที เพื่อสรุปความก้าวหน้า (Daily Scrum)
9. Sprint Review ใช้เวลาประมาณ 2 ชั่วโมงสำหรับ Sprint ขนาด 2 สัปดาห์
   - 9.1 ตรวจสอบงานว่ามีงานใดแล้วเสร็จบ้างและงานใดยังไม่แล้วเสร็จ
   - 9.2 ทีมอภิปรายประสิทธิภาพ
   - 9.3 ทำการ Demo ส่วนที่พัฒนาเพิ่ม
   - 9.4 สร้าง Burndown Chart ทั้งสองแบบ



9.5 อภิปรายงานที่เหลือและวางแผนสำหรับ Sprint ครั้งต่อไป

10. Sprint Retrospective ใช้เวลา 1.5 ชั่วโมง

10.1 ประชุมเพื่อประเมินประสิทธิภาพกันเองภายในกลุ่ม

10.2 ตรวจสอบกระบวนการทำงานและเครื่องมือ

10.3 เตรียมการประชุมครั้งต่อไป

การประมาณโครงการด้วย Use Case Point (UCP) สำหรับ Scrum

เมื่อ Product Owner ทำการบันทึก User Story ในรูปแบบของ Product Backlog Item และทีมเตรียม Product Backlog Item เป็น Sprint Backlog แล้ว จะมีการแปลง User Story แต่ละเรื่องให้เป็น Use Case ก่อน จากนั้นจึงสร้าง Activity Diagram ขึ้นมา เพื่อให้ได้ขั้นตอนการทำงาน แล้วจึงประมาณค่า UCP จากขั้นตอนเหล่านั้น

User Story

การเขียน User Story ทำโดย Product Owner ซึ่งเป็นการอธิบายสถานการณ์หนึ่งๆ เพื่อสร้างเป็นคุณสมบัติของซอฟต์แวร์ ดังตัวอย่างต่อไปนี้

Story #1 ระบบ Issue Tracker

ในบทบาทผู้ใช้ เราสามารถแจ้งข้อผิดพลาดของโปรแกรมเข้าสู่ระบบได้
เพื่อให้มีการติดตามงานและแก้ไขข้อผิดพลาดต่อไป

จากตัวอย่าง รูปแบบการเขียน User Story มีดังนี้

1. ในบทบาท <ชื่อบทบาท>.....
2. เรา <สามารถ, ต้องการ>.....
3. เพื่อให้.....

ตัวอย่าง

ข้อมูลจาก Product Backlog มี User Story จำนวน 10 Story และพบว่ามี Actor ที่เกี่ยวข้องจำนวน 2 Actor ซึ่งเป็นผู้ใช้ (น้ำหนัก =3) และเมื่อนำแต่ละ Story มาแปลงเป็น Use Case และสร้าง Activity Diagram ให้แต่ละ Use Case แล้วนั้น พบว่า

มี Use Case ขนาดน้อยกว่า 3 Transaction	7 Use Case	(น้ำหนัก 5)
มี Use Case ขนาด 4-7 Transaction	3 Use Case	(น้ำหนัก 10)

$$UUCP = \sum(W_a \bullet A_i) + \sum(W_u \bullet UC_i)$$

= (3X2) + [(5X7)+(10X3)]

= 71

ให้ค่าตัวคูณความซับซ้อนเชิงเทคนิคและตัวคูณความซับซ้อนแวดล้อม = 1

UCP = UUCP X TCF X UCF

= 71 X 1 X 1

= 71 points

กำหนดค่า Productivity Factor (PF) ของสมาชิกทีมพัฒนาที่ใช้เฟรมเวิร์คกลุ่ม Rapid MVC ไว้ที่ 2.5

ดังนั้น แรงงานที่ใช้ในการพัฒนาโครงการนี้

= UCP X PF

= 71 X 2.5

= 177.5 คน-ชั่วโมง

ให้ 1 สัปดาห์ ใช้เวลาพัฒนาซอฟต์แวร์ 30 ชั่วโมง

จะใช้เวลา 177.5/30 = 5.91 ≈ 6 คน-สัปดาห์

เพื่อให้การพัฒนาซอฟต์แวร์เป็นไปอย่างมีประสิทธิภาพ จึงควรแบ่งการทำงานออกเป็น 3 Sprint

โดยให้แต่ละ Sprint ยาว 2 สัปดาห์ และใช้นักพัฒนาจำนวน 1 คน

ถ้านักพัฒนามีเงินเดือน 14,000 บาท ต้นทุนของระบบจะเท่ากับ 6/(4 สัปดาห์ต่อเดือน)X 14,000 = 21,000 เดือน

การประมาณ User Story ด้วย Use Case Point (UCP) สำหรับ Scrum

การประมาณ User Story ทำได้โดยเมื่อนำ User Story ไปเขียนเป็น Use Case แล้ว จากนั้นทำการสร้าง Activity Diagram แบบมี Swim Lane ให้ Use Case ดังกล่าว เมื่อได้ Activity Diagram แล้วจึงใช้ขั้นตอนที่เขียนขึ้นใน Activity Diagram ในการประมาณค่า UCP โดยนับจำนวนขั้นตอนที่ต้องประมวลผลโดยใช้ Transaction

ตัวอย่าง

จากข้อมูลที่ได้จาก Product Backlog ซึ่งประกอบไปด้วย Backlog Item จำนวน 10 Story และมี Actor ที่เกี่ยวข้อง 2 Actor ดังนั้น Story/Actor คือ 5 ซึ่งจะเป็นค่าที่ใช้สำหรับหาค่า Weight Actor จากสูตรการคำนวณ Unadjusted UCP (UUCP)

$$UUCP = \sum(W_a \bullet A_i) + \sum(W_u \bullet UC_i)$$

จากตัวอย่างนี้ $W_a = 3$ เนื่องจากเป็นงานที่ผู้ใช้ติดต่อกับ GUI

A_i = 1 เพราะมี 1 Actor

ค่าเฉลี่ย Story/Actor = 5

กำหนดให้ Use Case "แจ้งข้อผิดพลาด" มีการทำงานที่ประกอบไปด้วย 2 Transaction

ตัวคูณน้ำหนัก Use Case จำนวน Transaction น้อยกว่า 3 จะมีค่า W_u = 5

ตัวคูณน้ำหนัก Use Case จำนวน Transaction 4-7 จะมีค่า W_u = 10

ตัวคูณน้ำหนัก Use Case จำนวน Transaction มากกว่า 7 จะมีค่า W_u = 15

$$UUCP = \sum(W_a \bullet A_i) + \sum(W_u \bullet UC_i)$$

= [(3X1)/5] + (5X1)

= 0.6 + 5

= 5.6

กำหนดค่า PF ของสมาชิกทีมพัฒนาที่ใช้เฟรมเวิร์กกลุ่ม Rapid MVC ไว้ที่ 2.5

ดังนั้น แรงงานที่ใช้ในการพัฒนา User Story นี้

= UCP X PF

= 5.6 X 2.5

= 14 คน-ชั่วโมง

Unit Testing

เป็นการทดสอบเชิงหน่วยในแต่ละส่วนของโปรแกรมโดยมีลักษณะสำคัญคือ ต้องสามารถแยกออกมาทดสอบเดี่ยวๆได้ สำหรับในบทนี้จะกล่าวถึงโดยสรุปเพื่อให้เห็นภาพการทำงานเบื้องต้น การทดสอบเชิงหน่วยนี้พัฒนาโดยผู้ที่เขียนโปรแกรมส่วนนั้นๆ ในแต่ละภาษาจะมีการสนับสนุน Unit Testing ในลักษณะที่คล้ายกันตามแนวคิดของ Kent Beck โดยภาษาโปรแกรมจะมีเฟรมเวิร์คสำหรับสนับสนุน Unit Testing ดังต่อไปนี้

Java	JUnit
C#	NUnit (ใช้ร่วมกับภาษาอื่นบน .NET ได้)
PHP	PHPUnit
Ruby	Test::Unit (อยู่ใน Standard Lib ของ Ruby)
JavaScript	JSUnit
Groovy	JUnit (อยู่ใน Standard Lib ของ Groovy)

ตัวอย่างการเขียน Unit Test

หน่วยย่อยสำหรับทดสอบโปรแกรม จะเรียกว่า Test Case โดยสามารถเขียนได้โดยใช้เฟรมเวิร์คสำหรับ Unit Testing ได้ดังนี้

Java:

```
package th.ac.sut.se.test;

import junit.framework.TestCase;

public class ShoppingCartTest extends TestCase {

    private ShoppingCart cart;
    private Product book1;

    // ตั้งค่าสิ่งที่จะทดสอบ
    // เมธอด setUp จะถูกเรียกใช้ก่อนการทดสอบทุก ๆ ครั้ง
    protected void setUp() {
        cart = new ShoppingCart();
        book1 = new Product("Space Valkyrie I", 295.0);
        cart.addItem(book1);
    }

    // ทดสอบว่า รถเข็น cart ว่างหรือไม่ เมื่อสั่ง cart.empty()
    public void testEmpty() {
        cart.empty();
        assertEquals(0, cart.getItemCount());
    }
}
```

PHP:

```
<?php
class ShoppingCartTest extends PHPUnit_Framework_TestCase {
    protected $cart;
    protected $book1;

    // ตั้งค่าสิ่งที่จะทดสอบ
    // เมธอด setUp จะถูกเรียกใช้ก่อนการทดสอบทุก ๆ ครั้ง
    protected function setUp() {
        $this->cart = new ShoppingCart();
        $this->book1 = new Product('Space Valkyrie I', 295);
        $this->cart->addItem($this->book1);
    }

    // ทดสอบว่า รถเข็น cart ว่างหรือไม่ เมื่อสั่ง cart.empty()
    public function testEmpty() {
        $this->cart->empty();
        $this->assertEquals(0, $this->cart->getItemCount());
    }
}
?>
```

Ruby:

```ruby
require 'test/unit'

class ShoppingCartTest < Test::Unit::TestCase

  # ตั้งค่าสิ่งที่จะทดสอบ
  # เมธอด setUp จะถูกเรียกใช้ก่อนการทดสอบทุก ๆ ครั้ง
  def setup
    @cart = ShoppingCart.new
    @book1 = Product.new('Space Valkyrie I', 295)
    @cart.add_item(@book1)
  end

  # ทดสอบว่า รถเข็น cart ว่างหรือไม่ เมื่อสั่ง cart.empty()
  def test_empty
    @cart.empty
    assert_equal(0, @cart.get_item_count)
  end

end
```

C#:

```csharp
using System;
using NUnit.Framework;
using SUT.SE;

namespace SUT.SE.Test {

   [TestFixture]
   public class ShoppingCartTest {

      private ShoppingCart cart;
      private Product book1;

      // เมธอด setUp จะถูกเรียกใช้ก่อนการทดสอบทุก ๆ ครั้ง
      [SetUp]
      public void SetUp() {
        cart = new ShoppingCart();
        book1 = new Product("Space Valkyrie I", 295.0);
      }

      // ทดสอบว่า รถเข็น cart ว่างหรือไม่ เมื่อสั่ง cart.Empty()
      [Test]
      public void TestEmpty( ){
         cart.Empty();
         Assert.AreEqual(0, cart.GetItemCount());
      }
   }
}
```

จะเห็นว่าการเขียน Unit Test มีองค์ประกอบคือ การเตรียมวัตถุเพื่อทดสอบ จากนั้นทำการเรียกใช้เมธอดที่ต้องการทดสอบ (ในที่นี้คือ empty) และตรวจสอบผลลัพธ์ว่า เป็นไปตามที่คาดไว้หรือไม่ โดยเราจะคาดไว้ว่า เมื่อเรียกเมธอด empty แล้วจำนวน item ใน cart ควรจะเป็น 0 เพื่อสะท้อนความหมายว่า ไม่มีสินค้าอยู่ในรถเข็น เป็นต้น

บทที่ 5
เครื่องมือทางวิศวกรรมซอฟต์แวร์
Software Engineering Tools

ในวิศวกรรมซอฟต์แวร์นั้นมีการใช้เครื่องมือจำนวนหนึ่งเพื่อช่วยให้กระบวนการพัฒนาเป็นไปได้อย่างราบรื่นขึ้น โดยเนื้อหาในส่วนนี้จะกล่าวถึงเครื่องมือที่ช่วยจัดการการพัฒนาซอฟต์แวร์แบบ Scrum, เครื่องมือสนับสนุนการจัดการความเปลี่ยนแปลง และเครื่องมือที่ใช้ทดสอบซอฟต์แวร์

เครื่องมือจัดการการพัฒนาซอฟต์แวร์แบบ Scrum

เป็นเครื่องมือทางวิศวกรรมซอฟต์แวร์ที่ใช้จัดการโครงการที่มีกระบวนการพัฒนาแบบ Scrum โดยเป็นส่วนหนึ่งของเครื่องมือทางวิศวกรรมซอฟต์แวร์ที่ใช้จัดการโครงการทั่วๆไปที่เรียกว่า Project Management Tools ซึ่งเป็นเครื่องมือที่ไม่มีความสามารถเฉพาะสำหรับกระบวนการพัฒนาซอฟต์แวร์ประเภทใดประเภทหนึ่ง เครื่องมือทางวิศวกรรมซอฟต์แวร์ที่ใช้จัดการโครงการทั่วๆไปมักจะมีคุณลักษณะดังต่อไปนี้

- มีระบบติดตามปัญหา (Issue Tracker)
- มี Gantt Chart และปฏิทินโครงการเพื่อสนับสนุนการวางแผน เป็นต้น

สำหรับกระบวนการพัฒนาซอฟต์แวร์แบบ Scrum จะมีเครื่องมืออีกกลุ่มหนึ่งที่เสริมขึ้นมาเพื่อให้สามารถจัดการกับโครงการและชิ้นงานในกระบวนการพัฒนาประเภทนี้ได้ดีขึ้น

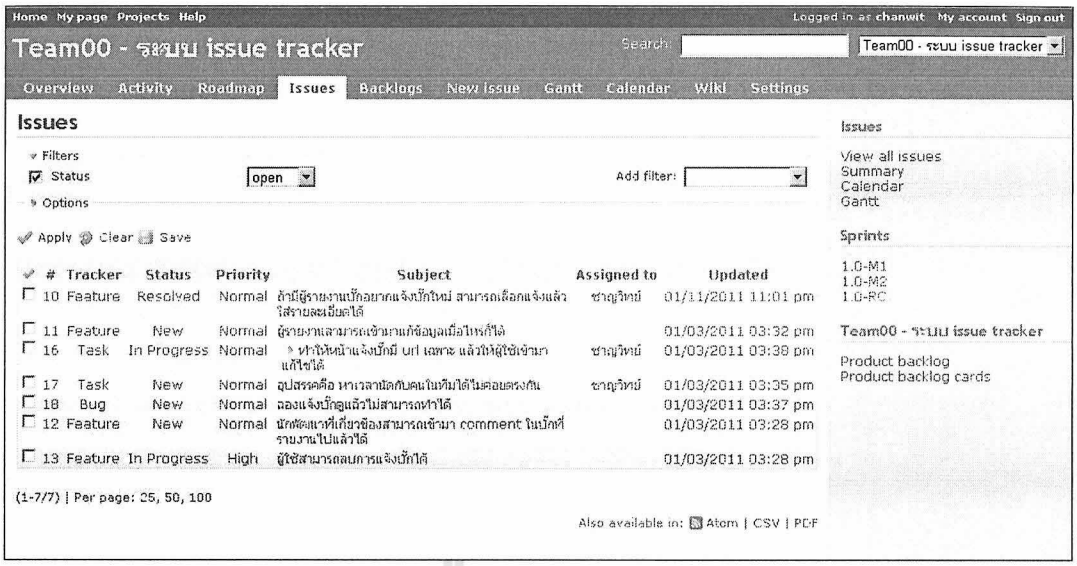
Redmine Backlogs

เป็นปลั๊กอินที่เพิ่มเข้าไปในโปรแกรม Redmine เพื่อให้ Redmine กลายเป็นเครื่องมือบริหารโครงการ Scrum ที่มีความสามารถต่อไปนี้

- สามารถสร้าง Story เป็น Issue ได้
- สามารถระบุ Point ให้กับ Story ได้
- มี Taskboard สำหรับแยก Story ออกเป็นงาน (Task)
- มีความสามารถช่วยในการจัด Product Backlog และ Sprint Backlog

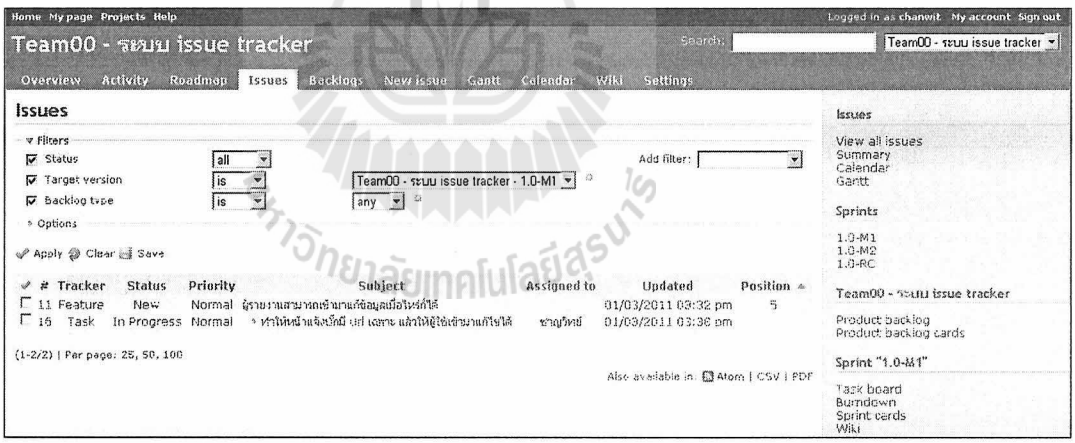
ใน Redmine Backlogs จะมีการใช้งานหลักดังต่อไปนี้

1. Issues แสดงรายการปัญหาทั้งหมดในโครงการ
   ในการตั้งค่าปกติของ Redmine Backlogs นั้นจะ แยกกลุ่มของ Issue ออกเป็น
   1. Story คือ กลุ่มของ Issue ที่ทำหน้าที่เป็น User Story หรือแต่ละรายการใน Product Backlog และ Sprint Backlog
   2. Task คือ กลุ่มของ Issue ที่ทำหน้าที่เป็นงานย่อยของ Story



เมื่อเลือก Tab Issues จะพบว่ารายการ Issue ที่แสดงนั้นจะระบุประเภท Tracker ของ Issue ต่างๆกัน โดยในตัวอย่างมีการตั้งค่า Issue ประเภท Feature และ Bug ให้สามารถเป็น “User Story” ใน Scrum ได้และ Issue ประเภท Task นั้นให้เป็น “Task” ใน Scrum เช่น Issue #11 เป็น User Story ประเภท Feature และมี Task ย่อย คือ Issue #16 เป็นต้น

ด้านขวาจะเป็นรายการของ Sprint ที่แสดงอยู่ ซึ่งตามตัวอย่างมี Sprint 1.0-M1, 1.0-M2 และ 1.0–RC ตามลำดับ เมื่อเลือก 1.0-M1 Redmine Backlogs จะแสดงเฉพาะ Story และ Task สำหรับ Sprint นั้นๆ

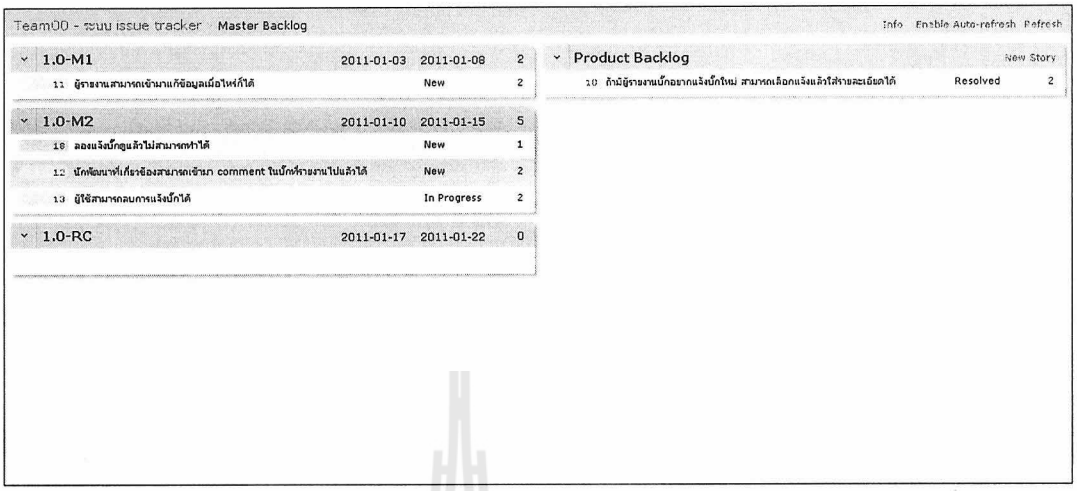


2. Backlogs แสดงรายการ Issue ในรูปแบบของ Backlog โดยจะแสดงเฉพาะ Issue ที่สามารถเป็น User Story ใน Scrum ได้เท่านั้น ในหน้า Backlogs นี้ จะเรียกว่าหน้า Master Backlog และหน้าจอจะแบ่งออกเป็น 2 ส่วน คือ

ด้านขวาจะเป็น Product Backlog

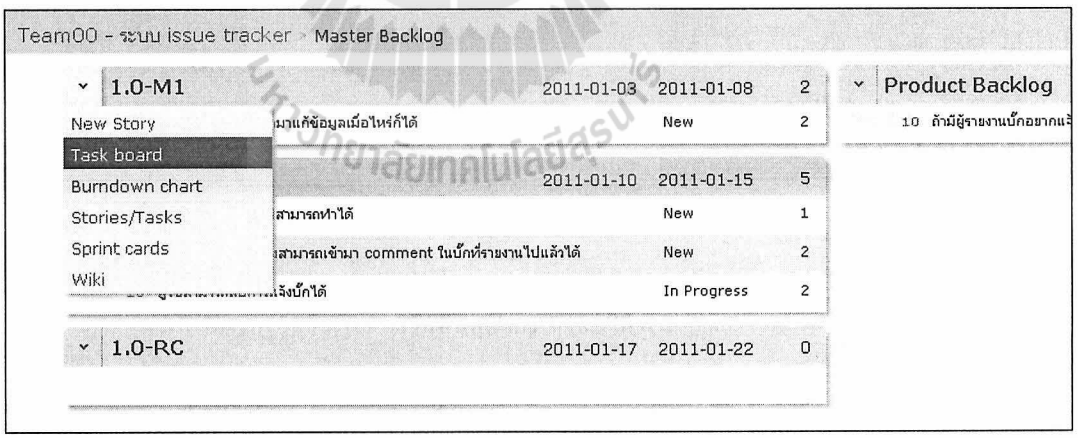
ด้านซ้ายจะเป็น Sprint Backlog แยกตาม Sprint

ใน Product Backlog จะสามารถเพิ่ม Story ใหม่เข้าไปใน Backlog โดยการเลือก “New Story” จากนั้นเลือกชนิดของ Issue ว่าเป็น Feature หรือ Bug แต่ละ Story โดยสามารถจัดเรียงก่อนหลังเพื่อลำดับความสำคัญได้

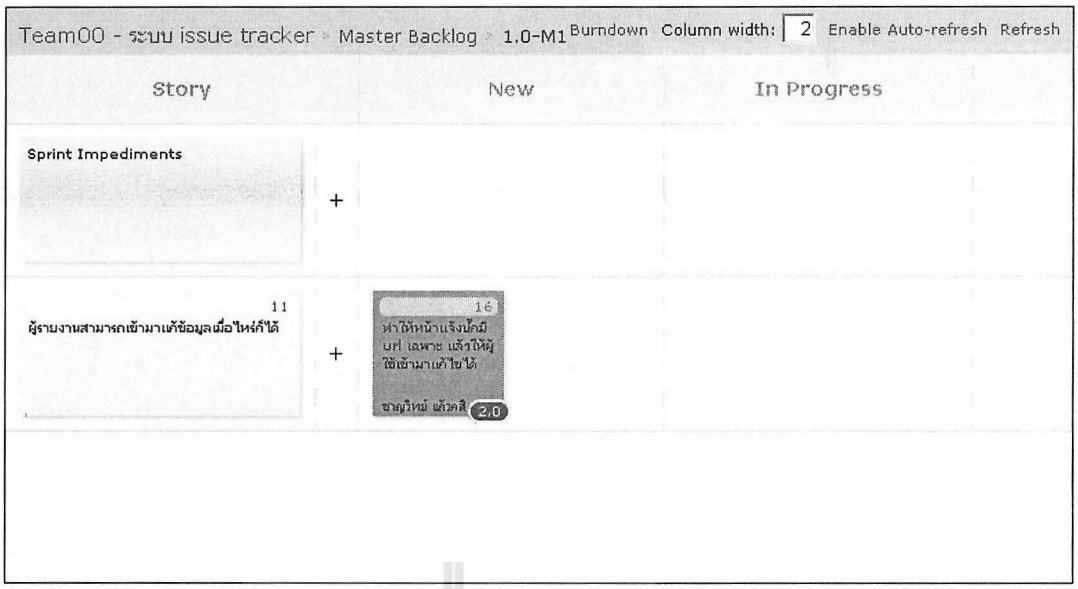


ใน Sprint Backlog ทางซ้าย จะเป็นที่เก็บ Backlog แยกตาม Sprint ชื่อ Sprint จะอยู่ในตำแหน่งบนซ้าย ตามด้วยตำแหน่งที่สามารถระบุวันเริ่มและวันสิ้นสุดของ Sprint ได้ ตามตัวอย่าง คือ วันที่ 2011-01-03 เป็นวันเริ่มต้นและวันที่ 2011-01-08 เป็นวันสิ้นสุดของ Sprint 1.0-M1 มุมบนขวาจะเป็น Point รวมของ Sprint ซึ่งอาจเป็น Story Point หรือ Use Case Point ก็ได้

มุมซ้ายของแต่ละ Sprint จะมีลูกศรชี้ลง v เพื่อแสดงเมนูรายการที่สามารถทำได้ เช่น การแสดง Task board ของแต่ละ Sprint เพื่อช่วยในการแบ่งงาน



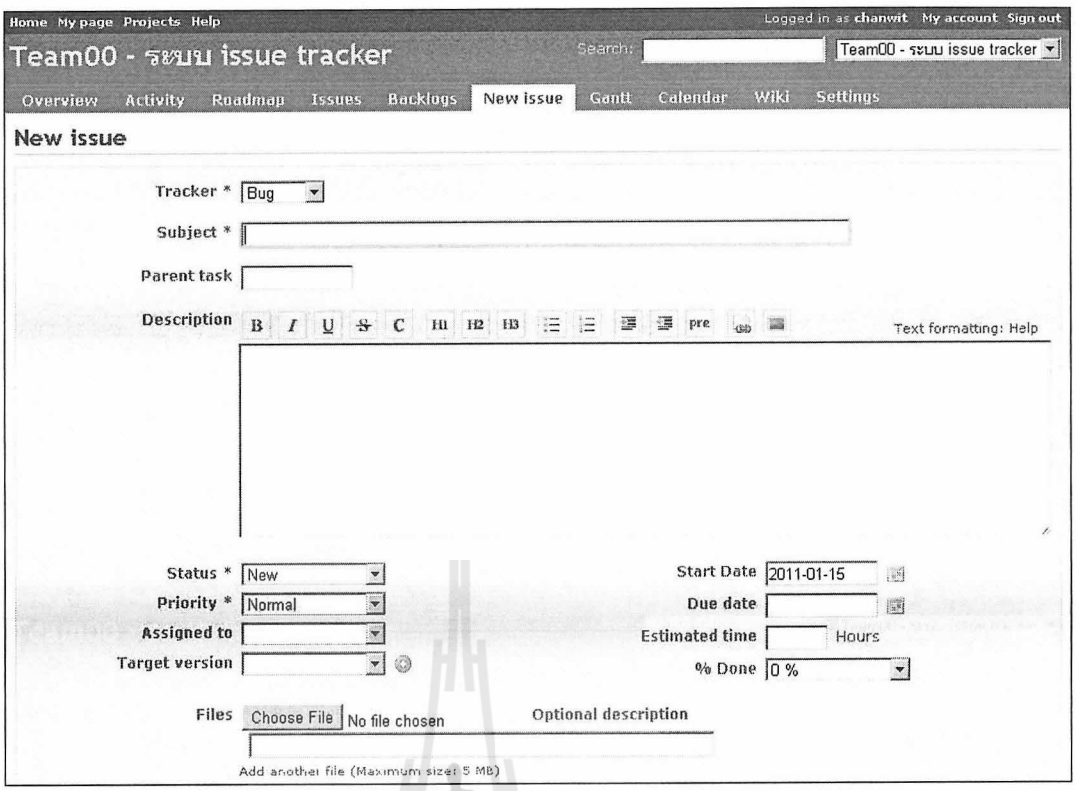
เมื่อเลือกเมนู Task board จะเป็นการเปิด Task Board ของ Sprint นั้นๆ ดังรูป



