Software Enginnering

เอกสารคำสอน

วิศวกรรมซอฟต์แวร์

https://gear.kku.ac.th/~witcha/Witcha

ผศ.ดร.วิชชา เฟื่องจันทร์

Software Enginnering

เอกสารคำสอน

วิศวกรรมซอฟต์แวร์

https://gear.kku.ac.th/~witcha/Witcha

ผศ.ดร.วิชชา เฟื่องจันทร์

กันยายน 2568

EN-813-306

Software Engineering

วิศวกรรมซอฟต์แวร์ 3(3-0-6)

วิศวกรรมซอฟต์แวร์ขั้นแนะนำ กระบวนการพัฒนาซอฟต์แวร์ เครื่องมือที่ช่วยในงานวิศวกรรมซอฟต์แวร์ การกำหนดความต้องการและข้อกำหนดของซอฟต์แวร์ การแปลภาษา การออกแบบซอฟต์แวร์ การบริหารโครงการพัฒนาซอฟต์แวร์ การทดสอบและการตรวจสอบความสมเหตุสมผลของซอฟต์แวร์ความทนทานต่อข้อผิดพลาดของซอฟต์แวร์ พัฒนาการซอฟต์แวร์ จรรยาบรรณวิศวกรรมซอฟต์แวร์

Introduction to software engineering, software processes, software tools and environments, software requirements and specifications, language translation, software design, software project management, software testing and validation, software fault tolerance, software evolution, software engineering ethic

2.1 วัตถุประสงค์ของรายวิชา

1.1. สามารถวิเคราะห์รายละเอียดและความต้องการของซอฟต์แวร์ที่พัฒนา

1.2. สามารถออกแบบซอฟต์แวร์และพัฒนาส่วนต่าง ๆ ของซอฟต์แวร์

1.3. สามารถทำงานเป็นทีมและควบคุมกระบวนการพัฒนาซอฟต์แวร์ให้มีคุณภาพได้

5.1 ตำราและเอกสารหลัก

I. Sommerville, Software Engineering, 10th edition. Harlow; Singapore: Pearson, 2015.

R. Pressman and B. Maxim, Software Engineering: A Practitioner’s Approach. 2019.

เอกสารคำสอน วิศวกรรมซอฟต์แวร์

ตำราอ่านประกอบ

Brooks, Frederick P., Jr.: "The Mythical Man-Month, Essays on Software Engineering", Addison-Wesley, 1975.

Schach, Stephen R.: "Object-Oriented and Classical Software Engineering",5th ed., McGraw-Hill, 2002.

เว็บไซต์

Software Engineering (http://gear.kku.ac.th/~witcha/Witcha/Courses.html)

สารบัญ

บทที่ 1 วิศวกรรมซอฟต์แวร์ขั้นแนะนำ Introduction to software engineering

วัตถุประสงค์

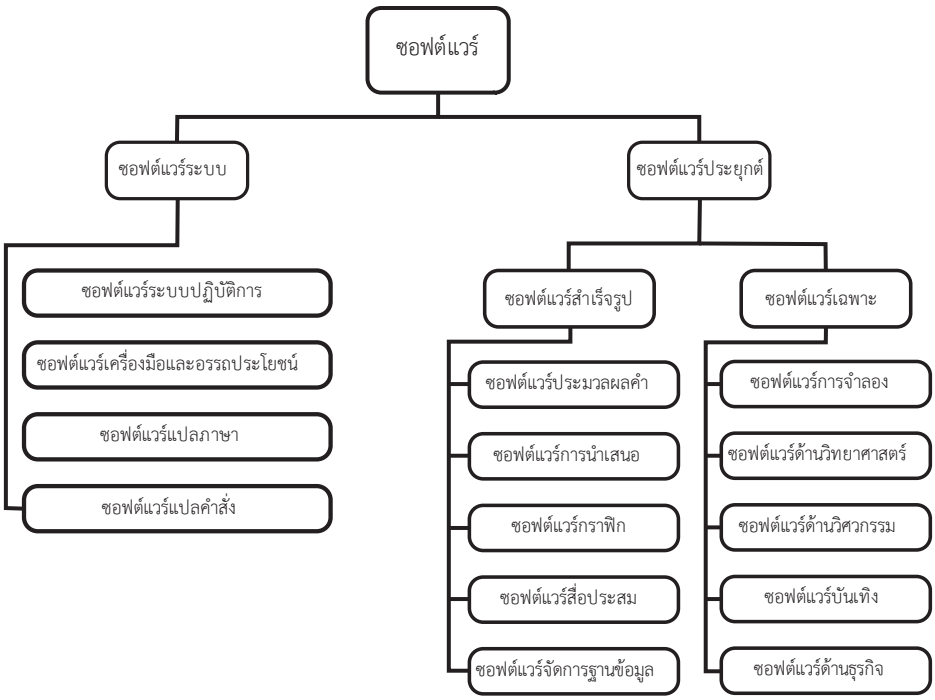
- เข้าใจความหมายและความสำคัญของวิศวกรรมซอฟต์แวร์
- เข้าใจความหมายซอฟต์แวร์ และกระบวนการพัฒนาซอฟต์แวร์
- เข้าใจจริยธรรมในวิศวกรซอฟต์แวร์

ปัจจุบันเราอาศัยอยู่ในโลกยุคที่การประมวลผลอยู่ทุกหนทุกแห่งรอบ ๆ ตัวเราและเกี่ยวข้องกับการใช้ชีวิตประจำวันของผู้คนส่วนใหญ่แทบทุกเพศ ทุกวัย ทุกอาชีพ ต้องเกี่ยวของกับสิ่งที่เรียกว่า “ซอฟต์แวร์” หรือโปรแกรมที่ทำงานบนเครื่องคอมพิวเตอร์หรืออุปกรณ์ประมวลผลรูปแบบต่าง ๆ เช่น โทรศัพท์เคลื่อนที่ นาฬิกา รถยนต์ และ อื่น ๆ อีกมากมายรอบตัวเรา หากปราศจากซอฟต์แวร์แล้ว เครื่องประมวลผลก็ไม่สามารถทำงานเพื่อตอบสนองการใช้งานหรืออำนวยความสะดวกต่าง ๆ ให้มนุษย์ การพัฒนาซอฟต์แวร์จึงเป็นอีกศาสตร์ที่สำคัญว่าด้วยการสร้างสรรค์ผลิตภัณฑ์ซอฟต์แวร์ที่มีประสิทธิภาพ ถูกต้อง ปลอดภัย เชื่อถือได้ สะดวกง่ายต่อการใช้งาน ทันสมัย ปรับเปลี่ยนง่าย และนำกลับมาใช้งานใหม่ได้ ซึ่งหนึ่งในศาสตร์ที่เกี่ยวกับการสร้างผลิตภัณฑ์ซอฟต์แวร์นั้นคือ วิศวกรรมซอฟต์แวร์และนั่นคือเนื้อหาหลักของเอกสารคำสอนนี้