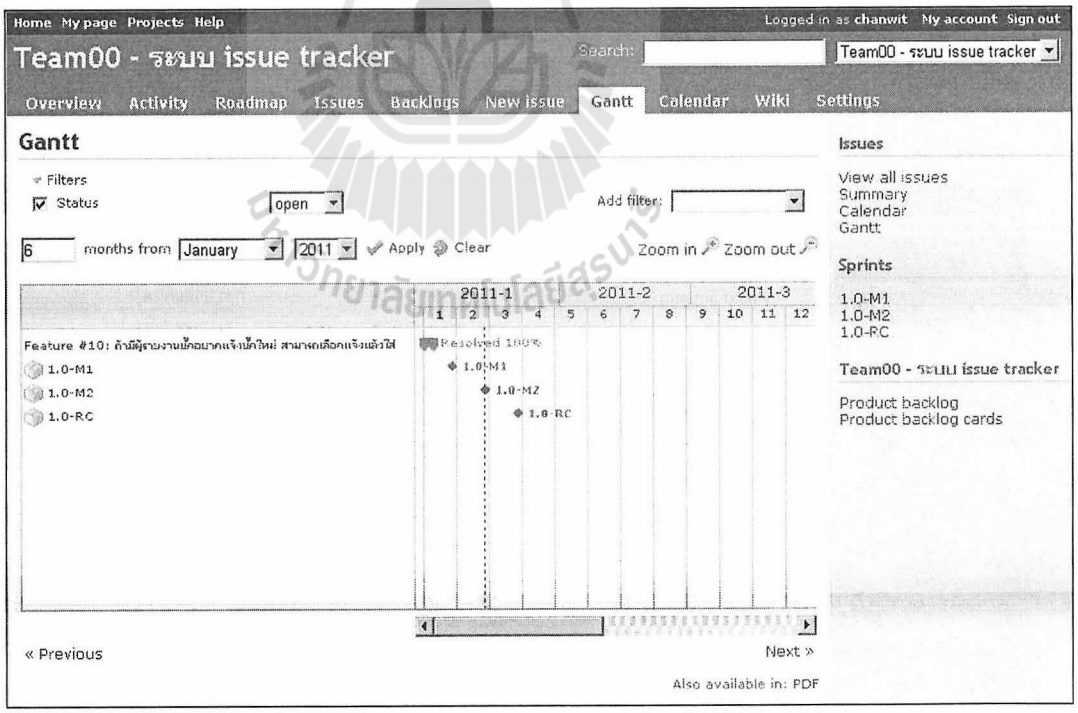
Task Board จะประกอบไปด้วย รายการ Story ในคอลัมน์ซ้ายสุด และสถานะของงานเรียงลำดับจาก New, In Progress ไปจนถึง Closed และ Rejected

ในแถวแรกจะเป็น Story พิเศษเพื่อใช้ระบุว่าใน Sprint นี้มีอุปสรรคอะไรบ้างและ Story จริงที่อยู่ใน Sprint Backlog จะเริ่มจากแถวที่สองเป็นต้นไป วิธีการเพิ่มงานให้ Story ทำได้โดยกดเครื่องหมาย + ที่อยู่ถัดจาก Story ในแต่ละแถว แล้วจึงใส่ข้อมูลของงานที่แตกออกจาก Story การเปลี่ยนสถานะของงานย่อยแต่ละงานสามารถทำได้โดย ลากงานไปยังสถานะอื่น เช่น จาก New ไปเป็น In Progress เมื่อเริ่มต้นทำงานนั้นๆ เป็นต้น

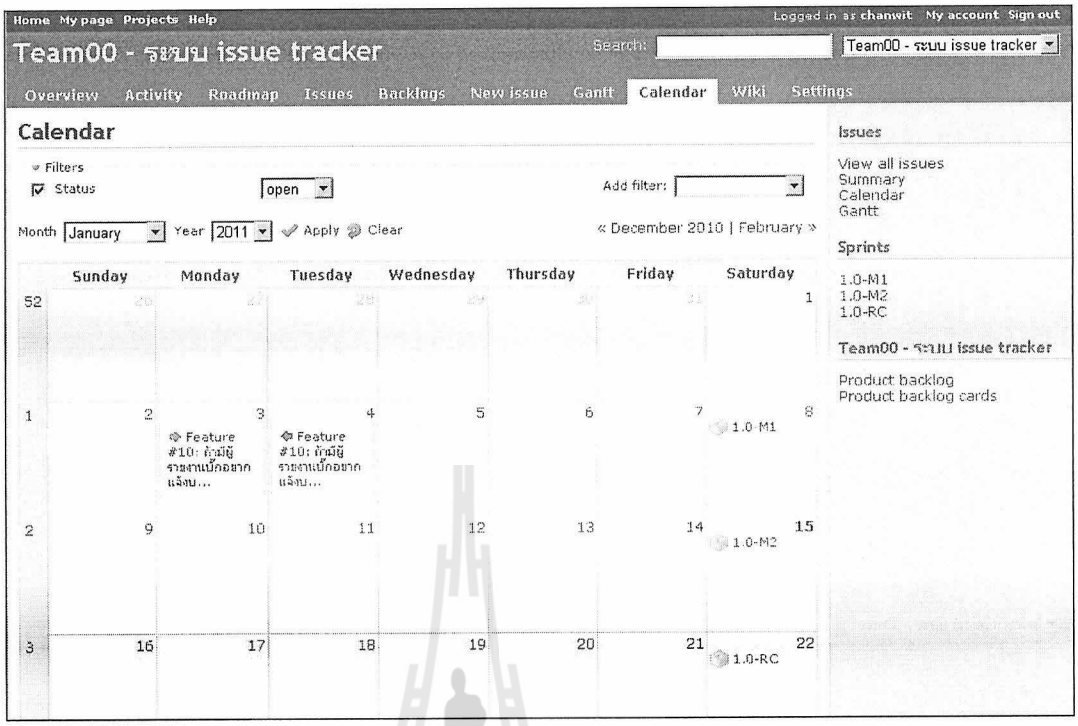
3. New issue เป็นการสร้าง Issue ใหม่ โดยปกติมักจะใช้ Tab นี้ในการสร้าง Issue ใหม่เพื่อแจ้ง Bug แต่ใน Redmine Backlogs จะใช้การสร้าง User Story เป็นตัวหลักในการจัดการ Issue



4. Gantt แสดง Gantt Chart ของโครงการ

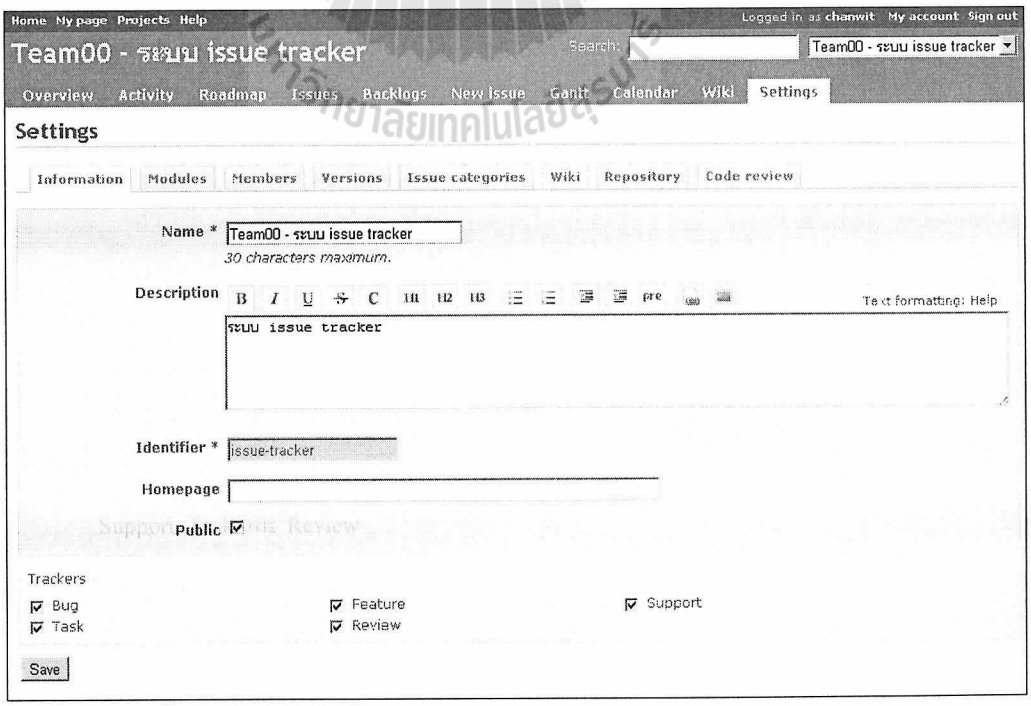


5. Calendar แสดงปฏิทินของโครงการ

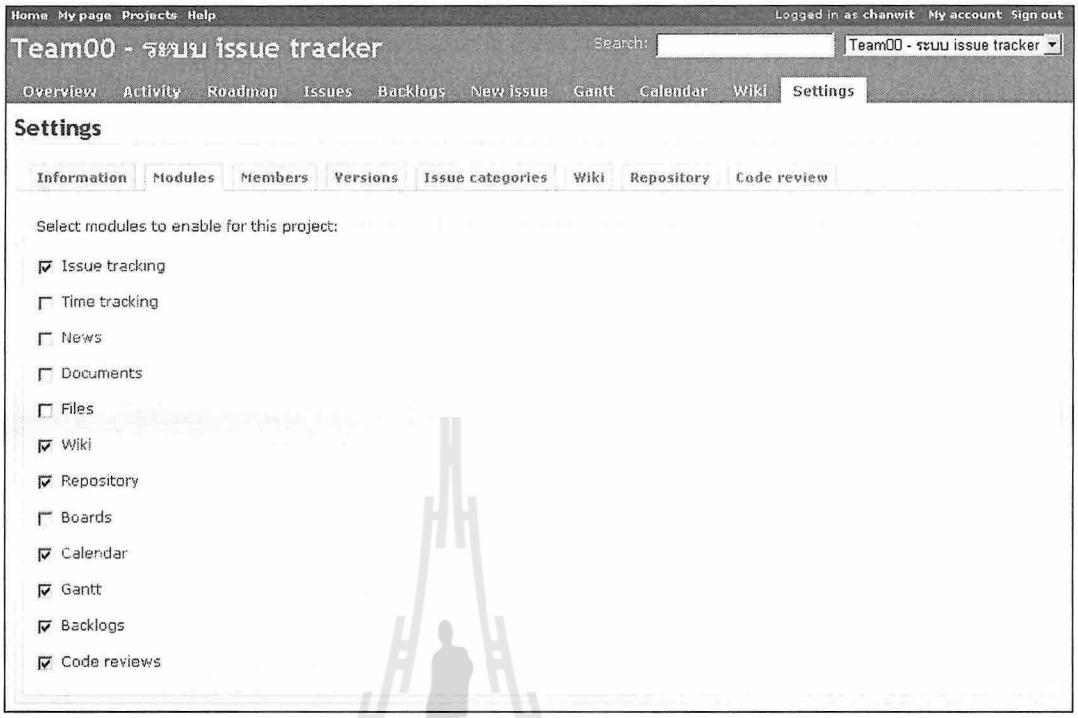


6. Settings ใช้สำหรับตั้งค่ารายละเอียดของโครงการโดยจะมี Tab ย่อยที่ใช้งานบ่อยดังนี้

1. Information ใช้สำหรับแก้ไขชื่อโครงการและรายละเอียด และจะมีรายการของ Checkbox แสดงตัว Tracker ที่ใช้อยู่สำหรับโครงการนี้ตามตัวอย่างมี Tracker 5 ประเภท คือ Bug, Feature, Support, Task และ Review

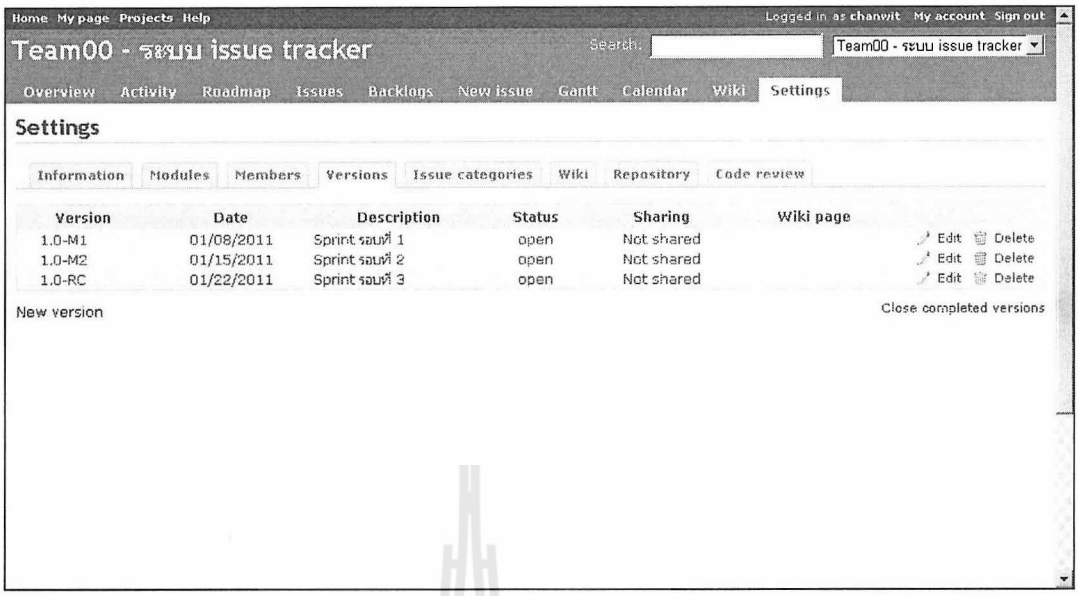


2. Modules ใช้สำหรับตั้งค่าโมดูลที่ต้องการใช้ในโครงการนี้ โดย Redmine Backlogs จะมีโมดูลชื่อ Backlogs เพิ่มจาก Redmine



3. Members ใช้สำหรับจัดการผู้ใช้หรือกลุ่มใดที่สามารถใช้งานโครงการนี้ได้บ้าง โดยผู้ที่มีสิทธิระดับ Admin หรือผู้จัดการกลุ่มจะสามารถเปลี่ยนแปลงระดับสิทธิเข้าถึงของแต่ละผู้ใช้ได้

4. Versions ใช้สำหรับเพิ่ม/ลบ/แก้ไข Sprint ใหม่ให้โครงการปัจจุบัน



5. Repository ใช้สำหรับเชื่อมโยงเครื่องมือประเภท Software Configuration Management เช่น Git เข้ากับโครงการ โดยต้องระบุ Path ของ Repository ไว้ในหน้านี้เพื่อจับคู่กับงาน

เครื่องมือกลุ่ม Software Configuration Management (SCM)

เป็นเครื่องมือทางวิศวกรรมซอฟต์แวร์ที่ใช้สำหรับช่วยดูแลรักษาต้นรหัส ควบคุมรุ่นของซอฟต์แวร์และอาจรวมถึงควบคุมกระบวนการสร้างด้วย

Git

เป็นเครื่องมือที่ช่วยจัดการควบคุมต้นรหัส ช่วยในการแตกสาขาของต้นรหัส และดูแลรักษา Patch ของต้นรหัสชุดอื่น

การเริ่มต้นใช้งาน Git สามารถทำได้โดยการสร้างแหล่งเก็บต้นรหัส (Repository) โดยคำสั่งการสร้างแหล่งเก็บต้นรหัส จะต้องสั่งในไดเร็คทอรี่ที่เตรียมไว้สำหรับทำงาน เช่น ถ้าโครงการเก็บไว้ที่ c:\dev\project จะต้องทำการเปลี่ยนไดเร็คทอรี่ด้วยคำสั่ง cd ไปยังไดเร็คทอรี่ดังกล่าวก่อน (บนระบบปฏิบัติการตระกูล Unix หรือ Linux อาจเป็น ~/dev/project) เมื่ออยู่ในไดเร็คทอรี่ที่ต้องการแล้ว
ทำการสร้างแหล่งเก็บต้นรหัสด้วยคำสั่งต่อไปนี้

```
$ git init
```

โดยคำสั่งนี้จะทำการสร้างไดเร็คทอรี่ย่อยชื่อ .git ซึ่งเก็บโครงสร้างไฟล์ที่จำเป็นในการควบคุมรุ่นของไฟล์

การควบคุมรุ่นของไฟล์ ทำได้โดยการเพิ่มไฟล์เข้าไปยังแหล่งเก็บต้นรหัสด้วยคำสั่ง git add และยืนยันด้วยคำสั่ง git commit ดังตัวอย่างต่อไปนี้

```
$ git add Main.java
$ git add README
$ git commit –m “init first commit”
```

ในกรณีที่มีต้นรหัสเก็บอยู่ในแหล่งเก็บต้นรหัสอื่น เช่น บนเซิร์ฟเวอร์ จะต้องทำการโคลนทั้งแหล่งเก็บต้นรหัสมาไว้ที่เครื่อง ด้วยคำสั่ง git clone โดยระบุ URL ของแหล่งเก็บต้นรหัสต่อจากคำสั่งดังนี้

```
$ git clone git@203.158.7.11:team00.git
```

โดยตัวอย่างคำสั่งนี้จะทำการสร้างไดเร็คทอรี่ชื่อ team00 ขึ้นแล้วทำการ initialize ตัว .git ไดเร็คทอรี่ให้ จากนั้นจึงทำการดึงต้นรหัสมาจากเซิร์ฟเวอร์ผ่านโปรโตคอล SSH ซึ่งนอกจากโปรโตคอลนี้แล้ว Git ยังสนับสนุนโปรโตคอลอื่นๆ ดังต่อไปนี้

โปรโตคอล git เป็นโปรโตคอลเฉพาะของ Git มีการใช้พอร์ตพิเศษ ซึ่งอาจเข้าถึงไม่ได้หากมีการตั้งค่า Firewall กันไว้ ตัวอย่าง URL ของโปรโตคอลนี เช่น

```
git://github.com/chanwit/test.git
```

โปรโตคอล http เป็นการใช้งาน Git ผ่านโปรโตคอลของ Web ซึ่งใช้พอร์ตปกติที่ Web Server ใช้ (พอร์ต 80) ตัวอย่าง URL ของโปรโตคอลนี้ เช่น

```
http://github.com/chanwit/test.git
```

ในลักษณะเดียวกันกับ http Git สนับสนุนโปรโตคอล https (พอร์ต 443) ด้วย

เมื่อใช้คำสั่งโคลนแล้ว จะมีการสร้างแหล่งเก็บต้นรหัสชื่อ origin และสาขา (Branch) ชื่อmaster ซึ่งเป็นชื่อที่ git กำหนดให้

เมื่อต้องการตรวจสอบสถานะของไฟล์ ใช้คำสั่งดังนี้

```
# On bra...
nothing to commit (working directory clean)
```

ในสถานะตามตัวอย่างหมายความว่า ไดเร็คทอรี่ปัจจุบันไม่มีไฟล์ค้างอยู่ เมื่อมีการสร้างไฟล์ใหม่เพิ่มเข้ามาในไดเร็คทอรี่นี้ การใช้คำสั่ง git status จะพบว่าสถานะของแหล่งเก็บต้นรหัสไม่เหมือนเดิม เนื่องจากมีไฟล์ใหม่ค้างอยู่ ตัวอย่างเช่น

```
$ git status
# On branch master
```

Untracked files:

#

MyClass.java

ไฟล์ที่สร้างขึ้นใหม่นี้เรียกว่า ไฟล์ที่ยังไม่ได้ตามการเปลี่ยนแปลง (Untracked file) ดังนั้นเมื่อต้องการตามการเปลี่ยนแปลงของไฟล์ จะใช้คำสั่ง git add ดังตัวอย่างต่อไปนี้

$ git add MyClass.java

เมื่อทำการสั่ง git status อีกครั้ง จะพบว่า สถานะของแหล่งเก็บต้นรหัสมีการเปลี่ยนแปลงไปดังนี้

$ git status

On branch master

Changes to be committed:

#

new file: MyClass.java

ไฟล์ที่ถูกเพิ่มเข้าไปใหม่นี้เรียกว่า ไฟล์ที่จัดแล้ว (Staged file) เมื่อมีการแก้ไขไฟล์ที่มีอยู่แล้ว การใช้คำสั่ง git status จะแสดงผลดังต่อไปนี้

$ git status

On master branch

Changes to be committed:

#

new file: MyClass.java

Changed but not updated:

modified: Main.java

ซึ่งหมายความว่ามีไฟล์ที่มีการแก้ไขเกิดขึ้น เมื่อต้องการจัดไฟล์เพื่อเตรียมยืนยันในการเก็บเข้า จะต้องใช้คำสั่ง git add เสมอ

$ git add Main.java

หลังจากทำการจัดไฟล์เรียบร้อย เมื่อใช้คำสั่ง git status จะแสดงผลดังต่อไปนี้

$ git status

On branch master

Changes to be committed:

#

modified: Main.java

new file: MyClass.java

ไฟล์ที่แก้ไขจะถูกนำเข้าไปอยู่ในพื้นที่จัดเตรียมเพื่อเตรียมยืนยันในการเก็บเข้า ดังนั้นการยืนยันในการเก็บเข้า จะเป็นการนำไฟล์ที่อยู่ในพื้นที่จัดเตรียมเก็บเข้าไปยังระบบเก็บข้อมูลของ git ซึ่งทำได้โดยใช้คำสั่ง git commit โดยทั่วไปจะใช้กับพารามิเตอร์ -m เพื่อระบุหมายเหตุของแต่ละการยืนยัน

git commit –m “updated 2 files for testing”

Git จะแสดงผลดังนี้

[master 40e22f2] updated 2 files for testing

2 files changed, 2 insertions (+), 0 deletions (-)

created mode 100644 MyClass.java

จากข้อความด้านบนหมายความว่า Git ได้ทำการสร้างการยืนยันใหม่ขึ้นมีหมายเลขคือ 40e22f2 เป็นการยืนยันบนสาขา master และมีการเปลี่ยนแปลง 2 ไฟล์

ถ้ามีการแก้ไขไฟล์ที่มีการตามการเปลี่ยนแปลงอยู่แล้ว สามารถใช้คำสั่ง git commit ได้โดยข้ามขั้นตอน git add ด้วยพารามิเตอร์ -a ดังตัวอย่างต่อไปนี้

$ git commit –a –m " made changes to Main.java"

คำสั่งอื่นๆที่เกี่ยวข้องกับการจัดการไฟล์ ได้แก่

คำสั่งสำหรับลบไฟล์

$ git rm ชื่อไฟล์

คำสั่งสำหรับย้ายไฟล์หรือเปลี่ยนชื่อไฟล์

$ git mv ไฟล์เก่า ไฟล์ใหม่

เมื่อต้องการดูรายการการยืนยันย้อนหลังเพื่อดูการเปลี่ยนแปลงที่เกิดขึ้น สามารถทำได้โดยใช้คำสั่ง git log ดังนี้

$ git log

Commit 40e22f25

Author: Chanwit Kaewkasi

Date: Tue Jan 18 11:34:07 2011 +0700

Updated 2 files for testing

Git อนุญาตให้มีการ Undo การเปลี่ยนแปลงของไฟล์ได้ อย่างไรก็ตามควรใช้คำสั่งเหล่านี้อย่างระมัดระวัง เพราะอาจทำให้งานบางส่วนหายไปได้หากใช้ผิดวิธี

นอกจากนี้ Git สามารถทำการยืนยันซ้ำไปบนการยืนยันล่าสุดได้ โดยใช้พารามิเตอร์ -- amend ดังตัวอย่างต่อไปนี้

$ git commit –m "init"

$ git add new_file.txt

$ git commit -- amend –m "init"

การสั่งยืนยันในบรรทัดที่ 3 จะทับการยืนยันในบรรทัดที่ 1 ทำให้ไม่เกิดการยืนยันใหม่เพิ่มขึ้น อย่างไรก็ตาม หมายเลขการยืนยันจะเปลี่ยนเป็นเลขชุดใหม่

สำหรับการดึงไฟล์ที่จัดแล้วกลับออกมาเป็นไฟล์ปกติ สามารถทำได้โดยใช้คำสั่ง git reset ดังตัวอย่างต่อไปนี้

$ git add Main.java

จะทำให้ไฟล์ Main.java ถูกจัดเข้าไปอยู่ในพื้นที่จัดเตรียม

$ git reset HEAD Main.java

จะเป็นการดึงไฟล์ Main.java ออกมาจากพื้นที่จัดเตรียม ซึ่งก็คือ การยกเลิกคำสั่ง git add ที่ใช้ไปก่อนหน้านี้ และหากต้องการให้เนื้อหาของไฟล์ย้อนกลับไปเป็นการยืนยันล่าสุดแทนสิ่งที่เพิ่งแก้ไขสามารถใช้คำสั่ง git checkout ดังตัวอย่างต่อไปนี้

$ git checkout -- Main.java

จะทำให้สิ่งที่เพิ่งแก้ไขในไฟล์ Main.java หายไปและกลับไปเหมือนสถานะที่เพิ่งยืนยันเข้าไปครั้งล่าสุด

การทำงานกับเซิร์ฟเวอร์ Git

Git อนุญาตให้ทำงานกับแหล่งเก็บต้นรหัสระยะไกลได้ โดยมีการใช้คำสั่ง git remote add เพื่อสร้างแหล่งเก็บต้นรหัสระยะไกล ชื่อ origin จากนั้นทำการดึงต้นรหัสด้วยคำสั่ง git fetch และสร้างสาขา master ขึ้นจาก origin/master โดยใช้คำสั่ง git checkout ดังนี้

$ git remote add origin "URL ปลายทาง"

$ git fetch origin

$ git checkout –b master origin/master

เมื่อทำการแก้ไขต้นรหัสบนเครื่องเรียบร้อยแล้ว

จะต้องผลักต้นรหัสกลับขึ้นไปยังแหล่งเก็บต้นรหัสระยะไกล โดยใช้คำสั่ง git push ดังตัวอย่างต่อไปนี้

$ git push origin master

ซึ่งหมายความว่า ต้นรหัสจากสาขา master บนเครื่องจะถูกผลักไปไว้ที่ origin/master

การจัดการสาขา (Branching)

เป็นเทคนิคสำคัญในการจัดการต้นรหัส โดยเครื่องมือ SCM บางตัวทำการจัดการสาขาได้ไม่ดีทำให้เกิดความเสียหายกับต้นรหัสได้ง่ายต่างกับ SCM ประเภทกระจาย เช่น Git หรือ Mercurial (hg) ที่สนับสนุนการแตกสาขาและการรวมกลับ (Merge) รวมถึงการปรับฐาน (Rebase) ได้อย่างมีประสิทธิภาพ ทำให้สามารถดูแลรักษาต้นรหัสได้ดีกว่าเครื่องมือชุดอื่น

โดยปกติสาขาหลักของต้นรหัสคือ สาขา master แต่ขณะทำงานไม่ควรใช้ต้นรหัสจากสาขา master โดยตรง ควรทำการแตกสาขาออกไปเป็นสาขา develop ก่อนแล้วจึงค่อยรวมกลับภายหลัง

วิธีการสร้างสาขาใหม่ทำได้โดยใช้คำสั่งดังนี้

$ git checkout –b develop

โดย -b หมายถึงการสร้างสาขาใหม่ ปกติคำสั่ง git checkout จะใช้เพื่อสลับระหว่างสาขา เช่น

$ git checkout master

ใช้สำหรับสลับมาที่สาขา master

$ git checkout develop

ใช้สำหรับสลับมาที่สาขา develop

เมื่อทำการแก้ไขต้นรหัสในสาขา develop แล้ว ต้นรหัสในสาขา master จะไม่มีการเปลี่ยนแปลง ดังนั้นเมื่อทำการเขียนและทดสอบต้นรหัสบนสาขา develop เรียบร้อยแล้ว จะต้องทำการรวมต้นรหัสกลับไปที่ master ซึ่งทำได้โดย checkout master ออกมาก่อนแล้วใช้คำสั่ง git merge

$ git checkout master

$ git merge develop

สาขาหัวข้อ (Topic Branch)

ในการพัฒนาซอฟต์แวร์ จะมีการพัฒนาตาม Story หรือ Issue ที่กำหนด ซึ่งวิธีการคือ ทำการแตกสาขาออกมาจากสาขา develop โดยจะตั้งชื่อสาขาที่แตกออกมาตามหมายเลข Story หรือ Issue ที่ต้องทำงานด้วย ซึ่งเรียกว่าการสร้างสาขาหัวข้อ โดยสาขาประเภทนี้จะถูกลบทิ้งเมื่อพัฒนาคุณสมบัติ (Feature) ตามที่ระบุไว้ใน Story หรือ Issue เสร็จ ดังตัวอย่างต่อไปนี้

```
$ git checkout develop
$git checkout –b iss-123
```

จากตัวอย่างจะมีการสร้างสาขา iss-123 สำหรับ Issue 123 จากนั้นเมื่อแก้ไขและ Commit เรียบร้อยแล้วจะทำการรวมกลับไปยังสาขา develop และลบสาขา iss-123 ทิ้งไป

```
$ git checkout develop
$ git merge iss-123
$ git branch –d iss-123
```

การปรับฐาน (Rebase)

เป็นการนำเอาต้นรหัสที่ได้มาจากผู้อื่นมาเป็นฐานให้กับการเปลี่ยนแปลงที่เรากำลังพัฒนาอยู่ เช่น ในทีมพัฒนามีสมาชิก 2 คน คนที่ 1 รับผิดชอบ Issue 101 ส่วนคนที่ 2 รับผิดชอบ Issue 105 ทั้งสองคนเริ่มทำงานพร้อมกัน คนที่ 1 ทำเสร็จก่อนและทำการรวม iss-101 กลับไปยังสาขา develop คนที่ 2 จึงจำเป็นต้องปรับฐานของสาขา iss-105 ซึ่งกำลังพัฒนาอยู่ให้ตรงกับ develop ใหม่ โดยการสลับไปยังสาขา develop แล้วดึงต้นรหัสล่าสุดมาจากเซิร์ฟเวอร์ จากนั้นสลับกลับมาที่ iss-105 แล้วใช้คำสั่ง git rebase กับสาขา develop ดังตัวอย่างต่อไปนี้

```
$ git checkout develop
$ git remote update
$ git pull origin develop
$ git checkout iss-105
$ git rebase develop
```

จากตัวอย่าง คำสั่งบรรทัดที่ 1 จะเป็นการสลับไปยังสาขา develop คำสั่งบรรทัดที่ 2 เป็นการปรับปรุงต้นรหัสจากเซิร์ฟเวอร์ คำสั่งบรรทัดที่ 3 จะเป็นการดึงต้นรหัสจาก origin มายัง develop คำสั่งบรรทัดที่ 4 เป็นการสลับกลับมาที่ iss-105 จากนั้นคำสั่งบรรทัดที่ 5 จะเป็นการปรับฐาน ผลที่ได้คือ ส่วนที่แก้ไขเพิ่มเติมไว้ใน iss-105 จะเรียงอยู่ด้านบนของ iss-101

เครื่องมือทดสอบ (Testing Tool)

เป็นเครื่องมือทางวิศวกรรมซอฟต์แวร์ที่ใช้ทดสอบคุณภาพ, ความถูกต้องของซอฟต์แวร์ ซึ่งแบ่งออกได้เป็นหลายระดับ เช่น เครื่องมือทดสอบระดับหน่วย (Unit Test) และเครื่องมือทดสอบเพื่อการยอมรับ (Acceptance Test)

การทดสอบระดับหน่วยด้วย xUnit

การทดสอบระดับหน่วยเป็นต้นรหัสที่เขียนโดยนักพัฒนาระบบ มีขนาดเล็ก มีความเจาะจงเพื่อจะทดสอบบางส่วนของซอฟต์แวร์เท่านั้น นั่นคือ การทดสอบระดับหน่วยจะใช้เพื่อทดสอบว่าต้นรหัสจุดนั้นๆเป็นไปตามที่ผู้พัฒนาตั้งใจจะให้เป็นหรือไม่

การทดสอบระดับหน่วยควรจะมีขนาดเล็กในระดับที่สามารถทดสอบได้หลายร้อยกรณีภายในไม่กี่วินาที หากมีการทดสอบใดที่ช้าควรแยกออกมาต่างหากและรันการทดสอบที่ช้าเหล่านั้นเพียงวันละ 1 ครั้ง เพื่อไม่ให้ถ่วงเวลาการพัฒนา

คลาสสำหรับทดสอบระดับหน่วยจะอยู่ในกลุ่มเฟรมเวิร์ค เรียกว่า xUnit ซึ่งขึ้นกับแต่ละภาษาที่ใช้พัฒนา เช่น ภาษา Java ใช้ JUnit, ภาษา PHP ใช้ PHPUnit เป็นต้น อย่างไรก็ตามหากพัฒนาซอฟต์แวร์ด้วยเฟรมเวิร์คเฉพาะ เช่น Grails, CakePHP หรือ Ruby on Rails นั้น ในตัวเฟรมเวิร์คเหล่านี้จะมีกลไกการทดสอบระดับหน่วยมาให้ใช้โดยเฉพาะ

จากตัวอย่าง User Story ต่อไปนี้

Story #20 ในบทบาทของลูกค้า ลูกค้าสามารถซื้อพิซซ่าได้ไม่เกิน 2 ถาดต่อครั้งเพื่อให้พิซซ่าเพียงพอกับลูกค้าอื่นๆ

จาก User Story นี้ สามารถเขียนการทดสอบระดับหน่วยได้ดังนี้

ตัวอย่าง Grails (ใช้ JUnit)

สร้าง Grails Application

$ grails create–app pizza–shop

$ cd pizza-shop

สร้างคลาสหลักที่เกี่ยวข้อง

$ grails create-domain-class PizzaOrder

$ grails create-domain-class Pizza

จากนั้นจึงทำการเขียนกรณีทดสอบ (Test Case) สำหรับ PizzaOrder ซึ่งอยู่ใน test/unit/pizza/shop/PizzaOrderTests.groovy แล้วเขียนกรณีทดสอบ ดังรูป

```
package pizza.shop

import grails.test.*

class PizzaOrderTests extends GrailsUnitTestCase {

    void testNoMoreTwoPizzaesPerOrder() {
        def pizza = [new Pizza(), new Pizza(), new Pizza()]
        mockDomain(Pizza, pizza)

        mockDomain(PizzaOrder)
        def po = new PizzaOrder().save()

        po.addToPizzas(pizza[0])
        po.addToPizzas(pizza[1])
        po.addToPizzas(pizza[2])

        assert po.validate() == false
    }
}
```

กรณีทดสอบในตัวอย่างนี้ทำการเตรียมวัตถุของ Pizza 3 วัตถุเพิ่มเข้าไปยังวัตถุของ PizzaOrder บรรทัดสุดท้ายจะเป็นการคาดหวังว่าการ Validate ควรจะเป็น false เพราะตาม User Story บอกไว้ว่า 1 Order มี Pizza ได้ไม่เกิน 2 ถาด แต่วัตถุที่ใส่มีจำนวนเป็น 3

ในการรันการทดสอบครั้งแรก Validate นี้จะเป็น true เพราะไม่มีการกำหนดเงื่อนไขไว้ในระบบเนื่องจากสร้างคลาสแล้วทดสอบก่อนที่จะเขียนซอฟต์แวร์ นักพัฒนามีหน้าที่แก้ไขคลาสให้เป็นไปตามผลลัพธ์ที่คาดหวังในกรณีทดสอบ และทำซ้ำจนกว่าการทดสอบจะรันผ่าน

เช่นเดียวกันกับบนเฟรมเวิร์ค CakePHP (ใช้ SimpleTest) เมื่อทำการสร้าง cake application และสร้างฐานข้อมูลตาราง pizzas และ pizza_orders จากนั้นทำการรันคำสั่ง

$ cake bake all Pizza

$ cake bake all PizzaOrder

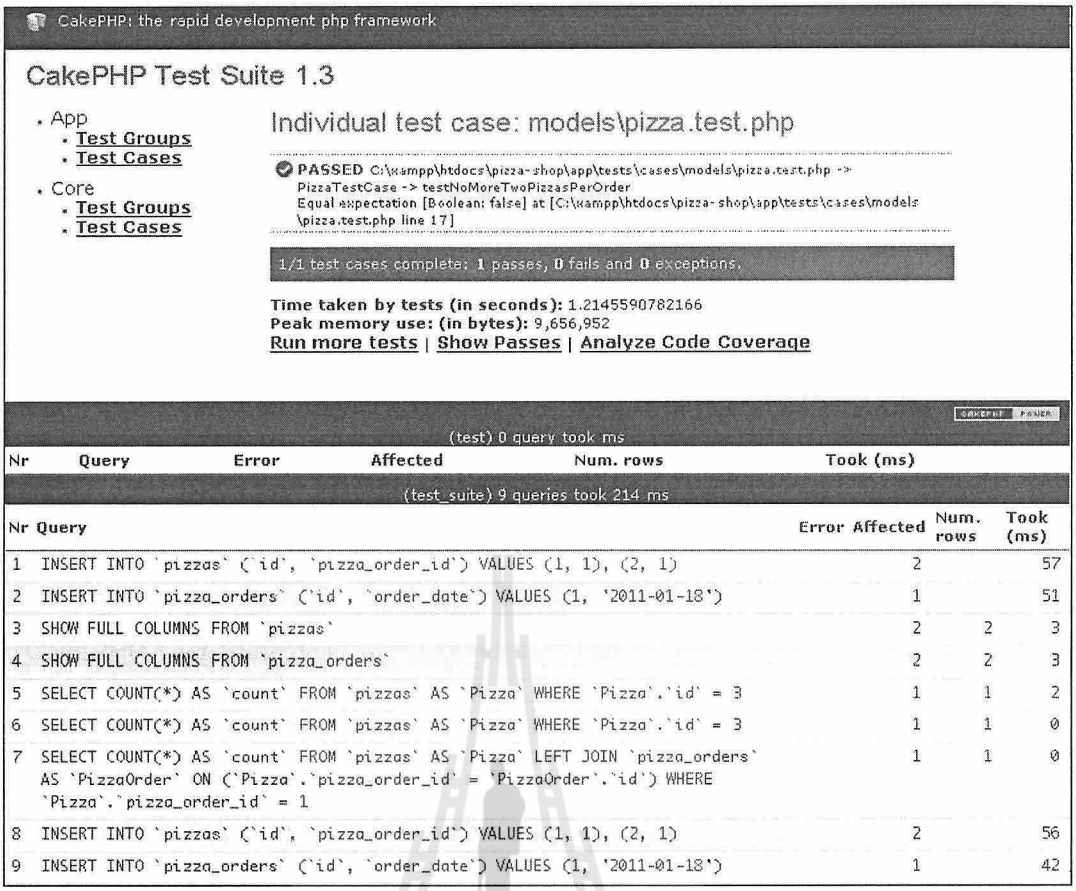
จะได้โมเดลสำหรับตารางทั้งสองตารางการทดสอบระดับหน่วยของ CakePHP นั้นจะใช้ SimpleTest 1.0.1 ซึ่งต้องดาวน์โหลดมาไว้ที่ app/vendors ก่อนใช้งาน ไฟล์การทดสอบระดับหน่วยของ Pizza จะอยู่ที่ app/tests/cases/models/pizza.test.php ซึ่งมีรายละเอียดดังรูป

```
<?php
App::import('Model', 'Pizza');

class PizzaTestCase extends CakeTestCase {
   var $fixtures = array('app.pizza', 'app.pizza_order');

   function startTest() {
      $this->Pizza =& ClassRegistry::init('Pizza');
   }

   function testNoMoreTwoPizzasPerOrder() {
      $result = $this->Pizza->save(
         array('Pizza' => array( 'id' => 3, 'pizza_order_id' => 1 ))
      );
      $expected = false;
      $this->assertEqual($result, $expected);
   }

   function endTest() {
      unset($this->Pizza);
      ClassRegistry::flush();
   }

}
?>
```

ใน CakePHP มีการแยกไฟล์ข้อมูลสำหรับทดสอบต่างหาก จึงทำให้ต้นรหัสส่วนที่เป็นกรณีทดสอบมีเพียงการเพิ่มข้อมูล Pizza ถาดที่ 3 และการเรียก assertEqual โดยข้อมูลทดสอบใน CakePHP เรียกว่า Test Fixtures อยู่ใน app/tests/fixtures

การทดสอบทำได้โดยชี้ไปยัง URL

http://localhost/<ชื่อโปรแกรม>/app/webroot/test.php แล้วเลือกการทดสอบที่ต้องการ



CakePHP: the rapid development php framework

CakePHP Test Suite 1.3

- App
  - Test Groups
  - Test Cases
- Core
  - Test Groups
  - Test Cases

Individual test case: models\pizza.test.php

PASSED C:\xampp\htdocs\pizza-shop\app\tests\cases\models\pizza.test.php -> PizzaTestCase -> testNoMoreTwoPizzasPerOrder
Equal expectation [Boolean: false] at [C:\xampp\htdocs\pizza-shop\app\tests\cases\models\pizza.test.php line 17]

1/1 test cases complete: 1 passes, 0 fails and 0 exceptions.

Time taken by tests (in seconds): 1.2145590782166
Peak memory use: (in bytes): 9,656,952
Run more tests | Show Passes | Analyze Code Coverage

(test) 0 query took ms

Nr	Query	Error	Affected	Num. rows	Took (ms)

(test_suite) 9 queries took 214 ms

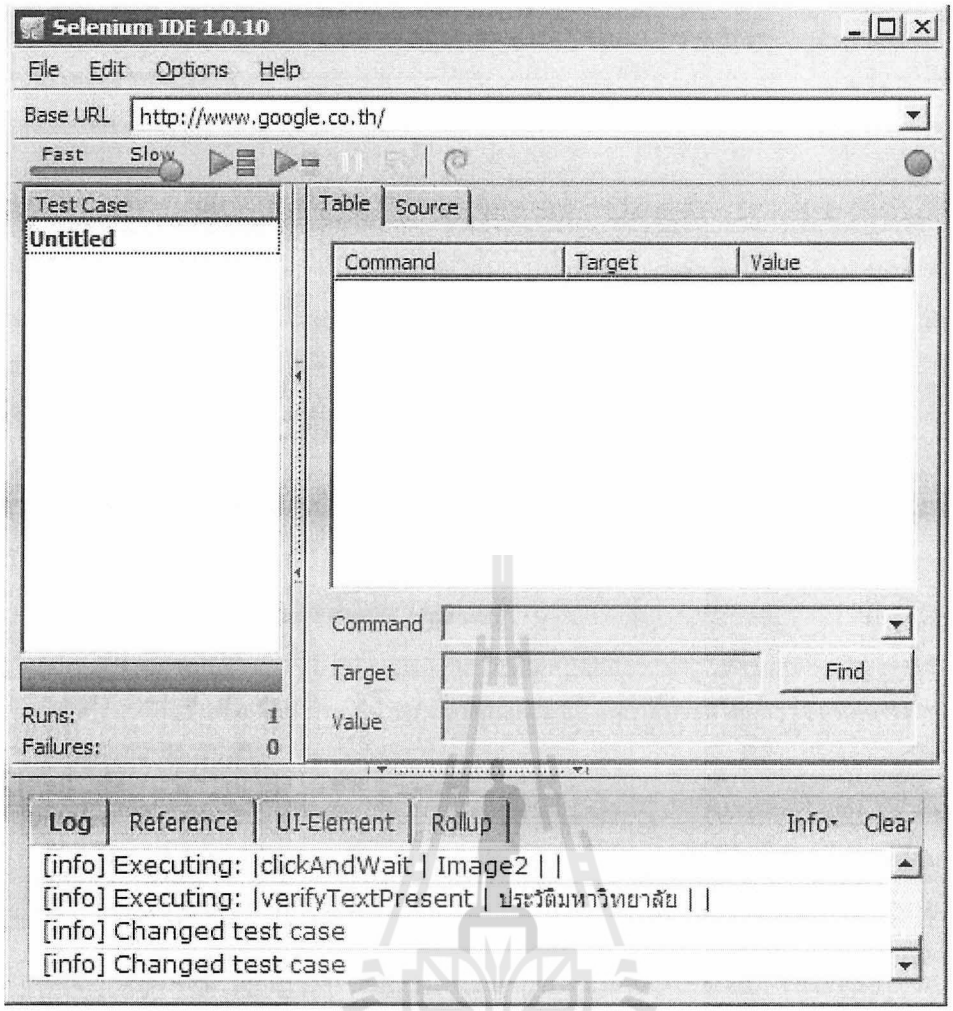
Nr	Query	Error	Affected	Num. rows	Took (ms)
1	INSERT INTO `pizzas` (`id`, `pizza_order_id`) VALUES (1, 1), (2, 1)		2		57
2	INSERT INTO `pizza_orders` (`id`, `order_date`) VALUES (1, '2011-01-18')		1		51
3	SHOW FULL COLUMNS FROM `pizzas`		2	2	3
4	SHOW FULL COLUMNS FROM `pizza_orders`		2	2	3
5	SELECT COUNT(*) AS `count` FROM `pizzas` AS `Pizza` WHERE `Pizza`.`id` = 3		1	1	2
6	SELECT COUNT(*) AS `count` FROM `pizzas` AS `Pizza` WHERE `Pizza`.`id` = 3		1	1	0
7	SELECT COUNT(*) AS `count` FROM `pizzas` AS `Pizza` LEFT JOIN `pizza_orders` AS `PizzaOrder` ON (`Pizza`.`pizza_order_id` = `PizzaOrder`.`id`) WHERE `Pizza`.`pizza_order_id` = 1		1	1	0
8	INSERT INTO `pizzas` (`id`, `pizza_order_id`) VALUES (1, 1), (2, 1)		2		56
9	INSERT INTO `pizza_orders` (`id`, `order_date`) VALUES (1, '2011-01-18')		1		42

ผลการรันกรณีทดสอบจะแสดงอยู่ในรูปข้างต้น

การทดสอบเพื่อการยอมรับด้วย Selenium

Selenium เป็นเฟรมเวิร์คสำหรับทดสอบเพื่อยอมรับหรือทดสอบเชิงฟังก์ชั่น การทดสอบประเภทนี้มักจะทำโดยผู้พัฒนาหลังจากระบบพัฒนาเสร็จแล้วหรือทดสอบโดยลูกค้าเพื่อรับมอบงานในช่วงปลายของโครงการ Selenium จะทำการรันและทดสอบซอฟต์แวร์ใน Browser โดยการควบคุม JavaScript ให้ทำงานตามคำสั่ง โดยสามารถเขียนกรณีทดสอบเพื่อส่งให้ Selenium ทำงานได้จากหลายๆภาษา ไม่ว่าจะเป็น Java, Groovy, Ruby หรือ Python เป็นต้น

จุดเด่นที่ชัดเจนของ Selenium คือ สามารถจำลองการใช้งานของผู้ใช้ได้และหยิบส่วนใดส่วนหนึ่งของหน้าเว็บมาเพื่อทวนสอบความถูกต้องได้ เครื่องมือที่ใช้ร่วมกับ Selenium เพื่อทำการบันทึกแล้วแปลงเป็น Script เรียกว่า Selenium IDE ทำงานบน Firefox Browser



บทที่ 6
การวางแผนโครงการ
Project Planning

ในบทนี้จะกล่าวถึงการวางแผนโครงการ ซึ่งเป็นกระบวนการที่เกิดขึ้นก่อนการเริ่มพัฒนาซอฟต์แวร์ รวมถึงการวัดซอฟต์แวร์ในลักษณะต่างๆ การประมาณโครงการซอฟต์แวร์ด้วยตัววัด ในการพัฒนาซอฟต์แวร์การวางแผนโครงการมีความจำเป็นเป็นอย่างยิ่ง เนื่องจากจะช่วยให้สามารถกำหนดระยะเวลา วิธีการทำงาน รวมทั้งสามารถช่วยควบคุมงบประมาณของโครงการก่อนการพัฒนาได้

การวัดซอฟต์แวร์

การวัดซอฟต์แวร์เป็นการใช้ตัววัดในลักษณะต่างๆกัน เพื่อให้ได้ปริมาณที่สามารถระบุขนาดของซอฟต์แวร์ในมิติที่ต้องการได้ โดยตัววัดที่น่าสนใจมีดังต่อไปนี้

1. **ตัววัดเชิงขนาด (Size-Oriented Metrices)**

   เป็นตัววัดที่คำนวณปริมาณเชิงคุณภาพหรือประสิทธิผล (Productivity) โดยพิจารณาจากขนาดของซอฟต์แวร์ที่สร้างขึ้น ซึ่งอาจจะเป็นการเก็บข้อมูลย้อนหลังเพื่อใช้ประมวลผลว่า ซอฟต์แวร์ที่สร้างขึ้นมีต้นทุนเท่าไร เป็นต้น

   เช่น โครงการ X มีจำนวนต้นรหัส (Source Code) 13,800 บรรทัด ใช้แรงงาน 24 คน-เดือน ใช้ต้นทุน 504,000 บาท มีเอกสาร 200 หน้า พบจำนวนข้อผิดพลาด (Errors) 96 ที่ จำนวนข้อบกพร่อง (Defects) 17 ที่ ใช้นักพัฒนา 3 คน
   จากข้อมูลนี้จะเห็นได้ว่า เวลาที่ใช้พัฒนาอาจเป็น 24/3 = 8 เดือน
   ค่าใช้จ่ายต่อเดือน 504,000/8 = 63,000 บาท เป็นต้น

   ตัววัดอื่นๆที่สามารถคำนวณได้จากขนาดซอฟต์แวร์ อาจมีได้ดังนี้
   - Errors ต่อ KLOC (Kilo Line of Code - 1,000 บรรทัดของต้นรหัส) สำหรับหาอัตราข้อผิดพลาด
   - Defects ต่อ KLOC สำหรับหาอัตราข้อบกพร่อง
   - ราคาต่อ KLOC
   - จำนวนหน้าของเอกสารต่อ KLOC
   - ข้อผิดพลาดต่อคน-เดือน
   - KLOC ต่อคน-เดือน
   - ราคาต่อหน้าเอกสาร

ตัวอย่างการวัดเชิงขนาด

โครงการ	LOC	แรงงาน	ค่าใช้จ่าย	เอกสาร(หน้า)	ข้อผิดพลาด	ข้อบกพร่อง	นักพัฒนา
A	12,100	24	168,000	365	134	29	3
B	27,200	62	440,000	1,224	321	86	5
C	20,200	43	314,000	1,050	256	64	6

2. ตัววัดเชิงฟังก์ชั่น (Function-Oriented Metrices)

เป็นการวัดขนาดของซอฟต์แวร์โดยใช้ความสามารถ (Functionality) ของซอฟต์แวร์นั้นๆ เป็นเกณฑ์ ซึ่งตัววัดประเภทนี้ที่เป็นที่นิยมคือ Function Point (FP) โดยการคำนวณค่า FP นั้นขึ้นอยู่กับลักษณะเฉพาะของโดเมนและความซับซ้อนของซอฟต์แวร์นั้นๆ และเมื่อใช้ข้อมูลที่สะสมมาในอดีตเข้าร่วมพิจารณาด้วย จะสามารถใช้ FP เพื่อ

1) ประมาณการต้นทุนที่จำเป็นต้องใช้ในการออกแบบ การพัฒนาและการทดสอบ
2) คาดการจำนวนของข้อผิดพลาดที่อาจเกิดขึ้นระหว่างการทดสอบ
3) ประมาณการจำนวนของชิ้นส่วนซอฟต์แวร์หรือจำนวนบรรทัดของต้นรหัสในระบบที่พัฒนาขึ้น

ค่า Function Point

สามารถหาได้จากการนับค่าโดเมนสารสนเทศและการวัดเชิงคุณภาพของความซับซ้อนของซอฟต์แวร์ โดยค่าโดเมนสารสนเทศนั้นมีคำจำกัดความดังต่อไปนี้

1. จำนวนของข้อมูลรับเข้าจากภายนอก (EI) โดยข้อมูลรับเข้าจากภายนอกนั้น หมายถึงข้อมูลหรือการควบคุมที่เกิดจากผู้ใช้หรือจากโปรแกรมประยุกต์อื่นๆ โดยการรับเข้าดังกล่าวมักจะทำให้เกิดการเปลี่ยนแปลงต่อแฟ้มภายในเชิงตรรกะ (ILF)
   การรับเข้าที่ใช้นับในค่าโดเมนสารสนเทศนี้จะพิจารณาเป็นค่าคนละประเภทกับการสอบถาม (EQ)
2. จำนวนของข้อมูลนำสู่ภายนอก (EO) โดยข้อมูลนำออกสู่ภายนอก หมายถึงสิ่งที่อยู่ภายในโปรแกรมประยุกต์ที่ให้ข้อมูลแก่ผู้ใช้ ในบริบทนี้ข้อมูลนำออกสู่ภายนอกจะเป็นหน้าจอ รายงาน หรือข้อความแสดงความผิดพลาด เป็นต้น
3. จำนวนของการสอบถามภายนอก (EQ) เป็นข้อมูลนำเข้าชนิดออนไลน์ที่ส่งผลลัพธ์ออกมาทันทีด้วยการตอบสนองของตัวซอฟต์แวร์ โดยการตอบกลับดังกล่าว จะอยู่ในรูปของข้อมูลนำออกชนิดออนไลน์ และมักจะเป็นการดึงข้อมูลมาจาก ILF
4. จำนวนของแฟ้มภายในเชิงตรรกะ (ILF) โดยแฟ้มภายในเชิงตรรกะ คือ กลุ่มของข้อมูลที่อยู่ภายในระบบงาน และข้อมูลดังกล่าวปรับปรุงผ่านการรับเข้าข้อมูลจากภายนอก
5. จำนวนแฟ้มต่อประสานภายนอก (EIF) โดยแฟ้มต่อประสานภายนอก คือ กลุ่มของข้อมูลที่อยู่ด้านนอกโปรแกรมประยุกต์ แต่ทำหน้าที่ให้ข้อมูลที่สามารถใช้ได้โดยตัวโปรแกรมดังกล่าว

เมื่อทำการรวบรวมข้อมูลตามค่าโดเมนสารสนเทศเรียบร้อยแล้ว ก็จะนำค่าไปใส่ไว้ในตาราง

จำนวนนับ		ตัวคูณความซับซ้อน			
		น้อย	ปานกลาง	มาก	
EI	X	3	4	6	=
EO	X	4	5	7	=
EQ	X	3	4	6	=
ILF	X	7	10	15	=
EIF	X	5	7	10	=

จำนวนรวม (ct) .....