ซอฟต์แวร์

ซอฟต์แวร์คือ ส่วนชุดคำสั่ง หากแปลตามราชบัณฑิตยสภา แต่คำนิยามโดยรวมของซอฟต์แวร์คือ โปรแกรมคอมพิวเตอร์หรือชุดคำสั่งที่เขียนด้วยภาษาคอมพิวเตอร์ เพื่อใช้ควบคุมการทำงานของเครื่องคอมพิวเตอร์ตามชุดคำสั่ง โดยปกติซอฟต์แวร์จะมีลักษณะดังต่อไปนี้ เป็นเพียงกลุ่มชุดคำสั่งดิจิทัลที่จับต้องไม่ได้ นับเป็นทรัพย์สินทางปัญญา สามารถทำซ้ำได้ง่าย กระจายไปใช้งานบนเครื่องคอมพิวเตอร์ต่าง ๆ ซอฟต์แวร์ไม่เสื่อมแต่อาจล้าสมัย ใช้เพียงเครื่องคอมพิวเตอร์และเครื่องมือพัฒนาในการผลิตซอฟต์แวร์ ต้นทุนในการผลิตส่วนใหญ่เป็นค่าแรงงานสมองของนักพัฒนา โดยปกติซอฟต์แวร์ที่ยังคงมีการใช้งานจะถูกพัฒนาให้ทันสมัยและตอบสนองความต้องการที่เปลี่ยนแปลงอยู่เสมอ ซอฟต์แวร์มีทั้งขนาดเล็กขนาดใหญ่ เรียบง่ายจนถึงซับซ้อนมาก

สำหรับวิศวกรรมซอฟต์แวร์นิยามซอฟต์แวร์คือ โปรแกรมคอมพิวเตอร์ที่ใช้ในการประมวลผลโดยคอมพิวเตอร์ ครอบคลุมทั้ง โปรแกรม ชุดคำสั่ง ข้อมูล คลังโปรแกรม เอกสาร และ สื่อดิจิทัล เราสามารถแบ่งซอฟต์แวร์ออกเป็น 2 ประเภท คือ ซอฟต์แวร์ระบบและซอฟต์แวร์ประยุกต์



รูปที่ 1.1 ประเภทของซอฟต์แวร์

ซอฟต์แวร์ระบบ

ซอฟต์แวร์ที่ใช้ในการควบคุมการทำงานหลักของเครื่องคอมพิวเตอร์ ได้แก่ซอฟต์แวร์ระบบปฏิบัติการ (operating system) และซอฟต์แวร์เครื่องมือพัฒนาซอฟต์แวร์ ได้แก่ ซอฟต์แวร์เครื่องมือและอรรถประโยชน์ (tools and utilities) ซอฟต์แวร์แปลภาษา (compilers) ซอฟต์แวร์แปลคำสั่ง (interpreter)

ซอฟต์แวร์ระบบปฏิบัติการ (operating system) หรือเรียกกันสั้น ๆ ว่า ระบบปฏิบัติการคอมพิวเตอร์คือ ซอฟต์แวร์สำคัญพื้นฐานที่ทำหน้าที่ควบคุมการทำงานหลักของคอมพิวเตอร์ บริหารจัดการทรัพยากรของเครื่องคอมพิวเตอร์ ติดต่อประสานระหว่างผู้ใช้ จัดการข้อมูล เหล่านี้เป็นหน้าที่หลักของระบบปฏิบัติการ เราจะสังเกตว่าเมื่อเครื่องคอมพิวเตอร์เริ่มเปิดขึ้นมา โปรแกรมเริ่มต้นหรือไบออส (bios) จะถูกโหลดขึ้นมาไว้ในหน่วยความจำหลัก (RAM) เพื่อเรียกใช้งานการตรวจสอบและเตรียมความพร้อมเบื้องต้นก่อนที่จะเริ่ม กระบวนการโหลด ระบบปฏิบัติการที่จากอุปกรณ์จัดเก็บข้อมูลหรือสื่อดิจิทัล มาทำงานบนหน่วยความจำหลักของคอมพิวเตอร์และเริ่มการทำงานของระบบปฏิบัติการ ซึ่งระบบปฏิบัติการจะทำหน้าที่ดูแลงานและบริการต่าง ๆ ของคอมพิวเตอร์ ทั้งยังอนุญาติให้ซอฟต์แวร์ระบบหรือซอฟต์แวร์ประยุกต์ทำงานอยู่บนระบบปฏิบัติการได้ดังแสดงในรูปที่ 1.1 ตัวอย่างระบบปฏิบัติการเช่น Microsoft Windows OSX IOS Android และ Linux เป็นต้น

ซอฟต์แวร์เครื่องมือและอรรถประโยชน์ (tools and utilities) เป็นซอฟต์แวร์ที่ช่วยจัดการงานพื้นฐานต่างๆซึ่งมักจะถูกจัดเตรียมไว้ในกับระบบปฏิบัติการเพื่อสนับสนุนการใช้งานพื้นฐานต่าง ๆ บนระบบปฏิบัติการ เช่น Windows Explorer ช่วยให้ผู้ใช้สามารถจัดการแฟ้มข้อมูลต่าง ๆ ได้อย่างสะดวก ไม่ว่าจะเป็นการลบคัดลอกหรือเปลี่ยนชื่อแฟ้มข้อมูล หรือซอฟต์แวร์จัดการอุปกรณ์จัดเก็บข้อมูลที่ทำหน้าที่ บริหารจัดการสื่อบันทึกข้อมูลไม่ว่าจะเป็นการ สร้าง ลบ จัดรูปแบบ เช่น Disk Utilities บนระบบปฏิบัติการ Apple OSX

ซอฟต์แวร์แปลภาษา (compiler) เป็นซอฟต์แวร์ที่ทำหน้าที่แปลงชุดคำสั่งที่เขียนโดยนักพัฒนาไปเป็นชุดคำสั่งที่เครื่องคอมพิวเตอร์สามารถเข้าใจเรียกว่ารหัสเครื่อง (machine code) ซอฟต์แวร์แปลภาษาจึงทำหน้าที่เป็นตัวกลางในการแปลจากภาษาที่นักพัฒนาใช้เขียนโปรแกรม ไปเป็นรหัสเครื่องเพื่อให้สามารถทำงานกับเครื่องคอมพิวเตอร์ได้โดย ซอฟต์แวร์แปลภาษานั้นก็มีหลากหลายแตกต่างกันไปขึ้นอยู่กับภาษาที่พัฒนา เช่นถ้านักพัฒนาด้วยภาษาซี (C language) จำเป็นต้องใช้ซอฟต์แวร์แปลภาษาที่รองรับการแปลภาษาซี มีหลักการทำงานคือดำเนินการแปลจากรหัสต้นฉบับ (source code) ไปไปเป็นภาษาเครื่องในคราวเดียวและสามารถนำโปรแกรมที่แปลเสร็จแล้วไปกระทำการ (execute) บนเครื่องคอมพิวเตอร์ได้ทันที

ซอฟต์แวร์แปลคำสั่ง (interpreter) เป็นซอฟต์แวร์ที่ทำงานในลักษณะเดียวกันกับซอฟต์แวร์แปลภาษา เพียงแต่ต่างกันตรงที่ซอฟต์แวร์แปลคำสั่งจะแปลงรหัสต้นฉบับที่นักพัฒนาเขียนทีละชุดคำสั่งไปเป็นภาษาเครื่อง ณ ขณะที่ทำการกระทำการโปรแกรมโดยในเครื่องคอมพิวเตอร์ที่จะใช้โปรแกรมที่ใช้งานร่วมกับซอฟต์แวร์แปลงคำสั่งจำเป็นต้องมีการติดตั้งซอฟต์แวร์แปลงคำสั่งหรือเครื่องเสมือน (virtual machine) ในคอมพิวเตอร์นั้นด้วย เช่น การใช้โปรแกรม ที่พัฒนาด้วย JAVA จำเป็นต้องกระทำการบน JVM (JAVA virtual machine)