และเราสามารถหาค่า FP ได้จากสมการต่อไปนี้

$$FP = ct \times \left[0.65 + 0.01 \times \sum (vaf_i) \right]$$

โดยค่า vaf_i เรียกว่าค่าตัวคูณปรับค่า (Value Adjustment Factor) มีจำนวณ 14 ค่า (vaf_1 ถึง vaf_{14}) ขึ้นกับข้อมูลที่เก็บรวบรวมจากคำถาม ซึ่งค่า vaf_i มีค่าที่เป็นไปได้ตั้งแต่ 0 ถึง 5 โดย 0 คือไม่สำคัญ และ 5 คือสำคัญมาก

วิธีการนับค่า EI, EO, EQ, ILF และ EIF มักจะนำมาจากแผนผังการไหลของข้อมูล (Data Flow Diagram) ซึ่งเป็นวิธีการวิเคราะห์และออกแบบระบบชนิดหนึ่ง ซึ่งมองการเปลี่ยนแปลงของข้อมูลในระบบเป็นหลัก

ตัวอย่าง

ระบบหนึ่งมีจำนวน EI = 3 ซึ่งมีกลไกความซับซ้อนน้อย มี EO = 2 ซึ่งมีความซับซ้อนปานกลาง มี EQ = 2 ซึ่งมีความซับซ้อนน้อย มี ILF = 1 ซึ่งมีความซับซ้อนปานกลาง และ EIF = 4 ซึ่งมีความซับซ้อนน้อย โดยกำหนดให้ตัวคูณปรับค่ามีค่า = 45 จงคำนวณหาค่า FP

EI	3	X	3	=	9
EO	2	X	5	=	10
EQ	2	X	3	=	6
ILF	1	X	10	=	10
EIF	4	X	5	=	20

Ct = 55

FP = 55X(0.65+(0.01X45)) = 60.5 ≈ 61

เมื่อได้ค่า FP จากการประมาณค่าจากการออกแบบ เช่น จาก Data Flow Diagram แล้วก็จะสามารถประเมินขนาดของโครงการได้โดยใช้ข้อมูลย้อนหลัง เช่น ในทีมพัฒนานี้ 1 FP แปลงเป็น 60 บรรทัดของต้นรหัสและ 1 คน-เดือน สามารถพัฒนาโปรแกรมได้ขนาด 12 FP จะทำให้ประเมินได้ดังต่อไปนี้

ขนาดโครงการ	61	FP	= 61X60	=	3,660 บรรทัด
effort	61	FP	= 61/12	=	5.08 คน-เดือน

นั่นคือนักพัฒนา 1 คน จะใช้เวลาประมาณ 5 เดือนเศษในการพัฒนาให้ซอฟต์แวร์นี้สำเร็จ

สำหรับคำถามเพื่อกำหนดค่าตัวคูณปรับค่าทั้ง 14 ตัวมีดังต่อไปนี้

1. ระบบต้องการการสำรองข้อมูลและการกู้คืนข้อมูลที่น่าเชื่อถือหรือไม่
2. มีความต้องการกระบวนการสื่อสารเฉพาะสำหรับถ่ายโอนข้อมูลกับตัวระบบงานหรือไม่
3. การประมวลผลเป็นแบบกระจายหรือไม่
4. สมรรถนะเป็นสิ่งสำคัญหรือไม่
5. ระบบจะต้องทำงานกับสภาวะการทำงานที่มีอยู่ ซึ่งถูกใช้งานอย่างหนักหรือไม่
6. ระบบต้องการการรับค่าแบบออนไลน์หรือไม่
7. ระบบรับค่าแบบออนไลน์ต้องการทรานแซคชั่น (transaction)
   การป้อนข้อมูลที่สร้างเป็นหน้าจอหลายหน้าจอหรือไม่
8. สามารถปรับปรุงข้อมูลแบบออนไลน์ไปยัง ILFได้หรือไม่
9. ข้อมูลรับเข้า ข้อมูลส่งออก ระบบแฟ้ม และการสอบถามซับซ้อนหรือไม่
10. กระบวนการประมวลผลภายในซับซ้อนหรือไม่
11. การออกแบบต้นรหัสสามารถใช้ซ้ำได้หรือไม่
12. การแปลงและการติดตั้งรวมอยู่ในการออกแบบหรือไม่
13. ระบบถูกออกแบบให้สามารถติดตั้งได้หลายครั้งในแต่ละองค์กรหรือบริษัทที่ต่างกันหรือไม่
14. ระบบออกแบบมาเพื่อรองรับความเปลี่ยนแปลงหรือให้ผู้ใช้สามารถใช้งานได้สะดวกหรือไม่

ความสัมพันธ์ระหว่างจำนวนบรรทัดและ FP

ความสัมพันธ์ระหว่างจำนวนบรรทัดและ FP นั้นขึ้นอยู่กับภาษาโปรแกรมที่ใช้ในการพัฒนาซอฟต์แวร์ และยังขึ้นกับคุณภาพของการออกแบบเช่นกัน อย่างไรก็ตามจาการศึกษาของกลุ่ม Quantitative Software Management ได้ ข้อสรุปสำหรับประมาณการจำนวนบรรทัดที่จำเป็นในการสร้างการทำงานให้ได้ 1 FP ในหลายๆภาษาดังต่อไปนี้ (ข้อมูลปี ค.ศ. 2009)

ภาษาโปรแกรม	จำนวนบรรทัดของต้นรหัส			
	ค่าเฉลี่ย	ค่ากลาง	ต่ำสุด	สูงสุด
C	148	107	22	704
C++	59	53	20	178
C#	58	59	51	66
Java	55	53	9	214
Perl	57	57	45	60
.NET	60	60	60	60
J2EE	57	50	50	67

FP และจำนวนบรรทัดมีความแม่นยำพอที่จะใช้ในการประเมินราคาและแรงงาน ที่จะใช้ในการพัฒนาซอฟต์แวร์ แต่ก็จำเป็นอย่างยิ่งที่ต้องใช้ข้อมูลย้อนหลังเป็นแนวทางในการประมาณค่า

3. ตัววัดเชิงวัตถุ (Object-Oriented Metrices)

แม้ว่าวิธีการดั้งเดิม เช่น FP หรือ LOC จะสามารถช่วยในการวัดขนาดของโครงการที่พัฒนาด้วยเทคนิคเชิงวัตถุได้ แต่ก็ไม่สามารถวัดลงไปในระดับรายละเอียดในโครงการประเภทนี้ได้ จึงจำเป็นต้องมีวิธีการวัดเฉพาะสำหรับโครงการประเภทเชิงวัตถุดังต่อไปนี้

1. จำนวนการบรรยายสถานการณ์ ซึ่งเทียบเท่ากับการอธิบาย Use Case ในรูปแบบของลำดับที่บอกการตอบโต้ระหว่างผู้ใช้กับระบบ แต่ละการบรรยายจะประกอบไปด้วย 3 องค์ประกอบ คือ ผู้กระทำ, การกระทำ, ผู้เกี่ยวข้อง จำนวนการบรรยายสถานการณ์จะสัมพันธ์โดยตรงกับขนาดของระบบงาน
2. จำนวนคลาสสำคัญ คลาสสำคัญ หมายถึง องค์ประกอบที่เป็นอิสระสูงที่อยู่ในระบบงาน มักเป็นศูนย์กลางของโดเมนที่เราสร้างระบบงานขึ้นมาเพื่อแก้ปัญหา ทำให้จำนวนคลาสสำคัญสะท้อนไปยังแรงงาน ที่ต้องใช้เพื่อพัฒนาระบบ รวมทั้งสะท้อนต่อไปยังความเป็นไปได้ในการใช้คลาสซ้ำระหว่างการพัฒนา ในบางเฟรมเวิร์คอาจเรียกคลาสกลุ่มนี้ว่า โดเมนคลาส หรือโมเดล
3. จำนวนคลาสสนับสนุน คลาสสนับสนุนเป็นคลาสที่ไม่ได้เกี่ยวข้องโดยตรงกับโดเมน ยกตัวอย่าง เช่น คลาสที่เกี่ยวข้องกับ GUI, คลาสที่ใช้เข้าถึงข้อมูล คลาสสนับสนุนอาจถูกพัฒนาขึ้นเพื่อสนับสนุนคลาสสำคัญแต่ละตัว เช่น คลาสควบคุม (Controller) มักจะสร้างเพื่อสนับสนุนคลาสสำคัญในการนำข้อมูลออกมาแสดงผล เป็นต้น ซึ่งอาจเป็นไปได้ว่าจะมีคลาสแสดงผลต่อเชื่อมจากคลาสควบคุมอีกทอดหนึ่ง
4. จำนวนเฉลี่ยของคลาสสนับสนุนต่อคลาสสำคัญ โดยทั่วไปจะมีการพัฒนาคลาสสำคัญขึ้นมาก่อนแล้วจึงสร้างคลาสสนับสนุนให้คลาสสำคัญแต่ละคลาส ซึ่งคลาสสำคัญ 1 คลาสอาจจะมีคลาสสนับสนุนมากกว่า 1 คลาสในระบบงานที่มี GUI มักจะมีคลาสสนับสนุน 2-3 คลาสต่อคลาสสำคัญ 1 คลาส และในระบบงานที่ไม่มี GUI จำนวนคลาสสนับสนุนมักจะเป็น 2 ต่อ 1
5. จำนวนระบบย่อย ระบบย่อยเป็นกลุ่มของคลาสที่สนับสนุนการทำงาน ฟังก์ชั่นหนึ่งๆที่ผู้ใช้ระบบใช้งานได้ ในระบบงานเชิงวัตถุมักจะแยกระบบใหญ่ให้ออกเป็นระบบย่อยเพื่อให้สามารถกำหนดปริมาณงานในการพัฒนาได้

4. ตัววัดเชิง use case (Use-Case-Oriented Metrices)

Use Case เป็นเทคนิคที่ใช้กันอย่างแพร่หลายสำหรับอธิบายความต้องการของผู้ใช้ในระดับที่ใกล้กับผู้ใช้หรือโดเมนธุรกิจโดย Use Case จะสะท้อนไปเป็นคุณสมบัติ (Feature) หรือฟังก์ชั่นการทำงานของระบบในระยะถัดไป การใช้ Use Case เป็นตัววัดจึงสมเหตุสมผลเช่นเดียวกันการใช้ LOC (Line of Code) หรือ FP และที่สำคัญ Use Case นั้นจะถูกกำหนดในช่วงแรกของกระบวนการพัฒนา ทำให้สามารถใช้ Use Case เป็นตัวประมาณค่าของส่วนอื่นๆของระบบได้ ประเด็นสำคัญอื่นๆ คือ Use Case ไม่ขึ้นกับภาษาโปรแกรม และจำนวน Use Case เป็นสัดส่วนโดยตรงกับจำนวนบรรทัดของโปรแกรมที่สร้างขึ้น อย่างไรก็ตาม Use Case ไม่มีมาตรฐานที่ชัดเจน เนื่องจากสามารถเขียนได้ด้วยระดับความละเอียดที่ต่างกันทำให้เป็นข้อด้อยในการใช้งานหากไม่มีการกำหนดมาตรฐานในการเขียน การคำนวณขนาดด้วย Use Case ที่ได้รับการยอมรับในช่วงที่ผ่านมา คือ หน่วยวัดที่เรียกว่า Use Case Points (UCP) ซึ่งเป็นการวัดขนาดของระบบงานโดยใช้ Use

Case และมีพื้นฐานมาจากการวิเคราะห์ FP และ Constructive Cost Model โดย UCP จะเริ่มด้วยการประมาณค่า Actor

การประมาณค่า Actor จะเป็นการนับจำนวน Actor

ที่มีในแบบจำลองจากนั้นจะทำการให้น้ำหนักตามตารางต่อไปนี้

ชนิดของ Actor	Wa น้ำหนัก
Actor ในรูปการต่อประสานกับชุดคำสั่ง	1
Actor ในรูปการต่อประสานผ่านการโต้ตอบหรือโปรโตคอล	2
Actor ในรูปการต่อประสานกับผู้ใช้แบบกราฟฟิก	3

จากตารางจะเห็นได้ว่า การต่อประสานกับ Actor ในรูปแบบต่างๆจะมีน้ำหนักไม่เท่ากัน โดยจะให้น้ำหนักของ Actor ที่เป็นบุคคลจริงมากที่สุด (3) ระบบภายนอกรองลงมา (2) และระบบภายในน้อยที่สุด (1)

อย่างไรก็ตามมีความจำเป็นที่จะต้องพึ่งข้อมูลย้อนหลังเพื่อปรับค่าตัวคูณน้ำหนักตามข้อมูลที่เก็บรวบรวมไว้

การให้น้ำหนัก Use Case สำหรับแต่ละ Use Case จะมีการประเมินว่ามีระดับความซับซ้อนน้อย, ปานกลาง หรือมาก โดยนับจำนวนทรานแซกชั่นที่เกิดขึ้นใน Use Case ทั้งในกรณีปกติ (Basic Flow) และกรณียกเว้น (Exceptional Flow) โดยการให้ น้ำหนัก Use Case ตามจำนวนทรานแซคชั่นตามตารางต่อไปนี้

ชนิดของ Use Case	Wu ตัวคูณน้ำหนัก
มีน้อยกว่า 3 ทรานแซคชั่น	5
มี 4 ถึง 7 ทรานแซคชั่น	10
มีมากกว่า 7 ทรานแซคชั่น	15

เมื่อได้ค่า UCP จาก Actor และจาก Use Case แล้วก็จะได้ค่ารวมกันเรียกว่า ยูสเคสพ้อยต์ที่ยังไม่ได้ปรับค่า (UUCP)

$$UUCP = \sum(W_a \bullet A_i) + \sum(W_u \bullet UC_i)$$

UUCP = Weighted Actor + Weighted Use Cases

การถ่วงน้ำหนักด้วยตัวคูณเชิงเทคนิค ตัวคูณความซับซ้อนเชิงเทคนิค (Technical Complexity Factor – TCF) เป็นค่าที่ใช้เพื่อปรับค่า UUCP ตามความซับซ้อนเชิงเทคนิคของโครงการ โดยจะมีลักษณะของค่าอยู่ในช่วง 0-5 (เช่นเดียวกับ VAF ของ FP) ข้อที่แตกต่างกับ VAF คือ ค่า TCF จะมีการกำหนดน้ำหนักให้แต่ละคำถาม สำหรับคำถามเพื่อให้ได้มาซึ่งค่า TCF มีทั้งหมด 13 ข้อ ดังต่อไปนี้

T1. ระบบต้องเป็นแบบกระจายหรือไม่ น้ำหนัก (W_t) = 2

T2. ระบบต้องมีสมรรถนะตามกำหนดหรือไม่ น้ำหนัก (W_t) = 1

T3. ระบบต้องมีประสิทธิภาพเชิงการใช้งานหรือไม่ น้ำหนัก (W_t) = 1

T4. การประมวลผลภายในซับซ้อนหรือไม่ น้ำหนัก (W_t) = 1

T5. ต้นรหัสต้องสามารถใช้ซ้ำได้หรือไม่	น้ำหนัก (W_t) = 1
T6. การติดตั้งสามารถทำได้ง่ายหรือไม่	น้ำหนัก (W_t) = 0.5
T7. การใช้งานง่ายหรือไม่	น้ำหนัก (W_t) = 0.5
T8. สามารถย้ายการทำงานข้ามแพล็ตฟอร์มได้หรือไม่	น้ำหนัก (W_t) = 2
T9. ง่ายต่อการเปลี่ยนแปลงหรือไม่	น้ำหนัก (W_t) = 1
T10. อนุญาตให้ใช้พร้อมกันหลายผู้ใช้หรือไม่	น้ำหนัก (W_t) = 1
T11. มีฟีเจอร์ด้านความปลอดภัยเป็นพิเศษหรือไม่	น้ำหนัก (W_t) = 1
T12. อนุญาตให้เข้าถึงได้จากบุคคลอื่นหรือไม่	น้ำหนัก (W_t) = 1
T13. จำเป็นต้องมีการฝึกการใช้งานเป็นพิเศษหรือไม่	น้ำหนัก (W_t) = 1

ค่า TFactor คือ ค่าผลรวมของ TCF ที่ถ่วงน้ำหนักแล้ว

$$TFactor = \sum(W_t \bullet T_i)$$

และ

TCF = 0.6+ (0.01 X TFactor)

การถ่วงน้ำหนักด้วยตัวคูณแวดล้อม ตัวคูณความซับซ้อนแวดล้อม (Environmental Complexity Factor – ECF) เป็นการประเมินประสบการณ์ของทีมพัฒนาต่อสภาวะแวดล้อมที่ใช้พัฒนาระบบงาน โดยมีคำถามและน้ำหนัก ดังนี้

	W_e
E1. ทีมพัฒนาคุ้นเคยกับ UML หรือไม่	1.5
E2. ทีมพัฒนาเป็นแบบ Part time หรือไม่	-1
E3. ทีมพัฒนามีความสามารถในการวิเคราะห์หรือไม่	0.5
E4. ทีมมีประสบการณ์ทางโปรแกรมประยุกต์หรือไม่	0.5
E5. ทีมมีประสบการณ์เชิงวัตถุหรือไม่	1
E6. ทีมมีความกระตือรือล้นหรือไม่	1
E7. ภาษาโปรแกรมที่ใช้ยากหรือไม่	-1
E8. ความต้องการเชิงซอฟต์แวร์แน่นอนหรือไม่	2

โดยใช้ลักษณะการตอบคำถามเป็นค่า 0-5 เช่นเดียวกับ TCF ซึ่งค่า ECF สามารถคำนวณได้จาก

$$ECF = 1.40 - 0.03 \times \sum(W_e \bullet E_i)$$

เมื่อคำนวณค่า UUCP, TCF และ ECF แล้วจะสามารถนำไปคำนวณค่า UCP ได้ ดังต่อไปนี้

UCP = UUCP X TCF X ECF

เมื่อได้ค่า UCP แล้วจึงนำไปประเมินว่าเวลาที่ใช้ในการพัฒนาเป็นเท่าไร โดยต้องใช้ค่า Productivity factor (PF) คืออัตราคน-ชั่วโมงของการพัฒนาต่อ 1 หน่วย UCP ซึ่งจำเป็นต้องได้จากข้อมูลย้อนหลัง ถ้าไม่มีข้อมูลย้อนหลัง สามารถทำได้ดังต่อไปนี้

1. คำนวณค่า UCP จากโครงการใดๆที่ผ่านมา
2. ใช้ค่า PF ในช่วง 15-30 ถ้าเป็นทีมใหม่อาจใช้ค่า 20 สำหรับโครงการแรก เมื่อโครงการเสร็จแล้วสามารถหาค่า PF ได้จาก (เวลาที่ใช้จริง)/UCP เพื่อใช้อ้างอิงต่อไป

ตัวอย่างการคำนวณค่า UCP

1. ระบบงานระบบหนึ่งมีผู้ใช้งานผ่าน User Interface ประกอบไปด้วย 3 Use Case โดย Use Case แรก มีความซับซ้อนปานกลาง อีก 2 Use Case มีความซับซ้อนน้อย จงคำนวณหาค่า UUCP

$$UUCP = \sum(W_a \bullet A_i) + \sum(W_u \bullet UC_i)$$

= (3X1) + [(10X1)+(5X2)]

= 4 + 20

= 24

2. จงคำนวณค่า TCF จากข้อมูลต่อไปนี้

T1. ระบบนี้ต้องการการประมวลผลแบบกระจาย	2 X 5 = 10
T2. ระบบนี้ต้องมีการตอบสนองกับผู้ใช้ภายใน 4 วินาที แม้ว่าผู้ใช้มีจำนวนมากกว่า 100,000 คนก็ตาม	1 X 5 = 5
T3. ระบบต้องมีประสิทธิภาพสำหรับผู้ใช้ทั่วไปในเกณฑ์ดี	1 X 2 = 2
T4. ระบบภายในมีความซับซ้อนในการประมวลผลข้อมูลมาก	1 X 5 = 5
T5. ต้นรหัสไม่จำเป็นต้องใช้ซ้ำได้	1 X 1 = 1
T6. การติดตั้งไม่จำเป็นต้องง่าย	0.5 X 1 = 0.5
T7. การใช้งานต้องง่าย	0.5 X 5 = 2.5
T8. ระบบไม่จำเป็นต้องย้ายการทำงานข้ามแพล็ตฟอร์มได้	2 X 1 = 2
T9. ระบบไม่จำเป็นต้องง่ายต่อการเปลี่ยนแปลง	1 X 1 = 1
T10. ระบบจำเป็นต้องรองรับผู้ใช้จำนวนมหาศาลได้พร้อมๆกัน	1 X 5 = 5
T11. ระบบมีคุณสมบัติทางด้านความปลอดภัยแบบมาตรฐาน	1 X 3 = 3
T12. บุคคลอื่นสามารถเข้าใช้บางส่วนของระบบได้แบบสาธารณะ	1 X 4 = 4
T13. ไม่จำเป็นต้องมีการฝึกเพื่อใช้งานระบบ	1 X 1 = 1

$$TFactor = \sum(W_t \bullet T_i)$$

TFactor = 42

TCF = 0.6+ (0.01 X TFactor)

TCF = 0.6 + (0.01 X 42)

= 1.02

3. จงคำนวณหาค่า ECF จากข้อมูลต่อไปนี้

E1. ทีมพัฒนาคุ้นเคยกับ UML เป็นอย่างดี	1.5 X 5 = 4.5
E2. ทีมพัฒนาทำงานแบบ Part time บ้าง	-1 X 3 = -3
E3. ทีมพัฒนามีความสามารถในการวิเคราะห์น้อย	0.5 X 2 = 1
E4. ทีมพัฒนามีประสบการณ์ด้านโปรแกรมประยุกต์ต่ำ	0.5 X 1 = 0.5
E5. ทีมพัฒนามีประสบการณ์ด้านเชิงวัตถุปานกลาง	1 X 3 = 3
E6. ทีมพัฒนามีความกระตือรือล้น	1 X 4 = 4

E7. ภาษาโปรแกรมที่ใช้ไม่ยาก -1 X 2 = -2

E8. ความต้องการเชิงซอฟต์แวร์ค่อนข้างคงที่ 2 X 4 = 8

$$ECF = 1.40 - 0.03 \times \sum (W_e \bullet E_i)$$

ECF = 1.40 – 0.03 X 16

= 1.12

4. จากค่า UUCP , TCF และ ECF จงคำนวณหาค่า UCP

UCP = UUCP X TCF X ECF

= 24 X 1.02 X1.12

= 27.41 ≈ 28

5. กำหนดให้ค่า PF = 20 จงคำนวณหาค่าแรงงานที่ใช้พัฒนาระบบนี้

แรงงาน คน-ชั่วโมง = UCP X PF

= 28 X 20

= 560 คน-ชั่วโมง

กำหนดให้เวลาพัฒนาโปรแกรมเป็น 30 ชั่วโมงต่อสัปดาห์

ดังนั้น เวลาที่ใช้พัฒนาระบบจะเท่ากับ 560/30 ≈ 18.67 คน-สัปดาห์

ทีมพัฒนามี 5 คน ดังนั้น ต้องใช้เวลาคนละ 18.67/5 = 3.73 สัปดาห์

กำหนดให้แต่ละคนเงินเดือน 14,000 บาท และทำงาน 4 สัปดาห์ต่อเดือน

ดังนั้น ค่าแรง = (3.73/4) X 5 X 14000 = 65,275 บาท

5. ตัววัดโครงการเว็บแอพพลิเคชั่น (WebApp Project Metrices)

จุดประสงค์ของเว็บแอพพลิเคชั่น คือ การรวมเอาเนื้อหา (Content) และฟังก์ชั่นเข้าด้วยกัน ทำให้การวัดบางประเภทไม่สามารถใช้ได้กับโครงการที่เป็นเว็บแอพพลิเคชั่น

ประเด็นที่สำคัญในการวัดขนาดเว็บแอพพลิเคชั่น คือ

1. จำนวนหน้าเว็บปกติ

เป็นหน้าเว็บที่ทำการเปลี่ยนแปลงแก้ไขเนื้อหาโดยตรง มีการนำเสนอเนื้อหาโดยไม่ผ่านการประมวลผลก่อน การวัดค่านี้จะเป็นการสะท้อนขนาดของแอพพลิเคชั่นทั้งหมดที่จะพัฒนา

2. จำนวนหน้าเว็บชนิดไดนามิค

เป็นหน้าเว็บที่เปลี่ยนแปลงเนื้อหาจากการประมวลผล และเป็นส่วนสำคัญในระบบธุรกิจที่เป็นเว็บแอพพลิเคชั่น เช่น Search Engine, โปรแกรมทางการเงิน เป็นต้น หน้าเว็บประเภทนี้สะท้อนถึงความซับซ้อนและต้องการแรงงานในการพัฒนามากกว่าหน้าเว็บปกติ

3. จำนวนการเชื่อมโยงภายใน

การเชื่อมโยงภายในมีผลต่อการออกแบบและสร้างระบบนำทางของเว็บแอพพลิเคชั่น โดยเมื่อการเชื่อมโยงภายในซับซ้อนขึ้น ระบบโดยรวมก็จะซับซ้อนขึ้นเช่นกัน

4. จำนวนวัตถุข้อมูลที่มีการจัดเก็บ

วัตถุข้อมูลอาจเข้าถึงได้ผ่านเว็บแอพพลิเคชั่นและขนาดของข้อมูลที่ตัวเว็บแอพพลิเคชั่น มีผลโดยตรงกับความซับซ้อนของตัวระบบ

5. จำนวนระบบภายนอกที่ต่อประสานด้วย

เว็บแอพพลิเคชั่นมักมีการเชื่อมต่อกับระบบภายนอกในลักษณะใดลักษณะหนึ่งเพื่อดึงข้อมูลมาแสดงผล เช่น ระบบด้านหลังสำหรับป้อนข้อมูล หรือแหล่งข้อมูลภายนอก เป็นต้น

6. จำนวนของวัตถุเนื้อหา

วัตถุเนื้อหาอาจอยู่ในรูปของข้อความ, ภาพ, วีดีโอ, เสียง ซึ่งรวมไว้เป็นส่วนหนึ่งของเว็บแอพพลิเคชั่น โดยหน้าเว็บหนึ่งอาจจะประกอบไปด้วยวัตถุเนื้อหาได้หลายวัตถุ

7. จำนวนฟังก์ชั่นที่สามารถทำงานได้

ฟังก์ชั่นที่สามารถทำงานได้มักอยู่ในรูปของ script หรือ flash application หรือ applet ซึ่งให้บริการการประมวลผลบางอย่างแก่ผู้ใช้ เมื่อจำนวนฟังก์ชั่นเหล่านี้เพิ่มขึ้น แรงงานที่ต้องใช้ในการพัฒนาย่อมเพิ่มขึ้นเช่นกัน

การวัดแต่ละค่าสามารถนำมาใช้คำนวณตัววัดที่น่าสนใจได้ เช่น

ให้ N_{sp} เป็นจำนวนหน้าเว็บปกติ

N_{dp} เป็นจำนวนหน้าเว็บแบบไดนามิค

กำหนดดัชนีปรับแต่ง C ดังนี้

$$C=\frac{N_{dp}}{N_{dp}+N_{sp}}$$

นั่นคือค่า C จะอยู่ในช่วง 0 ถึง 1 โดยเมื่อ C มีค่ามาก จะหมายถึงความสามรถในการปรับแต่งจะกลายเป็นประเด็นเชิงเทคนิคที่ต้องพิจารณามากขึ้น

การประมาณโครงการซอฟต์แวร์

แบบจำลองการประมาณโครงการซอฟต์แวร์จะได้มาจากสูตร ซึ่งเกิดจากการทดลอง ประสบการณ์และข้อมูลในอดีตเพื่อประเมินแรงงานที่ต้องใช้ โดยมักจะเป็นฟังก์ชั่นของ LOC, FP หรือ UCP โดยค่า LOC, FP หรือ UCP จะมาจากการประเมินโดยใช้การวัดตามที่อธิบายไว้แล้วในส่วนก่อนหน้านี้

ประเด็นที่น่าสนใจคือ ข้อมูลย้อนหลังที่ใช้เพื่อสนับสนุนแบบจำลองการประมาณโครงการนั้นมักจะมาจากข้อมูลของโครงการเพียงจำนวนหนึ่ง จึงไม่สามารถใช้แบบจำลองเดียวในการประมาณซอฟต์แวร์ทุกๆแบบได้ และแบบจำลองควรถูกปรับให้เหมาะกับบริบทการพัฒนา โดยอาจจะใช้ข้อมูลจากโครงการที่ทำเสร็จแล้วขององค์กรนั้นๆ มาเป็นแนวทางในการเทียบค่า ปรับแต่งพารามิเตอร์ และทดสอบซ้ำก่อนใช้งานจริง

โครงสร้างของแบบจำลองการประมาณ

โดยทั่วไปแบบจำลองการประมาณจะใช้ข้อมูลการวิเคราะห์ย้อนหลังเป็นตัวแปรประมาณค่า ซึ่งแบบจำลองมีโครงสร้างดังต่อไปนี้

$$E=A+B\bullet(e_v)^C$$

โดย A, B และ C เป็นค่าคงที่ ซึ่งได้จากการทดลองหรือประสบการณ์และ E จะเป็นแรงงานที่ใช้ในหน่วย คน-เดือน โดย ev เป็นตัวแปรประมาณค่า อาจเป็นได้ทั้ง LOC, FP หรือ UCP โดยการประมาณค่าที่ใช้ LOC จะมีดังต่อไปนี้

$E=5.2\times(KLOC)^{0.01}$ แบบจำลองของ Walston - Felix

$E=5.5+0.73\times(KLOC)^{1.16}$ แบบจำลองของ Bailey – Basili

สำหรับแบบจำลองที่อิงค่า FP มีตัวอย่างดังต่อไปนี้

$E = -91.4 + 0.355FP$ แบบจำลองของ Albrecht และ Gaffney

$E = -37 + 0.96FP$ แบบจำลองของ Kemerer

สำหรับแบบจำลองที่อิง UCP ยังไม่มีการศึกษาในรายละเอียด อย่างไรก็ตามโดยทั่วไปจะมีการประมาณค่า E ในหน่วย คน-ชั่วโมง โดยใช้ค่า Productivity Factor (PF) เป็นตัวคูณ