ซอฟต์แวร์ประยุกต์

ซอฟต์แวร์ที่ผู้ใช้นำมาใช้ช่วยทำงานด้านต่างๆตามความต้องการและวัตถุประสงค์ของงานหรือปัญหา โดยสามารถแบ่งซอฟต์แวร์ประยุกต์ออกเป็นประเภทหลัก ๆ ได้ดังนี้

ซอฟต์แวร์ประมวลคำ (word processing) เป็นซอฟต์แวร์สำหรับจัดการตัวหนังสือโดยรับเข้าผ่านทางแป้นพิมพ์ และนำไปแสดงผลผ่านจอแสดงผล และสามารถเพิ่มลบแก้ไขและบันทึกข้อมูล จัดรูปแบบการแสดงผล การพิมพ์ ยกตัวอย่างซอฟต์แวร์ประมวลผลคำ Microsoft Word

ซอฟต์แวร์ตารางคำนวณ (spreadsheet) เป็นซอฟต์แวร์ที่ใช้ในการประมวลข้อมูลในรูปแบบตารางคำนวณ ที่มาพร้อมฟังก์ชัน (function) ช่วยการคำนวณ ประมวลผล โดยสามารถแสดงผลลัพธ์ในรูปแบบต่าง ๆ เช่นตาราง หรือแผนภูมิแบบต่าง ๆ ยกตัวอย่างซอฟต์แวร์ตารางคำนวณ Microsoft Excel

ซอฟต์แวร์การนำเสนอ (presentation) เป็นซอฟต์แวร์ที่ใช้ในการนำเสนอข้อมูลในรูปแบบแผ่นภาพเลื่อน (slide) ที่สามารถแทรกสารสนเทศและสื่อดิจิทัลต่างๆเช่นแผนภูมิ ภาพ ภาพเคลื่อนไหว วิดิทัศน์ และเสียงประกอบการนำเสนอ ตัวอย่างซอฟต์แวร์นำเสนอ Microsoft PowerPoint

ซอฟต์แวร์กราฟิก (graphic) เป็นซอฟต์แวร์ที่ใช้ในการสร้างหรือแก้ไขกราฟิกทั้งแบบแรสเตอร์และแบบเวกเตอร์กราฟิกส์ ตัวอย่างซอฟต์แวร์ Adobe Photoshop Adobe Illustrator

ซอฟต์แวร์สื่อประสม (multimedia) เป็นซอฟต์แวร์ที่ใช้ในการสร้างหรือแก้ไขหรือเล่นสื่อประสม โดยส่วนมากถูกใช้เพื่อความบันเทิงเช่น ซอฟต์แวร์เล่นวิดิทัศน์ หรือเล่นเพลง ตัวอย่างซอฟต์แวร์สื่อประสมที่ใช้อย่างแพร่หลาย VLC media player

ซอฟต์แวร์เว็บเบราว์เซอร์ (web browser) เป็นซอฟต์แวร์ที่ใช้ในการแสดงผลและทำงานกับเว็บ ซึ่งในปัจจุบันเป็นซอฟต์แวร์ประยุกต์ที่ถูกใช้อย่างแพร่หลายมาก เมื่อการใช้งานคอมพิวเตอร์ส่วนมากมีการใช้งานบนเครือข่ายอินเทอร์เน็ต ไม่ว่าจะเป็นการ

ทำงานหรือสืบค้นสารสนเทศ และในปัจจุบันซอฟต์แวร์ให้บริการต่างๆรองรับการทำงานบน เว็บเบราว์เซอร์ไม่ว่าจะเป็นการท่องเว็บ เข้าถึงจดหมายอิเล็กทรอนิกส์ ชมวิดิทัศน์ ฟังเพลง ล้วนสามารถทำงานได้บนเว็บเบราว์เซอร์นับเป็นอีกรูปแบบของโปรแกรมประยุกต์ผ่านบริการออนไลน์ ตัวอย่างซอฟต์แวร์เว็บเบราว์เซอร์ที่ใช้อย่างแพร่หลาย Google Chrome และ Safari

ซอฟต์แวร์จัดการฐานข้อมูล (database management) เป็นซอฟต์แวร์ที่ใช้ในการบริหารจัดการข้อมูลโดยการจัดเก็บข้อมูลไว้ในฐานข้อมูล และสามารถเข้าถึงเพื่อแก้ไขหรืออ่านข้อมูลที่เก็บไว้ ซอฟต์แวร์จัดการฐานข้อมูลที่ใช้อย่างแพร่หลาย MySQL

ซอฟต์แวร์ประยุกต์อื่น ๆ ยังมีอีกมากมายทั้งแบบที่สร้างขึ้นมาเพื่อผู้ใช้ทั่วไป หรือออกแบบมาเฉพาะกลุ่มบุคคล โดยวัตถุประสงค์ส่วนใหญ่คือการสร้างซอฟต์แวร์ประยุกต์เพื่อตอบสนองความต้องการใช้งานของผู้ใช้ไม่ว่าจะเป็นสำหรับการใช้ชีวิตประจำวัน การทำงาน หรือการสันทนาการ

รูปแบบผลิตภัณฑ์ซอฟต์แวร์

ซอฟต์แวร์ยังสามารจำแนกได้ตามรูปแบบผลิตภัณฑ์เป็นสองประเภทได้แก่ ผลิตภัณฑ์ทั่วไปและผลิตภัณฑ์ที่กำหนดเอง ซึ่งผลิตภัณฑ์ทั้งสองแบบจะแตกต่างกันตรงวัตถุประสงค์ในการสร้างและบทบาทของผู้พัฒนาและผู้ใช้งาน

ผลิตภัณฑ์ทั่วไป (generic product) คือผลิตภัณฑ์ซอฟต์แวร์ที่พัฒนาโดยนักพัฒนาตามความต้องการของผู้พัฒนาเพื่อวางขายในตลาดแบบกว้าง ๆ ผู้ใช้สามารถหาซื้อมาใช้งานได้ทั้งสำหรับเครื่องคอมพิวเตอร์หรืออุปกรณ์เคลื่อนที่ เช่นซอฟต์แวร์ เล่นเพลง ตกแต่งรูป ประมวลผลคำ โดยจะสังเกตได้ว่าความต้องการของซอฟต์แวร์นั้นส่วนมากนักพัฒนามักเป็นผู้กำหนดเอง ผู้ใช้ไม่สามารถกำหนดความต้องการเองได้หรือร้องขอให้นักพัฒนาปรับซอฟต์แวร์ตามที่ตนต้องการได้ หรือกล่าวได้ว่าเป็นผลิตภัณฑ์ที่ผลิตออกมาขายสำหรับตลาดแนวกว้างสำหรับลูกค้าใด ๆ

ผลิตภัณฑ์ที่กำหนดเอง (customized product) คือผลิตภัณฑ์ที่นักพัฒนาผลิตขึ้นมาเฉพาะเพื่อลูกค้าเดียว โดยส่วนใหญ่จะเป็นการว่าจ้างออกแบบและผลิตซอฟต์แวร์ตามสัญญาจ้าง เช่น การผลิตซอฟต์แวร์เพื่อใช้เฉพาะในองค์กร ซอฟต์แวร์เฉพาะตามรูปแบบธุรกิจ หรือเฉพาะด้าน เช่นซอฟต์แวร์ควบคุมการเดินอากาศยาน