$E_{(man\text{-}hour)} = PF \times UCP$

และมีการประมาณค่า PF ให้อยู่ในช่วง 15-30 หรือ 20-28 ขึ้นกับข้อมูลจากหน่วยงานที่พัฒนาซอฟต์แวร์

แบบจำลอง COCOMO II

แบบจำลอง Constructive Cost Model หรือ COCOMO เป็นชุดประมาณขนาดซอฟต์แวร์ มี 2 รุ่น รุ่นแรกเรียกว่า COCOMO และรุ่นถัดมาเรียกว่า COCOMO II ซึ่งมีความสมบูรณ์มากขึ้นโดย COCOMO II เป็นกลุ่มการประมาณค่าซอฟต์แวร์ ซึ่งพยายามที่จะแก้ปัญหา โดยใช้แบบจำลองต่อไปนี้

1. แบบจำลองการประกอบโปรแกรมประยุกต์ ใช้ในช่วงแรกของกระบวนการ ในขณะเตรียมต้นแบบของส่วนติดต่อผู้ใช้
2. แบบจำลองขณะออกแบบช่วงต้น ใช้เมื่อความต้องการเชิงซอฟต์แวร์เริ่มคงที่และตัวสถาปัตยกรรมของระบบถูกเตรียมขึ้นแล้ว
3. แบบจำลองหลังระยะสถาปัตยกรรม ใช้ในช่วงกำลังพัฒนาซอฟต์แวร์

เช่นเดียวกันกับแบบจำลองอื่นๆ COCOMO II ต้องการข้อมูลเชิงขนาดเพื่อใช้ในการประมาณค่า ข้อมูลเชิงขนาดที่สามารถใช้ใน COCOMO II ได้ คือ

1. Object Point
2. Function Point
3. SLOC (Source Line of Code - จำนวนบรรทัดของต้นรหัส)

การใช้ Object Point ในการประมาณโครงการตามแบบจำลองการประกอบโปรแกรมประยุกต์ทาง COCOMO II โดย object point จะเป็นค่าสำหรับวัดขนาดซอฟต์แวร์ทางอ้อมโดยนับจำนวนวัตถุต่อไปนี้

1. จำนวนหน้าจอ (ที่ติดต่อกับผู้ใช้)
2. จำนวนรายงาน
3. จำนวนชิ้นส่วนซอฟต์แวร์ที่จำเป็นต้องใช้สร้างระบบงาน โดยจะแยกความซับซ้อนออกเป็น 3 ระดับ ดังตารางต่อไปนี้

ชนิดวัตถุ	ตัวคูณความซับซ้อน		
	น้อย	ปานกลาง	มาก
หน้าจอ	1	2	3
รายงาน	2	5	8
ชิ้นส่วนซอฟต์แวร์	-	-	10

เมื่อได้ข้อมูลครบถ้วนแล้ว

จะสามารถนำมาพิจารณาความสามารถในการใช้ซ้ำว่าใช้ซ้ำได้เป็นจำนวนเปอร์เซ็นต์เท่าใด จะได้ค่า New Object Point (NOP) เพื่อใช้ในการคำนวณในลำดับถัดไป

NOP = (Object Point) X [(100 - % reuse)/100]

จากนั้นจะใช้ค่า NOP เพื่อหาอัตราประสิทธิผล (Productivity Rate)

$$PROD=\frac{NOP}{person\text{-}month}$$

เมื่อได้ค่า PROD ของทีมพัฒนาแล้วก็จะสามารถใช้ค่านี้ในการประมาณค่าซอฟต์แวร์โครงการถัดๆไปได้ โดย E หรือค่าประมาณแรงงานที่ใช้พัฒนาจะเท่ากับ

$$E=\frac{NOP}{PROD}$$

อย่างไรก็ตามการประมาณค่าตามตัวอย่างนี้เป็นเพียงการใช้แบบจำลองส่วนหนึ่งของ COCOMO II

บทที่ 7
ความต้องการเชิงซอฟต์แวร์
Software Requirements

เป้าหมายในการพัฒนาซอฟต์แวร์ คือ การเปลี่ยนรูป (Transform) ความต้องการเชิงซอฟต์แวร์จากลูกค้าให้กลายเป็นซอฟต์แวร์ที่สามารถทำงานได้และเกิดประโยชน์ตามที่ลูกค้าต้องการ กิจกรรมย่อยๆทั้งหมดที่อยู่ในกระบวนการพัฒนาซอฟต์แวร์ คือ วิธีการที่ช่วยเปลี่ยนรูปความต้องการเชิงซอฟต์แวร์ให้ค่อยๆกลายเป็นส่วนของซอฟต์แวร์ โดยใช้เทคนิคที่เหมาะสมเข้ามาช่วยระหว่างกระบวนการเปลี่ยนรูปดังกล่าว จะเห็นได้ว่า ความต้องการเชิงซอฟต์แวร์เป็นจุดเริ่มต้นของกระบวนการทั้งหมด ซึ่งความต้องการเชิงซอฟต์แวร์นั้นหมายถึงข้อมูลเชิงพรรณาที่อธิบายว่าซอฟต์แวร์ที่ลูกค้าต้องการ ทั้งในแง่ของตัวสินค้าหรือการบริการมีลักษณะเป็นอย่างไร โดยความต้องการเชิงซอฟต์แวร์สามารถแบ่งออกได้เป็น 2 ประเภท คือ ความต้องการเชิงฟังก์ชั่น (Functional Requirements) และความต้องการที่ไม่ใช่ฟังก์ชั่น (Non-Functional Requirements)

ความต้องการเชิงฟังก์ชั่น

ความต้องการเชิงฟังก์ชั่นเป็นสิ่งที่ใช้อธิบายว่าระบบที่จะสร้างขึ้นต้องทำอะไร โดยความต้องการเชิงฟังก์ชั่นจะสะท้อนออกมาเป็นความสามารถของระบบ เช่น คุณลักษณะ (Feature) ซึ่งสามารถใช้ Use Case หรือ User Story จับ (Capture) ความต้องการประเภทนี้ได้

ความต้องการที่ไม่ใช่ฟังก์ชั่น

ความต้องการที่ไม่ใช่ฟังก์ชั่นเป็นสิ่งที่อธิบายประกอบเพื่อบอกลักษณะบางอย่างเพิ่มเติมให้แก่ตัวระบบ ซึ่งอาจเกี่ยวข้องกับเกณฑ์สมรรถนะของระบบ ความปลอดภัยของระบบ เช่น

- ระบบต้องรองรับผู้ใช้ได้ 1,000 คนพร้อมกัน
- การเข้ารหัสที่ใช้ในระบบต้องเป็นแบบ RSA 1024-bit ขึ้นไป
- ระบบต้องติดตั้งทั้งบนระบบปฏิบัติการ Linux, BSD และ Windows ได้

เป็นต้น

ความต้องการประเภทนี้มักเป็นส่วนเพิ่มเติมในรูปแบบเอกสารหรือปนอยู่ในคำอธิบาย Use Case หรือเป็นส่วนขยายอยู่ใน User Story เป็นต้น

วิธีการเก็บรวบรวมความต้องการเชิงซอฟต์แวร์ (Requirements Elicitation) เป็นกระบวนการเชิงวิศวกรรมที่พัฒนาขึ้นเพื่อช่วยเก็บรวบรวมข้อมูล โดยใช้เทคนิคต่างๆดังนี้

1. การสัมภาษณ์ เป็นการสื่อสารตัวต่อตัวระหว่างผู้เก็บข้อมูลและลูกค้า
2. การใช้สถานการณ์ เป็นการกำหนดสถานการณ์การใช้ระบบขึ้นเพื่อเก็บข้อมูลความต้องการจากลูกค้า
3. การใช้ต้นแบบ เป็นการสร้างต้นแบบของซอฟต์แวร์ในที่นี้อาจจะเป็นต้นแบบชนิดสร้างแล้วทิ้งเพื่อใช้สื่อสารกับลูกค้า ก่อนการสร้างจริง อย่างไรก็ตามมีเครื่องมือหลายชนิดที่สามารถสร้างต้นแบบที่ใช้พัฒนาต่อเป็นซอฟต์แวร์จริงได้

4. การประชุม เป็นการสื่อสารระดับกลุ่มโดยเชิญลูกค้าและทีมพัฒนามาพูดคุยกัน เพื่อให้ได้ข้อตกลงร่วมกัน ซึ่งจะนำไปเป็นความต้องการเชิงซอฟต์แวร์ จุดที่น่าสังเกตคือ แม้ว่าจะมีการตกลงเห็นด้วยในทุกกรณีแล้วก็ตาม จะมีการเปลี่ยนแปลงความต้องการเกิดขึ้นภายหลังเสมอ
5. การสังเกตการณ์ เป็นการเข้าไปสังเกตวิธีการทำงานของลูกค้าโดยตรงเพื่อเก็บข้อมูล

การวิเคราะห์ความต้องการเชิงซอฟต์แวร์

การวิเคราะห์ความต้องการเชิงซอฟต์แวร์เป็นกระบวนการที่นำเอาข้อมูล ซึ่งรวบรวมได้มาทำการวิเคราะห์และสร้างแบบจำลองความต้องการ บางครั้งเรียกกระบวนการนี้ว่าการวิเคราะห์ระบบ โดยเมื่อผ่านการวิเคราะห์และสร้างแบบจำลองแล้วจะได้ผลลัพธ์เป็นแบบจำลองในลักษณะดังต่อไปนี้

1. แบบจำลองเชิงสถานการณ์ เป็นแบบจำลองตามมุมมองของแต่ละ Actor ที่เกี่ยวข้องกับสถานการณ์นั้น
2. แบบจำลองเชิงข้อมูล เป็นแบบจำลองที่สะท้อนโดเมนข้อมูลที่วิเคราะห์ได้จากความต้องการเชิงซอฟต์แวร์
3. แบบจำลองเชิงคลาส เป็นแบบจำลองที่สะท้อนความต้องการเชิงซอฟต์แวร์ให้อยู่ในรูปของคลาสจากวิธีการเชิงวัตถุ โดยพิจารณาว่าแต่ละคลาสจะทำงานร่วมกัน เพื่อตอบสนองต่อความต้องการเชิงซอฟต์แวร์นั้นๆ
4. แบบจำลองเชิงการไหล เป็นการจำลองการไหลของข้อมูลผ่านฟังก์ชั่นของระบบและพิจารณาการเปลี่ยนรูปของข้อมูลนั้นๆ เมื่อผ่านฟังก์ชั่นแต่ละตัว
5. แบบจำลองเชิงพฤติกรรม เป็นแบบจำลองที่สะท้อนว่าระบบมีพฤติกรรมอย่างไรเมื่อเหตุการณ์ภายนอกระบบเกิดขึ้น

ข้อกำหนดความต้องการเชิงซอฟต์แวร์ (Requirement Specification)

ข้อกำหนดความต้องการเชิงซอฟต์แวร์เป็นกระบวนการร่างเอกสารข้อกำหนดความต้องการเพื่อใช้ทวนสอบกับตัวระบบที่พัฒนาขึ้น ซึ่งเอกสารดังกล่าวอาจจะประกอบไปด้วย การบรรยายแผนผังของแบบจำลอง หรือแม้แต่สมการคณิตศาสตร์ที่สามารถใช้ช่วยในการทวนสอบระบบ โดยโครงสร้างของเอกสารข้อกำหนดสามารถเป็นได้ดังต่อไปนี้

สารบัญ

ประวัติการแก้ไข (เพื่อบอกว่าเอกสารนี้แก้ไขเป็นครั้งที่เท่าใด/โดยใคร)

1. บทนำ
    1.1 วัตถุประสงค์
    1.2 ข้อตกลงที่ใช้ในเอกสาร
    1.3 กลุ่มเป้าหมายและเอกสารอ่านเพิ่มเติม
    1.4 ขอบเขตของโครงการ
    1.5 เอกสารอ้างอิง
2. อธิบายภาพรวม
    2.1 มุมมองของสินค้า
    2.2 คุณลักษณะของสินค้า

2.3 กลุ่มและลักษณะเฉพาะของผู้ใช้

2.4 สภาพแวดล้อมปฏิบัติการ

2.5 ข้อจำกัดในการออกแบบและพัฒนา

2.6 เอกสารสำหรับผู้ใช้

2.7 สมมติฐานและการขึ้นต่อกัน

3. คุณลักษณะของระบบ

3.1 คุณลักษณะที่ 1

3.2 คุณลักษณะที่ 2 (และถัดไป)

...

4. ความต้องการในการต่อประสานภายนอก

4.1 ส่วนติดต่อกับผู้ใช้

4.2 ส่วนต่อประสานกับอุปกรณ์

4.3 ส่วนต่อประสารกับซอฟต์แวร์

4.4 ส่วนต่อประสานกับการสื่อสาร

5. ความต้องการที่ไม่ใช่ฟังก์ชั่นอื่นๆ

5.1 ความต้องการเชิงสมรรถนะ

5.2 ความต้องการเชิงความปลอดภัย

5.3 ความต้องการเชิงความมั่นคง

5.4 ลักษณะเชิงคุณภาพของซอฟต์แวร์

6. ความต้องการอื่นๆ

ภาคผนวก ก. อภิธานศัพท์

ภาคผนวก ข. แบบจำลองการวิเคราะห์

ภาคผนวก ค. รายการประเด็น

การทวนสอบความต้องการ

การทวนสอบความต้องการเป็นกระบวนการในการตรวจสอบว่า แบบจำลองความต้องการที่สร้างขึ้นนั้นมีความไม่ถูกต้อง ไม่สมบูรณ์ หรือมีความกำกวมที่ใดบ้าง โดยกลไกหลักของการทวนสอบจะเรียกว่า การรีวิวเชิงเทคนิค โดยคณะผู้ทวนสอบจะประกอบไปด้วย วิศวกรซอฟต์แวร์ ลูกค้า และผู้ใช้ เพื่อทำการตรวจดูรายละเอียดของเอกสารข้อกำหนดว่าเป็นไปตามที่ต้องการหรือไม่ โดยดูเทียบเนื้อหาในเอกสาร รวมทั้งการตีความว่ามีส่วนใดที่ผิดพลาด หายไป ขัดแย้งกัน หรือเกินจริง เป็นต้น

หัวข้อที่จะตรวจสอบสามารถมีได้ดังต่อไปนี้

1. ความสมเหตุสมผล (Validity)
2. ความต้องกัน (Consistency) หมายถึง ไม่ขัดแย้งกัน
3. ความสมบูรณ์ (Completeness) หมายถึง มีครบถ้วน
4. ความเป็นไปได้ (Feasibility)
5. การพิสูจน์ได้ (Verifiability)

บทที่ 8
การออกแบบซอฟต์แวร์

Software Design

การออกแบบเป็นกลไกทางวิศวกรรมซอฟต์แวร์ที่อยู่ถัดจากการวิเคราะห์ความต้องการเชิงซอฟต์แวร์ การออกแบบจะทำการรับแบบจำลองระดับวิเคราะห์มาสร้างเป็นแบบจำลองระดับออกแบบ ซึ่งจะเป็นสิ่งที่วางรากฐานว่าจะสร้างโปรแกรมอย่างไร

แนวคิดการออกแบบ

แนวคิดพื้นฐานสำหรับการออกแบบซอฟต์แวร์นั้นมีการเปลี่ยนแปลงมาอย่างต่อเนื่องและมีแนวคิดใหม่ๆที่น่าสนใจเกิดขึ้นหลายแนวคิดที่ครอบคลุมทั้งการพัฒนาซอฟต์แวร์ทั่วไปและที่เกี่ยวข้องกับการพัฒนาซอฟต์แวร์เชิงวัตถุ

นามธรรม (Abstraction)

นามธรรมที่ใช้ในการออกแบบมีหลายระดับ ในระดับที่มีความเป็นนามธรรมสูง จะเป็นการอธิบายปัญหาด้วยคำกว้างๆ ในขณะที่ระดับที่มีความเป็นนามธรรมต่ำ จะเริ่มลงในรายละเอียดของวิธีการแก้ปัญหานั้นๆ ในการออกแบบทางซอฟต์แวร์จะพบคำว่า นามธรรมเชิงกระบวนการ (Procedural Abstraction) ซึ่งหมายถึงลำดับการทำงานหนึ่งๆและนามธรรมเชิงข้อมูล (Data Abstraction) ซึ่งหมายถึง สิ่งที่ใช้อธิบายวัตถุข้อมูล ตัวอย่างเช่น ปรุงอาหาร สามารถแบ่งออกเป็น นามธรรมเชิงกระบวนการ คือ “ปรุง” และนามธรรมเชิงข้อมูล คือ “อาหาร” ได้ โดยสามารถอธิบาย “ปรุง” เป็นขั้นตอนย่อยๆประกอบกันเพื่อให้ได้ตามที่ต้องการและสามารถอธิบาย “อาหาร” ในลักษณะว่า คืออะไร ประกอบด้วยอะไรบ้าง ซึ่งสิ่งเหล่านี้เป็นลักษณะของข้อมูลที่ใช้อธิบายอาหาร

สถาปัตยกรรม (Architecture)

สถาปัตยกรรมเป็นโครงสร้างและการจัดเรียงชิ้นส่วนของโปรแกรม โดยสนใจว่า แต่ละชิ้นส่วนของโปรแกรมนั้นติดต่อกันอย่างไร รวมถึงโครงสร้างของข้อมูลเป็นอย่างไรและชิ้นส่วนเหล่านั้นนำข้อมูลไปใช้อย่างไร ซึ่งคำว่าชิ้นส่วนของโปรแกรมในบริบทของสถาปัตยกรรมนั้นอาจหมายถึง องค์ประกอบของระบบขนาดใหญ่ในองค์กรหรือระบบย่อยที่ต้องทำงานร่วมกัน โดยควรจะมีสมบัติต่อไปนี้อยู่ในการออกแบบเชิงสถาปัตยกรรม

1. สมบัติเชิงโครงสร้าง

   สมบัติเชิงโครงสร้างเป็นการออกแบบเชิงสถาปัตยกรรมที่นิยามโครงสร้างของชิ้นส่วนในระบบว่าแต่ละชิ้นส่วนมีโครงสร้างอย่างไร และติดต่อไปยังชิ้นส่วนอื่นๆได้อย่างไร เช่น โปรเซสมีโครงสร้างที่เก็บข้อมูลบางอย่างไว้ภายในและจะทำงานได้ก็ต่อเมื่อมีเหตุการณ์จากภายนอกส่งเข้ามาเป็นข้อความที่โปรเซสนั้นเข้าใจ ซึ่งก็คือวิธีการติดต่อกันระหว่างโปรเซสในขณะที่วัตถุในภาษาโปรแกรมเชิงวัตถุมีโครงสร้างสำหรับเก็บทั้งข้อมูลและนิยามของเมธอดไว้ โดยการติดต่อกันระหว่างวัตถุก็คือการเรียกใช้เมธอดของวัตถุอื่น เป็นต้น

2. สมบัติเชิงฟังก์ชั่นพิเศษ

สมบัติเชิงฟังก์ชั่นพิเศษเป็นการออกแบบเชิงสถาปัตยกรรมที่อธิบายสถาปัตยกรรมที่สามารถช่วยแก้ปัญหาในเรื่องสมรรถนะ ความจุ ความเชื่อถือได้ หรือความมั่นคงของระบบ เป็นต้น มีตัวอย่างชัดเจนในอุตสาหกรรมหลายตัวอย่างที่สะท้อนให้เห็นว่าการตัดสินใจระดับสถาปัตยกรรมมีผลต่อสมบัติเชิงฟังก์ชั่นพิเศษ เช่น Facebook เริ่มต้นด้วยการเลือกใช้ PHP ในการทำเว็บ ต่อมาประสิทธิภาพของระบบไม่ดีอย่างที่ควรจะเป็น แม้ว่าจะทำการปรับแล้วก็ตาม สิ่งที่ Facebook ตัดสินใจทำก็คือ การสร้างคอมไพเลอร์เพื่อแปลง PHP ให้เป็น C++ และยอมเสียความสามารถบางอย่างของ PHP ไป เพื่อให้ได้ประสิทธิภาพที่สูงขึ้นบนฮาร์ดแวร์ชุดเดิม

3. กลุ่มของระบบที่เกี่ยวข้องกัน

การออกแบบเชิงสถาปัตยกรรมควรเป็นรูปแบบซ้ำๆที่สามารถพบได้ในกลุ่มของระบบที่เกี่ยวข้องหรือคล้ายกัน นั่นคือ การออกแบบเชิงสถาปัตกรรมที่ดีควรใช้ซ้ำได้ และเป็นองค์ประกอบให้กับระบบอื่นๆที่สร้างขึ้นใหม่ได้

รูปแบบ (Patterns)

รูปแบบหรือรูปแบบการออกแบบ (Design Pattern) เป็นสิ่งที่ได้รับการยอมรับแล้วว่าเป็นวิธีการแก้ปัญหาในบริบทหนึ่งๆ ซึ่งนำมาปรับใช้ได้ทันที รูปแบบที่พบมากที่สุดในกลุ่มการออกแบบส่วนติดต่อกับผู้ใช้หรือการเขียนโปรแกรมเชิงวัตถุ โดยแต่ละรูปแบบจะมีคำอธิบายเพื่อช่วยให้นักออกแบบเลือกว่าเหมาะสมกับงานที่ทำอยู่หรือไม่ และรูปแบบนี้สามารถใช้ซ้ำได้อย่างไร รวมทั้งบางรูปแบบอาจจะสามารถเป็นแนวทางในการสร้างรูปแบบอื่นๆที่มีความสามารถหรือโครงสร้างที่คล้ายกันได้

การแยกความเกี่ยวพัน (Separation of Concerns)

การแยกความเกี่ยวพันเชิงซอฟต์แวร์เป็นวิธีการที่แยกปัญหาทางซอฟต์แวร์ออกเป็นส่วนที่เล็กลงในระดับที่จัดการได้ง่ายขึ้น โดยนิยามของความเกี่ยวพันคือ คุณสมบัติ (Feature) หรือพฤติกรรม (Behavior) ที่เป็นส่วนหนึ่งของแบบจำลองความต้องการของซอฟต์แวร์ การแยกความเกี่ยวพันที่ดีจะทำให้ใช้เวลาและความพยายามน้อยลงในการแก้ปัญหา นั่นคือความซับซ้อนของซอฟต์แวร์โดยรวมจะลดลง

สภาพโมดูล (Modularity)

เป็นแนวคิดเชิงสถาปัตยกรรมหลักที่เกี่ยวข้องกับการแยกความเกี่ยวพัน โดยเป็นแนวคิดที่ทำการแบ่งซอฟต์แวร์ออกเป็นชิ้นส่วนและแต่ละชิ้นส่วนจะมีชื่อกำกับเพื่อให้สามารถอ้างอิงได้ เช่น คลาสในภาษา Java โมดูลในภาษา Ruby เป็นต้น ซอฟต์แวร์ที่มีขนาดใหญ่แต่ไม่มีการแยกเป็นโมดูลย่อยเรียกว่าซอฟต์แวร์ประเภทโมโนลิธิค เป็นสิ่งที่ควรหลีกเลี่ยงในการพัฒนาเพราะจะทำให้ดูแลยาก ในขณะเดียวกันการแยกโมดูลจนมีขนาดเล็กและย่อยจนเกินไปจะทำให้ต้นทุนของการรวมแต่ละโมดูลเข้าด้วยกันสูงจนทำให้การดูแลยากไปในอีกลักษณะหนึ่ง ดังนั้นจำนวนโมดูลจะมีค่าอยู่ในช่วงหนึ่งที่เหมาะสมและทำให้ต้นทุนการดูแลรักษาอยู่ในช่วงที่ต่ำ

การซ่อนข้อมูล (Information Hiding)

ประโยชน์ของการแยกระบบออกเป็นโมดูลนั้น ส่วนหนึ่งทำเพื่อเพิ่มความสามารถในการจัดการระบบใหญ่ด้วยการแบ่งเป็นส่วนๆ และอีกส่วนหนึ่งเพื่อซ่อนข้อมูลที่ไม่จำเป็นของแต่ละโมดูลไม่ให้สามารถเข้าถึงได้จากโมดูลอื่นๆเพื่อลดผลข้างเคียง (Side Effect) ซึ่งอาจจะทำให้เกิดข้อผิดพลาดที่ไม่จำเป็นขึ้นได้ การซ่อนข้อมูลจะทำให้การติดต่อกันระหว่างโมดูลมีประสิทธิภาพมากขึ้นได้อีกทางหนึ่ง เนื่องจากจุดที่เชื่อมต่อกันระหว่างโมดูลนั้นจะมีเท่าที่จำเป็น และเมื่อโมดูลอื่นไม่เห็นรายละเอียดภายในของอีกโมดูล ทำให้เมื่อมีความจำเป็นที่ต้องการเปลี่ยนแปลงแก้ไขภายในโมดูลนั้นจะไม่เกิดผลกระทบระหว่างกัน หากวิธีการติดต่อระหว่างโมดูลไม่เปลี่ยนแปลง

การไม่ขึ้นต่อกันเชิงฟังก์ชั่น (Functional Independence)

การไม่ขึ้นต่อกันเชิงฟังก์ชั่นเป็นแนวคิดการออกแบบที่สำคัญที่ต่อยอดออกมาจากแนวคิดของการแยกความเกี่ยวพัน สภาพโมดูล และการซ่อนข้อมูล เพื่อทำให้แต่ละโมดูลของซอฟต์แวร์เน้นไปยังการมีการทำงานที่เป็นไปในทางเดียวกัน นั่นคือ การออกแบบควรให้แต่ละโมดูลเจาะจงรับผิดชอบส่วนใดส่วนหนึ่งของความต้องการเชิงซอฟต์แวร์ และมีส่วนต่อประสาน (Interface) ที่เรียบง่ายที่สุดจากมุมมองของโมดูลอื่น เมื่อสามารถทำให้แต่ละโมดูลไม่ขึ้นต่อกันแล้วจะทำให้แต่ละโมดูลง่ายต่อการดูแลรักษา ง่ายต่อการทดสอบ ลดการแพร่ของข้อผิดพลาด และเพิ่มโอกาสการใช้ซ้ำ การไม่ขึ้นต่อกันสามารถวัดได้โดยใช้ปริมาณเชิงคุณภาพ 2 ค่า คือ Cohesion และ Coupling โดยค่า Cohesion เป็นค่าวัดความเป็นอันหนึ่งอันเดียวกันของหน้าที่ในโมดูล และค่า Coupling เป็นการวัดการขึ้นต่อกันระหว่างโมดูล Cohesion อาจมองได้ว่าเป็นแนวคิดเพิ่มเติมจากการซ่อนข้อมูล โมดูลในลักษณะที่มีค่า Cohesion สูง หมายถึงโมดูลที่รับผิดชอบงานเดี่ยว มีการติดต่อกับชิ้นส่วนซอฟต์แวร์อื่นๆในระบบน้อย อย่างไรก็ตามโมดูลหนึ่งมักจะมีฟังก์ชั่นการทำงานมากกว่า 1 อย่างเป็นปกติ แต่จะไม่มากจนทำได้ทุกอย่างในโมดูลเดียว ส่วน Coupling สามารถมองเป็นการเชื่อมต่อระหว่างโมดูล โดย Coupling จะเป็นความซับซ้อนของส่วนต่อประสานระหว่างโมดูล ซึ่งควรจะมีค่าต่ำในการออกแบบที่ดี การเชื่อมต่อระหว่างโมดูลที่เรียบง่ายจะทำให้สามารถเข้าใจแต่ละองค์ประกอบได้ง่าย และลดข้อผิดพลาดที่อาจแพร่จากโมดูลหนึ่งไปยังส่วนอื่นๆของระบบ

การแบ่งละเอียด (Refinement)

การแบ่งละเอียดเป็นกลยุทธ์การออกแบบที่อธิบายว่าซอฟต์แวร์ควรถูกพัฒนาในลักษณะที่ค่อยๆแบ่งรายละเอียดการทำงานลงไปเป็นระดับๆ การแบ่งละเอียดเป็นกระบวนการขยายความโดยอาจจะเริ่มจากฟังก์ชั่นที่มีความเป็นนามธรรมสูง นั่นคือไม่ได้ระบุรายละเอียดที่เป็นการทำงานภายในหรือชนิดของโครงสร้างข้อมูล จากนั้นจึงค่อยๆขยายความไปเรื่อยๆ เมื่อลงไปในระดับอื่นๆ จะเห็นได้ว่า การแบ่งละเอียดนั้นเป็นสิ่งที่ตรงข้ามกับนามธรรมหากมองเป็นสถาปัตยกรรมที่รู้จักกันในยุคปัจจุบัน อาจมองการประกาศส่วนต่อประสานของบริการใน Service-Oriented Architecture ว่ามีความเป็นนามธรรมในขณะที่การพัฒนาต้นรหัสสำหรับบริการนั้นๆเป็นการแบ่งละเอียด

การรีแฟ็คเตอร์ (Refactoring)

การรีแฟ็คเตอร์เป็นกลไกการออกแบบที่สำคัญซึ่งใช้มากในวิธีการแบบอาไจล์ โดยการรีแฟ็คเตอร์การจัดเรียงการออกแบบหรือต้นรหัสให้อยู่ในรูปแบบที่เข้าใจได้ง่ายขึ้นภายใต้เงื่อนไขที่จะไม่มีการเปลี่ยนแปลงความหมาย

หน้าที่และพฤติกรรมของตัวซอฟต์แวร์ นั่นคือเป็นวิธีการจัดโครงสร้างภายในเพื่อให้มีคุณภาพสูงขึ้น แต่มุมมองจากภายนอกจะไม่เห็นว่าพฤติกรรมของระบบเปลี่ยนแปลงไป ในขณะที่ทำการรีแฟ็คเตอร์ สิ่งที่ต้องแก้ไขคือความซ้ำซ้อน ส่วนของการออกแบบที่ไม่ได้ใช้ อัลกอริทึมที่ไม่มีประสิทธิภาพ การตั้งชื่อที่ไม่สื่อความหมาย เป็นต้น ตัวอย่างเช่น มีชิ้นส่วนซอฟต์แวร์ตัวหนึ่งได้รับการออกแบบครั้งแรกแล้วมีการทำงานภายใน 5 หน้า พบว่าแต่ละหน้าที่ไม่เกี่ยวข้องกันมากนัก โดยมี 2 หน้าที่เกี่ยวข้องกันและอีก 3 หน้าที่เป็นอิสระจากกัน ซึ่งสามารถรีแฟ็คเตอร์ชิ้นส่วนซอฟต์แวร์นี้ออกได้เป็น 4 โมดูล โดยโมดูลแรกมี 2 หน้าที่ และอีก 3 โมดูลที่เหลือมีอย่างละ 1 หน้าที่ เมื่อแยกออกมาแล้วจะสามารถดูแลรักษาและทดสอบได้ง่ายขึ้น

แนวคิดการออกแบบเชิงวัตถุ

วิธีการเชิงวัตถุนั้นใช้อย่างแพร่หลายในวิศวกรรมซอฟต์แวร์ยุคปัจจุบัน โดยแนวคิดการออกแบบเชิงวัตถุมีดังต่อไปนี้

1. คลาสและวัตถุ (Classes and Objects)

คลาสจะเป็นต้นแบบของวัตถุ ซึ่งมีการนิยามตัวเก็บข้อมูลและตัวอธิบายพฤติกรรมไว้ในคลาสที่เกี่ยวข้องกับแนวคิดการออกแบบเรื่องสภาพโมดูลและการซ่อนข้อมูล โดยสามารถมองได้ว่าสภาพโมดูลคือคลาส ซึ่งเป็นสิ่งที่มีการตั้งชื่อ และการเข้าถึงข้อมูลภายในคลาสนั้นจะแบ่งเป็นระดับ ซึ่งก็คือการซ่อนข้อมูล วัตถุเป็นตัวแปรของคลาส แต่ละวัตถุมีพื้นที่ส่วนตัวเพื่อเก็บสถานะที่เปลี่ยนได้ตามค่าของตัวเก็บข้อมูลตามที่ประกาศไว้ในคลาส

2. การสืบทอด (Inheritance)

คลาสในแนวคิดเชิงวัตถุนั้นสามารถสืบทอดกันได้ ซึ่งทำให้โมดูลในแนวคิดเชิงวัตถุนี้มีสภาพเป็นลำดับชั้นแบบต้นไม้ โดยปกติคลาสหนึ่งๆจะมีคลาสแม่ 1 คลาสและคลาสลูกได้หลายคลาส อย่างไรก็ตามมีแนวคิดของการสืบทอดจากคลาสแม่ที่มากกว่า 1 คลาส แต่พบว่าเป็นการออกแบบที่เพิ่มความซับซ้อนให้กับระบบจึงไม่เป็นที่นิยมใช้ ลักษณะพิเศษอีกประเด็นหนึ่งของการสืบทอดคือจะมีการลดความเข้มงวดของการซ่อนข้อมูลลงระหว่างคลาสที่สืบทอดกัน ทำให้คลาสลูกได้รับการนิยามข้อมูลและพฤติกรรมบางอย่างมาจากคลาสแม่ได้

3. ข้อความ (Messages)

กลไกสำคัญในการออกแบบเชิงวัตถุคือ การส่งข้อความจากวัตถุหนึ่งไปยังอีกวัตถุหนึ่ง เพื่อให้วัตถุที่รับข้อความทำพฤติกรรมบางอย่างตามที่กำหนดไว้ในคลาสของวัตถุนั้น ในระดับการเขียนโปรแกรมการส่งข้อความคือ การเรียกใช้เมธอดนั่นเอง

4. โพลิมอร์ฟิซึ่ม (Polymorphism)

โพลิมอร์ฟิซึ่มเป็นความสามารถในการเปลี่ยนรูปได้ของวัตถุ นั่นคือในบางครั้งวัตถุจะถูกอ้างอิงด้วยตัวแปรของคลาสแม่ แต่เมื่อมีการเข้าถึงข้อมูลภายในหรือมีการรับข้อความเพื่อให้ทำงานตามพฤติกรรมวัตถุดังกล่าวแม้จะอยู่ในรูปตัวแปรของคลาสแม่ก็ตามจะแปลงกลับไปเป็นวัตถุของคลาสลูกที่ถูกต้อง แม้ว่าคลาสแม่นั้นจะมีคลาสลูกสืบทอดหลายคลาส การแปลงรูปจะสามารถทำได้ถูกต้องเสมอ การแปลงรูปจากตัวแปรของคลาสแม่ไปเป็นวัตถุคลาสลูกได้อย่างถูกต้องนี่เองจึงเรียกว่า โพลิมอร์ฟิซึ่ม

แบบจำลองการออกแบบ

แบบจำลองการออกแบบเป็นผลของการแปลงแบบจำลองการวิเคราะห์โดยใช้แนวคิดการออกแบบต่างๆ โดยทั่วไปแต่ละองค์ประกอบของแบบจำลองเชิงวัตถุ ซึ่งใช้ภาษา เช่น UML ในการออกแบบมักจะใช้กลุ่มของสัญลักษณ์ชุดเดียวกันกับการวิเคราะห์ โดยในระดับการออกแบบนั้นจะมีรายละเอียดสูงกว่า เนื่องจากการอธิบายเพิ่ม (Elaboration) และการแบ่งละเอียด (Refinement) ซึ่งรายละเอียดดังกล่าวคือ สิ่งที่เกี่ยวข้องกับการสร้างมากขึ้น โดยแบบจำลองการออกแบบที่สำคัญมีดังต่อไปนี้

1. แบบจำลองการออกแบบข้อมูล (Data Design Model)

การออกแบบข้อมูลเป็นการสร้างแบบจำลองที่เป็นตัวแทนของข้อมูลตามมุมมองของผู้ใช้ระบบ โดยการออกแบบจะอยู่ในรูปแบบที่มีนามธรรมสูงก่อน แล้วจึงเปลี่ยนระดับลงไปหารูปแบบที่ใกล้เคียงการสร้างจริง เช่น การออกแบบ ER ไดอะแกรม อาจเป็นการออกแบบที่ไม่ได้ลงรายละเอียดเกี่ยวกับระบบฐานข้อมูลที่จะใช้ก่อน จากนั้นเมื่อมีการกำหนดระบบฐานข้อมูลแล้ว จึงสร้างแบบจำลองอีกชุดหนึ่งที่ระบุรายละเอียดที่เกี่ยวข้องกับฐานข้อมูลดังกล่าว ในระดับชิ้นส่วนซอฟต์แวร์อาจเป็นการออกแบบโครงสร้างข้อมูลและอัลกอริทึมที่เกี่ยวข้อง ในระดับโปรแกรมประยุกต์ อาจเป็นการออกแบบแบบจำลองข้อมูลเพื่อแปลงเป็นโครงสร้างฐานข้อมูลขององค์กรเพื่อนำไปสู่การสร้างคลังข้อมูลต่อไป

2. แบบจำลองการออกแบบเชิงสถาปัตยกรรม

แบบจำลองการออกแบบเชิงสถาปัตยกรรมเป็นแบบจำลองการออกแบบการเชื่อมต่อของระบบย่อย ซึ่งมักจะใช้ข้อมูลจากการวิเคราะห์แพ็คเกจ (Package) ในกระบวนการวิเคราะห์ ในแต่ละระบบย่อยจะสามารถมีสถาปัตยกรรมของตนเองได้ เช่น ระบบย่อยสำหรับติดต่อกับผู้ใช้ อาจใช้รูปแบบสถาปัตยกรรม MVC (Model-View-Controller) เป็นต้น

3. แบบจำลองการออกแบบส่วนเชื่อมประสาน

แบบจำลองการออกแบบส่วนเชื่อมประสานเป็นแบบจำลองเพื่อออกแบบการติดต่อกับส่วนอื่นๆ โดยองค์ประกอบที่สำคัญ คือ

- องค์ประกอบของการติดต่อกับผู้ใช้
- องค์ประกอบของการเชื่อมต่อภายนอก
- องค์ประกอบของการเชื่อมต่อภายใน

3.1 การออกแบบส่วนติดต่อกับผู้ใช้มีลักษณะเฉพาะสูง โดยในระบบย่อยที่มีหน้าที่ติดต่อกับผู้ใช้นั้นมักจะมีสถาปัตยกรรมเฉพาะ ซึ่งต่างกับการเชื่อมต่อประเภทอื่น การออกแบบที่ดีจะเป็นการเพิ่มความน่าใช้งาน (Usability) ของระบบขึ้น องค์ประกอบเชิงเทคนิคที่มักใช้ในการออกแบบส่วนติดต่อกับผู้ใช้ คือ UI patterns และระบบชิ้นส่วนซอฟต์แวร์ที่สามารถใช้ซ้ำได้ บางครั้งเรียกว่า UI toolkit

3.2 การออกแบบส่วนเชื่อมต่อภายนอก ต้องการการนิยามการเชื่อมต่อที่ชัดเจน ซึ่งควรมีข้อมูลอย่างละเอียดจากช่วงการเก็บความต้องการเชิงซอฟต์แวร์ โดยจำเป็นต้องมีการตรวจสอบข้อผิดพลาดรวมทั้งระมัดระวังในแง่ของความมั่นคงด้วย

3.3 การออกแบบส่วนเชื่อมต่อภายใน จะมีลักษณะคล้ายการออกแบบชิ้นส่วนซอฟต์แวร์ การออกแบบจะเป็นการนำข้อมูลของคลาสระดับวิเคราะห์มาพิจารณา เพื่อสร้างแนวทางการติดต่อ

และร่วมมือระหว่างคลาส เพื่อให้ตอบสนองกับหน้าที่ที่ระบุไว้ในความต้องการเชิงซอฟต์แวร์ให้ได้ดีที่สุด

4. การออกแบบระดับชิ้นส่วน

การออกแบบระดับชิ้นส่วนเป็นการออกแบบที่ระบุรายละเอียดภายในเกี่ยวกับชิ้นส่วนซอฟต์แวร์แต่ละชิ้น โดยทำการระบุวิธีการเก็บข้อมูลภายในของแต่ละชิ้นส่วนรวมถึงรายละเอียดเชิงการประมวลผลภายในและส่วนเชื่อมต่อที่อนุญาตให้ชิ้นส่วนอื่นเข้ามาใช้บริการของชิ้นส่วนนั้นๆได้

5. การออกแบบระดับการนำไปใช้

การออกแบบองค์ประกอบระดับการนำไปใช้จะเป็นการอธิบายว่า ฟังก์ชั่นการทำงานและระบบย่อยต่างๆสามารถนำไปใช้งานในสภาพแวดล้อมจริงอย่างไร โดยในบริบทของการออกแบบเชิงวัตถุด้วยภาษา UML จะมีแผนผังที่เรียกว่าแผนผังการนำไปใช้เพื่อแสดงองค์ประกอบจริง เช่น เครื่องแม่ข่าย (Server) สภาพแวดล้อมการทำงาน (Execution environment) และชิ้นงาน (Artifact) ซึ่งเป็นตัวแทนระบบที่พัฒนาขึ้น โดยแผนผังดังกล่าวจะสนับสนุนการแสดงการเชื่อมต่อและความสัมพันธ์ขององค์ประกอบเหล่านี้

บทที่ 9

การสร้างซอฟต์แวร์

Software Implementation

การสร้างซอฟต์แวร์เป็นกระบวนการหลักในการพัฒนาซอฟต์แวร์ โดยเป็นการสร้างแอพพลิเคชั่นที่ต้องการให้กลายเป็นสิ่งที่ทำงานได้จริง ซึ่งเป็นการนำแบบจำลองการออกแบบมาแปลงเป็นต้นรหัส (Source Code)

เครื่องมือในการพัฒนาซอฟต์แวร์

เครื่องมือในการพัฒนาซอฟต์แวร์ คือ เครื่องมือที่นักพัฒนาใช้เพื่อสร้างต้นรหัส ซึ่งสามารถแบ่งออกเป็น 2 กลุ่ม คือ ภาษาโปรแกรมและเฟรมเวิร์คสำหรับพัฒนาซอฟต์แวร์

ภาษาโปรแกรม

ภาษาโปรแกรมเป็นองค์ประกอบสำคัญที่สุดในการสร้างซอฟต์แวร์ ซึ่งภาษาโปรแกรมที่ได้รับความนิยมมากที่สุดในปัจจุบันคือ

1. ภาษา Java เป็นภาษาที่เริ่มต้นพัฒนาโดย James Gosling ที่ Sun Microsystems เริ่มใช้ครั้งแรกในปี ค.ศ. 1995 โดยลักษณะไวยากรณ์หลักนั้นมีลักษณะคล้ายภาษา C และ C++ ภาษา Java ทำงานบนเครื่องจักรเสมือนที่เรียกว่า Java Virtual Machine (JVM) ซึ่งสามารถทำงานบนสถาปัตยกรรมของฮาร์ดแวร์ต่างๆ ได้หลายชนิด ในปี ค.ศ. 2007 ได้มีการเปิดต้นรหัส Java Development Kit (JDK) ภายใต้สัญญาอนุญาต GNU GPL เรียกว่า OpenJDK เทคโนโลยีภาษา Java ครอบคลุมหลายพื้นที่ที่สำคัญ เช่น Web Application, Mobile Application เป็นต้น อีกทั้งยังมีการปรับใช้ภาษา Java ในคอมไพเลอร์อื่นๆนอกเหนือจาก JDK เองคือ คอมไพเลอร์ของ GWT และ Dalvik Virtual Machine บนระบบปฏิบัติการ Android เป็นต้น
2. ภาษา C เป็นภาษาที่คิดค้นขึ้นในช่วงปี ค.ศ. 1969-1973 โดย Dennis Ritchie เพื่อพัฒนาระบบปฏิบัติการ ทำให้โปรแกรมที่เขียนด้วยภาษา C สามารถคอมไพเลอร์บนแพล็ตฟอร์มประเภทใดก็ได้ มาตรฐานภาษา C ที่สำคัญคือ C99 การใช้งานทั่วไปของภาษา C คือ ใช้พัฒนาซอฟต์แวร์ระบบและซอฟต์แวร์ที่ต้องการคอมไพล์ได้บนหลายแพล็ตฟอร์มตั้งแต่ไมโครโพรเซสเซอร์ไปจนถึงเครื่องระดับซูเปอร์คอมพิวเตอร์ คอมไพเลอร์ภาษา C ที่เป็นที่นิยมในปัจจุบัน มีดังนี้
   - Clang เป็นคอมไพเลอร์ภาษา C ที่เริ่มพัฒนาโดย University of Illinois และสนับสนุนโดย Apple
   - GCC มีคอมไพเลอร์ภาษา C/C++ ในชุดคอมไพเลอร์ของ GNU ใช้กันเป็นมาตรฐานทั่วไปบน Linux
   - MSVC เป็นคอมไพเลอร์ภาษา C/C++ ของ Microsoft
3. ภาษา C++ เป็นภาษาเชิงวัตถุที่พัฒนาโดย Bjarne Stroustrup เดิมเรียกว่า C with Classes เริ่มสร้างในปี ค.ศ. 1979 ต่อมาเปลี่ยนชื่อภาษาเป็น C++ ในปี ค.ศ. 1983 ภาษา C++ เป็นภาษาที่ได้รับความนิยมมาอย่างยาวนาน ใช้สร้างระบบงานมากมายและเป็นภาษาหลักที่มีอิทธิพลต่อการออกแบบภาษาอื่น เช่น Java หรือ C# ภาษา C++ มีมาตรฐาน คือ ISO/IEC 14882 ปี ค.ศ. 1998 มาตรฐานถัดมาที่อยู่ระหว่างการพัฒนา มีชื่อเรียกกันอย่างไม่เป็นทางการว่า C++0x เริ่มในปี ค.ศ. 2003 จนถึงปัจจุบัน คอมไพเลอร์ภาษา C++ ที่ได้รับความนิยมในปัจจุบันเป็นกลุ่มคอมไพเลอร์เดียวกันกับภาษา C

4. ภาษา PHP เป็นภาษาที่ถูกออกแบบสำหรับการพัฒนาเว็บที่เป็นไดนามิค พัฒนาขึ้นในปี ค.ศ. 1995 โดย Rasmus Lerdorf ในยุค PHP 3 นั้น PHP ได้รับการพัฒนาใหม่โดย Andi Gutmans และ Zeev Suraski จนกลายเป็น PHP5 ที่มีการเพิ่มความสามารถด้านการโปรแกรมเชิงวัตถุและเป็นรุ่นหลักที่ใช้กันในปัจจุบัน นอกจาก Zend Engine แล้วคอมไพเลอร์อื่นๆที่เป็นที่นิยมใช้ในปัจจุบันซึ่งสนับสนุน PHP คือ Phalange สำหรับ .Net, Quercus ซึ่งเป็น PHP บน Java แพล็ตฟอร์มและ HipHop เป็น PHP Source Translator แปลง PHP เป็น C++ เพื่อแก้ปัญหาประสิทธิภาพของ PHP พัฒนาโดย Facebook

5. ภาษา Python เป็นภาษาโปรแกรมระดับสูงที่มีแนวคิดด้านการทำให้ต้นรหัสอ่านได้ง่าย Python พัฒนาขึ้นในปี 1989 โดย Guido van Rossum เป็นภาษาที่มีลักษณะของทั้งการโปรแกรมเชิงวัตถุ เชิงฟังก์ชั่น และแบบขั้นตอน ภาษา Python ได้รับอิทธิพลจากภาษา C, C++ และ Perl รวมทั้ง Java ภาษา Python เป็นภาษาประเภทแปลทีละคำสั่งไม่ใช่คอมไพเลอร์ในลักษณะ C หรือ C++ ทำให้ประสิทธิภาพของ Python ไม่เร็วเท่าภาษากลุ่มนี้ อย่างไรก็ตามโมดูลในภาษา Python มักจะเขียนด้วย C หรือ C++ เพื่อแก้ปัญหาเชิงประสิทธิภาพ

ภาษาที่น่าสนใจ

1. ภาษา Scala เป็นภาษาโปรแกรมที่มีความหลากหลายทางกระบวนทัศน์ ทำงานบน Java แพล็ตฟอร์ม มีทั้งความเป็นเชิงฟังก์ชั่น เชิงวัตถุ และแบบขั้นตอน Scala เริ่มต้นพัฒนามาในปี ค.ศ. 2001 และรุ่นแรกปรากฏในปี ค.ศ. 2003 และในปี ค.ศ. 2009 บริษัท Twitter ประกาศการเปลี่ยนแปลงระบบ Web หลายส่วนจากเดิม ซึ่งเขียนในภาษา Ruby มาเป็นภาษา Scala ด้วยเหตุผลทางประสิทธิภาพและการบำรุงรักษา ผู้ออกแบบภาษา Scala คือ Martin Odersky ซึ่งเป็นผู้พัฒนาคอมไพเลอร์ภาษา Java ในรุ่น 5.0 ให้กับ Sun Microsystems

2. ภาษา Groovy เป็นภาษาโปรแกรมที่มีความหลากหลายทางกระบวนทัศน์ เช่นเดียวกับ Scala ต่างกันตรงที่ Groovy เป็นภาษาประเภทไดนามิคที่ทำงานบน Java แพล็ตฟอร์ม มีจุดเด่นเรื่องความสามารถในการช่วยสร้างภาษาเฉพาะทาง (Domain Specific Language) ภาษา Groovy ได้รับอิทธิพลโดยตรงจากภาษา Java รวมทั้ง Ruby และ Perl Groovy 1.0 ปรากฏในปี ค.ศ. 2007

3. ภาษา Go เป็นภาษาที่พัฒนาโดย Google เปิดตัวครั้งแรกในปี ค.ศ. 2009 มีความหลากหลายทางกระบวนทัศน์ แต่ไม่เป็นภาษาเชิงวัตถุ โดยมีความเป็นเชิงโครงสร้างและแบบขั้นตอน สิ่งที่เพิ่มเติมเข้ามาในภาษา Go คือ การสนับสนุน Concurrency ภาษา Go ได้รับอิทธิพลมาจากภาษา C, Limbo, Newspeak เป็นต้น

4. ภาษา Erlang เป็นภาษาที่ถูกพัฒนาขึ้นในปี ค.ศ. 1986 ที่ Ericsson โดยเปิดต้นรหัสในปี ค.ศ 1998 และเพิ่มความนิยมมากขึ้น เนื่องจากมีความสามารถในการเขียนโปรแกรมเชิงขนานได้ดี โดย Erlang เป็นภาษาเชิงฟังก์ชั่นที่สนับสนุนการตั้งค่าตัวแปรได้ครั้งเดียวทำให้เหมาะสมต่อการใช้งานระบบที่มีโปรเซสทำงานร่วมกันเป็นจำนวนมาก กลไกสำคัญของ Erlang อีกจุดหนึ่งคือ การใช้เทคนิคการส่งข้อความระหว่างโปรเซสแทนการใช้ตัวแปรร่วมกัน รูปแบบการเขียนโปรแกรมลักษณะนี้ไม่ต้องการการล็อคตัวแปร ทำให้ได้ประสิทธิภาพเชิงขนานสูง ภาษา Erlang นิยมใช้มากในงานทางโทรคมนาคม โดยเฉพาะการพัฒนาเครื่องแม่ข่ายที่ต้องรองรับผู้ใช้จำนวนมาก

เฟรมเวิร์คสำหรับพัฒนาซอฟต์แวร์

เฟรมเวิร์คเป็นกลุ่มของต้นรหัสที่สามารถใช้ซ้ำได้ ผู้ใช้เฟรมเวิร์คสามารถเลือกที่จะเขียนต้นรหัสทับบางส่วนของเฟรมเวิร์คหรือปรับเปลี่ยนต้นรหัสส่วนที่ต้องการได้โดยไม่ต้องแก้ไขตัวเฟรมเวิร์คโดยตรง เฟรมเวิร์คสำหรับ

การพัฒนาซอฟต์แวร์มีหลายกลุ่ม เช่น เฟรมเวิร์คสำหรับแอพพลิเคชั่นตั้งโต๊ะ เฟรมเวิร์คสำหรับเว็บแอพพลิเคชั่น และเฟรมเวิร์คสำหรับอุปกรณ์เคลื่อนที่

เฟรมเวิร์คสำหรับแอพพลิเคชั่นตั้งโต๊ะ

เฟรมเวิร์คสำหรับแอพพลิเคชั่นตั้งโต๊ะนั้นมีอยู่มากมายหลายเฟรมเวิร์คแยกตามภาษาและแพล็ตฟอร์มที่ใช้ เช่น

Swing เป็นเฟรมเวิร์คสำหรับพัฒนาแอพพลิเคชั่นตั้งโต๊ะบนแพล็ตฟอร์ม Java เป็นส่วนหนึ่งของ Java Foundation Class ซึ่งเป็นชุดคำสั่งสำหรับสร้างส่วนติดต่อผู้ใช้ (Graphical User Interface - GUI) โดยโครงสร้างเชิงสถาปัตยกรรมของ Swing เป็นประเภท MVC (Model-View-Controller) สำหรับแอพพลิเคชั่นที่สร้างด้วย Swing สามารถทำงานได้บนทุกแพล็ตฟอร์มที่ Java สนับสนุน โดยรูปแบบของเฟรมเวิร์ค Swing นั้นมีอิทธิพลต่อเฟรมเวิร์คอื่นบนแพล็ตฟอร์ม Java เช่น SWT

WinForms เป็นเฟรมเวิร์คสำหรับพัฒนาแอพพลิเคชั่นตั้งโต๊ะในลักษณะเดียวกับ Swing แต่ WinForms เป็นส่วนหนึ่งของ .Net Framework และทำงานเฉพาะบน .Net เท่านั้น โครงสร้างเชิงสถาปัตยกรรมเป็นประเภทเดียวกับ MVC โดยมีจุดต่างคือ มีกระบวนการโยงข้อมูล (Data binding) ที่มีประสิทธิภาพกว่า MVC ที่ใช้ใน Swing ซึ่งบางครั้งเรียกแยกออกเป็นสถาปัตยกรรมแบบ MVP (Model-View-Presenter) สำหรับแอพพลิเคชั่นที่สร้างด้วย WinForms จะมีการยึดติดกับระบบปฏิบัติการ Microsoft Windows ค่อนข้างสูง หากพัฒนาโดยใช้ชุดคำสั่งเฉพาะกับ Microsoft Windows ก็จะไม่สามารถนำไปใช้งานบน .Net Framework หรือรันไทม์ตัวอื่น เช่น Mono ซึ่งทำงานบนระบบปฏิบัติการอื่นๆ ได้

Qt เป็นเฟรมเวิร์คสำหรับพัฒนาแอพพลิเคชั่นตั้งโต๊ะ โดยตัวเฟรมเวิร์คพัฒนาด้วยภาษา C++ มีรูปแบบการโปรแกรมเพื่อจัดการเหตุการณ์เฉพาะแบบเรียกว่าการเชื่อม slot ซึ่งต่างกับรูปแบบที่พบใน Swing หรือ WinForms ระบบจัดการ Layout จะมีลักษณะคล้ายกับ Swing คือมีระบบจัดการ Layout แบบบังคับต่างจาก WinForms ซึ่งมักเป็นแบบอิสระ เหตุผลที่ Qt เน้นการจัดการ Layout ประภทนี้ เนื่องจากเป็นเฟรมเวิร์คที่ทำงานได้บนระบบปฏิบัติการหลายประเภท ซึ่งต้องพึ่งระบบจัดการ Layout แบบบังคับเพื่อให้การแสดงผลสามารถทำได้ถูกต้องเมื่อความละเอียดหน้าจอและขนาดตัวหนังสือต่างกัน ระบบที่ใช้งาน Qt ได้แก่ KDE ซึ่งเป็นระบบจัดการหน้าต่างบน Linux รวมทั้งแอพพลิเคชั่นขนาดใหญ่อีกหลายตัว นอกจาก C++ แล้วภาษา Python, Ruby และ Java ยังสามารถใช้เพื่อเขียน Qt ได้เช่นกัน

Gtk เป็นเฟรมเวิร์คสำหรับพัฒนาแอพพลิเคชั่นตั้งโต๊ะ เขียนด้วยภาษา C++ พัฒนาขึ้นเพื่อใช้สร้างแอพพลิเคชั่นตัวหนึ่งที่เป็นประเภทเปิดต้นรหัสคือ GIMP ซึ่งเป็นโปรแกรมแต่งภาพคล้าย Photoshop ในระยะถัดมามีการจัดกลุ่มและแยกส่วนที่ใช้ซ้ำในโครงการ GIMP ออกมาเป็น Gtk โดยโปรแกรมที่ใช้ Gtk อีกตัวหนึ่งที่ได้รับความนิยม คือ Pidgin ซึ่งเป็น Instant Messenger Client เฟรมเวิร์ค Gtk ทำงานได้บนระบบปฏิบัติการหลายตัวรวมทั้ง Mono ซึ่งเป็น .Net Framework สำหรับ Linux โดยบน Mono จะสามารถเขียนแอพพลิเคชั่นด้วย Gtk ได้โดยใช้ภาษา C# และชุดคำสั่งของ Gtk บน Mono เรียกว่า Gtk#