ในปัจจุบันผลิตภัณฑ์ทั้งสองแบบนี้ก็ยังมีผลิตภัณฑ์ที่เป็นทั้งสองรูปแบบโดยการสร้างผลิตภัณฑ์ทั่วไปเป็นพื้นฐาน และทำการปรับให้เข้ากับลูกค้าแต่ละรายตามข้อกำหนดเช่น ผลิตภัณฑ์การบริหารทรัพยากรขององค์กร (ERP enterprise resource planning) ที่นักพัฒนาจะทำการปรับแก้ไขซอฟต์แวร์ให้ตรงกับความต้องการของลูกค้าเช่นการสร้างรายงาน การจัดการฐานข้อมูลหรือกระบวนการทำงาน ยกตัวอย่างระบบที่เป็นที่รู้จัก คือ SAP และ Oracle

จากที่กล่าวมาในขั้นต้นนั้นจะเห็นว่าซอฟต์แวร์นั้นมีความสำคัญและเข้ามาเกี่ยวข้องกับชีวิตประจำวันของมนุษย์มาก ทั้งการใช้ชีวิตการทำงานและระบบต่าง ๆ รอบ ๆ ตัว ความปลอดภัยในการใช้งานซอฟต์แวร์ก็ถือว่าเป็นสิ่งสำคัญ เช่นซอฟต์แวร์ควบคุมการขับขี่รถยนต์ก็ควรมีความปลอดภัยและเชื่อถือได้สูงเช่นเดียวกับซอฟต์แวร์สำหรับดำเนินการธุรกรรมทางการเงิน รวมทั้งต้องคำนึงถึงประสิทธิภาพของซอฟต์แวร์เช่นกัน ดังนั้นนักพัฒนาควรจะศึกษาวิศวกรรมซอฟต์แวร์เพื่อเรียนรู้กระบวนการสร้างซอฟต์แวร์ที่ดี แม้กระทั้งลูกค้าในปัจจุบันลูกค้าก็จำเป็นต้องรู้พื้นฐานวิศวกรรมซอฟต์แวร์เพื่อที่จะสามารถทำงานร่วมกับนักพัฒนา

ประเภทผลิตภัณฑ์ซอฟต์แวร์

ประเภทซอฟต์แวร์ยังสามารจำแนกได้ตามประเภทต่าง ๆ ได้แก่ ซอฟต์แวร์แบบเดี่ยว ซอฟต์แวร์ธุรกรรมแบบปฏิสัมพันธ์ ซอฟต์แวร์ควบคุมแบบฝังตัว ซอฟต์แวร์ประมวลผลกลุ่ม ซอฟต์แวร์เก็บรวบรวมข้อมูล และรวมถึงระบบซอฟต์แวร์ใหญ่ที่มีระบบย่อยอยู่ภายในระบบ

ซอฟต์แวร์แบบเดี่ยว คือซอฟต์แวร์ที่ทำงานบนเครื่องคอมพิวเตอร์แบบเดี่ยวได้เอง ไม่จำเป็นต้องพึ่งใช้งานเครือข่าย และประกอบด้วยฟังก์ชันการทำงานที่เพียงพอต่อการทำงาน

ซอฟต์แวร์ธุรกรรมแบบปฏิสัมพันธ์ ซอฟต์แวร์ประเภทนี้ส่วนใหญ่จะทำงานร่วมกับการประมวลผลทางไกลผ่านเครือข่ายโดยใช้เครื่องคอมพิวเตอร์ของผู้ใช้เป็นเพียงจุดเชื่อมต่อการใช้งาน เช่นการทำธุรกรรมการพาณิชย์อิเล็กทรอนิกส์ หรือการใช้งานโปรแกรมประยุกต์ผ่านเว็บ

ซอฟต์แวร์ควบคุมแบบฝังตัว ซอฟต์แวร์ฝังอยู่ในอุปกรณ์สำหรับควบคุมหรือบริหารจัดการอุปกรณ์ เช่นซอฟต์แวร์ในเครื่องคิดเลข ซอฟต์แวร์ในกล่องควบคุมในรถยนต์ ซอฟต์แวร์ควบคุมเครื่องใช้ไฟฟ้า ซึ่งปัจจุบันซอฟต์แวร์ควบคุมแบบฝังตัวนั้นถูกใช้งานในชีวิตประจำวันเราอย่างมากมาย

ซอฟต์แวร์ประมวลผลกลุ่ม เป็นซอฟต์แวร์ประเภทที่ถูกใช้งานมากในธุรกิจเพื่อประมวลผลข้อมูลในลักษณะกลุ่ม ประมวลผลข้อมูลปริมาณมาก ๆ เช่นซอฟต์แวร์ประมวลผลเงินเดือน ซอฟต์แวร์ประมวลผลค่าบริการต่างๆ

ซอฟต์แวร์ทางวิทยาศาสตร์ เป็นซอฟต์แวร์ประเภทที่ถูกใช้งานในด้านงานวิจัยการศึกษาเพื่อสร้างและจำลองแบบจำลองทางวิทยาศาสตร์สำหรับศึกษา แก้ปัญหา หรือหาผลเฉลย ด้านวิทยาศาสตร์ โดยผู้ใช้งานส่วนใหญ่เป็นวิศวกร หรือนักวิทยาศาสตร์ เช่น ซอฟต์แวร์จำลองอากาศพลศาสตร์เพื่อช่วยออกแบบใบพัดหรือปีกเครื่องบิน ซอฟต์แวร์พยากรณ์สภาพอากาศ

ซอฟต์แวร์เก็บรวมรวมข้อมูล เป็นซอฟต์แวร์ที่ใช้ในการเก็บรวบรวมข้อมูลสภาพแวดล้อม จากตัวรับรู้เพื่อนำมาเก็บไว้และส่งต่อให้ซอฟต์แวร์หรือระบบอื่นนำไปใช้ต่อ เช่นซอฟต์แวร์เก็บรวบรวมข้อมูลสภาพอากาศจากอุปกรณ์ตัวรับรู้ในสถานีตรวจวัดสภาพอากาศณ์ ซอฟต์แวร์เฝ้าสังเกตการทำงานเครื่องจักรต่างๆในโรงงานอุตสาหกรรม

วัฏจักรชีวิตการพัฒนาซอฟต์แวร์ (software development life cycle)

วัฏจักรชีวิตการพัฒนาซอฟต์แวร์หรือเรียกสั้น ๆ ว่า SDLC นั้นคือกระบวนการในการพัฒนาหรือปรับปรุงซอฟต์แวร์ ไม่ว่าจะเป็นการผลิตซอฟต์แวร์ขึ้นมาใหม่หรือการปรับปรุงปรับปรุงซอฟต์แวร์ที่มีอยู่เดิมเพื่อให้ตอบสนองการเปลี่ยนแปลงหรือความต้องการของลูกค้า โดยจะมีกระบวนการหลัก 8 ขั้นตอนในวัฏจักรชีวิตการพัฒนาซอฟต์แวร์

การวางแผน (planning) คือการวางแผนการพัฒนาซอฟต์แวร์ โดยการวางแผนงาน ประมาณการใช้ทรัพยากร วางแผนรูปแบบการพัฒนา กำหนดกระบวนการ กำหนดเครื่องมือ กำหนดระยะเวลา และกลยุทธ์เกี่ยวกับการพัฒนาซอฟต์แวร์

การเก็บรวบรวมความต้องการ (requirements gathering) การเก็บรวบรวมความต้องการของลูกค้า ผู้ใช้งาน และความต้องการของระบบ ปัญหาและแนวทางพัฒนาตลอดจนความเป็นไปได้ในการดำเนินโครงการ เพื่อสร้างนิยามความต้องการ ซึ่งสามารถนำข้อมูลต่าง ๆ ไปประกอบร่วมกับการวางแผน

การวิเคราะห์ (analysis) คือกระบวนการวิเคราะห์ปัญหา นิยามความต้องการเพื่อทำความเข้าใจทั้งระบบ ความต้องการลูกค้า ปัญหา และข้อจำกัดต่างๆทั้งทางเทคนิคและทางนโยบาย เพื่อนำไปสู่การวิเคราะห์ระบบเพื่อเป็นแนวทางในการสร้างซอฟต์แวร์ กำหนดฟังก์ชันการทำงานหลักๆ โดยจะจัดทำเอกสารข้อกำหนดความต้องการซอฟต์แวร์เพื่อใช้ ในการจัดทำสัญญาจ้างพัฒนาซอฟต์แวร์และ เป็นเอกสารที่ใช้ในการตรวจรับซอฟต์แวร์ว่าสามารถใช้งานได้ตรงกับความต้องการของลูกค้าหรือไม่

การออกแบบ (design) หลังจากได้เอกสารข้อกำหนดความต้องการซอฟต์แวร์จากขั้นตอนการวิเคราะห์ นักพัฒนาจะนำรายละเอียดเหล่านี้ไปทำการออกแบบซอฟต์แวร์ให้ได้ตามความต้องการ โดยพิจารณาจาก การทำงานของระบบ รูปแบบการใช้งาน การออกแบบสถาปัตยกรรมซอฟต์แวร์ การออกแบบส่วนประกอบ การออกแบบกระบวนการทำงาน การออกแบบโครงสร้างข้อมูล ฐานข้อมูล การออกแบบการปฏิสัมพันธ์ การออกแบบส่วนประสานกับผู้ใช้ การออกแบบฟอร์มและรายงาน ข้อกำหนด ข้อจำกัด โดยออกแบบไว้เป็นรายละเอียดของซอฟต์แวร์เพื่อนำไปใช้ในการพัฒนาซอฟต์แวร์ในลำดับต่อไป

การพัฒนา (development) คือขั้นตอนกับเขียนโปรแกรมเพื่อสร้างซอฟต์แวร์ จากการนำรายละเอียดซอฟต์แวร์จากขั้นตอนการออกแบบมาเขียนโปรแกรมเพื่อสร้างซอฟต์แวร์และทำเอกสารประกอบ

การทดสอบ (testing) เป็นขั้นตอนที่สามารถดำเนินงานควบคู่ไปพร้อมๆกับกระบวนการพัฒนาโดย วัตถุประสงค์ของการทดสอบนั้นคือตรวจสอบความถูกต้องของซอฟต์แวร์ที่ถูกสร้างว่าทำงานได้ถูกต้องตามรายละเอียดของซอฟต์แวร์ และตรงกับความต้องการของลูกค้า และจัดทำเอกสารรายงานผลการทดสอบซอฟต์แวร์

การส่งมอบ (deployment) หลังจากผ่านการทดสอบแล้วซอฟต์แวร์จะถูกส่งมอบให้กับลูกค้าโดยการติดตั้งและอบรมการใช้งานให้ลูกค้าและผู้ใช้ พร้อมทั้งส่งมอบเอกสารประกอบการใช้งานของระบบ และเอกสารอื่น ๆ ทั้งหมดให้กับลูกค้า

การให้ความช่วยเหลือ (support) เป็นขั้นตอนการสนับสนุนและให้ความช่วยเหลือผู้ใช้เหรือลูกค้า เมื่อพบปัญหา ข้อผิดพลาดในการใช้งานซอฟต์แวร์ การปรับเปลี่ยนความต้องการ เปลี่ยนเทคโนโลยี ปรับสภาพแวดล้อม เพิ่มประสิทธิภาพ การแก้ไขหรือปรับเปลี่ยนซอฟต์แวร์ หรืออาจเรียกว่าขั้นตอนการบำรุงรักษาซอฟต์แวร์ซึ่งโดยปกติขั้นตอนนี้จะใช้ระยะเวลานานที่สุดกล่าวคือตั้งแต่เริ่มใช้งานซอฟต์แวร์ไปจนกระทั่งสิ้นสุดการใช้งานหรือถูกแทนที่ด้วยซอฟต์แวร์อื่น และส่วนมากขั้นตอนบำรุงรักษานี้จะใช้งบประมาณสูงกว่ากระบวนการอื่น ๆ

ผลิตภัณฑ์ซอฟต์แวร์นั้นจับต้องไม่ได้ดังนั้นปกติการจ้างผลิตซอฟต์แวร์นั้นจะใช้เอกสารในการตรวจสอบ ควบคู่ไปกับการส่งมอบซอฟต์แวร์หรือระบบซอฟต์แวร์ การจัดทำเอกสารก็ยังถือว่ามีความสำคัญอย่างน้อยสุดสัญญาระหว่างลูกค้าและนักพัฒนาที่ระบุขอบเขตงานตามเอกสารความต้องการที่ควรระบุไว้อย่างชัดเจน เพื่อป้องกันข้อพิพาทภายหลัง และเอกสารส่งมอบงานก็ควรที่จะครบถ้วนให้ลูกค้าหรือผู้ใช้สามารถใช้งานหรือส่งผ่านงานให้ทีมช่วยเหลือดูแลต่อได้ ดังนั้นเอกสารควรส่งมอบอย่างครบถ้วน เช่น เอกสารการออกแบบ แบบจำลอง แผนภาพ รหัสต้นฉบับ เอกสารการทดสอบ

วัฏจักรชีวิตการพัฒนาซอฟต์แวร์เป็นภาพรวมของขั้นตอนในการพัฒนาซอฟต์แวร์ซึ่งอาจมีมากกว่าหรือน้อยกว่า 8 ขั้นตอนที่นำเสนอได้ ในเอกสารคำสอนนี้นำเสนอวัฏจักรชีวิตการพัฒนาซอฟต์แวร์นี้เพื่อให้เห็นภาพโดยรวมและทำความเข้าใจถึงขั้นตอนการพัฒนาซอฟต์แวร์ในบริบททั่ว ๆ ไป สำหรับวิศวกรรมซอฟต์แวร์นั้นจะกล่าวถึงกระบวนการผลิตซอฟต์แวร์เป็นหลักโดยใช้กระบวนการซอฟต์แวร์ ระเบียบวิธี แบบจำลอง ตามหลักการทางวิศวกรรมซอฟต์แวร์

บุคคลผู้เกี่ยวข้องกับการผลิตซอฟต์แวร์

ในกระบวนการพัฒนาซอฟต์แวร์นั้นมีผู้เกี่ยวข้องที่หลากหลาย และมีบทบาท หน้าที่ และความรับผิดชอบที่ต่างกัน โดยเฉพาะเมื่อการผลิตซอฟต์แวร์นั้นใช้วิศวกรรมซอฟต์แวร์ในการผลิต จะมีผู้เกี่ยวข้องพอสมควร แตกต่างจากการเขียนโปรแกรมซอฟต์แวร์ที่ไม่ใช้วิศวกรรมซอฟต์แวร์ที่ใช้คนน้อยอาจมีเพียงผู้ใช้และนักพัฒนา