เฟรมเวิร์คสำหรับพัฒนาเว็บแอพพลิเคชั่น

เฟรมเวิร์คสำหรับพัฒนาเว็บแอพพลิเคชั่นสามารถแบ่งออกได้เป็น 2 กลุ่มคือ เว็บเฟรมเวิร์คแบบดั้งเดิม และเฟรมเวิร์คประเภท Rapid MVC

1. เว็บเฟรมเวิร์คดั้งเดิม

เฟรมเวิร์คในยุคแรกของการพัฒนาเว็บแอพพลิเคชั่นที่ได้รับความนิยมนอกเหนือจากการเขียนเว็บด้วยภาษา PHP คือ Java Servlet, JSP, ASP และ ASP.Net

Java Servlet และ JSP

Java Servlet และ JSP เป็นเฟรมเวิร์คภาษา Java ที่เกิดขึ้นในช่วงปี ค.ศ. 1997 โดยวัตถุ Servlet มีหน้าที่รับการร้องขอแบบ HTTP เพื่อประมวลผลและทำการตอบกลับ โดยเทคโนโลยีถัดจาก Servlet คือ JSP ซึ่งเป็นการเขียนหน้าเว็บในลักษณะ HTML ที่ยอมให้ฝังชุดคำสั่งภาษา Java ลงไป (เป็นลักษณะเช่นเดียวกับ PHP) โดยไฟล์ JSP จะถูกแปลงเป็น Servlet อีกทอดหนึ่ง ในปัจจุบันมีการใช้งาน Servlet ลดลง โดยมักเป็นการใช้เพื่อเป็นตัวประมวลผลภายในของเฟรมเวิร์คอื่นมากกว่าเป็นการใช้งานโดยตรง ในขณะที่ JSP ยังเป็นที่นิยมใช้อยู่ในเฟรมเวิร์คยุคใหม่อย่างเช่น Spring ROO และ Grails

ASP และ ASP.Net

ASP เป็นเฟรมเวิร์คที่พัฒนาขึ้นโดย Microsoft ในปี ค.ศ. 1998 โดยสนับสนุนภาษาหลักคือ VBScript และสนับสนุนภาษาเพิ่มเติมเช่น JScript และ PerlScript เป็นต้น ASP และ JSP มีลักษณะการเขียนแบบเดียวกันคือ เป็นการเขียนชุดคำสั่งภาษา VBScript ฝังลงไปในหน้า HTML สำหรับ ASP.Net เป็นอีกเฟรมเวิร์คหนึ่งที่พัฒนาเพื่อใช้แทน ASP โดยมีจุดที่แตกต่าง คือ ASP.Net มีโครงสร้างเป็นประเภทคอมโพเนนท์มากขึ้นและสามารถทำงานกับภาษาใดๆก็ได้บน .Net แพล็ตฟอร์ม

2. เฟรมเวิร์คประเภท Rapid MVC

เว็บเฟรมเวิร์คกลุ่มนี้มีต้นแบบมาจากเฟรมเวิร์ค Ruby on Rails ซึ่งพัฒนาขึ้นในปี ค.ศ. 2004 โดยใช้ภาษา Ruby on Rails กับเว็บเฟรมเวิร์คในยุคก่อนหน้านี้คือ แนวคิดและการออกแบบ แม้ว่า Ruby on Rails จะใช้สถาปัตยกรรมแบบ MVC เหมือนกับเว็บเฟรมเวิร์คหลายๆตัว แต่สิ่งที่น่าสนใจคือ แนวคิดเรื่องการใช้ข้อตกลงก่อนการปรับแต่ง (Convention over Configuration) และกฎการไม่ทำงานซ้ำซาก (Don't Repeat Yourself)

การใช้ข้อตกลงก่อนการปรับแต่ง เป็นการลดงานที่เกี่ยวข้องกับการปรับแต่งโดยตั้งข้อตกลงขึ้นมาใช้ร่วมกัน (ระหว่างนักพัฒนากับเฟรมเวิร์ค) เช่น ถ้ามีคลาสชื่อ Book แล้วตารางในฐานข้อมูลจะมีชื่อว่า books หรือถ้าคลาสชื่อ BookController จะใช้เป็นตัวควบคุมเว็บ เป็นต้น เมื่อมีการตั้งข้อตกลงในลักษณะนี้แล้วจะทำให้ไม่มีความจำเป็นที่ต้องเขียนตัวอธิบายสำหรับการปรับแต่ง ซึ่งใช้มากในเฟรมเวิร์คภาษา Java หลายตัว

กฎการไม่ทำงานซ้ำซาก เป็นอีกกลไกหนึ่งเพื่อลดงานของนักพัฒนาโดยจะมีการเตรียมให้แก้ไขงานเพียงจุดเดียวในระบบเพื่อทำงาน 1 อย่าง เนื่องจากในบางเฟรมเวิร์คมีความจำเป็นที่ต้องแก้ไขไฟล์หลายๆไฟล์เพียงเพื่อจะทำงานง่ายๆ 1 อย่าง ทำให้นักพัฒนาต้องทำงานที่ไม่จำเป็นซ้ำๆ

เฟรมเวิร์ค Ruby on Rails

ได้รับความนิยมมากจนมีการสร้างเฟรมเวิร์คในลักษณะเดียวกันบนแพล็ตฟอร์มและภาษาอื่นๆ เช่น

- Grails ใช้ภาษา Groovy ทำงานบน Java แพล็ตฟอร์ม พัฒนาขึ้นในปี ค.ศ. 2006
- Spring ROO ใช้ภาษา Java และ AspectJ ทำงานบน Java แพล็ตฟอร์ม
- Django ใช้ภาษา Python ทำงานบน CPython และ Jython

- CakePHP, CodeIgniter, Kohana ใช้ภาษา PHP ทั้ง 3 เฟรมเวิร์ค โดย Kohana ใช้ PHP5 และสร้างต่อมาจาก CodeIgniter
- ASP.Net MVC ใช้ภาษา C#, VB.Net และอื่นๆ ทำงานบน .Net Framework

เฟรมเวิร์คสำหรับอุปกรณ์เคลื่อนที่

เฟรมเวิร์คกลุ่มนี้มักเป็นเฟรมเวิร์คที่มีความเฉพาะตัวสูงและใช้สำหรับอุปกรณ์ประเภทใดประเภทหนึ่ง อย่างไรก็ตามเฟรมเวิร์คบางตัวสามารถเริ่มทำงานข้ามประเภทอุปกรณ์โดยใช้เทคโนโลยี เช่น HTML5 ได้

iOS SDK เป็นเฟรมเวิร์คสำหรับพัฒนาแอพพลิเคชั่นบนมือถือตระกูล iPhone และ Tablet iPad ของ Apple โดยใช้ภาษาหลักคือ Objective-C ซึ่งเป็นภาษาเชิงวัตถุที่สร้างขึ้นในยุคใกล้เคียงกับ C++ โปรแกรมที่สร้างด้วย iOS SDK นั้นสามารถใช้ความสามารถของอุปกรณ์ได้เกือบทุกความสามารถเนื่องจากเป็นการสร้างแอพพลิเคชั่นชนิดเนทีฟ

Android SDK เป็นเฟรมเวิร์คสำหรับพัฒนาแอพพลิเคชั่นบนมือถือที่ใช้ระบบปฏิบัติการ Android ของ Google ซึ่งตัวระบบปฏิบัติการจะมีองค์ประกอบหลัก 2 ส่วน คือ 1) Kernel จะเป็น Linux และ 2) Virtual Machine เป็น Dalvik Virtual Machine เฉพาะที่พัฒนามาสำหรับ Android โดย SDK จะใช้คอมไพเลอร์ภาษา Java แปลงเป็นชุดคำสั่งเฉพาะที่ทำงานบน Virtual Machine ตัวนี้ได้ SDK ของ Android ยอมให้เข้าถึงทรัพยากรส่วนใหญ่ของอุปกรณ์ อย่างไรก็ตามถ้าจะเข้าถึงการทำงานระดับฮาร์ดแวร์ได้ทุกอย่างนั้นจะต้องใช้ SDK อีกตัวหนึ่ง ชื่อ Android NDK

Rhode เป็นเฟรมเวิร์คที่พัฒนาแอพพลิเคชั่นข้ามประเภทอุปกรณ์ได้โดยใช้ Ruby on Rails เป็นต้นแบบในการออกแบบการพัฒนาตัวแอพพลิเคชั่นจะสามารถทำงานบนชุดคำสั่งภาษา Ruby ที่เตรียมไว้ให้เพื่อเข้าถึงทรัพยากรของอุปกรณ์ โดยจะทำงานได้ทั้งบนอุปกรณ์กลุ่ม iPhone, Android และอื่นๆ

Phonegap เป็นเฟรมเวิร์คที่ใช้ JavaScript ครอบการเข้าถึงทรัพยากรของตัวอุปกรณ์ ทำให้สามารถพัฒนาแอพพลิเคชั่นข้ามประเภทอุปกรณ์ได้โดยใช้ภาษา HTML และ JavaScript โดยตัวแสดงผลของ Phonegap นั้นจะเป็นเว็บเบราเซอร์ตระกูล WebKit ซึ่งเป็นระบบแบบเดียวกับที่ใช้ใน Google Chrome ทำให้ตัวแอพพลิเคชั่นมีประสิทธิภาพสูงใกล้เคียงกับการพัฒนาด้วยภาษา เช่น Objective-C หรือ Java ได้

บทที่ 10

การทดสอบซอฟต์แวร์

Software Testing

การทดสอบซอฟต์แวร์เป็นกระบวนการดำเนินการตรวจสอบเพื่อให้ผู้ที่เกี่ยวข้องกับระบบ เช่น เจ้าของหรือลูกค้าได้รับทราบถึงระดับคุณภาพของซอฟต์แวร์นั้นๆ รวมทั้งให้บุคคลทางฝั่งธุรกิจที่เกี่ยวข้องกับระบบได้เข้าใจความเสี่ยงในการสร้างซอฟต์แวร์ การทดสอบซอฟต์แวร์มีหลายเทคนิคที่สามารถทำได้ เช่น การสั่งให้ซอฟต์แวร์ทำงานเพื่อที่จะค้นหาข้อบกพร่อง หรือการพิสูจน์ด้วยวิธีการทางฟอร์มอลเมธอด (Formal method) เป็นต้น การทดสอบซอฟต์แวร์นั้นขึ้นกับวิธีและกระบวนการพัฒนา โดยอาจจะเป็นระยะหลังจากการสร้างซอฟต์แวร์แล้วหรืออาจเป็นส่วนหนึ่งของช่วงการพัฒนาก็ได้ ในขณะที่วิธีการพัฒนาบางแนวคิดนั้นอาจทำการทดสอบก่อนที่จะเขียนโปรแกรม เช่น Test-Driven Development

ระดับของการทดสอบ

การทดสอบซอฟต์แวร์สามารถแบ่งออกเป็นระดับได้ดังนี้

1. การทดสอบระดับหน่วย (Unit Testing)

   การทดสอบระดับหน่วยเป็นการทดสอบระดับที่ใกล้ผู้พัฒนามากที่สุด เพื่อตรวจทานความสามารถบางส่วนของต้นรหัสที่สร้างขึ้น เช่น การทดสอบฟังก์ชั่นหรือการทดสอบเมธอด เป็นต้น โดยเมธอดหรือฟังก์ชั่นอาจมีการทดสอบมากกว่า 1 แบบ เพื่อให้ครอบคลุมกรณีที่ต้องระมัดระวังเป็นพิเศษ คุณสมบัติที่สำคัญของการทดสอบระดับหน่วย คือ ความเป็นอิสระต่อกัน ซึ่งจะทำให้สามารถมั่นใจได้ว่า พฤติกรรมของระบบได้รับการทดสอบเป็นส่วนๆแล้ว

2. การทดสอบระดับบูรณาการ (Integration Testing)

   การทดสอบระดับบูรณาการเป็นระดับของการทดสอบที่ใช้เพื่อตรวจทานการต่อประสานระหว่างชิ้นส่วนซอฟต์แวร์ว่าเป็นไปตามการออกแบบหรือไม่ การทดสอบเชิงบูรณาการจะเป็นระยะที่นำเอาโมดูลหรือชิ้นส่วนทางซอฟต์แวร์แต่ละตัวมารวมและทำการทดสอบร่วมกัน การทดสอบที่ตั้งขึ้นจะเป็นการทดสอบที่ครอบคลุมการทำงานของทุกๆชิ้นส่วน ซึ่งต้องมีการเรียกใช้คำสั่งข้ามชิ้นส่วนกัน การทดสอบประเภทนี้มักกระทำหลังจากได้ทำการทดสอบระดับหน่วยเรียบร้อยแล้ว

   แนวทางการทดสอบระดับบูรณาการจะมี 3 แนวทาง คือ

   1) Big Bang เป็นการประกอบระบบแล้วทดสอบด้วยข้อมูลการใช้งานจริง
   2) การทดสอบจากบนลงล่าง เป็นการทดสอบระบบรวมก่อนระบบย่อย
   3) การทดสอบจากล่างขึ้นบน เป็นการทดสอบระบบย่อยก่อนแล้วค่อยๆทดสอบระบบที่ใหญ่ขึ้น

3. การทดสอบระดับระบบ

   การทดสอบระดับระบบเป็นการทดสอบระบบที่ประกอบกันอย่างสมบูรณ์แล้วว่าสามารถทำงานได้ตามข้อกำหนดเชิงฟังก์ชั่นหรือไม่ การทดสอบในลักษณะนี้จะเป็นการทดสอบพฤติกรรมของระบบว่าเป็นไปตามความคาดหวังของผู้ใช้หรือลูกค้าหรือไม่ โดยการทดสอบระดับระบบจะทำหลังการทดสอบระดับบูรณาการ

4. การทดสอบระดับบูรณาการระบบ

การทดสอบระดับบูรณาการระบบเป็นกระบวนการทดสอบระบบซอฟต์แวร์ในบริบทของการเชื่อมต่อกับระบบภายนอกอื่นๆ ที่ระบุไว้ในเอกสารข้อกำหนด เนื่องจากการทดสอบระดับนี้จะมีการใช้ระบบจริงซึ่งเชื่อมต่ออยู่กับระบบอื่นมาใช้เป็นตัวทดสอบดังนั้นข้อมูลสำหรับใช้ในการทดสอบระดับนี้ควรมีขนาดเล็กที่สุดที่จะครอบคลุมกรณีทดสอบเพื่อลดเวลาในแต่ละระบบของการทดสอบ

5. การทดสอบแบบถดถอย

การทดสอบแบบถดถอยเป็นการทดสอบที่ทำซ้ำหลังมีการเปลี่ยนแปลงแก้ไขต้นรหัสเกิดขึ้น โดยปกติในกระบวนการทดสอบจะมีการทดสอบระดับหน่วยสำหรับชุดของต้นรหัสที่เขียนเพิ่มขึ้น และเพื่อให้แน่ใจว่า ส่วนที่เพิ่มขึ้นมานั้นไม่ไปทำให้เกิดผลกระทบต่อต้นรหัสทั้งหมดที่มีอยู่ก่อนแล้ว นักพัฒนาจะทำการทดสอบเชิงถดถอยโดยนำเอากรณีทดสอบเชิงหน่วยที่สร้างไว้ทั้งหมดมาทดสอบซ้ำ

การทดสอบแอพพลิเคชั่นทั่วไป

การทดสอบแอพพลิเคชั่นทั่วไปสามารถแบ่งออกได้เป็น 2 ประเภท ดังนี้

1. การทดสอบแบบ White-box

การทดสอบแบบ White-box เป็นวิธีการที่ใช้ทดสอบโครงสร้างภายในของซอฟต์แวร์ โดยการสร้างกรณีทดสอบประเภทนี้ผู้เขียนกรณีทดสอบจำเป็นต้องมีความรู้ความเข้าใจเกี่ยวกับระบบภายในของซอฟต์แวร์นั้นๆเป็นอย่างดีรวมทั้งต้องมีทักษะการพัฒนาโปรแกรมด้วย ในการทดสอบลักษณะนี้ผู้พัฒนามักจะเป็นผู้เตรียมข้อมูลทดสอบเอง อย่างไรก็ตามแม้ว่าการทดสอบประเภท White-box จะสามารถนำไปใช้ได้ในทุกๆระดับของการทดสอบ แต่ก็มักจะทำในการทดสอบระดับหน่วย ตัวอย่างเครื่องมือประเภท White-box เช่น JUnit

2. การทดสอบแบบ Black-box

การทดสอบแบบ Black-box เป็นวิธีการทดสอบซอฟต์แวร์ที่ทดสอบเฉพาะฟังก์ชั่นการทำงานของซอฟต์แวร์โดยไม่สนใจรายละเอียดภายในหรือโครงสร้างภายในของโปรแกรม การทดสอบจะถูกเตรียมขึ้นจากข้อมูลข้อกำหนดความต้องการเชิงซอฟต์แวร์ รวมถึงการออกแบบ โดยปกติการทดสอบประเภท Black-box มักจะเป็นการทดสอบความต้องการเชิงฟังก์ชั่น แต่ก็จะสามารถทดสอบความต้องการส่วนที่ไม่ใช่ฟังก์ชั่นได้เช่นกัน ตัวอย่างเครื่องมือประเภท Black-box เช่น Selenium, Sikuli เป็นต้น

การทดสอบแอพพลิเคชั่นเชิงวัตถุ

หน่วยย่อยที่สุดที่สามารถทดสอบได้ในการโปรแกรมเชิงวัตถุคือคลาส จึงทำให้คุณสมบัติโดยเฉพาะ encapsulation เข้ามาเกี่ยวข้องกับการทดสอบโดยตรง นั่นแปลว่าจะทำให้ไม่สามารถทดสอบเมธอดเดี่ยวๆ ได้ในภาวะที่แยกออกจากระบบอย่างสิ้นเชิงได้เนื่องจากการทดสอบเมธอดหนึ่งๆของคลาสจะทำให้ "สถานะ" ของวัตถุเปลี่ยนแปลงไป

แนวคิดในการออกแบบกรณีทดสอบสำหรับระบบเชิงวัตถุจึงควรคำนึงถึงหลักดังต่อไปนี้

1. แต่ละกรณีทดสอบควรมีความเฉพาะและเกี่ยวข้องกันอย่างชัดเจนต่อคลาสที่นำมาทดสอบ ในเฟรมเวิร์คบางประเภทจะมีข้อตกลงในการทดสอบในลักษณะดังต่อไปนี้ เช่น คลาส CarTest จะบรรจุกรณีทดสอบของคลาส Car

2. จุดประสงค์ของการทดสอบควรระบุไว้อย่างชัดเจน
3. แต่ละกรณีทดสอบประกอบไปด้วยขั้นตอนการทดสอบโดยแต่ละขั้นตอนควรมี
    - วัตถุในสถานะต่างๆกันของคลาสที่จะนำมาทดสอบ ในเฟรมเวิร์คสำหรับการทดสอบอาจเรียกวัตถุเหล่านี้ว่า Test Fixture
    - ชุดของเมธอดหรือโอเปอร์เรชั่นที่จะถูกเรียกใช้เป็นลำดับเพื่อทดสอบ
    - ชุด Exception ที่อาจจะเกิดขึ้นได้จากคลาสที่กำลังทดสอบ
    - ข้อมูลเพิ่มเติมที่จะช่วยทำความเข้าใจต่อการทดสอบ

การทดสอบเว็บแอพพลิเคชั่น

การทดสอบเว็บแอพพลิเคชั่นเป็นกลุ่มของกิจกรรมที่ใช้เพื่อทดสอบเนื้อหา, การเชื่อมต่อ, การออกแบบ, ส่วนติดต่อผู้ใช้, ฟังก์ชั่นการทำงาน, การนำทาง, สภาพการปรับแต่งที่ต่างกัน (เช่น เว็บเบราเซอร์ต่างกัน) รวมทั้งความมั่นคง, สมรรถนะ, และทดสอบการใช้งานด้วยผู้ใช้จริง

การทดสอบด้านเนื้อหา

การทดสอบด้านเนื้อหาเป็นการทดสอบเพื่อหาข้อผิดพลาดในเชิงการพิมพ์ผิด หรือการสะกดคำไม่ถูกต้อง รวมถึงความหมายที่คลาดเคลื่อนของข้อความ และการจัดเรียงเนื้อหาที่แสดงต่อผู้ใช้ ระบบเว็บแอพพลิเคชั่นทั่วไปในปัจจุบันเก็บข้อมูลที่เป็นเนื้อหาไว้ในฐานข้อมูล การทดสอบเนื้อหาจึงอาจจะจำเป็นต้องทดสอบโดยผ่านการดึงข้อมูลมาจากฐานข้อมูลด้วย ซึ่งข้อผิดพลาดพลาดที่เกิดขึ้นเมื่อมีระบบฐานข้อมูลมาเกี่ยวข้อง คือ การ Encode ข้อมูลตามรหัสภาษา เช่น TIS-620 หรือ UTF-8 เป็นต้น

การทดสอบส่วนติดต่อกับผู้ใช้

การทดสอบส่วนติดต่อกับผู้ใช้เป็นการทดสอบเพื่อค้นหาข้อผิดพลาดในส่วนของการแสดงผลและกลไกการนำทางในเว็บแอพพลิเคชั่น

- ส่วนติดต่อกับผู้ใช้จะถูกทดสอบเพื่อให้แน่ใจว่าการแสดงผลที่ช่วยในการนำเสนอเนื้อหาสามารถใช้งานได้ถูกต้อง โดยรวมถึงชนิดของฟอนต์, การใช้สี, การจัดแบ่งเฟรม, รูปภาพหรือตาราง โดยเทคโนโลยีหลักที่เกี่ยวข้อง คือ Cascaded Style Sheet, HTML
- การทดสอบส่วนติดต่อหนึ่งๆ ควรกระทำในลักษณะเชิงหน่วย โดยการแยกทดสอบการแสดงผลเฉพาะส่วน ในกรณีนี้สามารถใช้เครื่องมือ เช่น Selenium ประกอบกับเฟรมเวิร์คการทดสอบระดับหน่วย เช่น Junit มากระทำการทดสอบได้

การทดสอบการออกแบบ

การทดสอบการออกแบบเป็นการทดสอบการออกแบบเพื่อหาข้อผิดพลาดเกี่ยวกับการใช้งาน เช่น การโต้ตอบมีลักษณะการใช้งานอย่างไร เข้าใจง่ายหรือไม่ การวางเลย์เอ้าต์ของแอพพลิเคชั่นเป็นอย่างไร มีกลไกการนำทางอย่างไร เนื้อหาเข้าถึงได้ง่ายหรือไม่ ข้อความอ่านง่าย ทำความเข้าใจง่ายหรือไม่ รวมทั้งกราฟฟิกสามารถแทนสิ่งที่ต้องการสื่อได้หรือไม่ นอกจากนี้ยังรวมถึงลักษณะการแสดงผลอื่นๆ เช่น สี ตัวอักษร และการใช้งานภายใต้ขนาดจอภาพหรือความละเอียดจอที่เว็บแอพพลิเคชั่นต้องสนับสนุน การทดสอบการออกแบบอาจครอบคลุมถึงการทดสอบเวลาในการตอบสนองว่าผู้ใช้สามารถเข้าถึงฟังก์ชั่นหรือเนื้อหาที่ต้องการภายในเวลาที่เหมาะสมหรือไม่

การทดสอบฟังก์ชั่นของคอมโพเน้นท์

การทดสอบฟังก์ชั่นของคอมโพเน้นท์เป็นการทดสอบเพื่อหาข้อผิดพลาดในฟังก์ชั่นการทำงานของเว็บแอพพลิเคชั่นแต่ละฟังก์ชั่นในเว็บแอพพลิเคชั่นสามารถพิจารณาเป็น Black-box และใช้เทคนิคการทดสอบสำหรับ Black-box เข้ามาช่วยได้ โดยทั่วไปจะเป็นการจำลองการป้อนข้อมูลของผู้ใช้และตรวจสอบผลลัพธ์ที่แต่ละฟังก์ชั่นตอบกลับออกมา เครื่องมือที่เกี่ยวข้องสำหรับขั้นตอนนี้ เช่น Selenium

การทดสอบอื่นๆ

- เพื่อให้เว็บแอพพลิเคชั่นมีความถูกต้องตามข้อกำหนด จึงจำเป็นต้องทำการทดสอบระบบนำทางให้ทั่วทั้งแอพพลิเคชั่น
- เว็บแอพพลิเคชั่นจำเป็นต้องทดสอบความเข้ากันได้กับสภาพแวดล้อมการทำงานที่ต่างชนิดกัน บางครั้งจะเรียกการทดสอบแบบนี้ว่า การทดสอบความเข้ากันได้ข้ามเบราเซอร์
- การทดสอบความมั่นคงก็เป็นสิ่งจำเป็นต่อเว็บแอพพลิเคชั่น เนื่องจากเว็บแอพพลิเคชั่นมักสามารถเข้าถึงได้โดยบุคคลภายนอก
- การทดสอบเชิงสมรรถนะเป็นอีกแง่มุมหนึ่งที่สำคัญ เพื่อให้ประมาณการได้ว่าผู้ใช้จำนวนเท่าใดจึงจะทำให้เว็บถึงขีดจำกัดของการบริการ

การเขียนการทดสอบระดับหน่วย

การเขียนการทดสอบระดับหน่วยทั่วๆไปจะทำเพื่อทดสอบเงื่อนไขหรือข้อกำหนดที่กระทำบนคลาสหลัก ซึ่งก็คือ โดเมนคลาส หรือ โมเดลตามหลักของ MVC

ตัวอย่างการเขียนการทดสอบระดับหน่วย

กำหนดให้มี User Story ดังต่อไปนี้

"ในบทบาทของพนักงาน เราต้องการเก็บข้อมูลเอกสาร โดยเอกสารมีรหัสเป็นเลข 8 ตัวและห้ามซ้ำกัน เพื่อให้สามารถค้นหาได้ภายหลัง"

จากตัวอย่าง User Story ข้างต้น จะสามารถเริ่มเขียนกรณีทดสอบระดับหน่วยได้ดังนี้

```
คลาส ทดสอบ_เอกสาร {
    กรณีทดสอบ_รหัสเอกสาร( ) {
        ตัวแปร เอกสาร1 = สร้างเอกสาร(รหัส: "12345678")
        ยืนยัน เอกสาร1.ตรวจสอบความถูกต้อง( ) = = จริง
        ยืนยัน เอกสาร1.รหัส = = "12345678"
    }
}
```

กรณีทดสอบแรกนี้เป็นการสร้างเพื่อทดสอบการทำงานที่เป็นปกติของระบบ คือ รับรหัสเข้าไปและควรจะบันทึกลงฐานข้อมูลได้อย่างถูกต้อง เมื่อนำกรณีทดสอบนี้ไปรันครั้งแรกจะปรากฏผลออกมาว่า "ไม่ผ่าน" เนื่องจาก

1. ยังไม่มีการสร้างคลาส "เอกสาร"
2. คลาส "เอกสาร" ยังไม่มีการประกาศรหัส

```
คลาส เอกสาร {
    รหัส เป็นข้อความ
}
```

จากกรณีทดสอบและคลาส เอกสาร จะทำให้การทดสอบปรากฏผลออกมาเป็น “ผ่าน” จากนั้นจึงจะเริ่มพิจารณาส่วนอื่นของ User Story ต่อไป ขั้นตอนถัดไป คือ การเขียนโปรแกรมสำหรับส่วน “รหัสเป็นตัวอักษร 8 ตัว” โดยเริ่มจากการเขียนกรณีทดสอบคลาส เอกสาร ให้รับรหัสเป็นตัวอักษร 8 ตัว โดยเพิ่มกรณีทดสอบเข้าไปยังคลาส ทดสอบ_เอกสาร

```
คลาส ทดสอบ_เอกสาร {
    กรณี ทดสอบ_รหัส( ) { ... }
    กรณี ทดสอบ_ความยาวรหัส( ) {
            ตัวแปร เอกสาร1 = สร้างเอกสาร(รหัส: ”1234567”)
            ยืนยัน เอกสาร1.ตรวจสอบความถูกต้อง( ) = = เท็จ
            ตัวแปร เอกสาร2 = สร้างเอกสาร(รหัส: ”123456789”)
            ยืนยัน เอกสาร2.ตรวจสอบความถูกต้อง( ) = = เท็จ
    }
}
```