1. **ลูกค้า** คือผู้ว่าจ้างหรือให้การสนับสนุนด้านการเงินสำหรับการผลิตซอฟต์แวร์หรือผู้เป็นเจ้าของผลิตภัณฑ์
2. **ผู้ใช้** คือผู้ใช้ผลิตภัณฑ์ซอฟต์แวร์ที่พัฒนาสำเร็จแล้วโดยอาจเป็นบุคคลเดียวกับลูกค้า
3. **ผู้จัดการโครงการ** เป็นผู้บริหารโครงการทำหน้าที่วางแผนและควบคุมการพัฒนาซอฟต์แวร์ให้เป็นไปตามแผน
4. **วิศวกรซอฟต์แวร์** คือผู้ที่ทำหน้าที่ดูแลการผลิตซอฟต์แวร์ภายใต้หลักกระบวนการทางวิศวกรรมซอฟต์แวร์
5. **นักพัฒนาซอฟต์แวร์** คือผู้ที่มีส่วนเกี่ยวข้องในกิจกรรมการพัฒนาซอฟต์แวร์ ตั้งแต่การเก็บรวบรวมข้อมูล วิเคราะห์ ออกแบบ พัฒนา และทดสอบซอฟต์แวร์ จึงเรียกบุคคลกลุ่มนี้โดยรวมว่านักพัฒนาซอฟต์แวร์
    a. **นักวิเคราะห์** ทำหน้าที่รวบรวมและวิเคราะห์ระบบ เพื่อทำความเข้าใจความต้องการของลูกค้า
    b. **นักออกแบบระบบ** ทำหน้าที่ออกแบบระบบเพื่อสนองความต้องการของลูกค้าภายใต้ข้อกำหนด เทคโนโลยี และทรัพยากร
    c. **นักออกแบบประสบการณ์ผู้ใช้** ทำหน้าที่ออกแบบผลิตภัณฑ์ซอฟต์แวร์ให้ได้ตรงตามความต้องการของผู้ใช้ และผู้ใช้งานมีความพึงพอใจและได้ประสบการณ์ใช้งานที่ดี โดยอาจเพิ่มเติมการออกแบบส่วนประสานผู้ใช้งานเข้าไปด้วยเพื่อเป้าหมายคือใช้งานได้ สะดวก สวยงาม และมีประสิทธิภาพ
    d. **นักเขียนโปรแกรม** ทำหน้าที่เขียนรหัสฉบับเพื่อสร้าง แก้ไข ปรับปรุง และพัฒนาซอฟต์แวร์ให้เป็นไปตามข้อกำหนดของโปรแกรม
    e. **นักทดสอบ** ทำหน้าที่ทวนสอบหรือตรวจสอบซอฟต์แวร์หาจุดบกพร่องโดยิงตามรายละเอียดข้อกำหนดซอฟต์แวร์ และความต้องการของลูกค้า
    f. **นักสนับสนุน** ทำหน้าที่สนับสนุนและบำรุงรักษาซอฟต์แวร์ให้ทำงานได้อย่างราบรื่น ทั้งการแก้ไขข้อบกพร่องซอฟต์แวร์ ปรับปรุงประสิทธิภาพ หรือปรับเปลี่ยนตามความต้องการของลูกค้าและผู้ใช้

บุคคลที่เกี่ยวข้องอาจมีเพิ่มเติมจากที่นำเสนอข้างต้นขึ้นอยู่กับแต่ละโครงการ รูปแบบองค์กร ลักษณะทีมพัฒนาซึ่งแต่ละที่จะมีการจัดทรัพยากรบุคคล บทบาท หน้าที่ที่แตกต่างกัน เช่น บางทีมพัฒนาขนาดใหญ่อาจมีการเพิ่มผู้ผลิต (producer) หลายคนมาช่วยบริหารและสนับสนุนจัดการกลุ่มทีมพัฒนาย่อยเป็นต้น

วิศวกรรมซอฟต์แวร์

วิศวกรรมซอฟต์แวร์เริ่มมีการพูดถึงในช่วงปี ค.ศ. 1968 ในการประชุมโดยเริ่มจากหัวข้อคำว่า “software crisis” ที่เกิดความลำบากในการพัฒนาซอฟต์แวร์ในช่วงนั้นอันเนื่องมาจากการพัฒนาที่ไม่รองรับการขยายขนาด การที่ซอฟต์แวร์ขนาดใหญ่ขึ้นซับซ้อนขึ้นซึ่งนำไปสู่ความไม่แน่นอนในการการประเมินค่าใช้จ่ายและเวลาในการพัฒนาซอฟต์แวร์ ดังนั้นจึงมีการนำเสนอแนวคิดในการใช้หลักทางวิศวกรรมมาประยุกต์ในกระบวนการพัฒนาซอฟต์แวร์ [1]

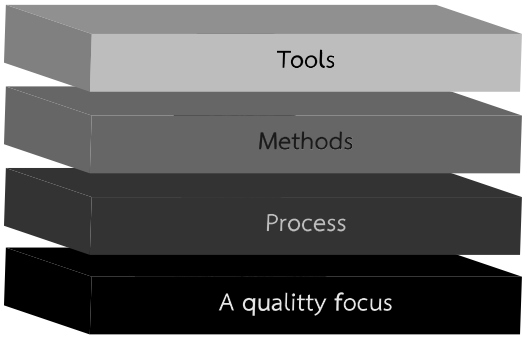
ในช่วงทศวรรษ 1970s-1980s มีการแนะนำกระบวนการวิศวกรรมซอฟต์แวร์ออกมาหลากหลาย เช่นการเขียนโปรแกรมแบบมีโครงสร้าง (structured programming) การพัฒนาเชิงวัตถุ (object-oriented development) เครื่องมือ นิยาม สัญกรณ์ และมาตรฐานในด้านวิศวกรรมซอฟต์แวร์ และยังคงถูกใช้อย่างกว้างขวางถึงปัจจุบัน

วิศกรรมซอฟต์แวร์คือ ระเบียบวินัยทางด้านวิศวกรรมที่เกี่ยวข้องกับการสร้างซอฟต์แวร์ในทุกๆด้านตั้งแต่เริ่มกระบวนการออกแบบพัฒนาทดสอบและการซ่อมบำรุง โดยวิศวกรรมซอฟต์แวร์จะเกี่ยวข้องกับทุกๆกิจกรรมในการผลิตซอฟต์แวร์ ถ้ากล่าวถึงระเบียบวินัยด้านวิศวกรรม หมายถึงการทำให้เกิดงาน หรือแก้ไขปัญหาได้อย่างลุล่วงด้วยการประยุกต์ใช้ทฤษฎี กรรมวิธี กระบวนการ และเครื่องมือทางวิศวกรรม โดยมุ่งไปที่การหาคำตอบในการแก้ไขปัญหาอย่างเหมาะสมภายใต้ข้อจำกัดและทรัพยากรที่มีอยู่

การเกี่ยวข้องกับการสร้างซอฟต์แวร์ในทุกๆด้านนั้นหมายรวมถึงทุกๆส่วนที่เกี่ยวข้องทั้งด้านเทคนิคและกระบวนการพัฒนา ตลอดจนกิจกรรมต่าง ๆ ที่เกี่ยวข้องในการสร้างผลิตภัณฑ์ซอฟต์แวร์ เช่นการบริหารจัดการโครงการ เครื่องมือที่ใช้ในการพัฒนา กรรมวิธี ทฤษฎี และทรัพยากรที่ใช้ในการพัฒนา แล้วถ้าพิจารณาถึงคุณภาพของผลิตภัณฑ์แน่นอนว่า ซอฟต์แวร์ควรจะต้องมีคุณภาพสูง หรือได้มาตรฐานแต่อาจต้องแลกด้วยการใช้ทรัพยากรจำนวนมากในการสร้าง ดังนั้นหลักการของวิศวกรรมซอฟต์แวร์จึงรวมไปถึงการเลือกทางเลือกที่เหมาะสมสำหรับการแก้ไขปัญหา

ดังนั้นจะเห็นว่านิยามของวิศวกรรมซอฟต์แวร์นั้นค่อนขัดเจนในความหมายแต่การนำไปประยุกต์ใช้นั้น ต้องคำนึงถึงข้อจำกัดทางด้านต่าง ๆ โดยเฉพาะค่าใช้จ่ายทรัพยากรที่ควรใช้อย่างมีประสิทธิภาพ ซึ่งอาจจะกล่าวได้ว่า วิศวกรรมซอฟต์แวร์นั้นหมายถึงระเบียบวินัยด้านวิศวกรรมที่เกี่ยวข้องกับการสร้างซอฟต์แวร์ โดยคำนึงถึงการใช้ค่าใช้จ่ายในการพัฒนาอย่างมีประสิทธิภาพ

ทำไมนักพัฒนาควรใช้วิศวกรรมซอฟต์แวร์ คำตอบคือในสังคมปัจจุบันระบบซอฟต์แวร์มีความซับซ้อนและก้าวหน้ามาก ซึ่งจำเป็นต้องมีกระบวนการพัฒนาที่รวดเร็วเชื่อถือได้มีรูปแบบมาตรฐานและเหมาะสมทางเศรษฐศาสตร์ และการใช้วิศวกรรมซอฟต์แวร์มีค่าใช้จ่ายในระยะยาวถูกกว่าการพัฒนาโดยนักพัฒนาที่แค่เขียนโปรแกรมโดยในระยะยาวถ้าไม่มีการใช้วิศวกรรมซอฟต์แวร์นั้นอาจนำไปสู่ค่าใช้จ่ายที่สูงในด้านการทดสอบการเปลี่ยนแปลงและการบำรุงรักษา



รูปที่ 1.2 เทคโนโลยีแบบชั้น [2]

วิศวกรรมซอฟต์แวร์นับว่าเป็น เทคโนโลยีแบบชั้น (layered technology) โดยมีเทคโนโลยีซ้อนกันเป็นชั้นดังรูป 1.2 โดยมีพื้นฐานคือชั้น กระบวนการวิศวกรรมซอฟต์แวร์ทำหน้าที่ผสานชั้นต่าง ๆ ไว้ด้วยกัน โดยกระบวนการจะถูกนิยามเป็นกรอบ

การทำงานสำหรับการผลิตซอฟต์แวร์ซึ่งประกอบไปด้วย การบริหารจัดการ ควบคุมการพัฒนาซอฟต์แวร์ด้วยการใช้ แบบจำลอง เอกสาร ข้อมูล รายงาน เพื่อสนับสนุนกระบวนการพัฒนาซอฟต์แวร์

กรรมวิธีทางวิศวกรรมซอฟต์แวร์จะเป็นชั้นที่รวบรวม กรรมวิธีในการพัฒนาซอฟต์แวร์โดยทำการกำหนดรายละเอียดวิธีการทำงาน เช่นการสื่อสารภายในและภายนอกทีม การการวิเคราะห์ความต้องการ การออกแบบ การสร้าง การทดสอบ และการสนับสนุนซอฟต์แวร์ โดยจะใช้การนิยามหลักการทำงาน เทคโนโลยี กิจกรรม และเทคนิคในการพัฒนาซอฟต์แวร์

เครื่องมือทางวิศวกรรม คือชุดเครื่องมือที่ถูกนำมาช่วยในกระบวนการและกรรมวิธีพัฒนาซอฟต์แวร์ ไม่ว่าจะเป็นเครื่องมือแบบอัตโนมัติหรือกึ่งอัตโนมัติก็ตาม ล้วนแล้วแต่ออกแบบมาเพื่อช่วยอำนวยความสะดวกและสนับสนุนการพัฒนาซอฟต์แวร์ โดยจะเรียกเครื่องมือประเภทนี้ว่าเคส (computer-aided software engineering: CASE)

คุณภาพซอฟต์แวร์คือชั้นล่างสุดซึ่งเป็นรากฐานสำคัญสำหรับเทคโนโลยีแบบชั้นกล่าวคือ วิศวกรรมซอฟต์แวร์นั้นตั้งอยู่บนรากฐานของคุณภาพเป็นสำคัญโดยซอฟต์แวร์ต้องถูกต้องและตอบสนองความต้องการของลูกค้าในระดับที่ยอมรับได้ด้วยคุณภาพที่เป็นไปตามมาตรฐาน ซอฟต์แวร์มีประสิทธิภาพ ถูกต้อง ใช้งานได้ บำรุงรักษาได้ นำกลับมาใช้ใหม่ได้ และ คุ้มค่า

กระบวนการวิศกรรมซอฟต์แวร์ (software engineering process)

แนวทางการพัฒนาซอฟต์แวร์อย่างเป็นระบบในวิศวกรรมซอฟต์แวร์เราจะเรียกว่ากระบวนการซอฟต์แวร์ที่มีกิจกรรมต่าง ๆ บรรจุในการบวนการพัฒนา ในกระบวนการซอฟต์แวร์ส่วนใหญ่มักจะประกอบด้วย 4 กิจกรรมหลักที่ใช้ในการพัฒนาซอฟต์แวร์เรียงลำดับ ดังต่อไปนี้

1. ข้อกำหนดซอฟต์แวร์ (software specification) เป็นกิจกรรมที่ลูกค้าและวิศวกรซอฟต์แวร์ช่วยกันนิยามคุณลักษณะและข้อกำหนดข้อจำกัดและการทำงานของซอฟต์แวร์ที่กำลังจะถูกพัฒนาขึ้น
2. การพัฒนาซอฟต์แวร์ (software development) เป็นกิจกรรมในการออกแบบและพัฒนาซอฟต์แวร์ตามมาตรฐานหรือกระบวนการทางวิศวกรรม
3. การตรวจสอบความถูกต้องของซอฟต์แวร์ (software validation) เป็นกิจกรรมว่าด้วยการทดสอบและตรวจสอบความถูกต้องของซอฟต์แวร์ว่าเป็นไปตามความต้องการของลูกค้าหรือไม่
4. การวิวัฒนาการซอฟต์แวร์ (software evolution) เป็นกิจกรรมที่เกี่ยวข้องกับการปรับเปลี่ยนแก้ไขซอฟต์แวร์เพื่อสนองการเปลี่ยนแปลงที่เกิดขึ้นจากความต้องการที่เปลี่ยนไปของลูกค้า รูปแบบธุรกิจหรือสภาพตลาดที่เปลี่ยนไป