เมื่อนำกรณีการทดสอบทั้งสองมารัน กรณีทดสอบ_รหัสเอกสารจะ “ผ่าน” แต่กรณีทดสอบ_ความยาวรหัสที่เขียนเพิ่มเข้าไปจะ “ไม่ผ่าน” เนื่องจากคลาสเอกสารในขณะนี้รับความยาวรหัสทุกแบบโดยไม่มีเงื่อนไข จึงต้องทำการแก้เงื่อนไขของรหัสของคลาส เอกสาร ให้รับข้อมูลความยาว 8 ตัวอักษรเท่านั้น ดังนี้

```
คลาส เอกสาร {
    รหัส เป็นข้อความ; ความยาว 8 เท่านั้น
}
```

เมื่อแก้ไขคลาสเอกสารแล้ว กรณีทดสอบทั้งสองจะให้ผลลัพธ์เป็น “ผ่าน” ขั้นตอนถัดไปเขียนโปรแกรมสำหรับส่วน “รหัสห้ามซ้ำกัน” เริ่มด้วยการเขียนกรณีทดสอบสำหรับคลาส เอกสาร ให้ป้องกันการที่เอกสารจะมีเลขซ้ำกัน

```
คลาส ทดสอบ_เอกสาร {
    กรณี ทดสอบ_รหัสเอกสาร ( ) {...}
    กรณี ทดสอบ_ความยาวรหัส ( ) {...}
    กรณี ทดสอบ_รหัสต้องไม่ซ้ำกัน ( ) {
            ตัวแปร เอกสาร1 = สร้าง เอกสาร(รหัส: ”12345678”)
            เอกสาร1.บันทึก( )
            ตัวแปร เอกสาร2 = สร้าง เอกสาร(รหัส: ”12345678”)
            ยืนยัน เอกสาร2.ตรวจความถูกต้อง( ) = = เท็จ
    }
}
```

เมื่อนำกรณีทดสอบทั้ง 3 กรณีไปรัน จะผ่าน 2 กรณีและกรณีทดสอบที่เพิ่มเข้าไปใหม่ คือ กรณีทดสอบ_รหัสต้องไม่ซ้ำ นั้นจะมีผลเป็น "ไม่ผ่าน" จากนั้นจึงต้องทำการแก้เงื่อนไขให้คลาส เอกสาร มีรหัสไม่ซ้ำกัน

```
คลาส เอกสาร {
    รหัส เป็น ข้อความ; ความยาว 8 เท่านั้น; ไม่ซ้ำ
}
```

เมื่อทำการรันกรณีทดสอบทั้ง 3 กรณีจะได้ผลลัพธ์เป็น "ผ่าน" ทั้งหมด ขั้นตอนถัดไปคือ เขียนโปรแกรมสำหรับส่วน "รหัสต้องเป็นตัวเลข" โดยเริ่มจากการเพิ่มกรณีทดสอบ_รหัสต้องเป็นตัวเลข เข้าไปยังคลาส ทดสอบ_เอกสาร

```
คลาส ทดสอบ_เอกสาร {
    กรณี ทดสอบ_รหัส( ) {...}
    กรณี ทดสอบ_ความยาวรหัส( ) {...}
    กรณี ทดสอบ_รหัสต้องไม่ซ้ำกัน( ) {...}
    กรณี ทดสอบ_รหัสต้องเป็นตัวเลข( ) {
        ตัวแปร เอกสาร1 = สร้าง เอกสาร(รหัส: "AB345678")
        ยืนยัน เอกสาร1.ตรวจสอบความถูกต้อง( ) = = เท็จ
    }
}
```

เมื่อสั่งทดสอบระดับหน่วยจะพบว่า กรณีทดสอบจะ "ผ่าน" 3 กรณีและ "ไม่ผ่าน" 1 กรณี เนื่องจากคลาส เอกสาร ยังไม่ได้รองรับการป้องกันการใส่รหัสที่เป็นตัวอักษรอื่น จึงจำเป็นต้องแก้ไข คลาส เอกสาร ดังนี้

```
คลาส เอกสาร {
    รหัส เป็น ข้อความ; ความยาว 8 เท่านั้น; ไม่ซ้ำ; ต้องเข้ากับ /\d+/
}
```

โดย /\d+/ เป็น regular expression และ \d คือ ตัวเลข และ \d+ คือ ตัวเลขมากกว่า 1 ตัวขึ้นไป เมื่อทำการแก้ไขคลาสเอกสารให้รับเฉพาะตัวเลขเข้าเป็นรหัสแล้ว จะทำการรันกรณีทดสอบอีกครั้ง ซึ่งจะได้ผลลัพธ์เป็น "ผ่าน" ทั้ง 4 กรณี เมื่อทดสอบเงื่อนไขครบถ้วนแล้ว จะได้โปรแกรมที่ทำงานได้และการทดสอบระดับหน่วยที่รับประกันว่าโปรแกรมทำงานได้ถูกต้องตาม User Story ที่กำหนด

วิธีการทดสอบเว็บแอพพลิเคชั่นโดยการใช้ Selenium

Selenium เป็นเครื่องมือทดสอบเว็บแอพพลิเคชั่น โดยมีโปรแกรม Selenium IDE สำหรับช่วยเตรียมกรณีทดสอบ Selenium IDE นั้นทำงานบนเว็บเบราเซอร์ Firefox ซึ่งสามารถทำการบันทึกการทำงานของผู้ใช้ รวมทั้งสร้างกรณีทดสอบขึ้นเองด้วยคำสั่ง

ตัวอย่างคำสั่งใน Selenium

คำสั่งใน Selenium ประกอบไปด้วย 3 ส่วน คือ

1. ชื่อคำสั่ง
2. เป้าหมายของคำสั่ง
3. ข้อความ

ชื่อคำสั่ง	เป้าหมายของคำสั่ง	ข้อความ
Open	/	
WaitForPageToLoad		
AssertTitle	Download	
ClickAndWait	xpath = id('header')/button	

เช่น คำสั่ง open มีเป้าหมายคือ / หมายถึงเปิดหน้าแรกของเว็บ คำสั่ง waitForPageToLoad เป็นคำสั่งที่ไม่มีพารามิเตอร์จึงไม่ต้องใช้ค่าเป้าหมาย

คำสั่งใน Selenium แบ่งออกได้เป็น 3 กลุ่ม คือ

1. Action ใช้สำหรับเปลี่ยนแปลงสถานะของโปรแกรมที่กำลังทดสอบ เช่น open, clickAndWait, type
2. Accessor ใช้สำหรับเข้าถึงสถานะบางส่วนของโปรแกรมแล้วเก็บค่าเข้าตัวแปร
3. Assertion ใช้สำหรับทวนสอบสถานะของโปรแกรมว่าเป็นไปตามที่คาดหวังหรือไม่ ใน Assertion ยังแบ่งออกได้เป็น 3 โหมด คือ
    a. โหมด assert ถ้าค่าที่ต้องการทดสอบไม่เป็นไปตามที่คาดหวัง การทดสอบที่กระทำอยู่จะถูกยกเลิก
    b. โหมด verify ถ้าค่าที่ต้องการทดสอบไม่เป็นไปตามที่คาดหวังการทดสอบจะกระทำต่อไปเรื่อยๆ
    c. โหมด waitFor เป็นโหมดที่การทดสอบจะคอยจนกว่าบางเงื่อนไขจะเป็นจริง ใช้มากในการทดสอบโปรแกรมประเภท AJAX

คำสั่งที่ใช้บ่อย

open สำหรับเปิดหน้าเว็บโดยใช้ URL

click สำหรับทำการคลิ๊ก

clickAndWait เหมือนคำสั่ง click แต่จะเพิ่มการรอให้หน้าเว็บโหลด

verifyTitle/assertTitle ตรวจสอบชื่อ (Title) ของหน้าเว็บ

verifyTextPresent ตรวจสอบว่ามีข้อความอยู่บนหน้าเว็บหรือไม่

verifyElementPresent ตรวจสอบว่ามีองค์ประกอบของ UI ที่ประกาศโดย HTML บนหน้าเว็บหรือไม่

verifyText ตรวจสอบข้อความและแท็กที่เกี่ยวข้อง

verifyTable ตรวจสอบข้อความในตาราง

verifyForPageToLoad หยุดการทดสอบชั่วคราว จนกระทั่งเว็บหน้าใหม่โหลดเรียบร้อยแล้วจึงทำการทดสอบต่อ

waitForElementPresent หยุดการทดสอบชั่วคราวจนกว่าองค์ประกอบ UI จะปรากฏบนหน้าเว็บ

การตรวจสอบองค์ประกอบบนหน้าเว็บ

การตรวจสอบองค์ประกอบบนหน้าเว็บเป็นสิ่งหลักๆที่ต้องทำด้วย Selenium องค์ประกอบบนหน้าเว็บสามารถอ้างถึงได้ด้วยวิธีการที่แตกต่างกัน

1. การอ้างถึงด้วย Identifier (ค่าใน Attribute "id" ของแต่ละแท็ก) สามารถระบุดังนี้ เช่น identifier = loginForm
2. การอ้างอิงด้วย name (ค่าใน Attribute "name" ของแต่ละแท็ก) สามารถระบุดังนี้ เช่น name = username

3. การอ้างอิงด้วย xpath ซึ่งเป็นเส้นทางระบุตำแหน่งในลักษณะต้นไม้เริ่มต้นจากราก (Root) ของไฟล์ HTML สามารถระบุได้ดังนี้ เช่น xpath = //form[@id = 'loginForm']
   การอ้างอิงด้วยวิธีการนี้มีประโยชน์ชัดเจนเมื่อองค์ประกอบที่ต้องการตรวจสอบไม่มีการระบุ id หรือ name ตามข้อ 1) หรือ 2)

ตัวอย่างการทวนสอบหน้าเว็บ

หน้า Facebook.com มีการล็อกอินด้วยอีเมล์และรหัสผ่าน เพื่อเข้าสู่ระบบการทดสอบ การทดสอบนี้เป็นการทดสอบในกรณีที่มีการใส่ Username และรหัสผ่านไม่ถูกต้องแล้วหน้าเว็บเปลี่ยนไปยังหน้าแจ้งข้อความผิดพลาด

Base URL http://www.facebook.com/

ชื่อคำสั่ง	เป้าหมายของคำสั่ง	ข้อความ
Open	/index.php	
Type	Email	user@nowhere.com
Type	Pass	test
clickAndWait	xpath = //input[@value = 'Login']	
AssertTextPresent	Incorrect Email	

บรรทัดที่ 1 ทำการเปิดเว็บ http://www.facebook.com/index.php

บรรทัดที่ 2 พิมพ์คำว่า user@nowhere.com ลงในช่องอีเมล์ในบรรทัดนี้ Identifier เป็นตัวระบุตำแหน่ง เป็นตัวระบุตำแหน่งให้กับการพิมพ์

บรรทัดที่ 3 พิมพ์คำว่า test ลงในช่องรหัสผ่านใช้ Identifier เพื่อระบุตำแหน่งเช่นกัน

บรรทัดที่ 4 คลิกปุ่ม Login แล้วรอโหลดหน้าใหม่ โดยใช้ตัวระบุตำแหน่งของปุ่มเป็น xpath = //input[@value = 'Login'] หมายความว่า ปุ่มที่ต้องการเป็นแท็ก input ที่มี Attribute 'value' เป็นค่า "Login" เนื่องจากไม่สามารถใช้ id ในการระบุตำแหน่งของปุ่มได้

บรรทัดที่ 5 เป็นการยืนยันว่าการเข้าระบบไม่ได้นั้นถูกตามขั้นตอนหรือไม่โดยการตรวจสอบคำว่า "Incorrect Email" บนหน้าเว็บที่เปิดขึ้นใหม่จากการคลิกปุ่ม Login

บทที่ 11

การนำไปใช้และการบำรุงรักษาซอฟต์แวร์

Software Deployment and Maintenance

การนำไปใช้

การนำไปใช้คือ กิจกรรมที่นำระบบซอฟต์แวร์ที่พัฒนาขึ้นไปติดตั้งเพื่อการใช้งาน เป็นกระบวนการที่เกี่ยวข้องกับหลายๆกิจกรรม เช่น การปล่อยซอฟต์แวร์ การปรับปรุงรุ่น เป็นต้น โดยกิจกรรมย่อยๆนั้นขึ้นกับซอฟต์แวร์นั้นๆ เนื่องจากในทางปฏิบัติแล้วการนำซอฟต์แวร์ไปใช้มีความแตกต่างกันค่อนข้างมาก และมักขึ้นกับเครื่องมือทางวิศวกรรมซอฟต์แวร์ที่นำมาจัดการกระบวนการด้วย

กิจกรรมในกระบวนการนำไปใช้มีดังต่อไปนี้

1. การปล่อยซอฟต์แวร์

การปล่อยซอฟต์แวร์เป็นสิ่งที่กระทำหลังจากกระบวนการพัฒนาเสร็จสิ้นลง การปล่อยซอฟต์แวร์เป็นขั้นตอนการเตรียมความพร้อมของระบบและประกอบทุกอย่างเข้าด้วยกัน ก่อนนำตัวระบบไปจัดจุดติดตั้งของผู้ใช้ ในกระบวนการนี้จึงมีความจำเป็นในการจัดสรรทรัพยากรบุคคลที่ต้องไปทำงาน ณ จุดติดตั้งโปรแกรม รวมทั้งดำเนินกิจกรรมอื่นๆเพื่อสนับสนุนการนำไปใช้ของลูกค้า

2. การติดตั้งและเปิดใช้งาน

การเปิดใช้งานเป็นกิจกรรมเพื่อเริ่มต้นใช้งานชิ้นส่วนของซอฟต์แวร์ในระบบ โดยปกติการติดตั้งและเปิดใช้งานของระบบขนาดเล็กจะเป็นการใช้คำสั่งเพื่อติดตั้ง เช่น ระบบจัดการทรัพยากรวิสาหกิจ อาจใช้เครื่องแม่ข่ายหลายเครื่องสำหรับฐานข้อมูล และแอพพลิเคชั่นแม่ข่าย กรณีที่เป็นระบบขนาดใหญ่มาก เช่น Twitter ที่มีซอฟต์แวร์กระจายตัวอยู่ตามเครื่องต่างๆในระดับพัน หรือหมื่นเครื่อง กลไกการติดตั้งจะต่างออกไป ซึ่งอาจจะใช้ระบบกระจายประเภท Peer-to-Peer ช่วยในการติดตั้ง เป็นต้น

3. การปิดใช้งาน

การปิดใช้งานเป็นกิจกรรมที่หมายถึงการปิดใช้ชิ้นส่วนของซอฟต์แวร์ในระบบ กิจกรรมนี้มักจะทำเพื่อให้สามารถดำเนินกิจกรรมอื่นๆต่อไปได้ เช่น ต้องปิดการใช้งานก่อนจึงจะปรับปรุงชิ้นส่วนของซอฟต์แวร์ที่เกี่ยวข้องได้ เป็นต้น

4. การดัดแปลง

การดัดแปลงเป็นกระบวนการในการปรับเปลี่ยนระบบซอฟต์แวร์ที่ติดตั้งอยู่ก่อนแล้วให้สามารถทำงานได้ในสภาวะการทำงานอื่นๆ ที่ต้องการ เช่น ปรับชิ้นส่วนซอฟต์แวร์ภายนอกบางชิ้น หรือการเปลี่ยนรุ่นของระบบปฏิบัติการที่ซอฟต์แวร์ทำงานอยู่ เป็นต้น

การดัดแปลงจะต่างจากการปรับปรุงตรงที่การปรับปรุงจะเริ่มจากการติดตั้งซอฟต์แวร์ที่นำมาจากผู้พัฒนา

5. การปรับปรุง

การปรับปรุงเป็นการเปลี่ยนหรือแทนที่ซอฟต์แวร์รุ่นก่อนหน้าด้วยรุ่นที่ใหม่กว่า โดยทั่วไปจะมีทั้งการแทนที่ทั้งระบบซอฟต์แวร์หรือแทนที่เฉพาะบางส่วน

6. การติดตามรุ่น

ระบบติดตามรุ่นจะช่วยให้ผู้ใช้งานระบบในการค้นหาและติดตั้งการปรับปรุงของชิ้นส่วนซอฟต์แวร์ของระบบ โดยปกติระบบในลักษณะนี้จะสร้างสำหรับแอพพลิเคชั่นประเภทตั้งโต๊ะ ตัวอย่างเช่น

เว็บเบราเซอร์ Google Chrome มีระบบติดตามรุ่นที่ตรวจสอบและดาวน์โหลดซอฟต์แวร์รุ่นใหม่ทำงานอยู่เบื้องหลังขณะผู้ใช้ใช้งานเบราเซอร์

7. การยกเลิกการติดตั้ง

การยกเลิกการติดตั้งเป็นกิจกรรมที่ตรงข้ามกับการติดตั้ง โดยการลบชิ้นส่วนหนึ่งหรือทั้งหมดของระบบออกเมื่อไม่ต้องการใช้แล้ว สิ่งที่ต้องระมัดระวังคือ การยกเลิกการติดตั้งอาจกระทบกับการปรับแต่งของระบบโดยรวม โดยชิ้นส่วนของซอฟต์แวร์ส่วนอื่นอาจใช้งานไม่ได้หากส่วนใดส่วนหนึ่งของระบบถูกลบออกจากการยกเลิกการติดตั้งซอฟต์แวร์อื่นได้

8. การสิ้นสุดการสนับสนุน

การสิ้นสุดการสนับสนุนเป็นกลไกการกำหนดให้ซอฟต์แวร์นั้นล้าสมัยและประกาศไม่สนับสนุนต่อโดยผู้พัฒนา ซอฟต์แวร์หลายตัว เช่น Windows XP หรือ Java Development Kit 1.4 มีลักษณะเป็นซอฟต์แวร์ที่สิ้นสุดการสนับสนุนแล้ว เป็นต้น

บทบาทในกระบวนการนำไปใช้

เนื่องจากกระบวนการนำไปใช้เกี่ยวข้องโดยตรงกับความหลากหลายของผลิตภัณฑ์ เมื่อต้องจัดการความซับซ้อนที่เกิดขึ้น ทำให้มีการกำหนดบทบาทเฉพาะขึ้นสำหรับจัดการกระบวนการนำไปใช้

ผู้พัฒนาแอพพลิเคชั่น

ผู้พัฒนาแอพพลิเคชั่นเป็นบทบาทในระยะ Pre-production โดยผู้พัฒนามีหน้าที่สร้างระบบตามการวิเคราะห์และการออกแบบ

วิศวกรสร้างและปล่อยซอฟต์แวร์

วิศวกรสร้างและปล่อยซอฟต์แวร์เป็นบทบาทที่มีหน้าที่คอมไพล์และประกอบต้นรหัสให้อยู่ในรูปของผลิตภัณฑ์ที่เสร็จสมบูรณ์ โดยมีแง่มุมที่น่าสนใจในกระบวนการ เช่น ความสามารถในการทำซ้ำ เป็นการประกอบต้นรหัสชิ้นส่วนซอฟต์แวร์อื่น รวมถึงข้อมูลและระบบภายนอกเข้าด้วยกัน เพื่อให้สามารถรับประกันความมีเสถียรภาพในการทำงานของระบบได้ หรือความสอดคล้องซึ่งเป็นการสร้างกรอบงานที่มีเสถียรภาพสำหรับการพัฒนาและประกอบชิ้นส่วนของซอฟต์แวร์เข้าด้วยกัน บทบาทนี้อยู่ในระยะ Pre-production

ผู้จัดการการปล่อยซอฟต์แวร์

ผู้จัดการการปล่อยซอฟต์แวร์เป็นผู้ทำหน้าที่ประสานงานระหว่างแต่ละหน่วยในการพัฒนาเพื่อให้การปล่อยซอฟต์แวร์และชุดของการปรับปรุงสามารถนำส่งลูกค้าได้อย่างเรียบร้อยและทันเวลา รวมทั้งมีส่วนช่วยในการตั้งกระบวนการเพื่อให้การปล่อยซอฟต์แวร์เป็นไปอย่างมีประสิทธิภาพ บทบาทนี้อยู่ในระยะ Pre-production ประเด็นสำคัญที่ต้องคำนึงถึงในบทบาทของผู้จัดการการปล่อยซอฟต์แวร์ อาจรวมถึงทรัพย์สินทางปัญญาที่เกี่ยวข้อง ความเสี่ยง การเปลี่ยนแปลงที่ลูกค้าต้องการเร่งด่วน การพัฒนาคุณลักษณะเพิ่มเติม เป็นต้น

ผู้ดูแลระบบ

ผู้ดูแลระบบเป็นผู้มีหน้าที่ดูแลรักษาระบบคอมพิวเตอร์ ซึ่งอาจรวมถึงระบบเครือข่าย บทบาทนี้มีหน้าที่อยู่ในระยะ Production โดยต้องรับผิดชอบดูแลให้ระบบของลูกค้าสามารถทำงานได้อย่างราบรื่น และผู้ดูแล

ระบบอาจมีความจำเป็นต้องแก้ปัญหาเฉพาะหน้า เมื่อเกิดเหตุการณ์ที่ไม่คาดคิดขึ้น เทคโนโลยีในปัจจุบัน เช่น Virtualization มีบทบาทในการลดความซับซ้อนของการดูแลระบบ

ผู้ดูแลระบบฐานข้อมูล

ผู้ดูแลระบบฐานข้อมูลมีหน้าที่รับผิดชอบในการบำรุงรักษาและดูแลระบบฐานข้อมูลของซอฟต์แวร์รวมถึงการทำงานร่วมกับผู้พัฒนาในการกำหนดสิทธิ์การเข้าถึงฐานข้อมูลแต่ละส่วน ผู้ดูแลฐานข้อมูลอาจมีหน้าที่เพิ่มเติม เช่น การสำรองและการกู้คืนข้อมูลของระบบ รวมทั้งการเตรียมทรัพยากรระดับฮาร์ดแวร์ เพื่อทำซ้ำข้อมูลขณะทำงานจริงเพื่อรับจำนวนผู้ใช้ที่อาจจะเพิ่มขึ้น เป็นต้น บทบาทนี้เป็นบทบาทในระยะ Production

ผู้ประสานงานการปล่อยซอฟต์แวร์

ผู้ประสานงานการปล่อยซอฟต์แวร์เป็นบทบาทที่มีหน้าที่ประสานงานการนำซอฟต์แวร์ไปใช้จากระยะ Pre-production ไปสู่ระยะ Production โดยจะประสานงานระหว่างกลุ่มทำงานหลายกลุ่มเพื่อให้เป้าหมายในการนำซอฟต์แวร์ไปใช้นั้นบรรลุผลได้ ผู้ประสานงานการปล่อยซอฟต์แวร์ต่างจากผู้จัดการการปล่อยซอฟต์แวร์ในจุดที่ผู้จัดการการปล่อยซอฟต์แวร์มักเน้นไปยังการวางแผนรับมือต่อการเปลี่ยนแปลงของซอฟต์แวร์ บทบาทนี้อยู่ในระยะ Production

การบำรุงรักษาซอฟต์แวร์

การบำรุงรักษาซอฟต์แวร์เป็นกระบวนการทางวิศวกรรมซอฟต์แวร์ เพื่อแก้ไขข้อผิดพลาดของซอฟต์แวร์ที่ส่งมอบแล้วให้สามารถทำงานได้ถูกต้อง หรือเพื่อทำการปรับปรุงประสิทธิภาพของซอฟต์แวร์และอาจรวมถึงคุณสมบัติด้านอื่นๆ เช่น การแก้ไขช่องโหว่ที่มีผลต่อความมั่นคงของซอฟต์แวร์ เป็นต้น

กระบวนการในการบำรุงรักษาซอฟต์แวร์แบ่งออกเป็นขั้นตอนดังนี้

1. วางแผนกระบวนการเตรียมความพร้อมของซอฟต์แวร์และถ่ายโอนซอฟต์แวร์ วางแผนการบำรุงรักษาการเตรียมการสำหรับจัดการปัญหาที่จะพบระหว่างการพัฒนาและติดตามการจัดการ Configuration
2. วิเคราะห์ปัญหาและการแก้ไข นักพัฒนาที่รับผิดชอบการบำรุงรักษาจะทำการวิเคราะห์ การร้องขอจากลูกค้าเพื่อเปลี่ยนแปลงตัวซอฟต์แวร์ จากนั้นทำการยืนยันว่ามีปัญหาเกิดขึ้นจริงตามการร้องขอโดยการทำซ้ำปัญหานั้นๆ และวิเคราะห์ว่าปัญหาดังกล่าวสมเหตุสมผลหรือไม่ เมื่อวิเคราะห์แล้วนักพัฒนาจะเสนอทางแก้และสร้างต้นรหัสเพื่อแก้ปัญหานั้นเพื่อขออนุมัติการแก้ไข
3. พิจารณาการดำเนินการแก้ไข
4. ยอมรับกระบวนการแก้ไขโดยทำการยืนยันว่าสิ่งที่เปลี่ยนแปลงในระบบสามารถแก้ไขปัญหาได้จริง
5. การบำรุงรักษาอาจรวมถึงการย้ายแอพพลิเคชั่นข้ามแพล็ตฟอร์ม ซึ่งเป็นกระบวนการพิเศษที่อยู่นอกเหนือการบำรุงรักษาปกติ
6. การสิ้นสุดการสนับสนุนของซอฟต์แวร์ก็เป็นกระบวนการบำรุงรักษาเช่นกัน โดยเป็นสิ่งที่เกิดขึ้นเพียงครั้งเดียวต่อซอฟต์แวร์ 1 ระบบ เพื่อถอนการติดตั้งและประการยกเลิกการสนับสนุนโดยผู้พัฒนา

ประเภทของการบำรุงรักษาซอฟต์แวร์

1. การบำรุงรักษาเชิงแก้ไข คือ การเปลี่ยนแปลงที่กระทำต่อระบบหลังส่งมอบแล้ว เพื่อแก้ไขปัญหาโดยตรงหลังจากพบแล้วว่าปัญหาคืออะไร
2. การบำรุงรักษาเชิงดัดแปลงเป็นการเปลี่ยนแปลงที่กระทำต่อระบบหลังส่งมอบแล้ว เพื่อให้ซอฟต์แวร์สามารถทำงานได้ในสภาพแวดล้อมที่เปลี่ยนไปจากเดิม

3. การบำรุงรักษาเชิงสมบูรณ์ คือการเปลี่ยนแปลงที่กระทำต่อระบบหลังส่งมอบแล้วเพื่อทำให้ระบบสมบูรณ์ยิ่งขึ้น เช่น การปรับปรุงประสิทธิภาพหรือเพิ่มความสามารถที่ช่วยในการบำรุงรักษาเข้าสู่ตัวระบบ

4. การบำรุงรักษาเชิงป้องกัน คือ การเปลี่ยนแปลงที่กระทำต่อระบบหลังส่งมอบแล้ว เพื่อตรวจสอบหรือแก้ไขข้อผิดพลาดที่แฝงอยู่ก่อนที่จะกลายเป็นข้อผิดพลาดร้ายแรงที่อาจก่อให้เกิดความเสียหายต่อระบบได้

บรรณานุกรม

1. Agile Manifesto. http://agilemanifesto.org. Last Accessed February 2012.
2. Amber, S. **Agile Modeling**. http://agilemodeling.com. Last Accessed February 2012.
3. Chacon, S. **Pro Git**. APress Publishing. 2009.
4. Clemmons, R. K. **Project Estimation with Use Case Points**. The Journal of Defense Software Engineering. 2006.
5. Cohn, M. **Succeeding with Agile**: Software Development Using Scrum. 2009.
6. Deemer, P., Benefield G., and Larman, C. **The Scrum Primer Version 1.1**. 2008.
7. Lang, J-P. **Redmine Project**. http://www.redmine.org. Last Accessed February 2012.
8. OpenUP. http://eclipse.org/epf. Last Accessed October 2011.
9. Pressman, R. S. **Software Engineering : A Practitioner's Approach** 6th Edition.
10. Redmine Backlogs. http://www.redminebacklogs.net. Last Accessed February 2012.
11. Schwaber, K. and Sutherland, J. **The Scrum Guide**. October 2011.
12. Scrum.org. http://scrum.org. Last Accessed February 2012.