ทั้งสี่กิจกรรมหลักข้างต้น จัดเป็นกิจกรรมหลักที่พบได้ในกระบวนการซอฟต์แวร์ใด ๆ ซึ่งบางกระบวนการอาจจะเรียกชื่อกิจกรรมแตกต่างออกไปแต่โดยรวมจะมีการประยุกต์ใช้สี่กิจกรรมนี้เสมอแต่อาจปรับรายละเอียดหรือชื่อกิจกรรมแตกต่างออกไป ดังนั้นอาจกล่าได้ว่าสาระสำคัญหลักของวิศวกรรมซอฟต์แวร์นั้นจะเกี่ยวข้องกับสี่กิจกรรมนี้ที่เป็นแกนสำคัญในกระบวนการซอฟต์แวร์ ซึ่งจะได้อธิบายรายละเอียดในบทที่ 2

จริยธรรมทางวิศวกรรมซอฟต์แวร์ (software engineering ethic)

จริยธรรม ตามความหมายพจนานุกรม ฉบับราชบัณฑิตยสถาน พ.ศ. 2554 กล่าวว่า “ธรรมที่เป็นข้อประพฤติปฏิบัติ ศีลธรรม กฎศีลธรรม” [3] ดังนั้นถ้ากล่าวถึงจริยธรรมทางวิศวกรรมซอฟต์แวร์นั้น หมายถึง แนวทางประพฤติปฏิบัติตนเพื่อบรรลุการเป็นวิศวกรซอฟต์แวร์อาชีพในกรอบศีลธรรมอันสุจริตเชกเช่นจริยธรรมทางวิศวกรรมสาขาอื่น ๆ โดยที่จริยธรรมทาวิศวกรรมซอฟต์แวร์เน้นรายละเอียดที่เกี่ยวข้องกับซอฟต์แวร์และระบบคอมพิวเตอร์ โดยจะมีหลักการหลัก ๆ ดังนี้

1. **การเก็บความลับ** การเก็บความลับของลูกค้าถือเป็นหลักสำคัญที่จะต้องเก็บไม่ให้เผยแพร่ออกไปเพื่อรักษาผลประโยชน์ของลูกค้าเป็นสำคัญถึงแม้จะไม่มีการเซ็นข้อตกลงในการรักษาความลับก็ตาม
2. **ความสามารถที่แท้จริง** วิศวกรซอฟต์แวร์ต้องประเมินและรู้ถึงความสามามรถที่แท้จริง และไม่ควรบิดเบือนระดับความสามารถของตน ถ้าความสามารถไม่ถึงพอที่จะรับงานลูกค้าได้ก็ควรแจ้งไปตามความจริง
3. **เคารพในทรัพย์สินทางปัญญา** ควรเคารพและตระหนักถึงทรัพย์สินทางปัญญาไม่ว่าจะเป็น สิทธิบัตร ลิขสิทธิ์ ให้เป็นไปตามกฎหมาย และเพื่อให้แน่ใจว่าทรัพย์สินทางปัญญาของลูกค้าและนายจ้างได้รับการคุ้มครอง
4. **การใช้คอมพิวเตอร์ในทางที่ผิด** วิศวกรซอฟต์แวร์ไม่ควรใช้ความรู้และทักษะคอมพิวเตอร์เพื่อนำไปใช้ในทางที่ผิด การใช้คอมพิวเตอร์ในทางที่ผิดมีตั้งแต่เรื่องเล็กน้อย การเล่นเกมจองตั๋วภาพยนต์จากเครื่องคอมพิวเตอร์ของนายจ้าง จนถึงขั้นร้ายแรง เช่น การจารกรรมข้อมูลแพร่กระจายของไวรัส หรือ มัลแวร์ ในระบบคอมพิวเตอร์

จริยธรรมทางวิศวกรรมซอฟต์แวร์ ในระดับสากลใช้ประมวลจริยธรรม (code of ethics) ที่กำหนดโดยสมาคมคอมพิวเตอร์ (association for computing machinery: ACM) ร่วมกับสถาบันวิชาชีพวิศวกรไฟฟ้าและอิเล็กทรอนิกส์ (institute of electrical and electronics engineers: IEEE) วิศวกรซอฟต์แวร์ต้องมุ่งมั่นที่จะทำการวิเคราะห์ การออกแบบ การพัฒนา การทดสอบ และการบำรุงรักษาซอฟต์แวร์ เพื่อก่อให้เกิดประโยชน์และเคารพในวิชาชีพ นอกจากนั้นยังต้องคำนึงถึง สุขภาพ ความปลอดภัย และสวัสดิภาพของประชาชนและส่วนรวม วิศวกรซอฟต์แวร์ต้องปฏิบัติตามหลักการแปดประการดังต่อไปนี้ [4]

1. วิศวกรซอฟต์แวร์ควรดำเนินงานโดยคำนึงถึงประโยชน์ส่วนรวมเสมอ
2. วิศวกรซอฟต์แวร์ควรดำเนินงานโดยคำนึงถึงความต้องการและประโชน์ของลูกค้าและนายจ้าง โดยสอดคล้องกับประโยชน์ส่วนรวม
3. วิศวกรซอฟต์แวร์ควรมั่นใจว่าการผลิตหรือแก้ไขซอฟต์แวร์นั้นเป็นไปตามตามมาตรฐานวิชาชีพสูงสุด
4. วิศวกรซอฟต์แวร์จะต้องรักษาความซื่อสัตย์และมีความเป็นอิสระในการตัดสินอย่างมืออาชีพ
5. ผู้จัดการและผู้นำควรส่งเสริมแนวทางจริยธรรมในทีม เพื่อใช้ในการจัดการ การพัฒนา และซ่อมบำรุงซอฟต์แวร์
6. วิศวกรซอฟต์แวร์ควรส่งเสริมพัฒนาและรักษาชื่อเสียงในวิชาชีพ โดยคำนึงถึงประโยชน์ส่วนรวมเสมอ
7. วิศวกรซอฟต์แวร์ต้องยุติธรรมและให้การสนับสนุนเพื่อนร่วมงาน
8. วิศวกรซอฟต์แวร์ควรพัฒนาตนและเรียนรู้เกี่ยวกับวิชาชีพของตนอย่างสม่ำเสมอ พร้อมทั้งส่งเสริมแนวทางจริยธรรมในการประกอบวิชาชีพ

นอกจากการปฏิบัติตามจริยธรรมทางวิศวกรรมซอฟต์แวร์แล้ว วิศวกรซอฟต์แวร์ยังต้องปฏิบัติตามกฎหมาย โดยเฉพาะราชอาณาจักรไทยมีพระราชบัญญัติว่าด้วยการกระทำความผิดเกี่ยวกับคอมพิวเตอร์ โดยวิศวกรซอฟต์แวร์ต้องทราบและปฏิบัติตาม พ.ร.บ.อย่างเคร่งครัดโดยมีหลักสำคัญพอสรุปได้ดังนี้

1. วิศวกรซอฟต์แวร์ ต้องไม่เข้าถึงระบบ หรือข้อมูลของผู้อื่นโดยมิชอบ
2. วิศวกรซอฟต์แวร์ ต้องไม่ดัดแปลง แก้ไข จารกรรม หรือทำลายข้อมูลซึ่งทำให้ผู้อื่นเสียหาย
3. วิศวกรซอฟต์แวร์ ต้องไม่ส่งจดหมายอิเล็กทรอนิกส์ขยะ ที่ทำให้เดือดร้อนรำคาญ
4. วิศวกรซอฟต์แวร์ ต้องไม่เข้าถึงระบบ หรือข้อมูลทางด้านความมั่นคงโดยมิชอบ
5. วิศวกรซอฟต์แวร์ ต้องไม่จำหน่ายหรือเผยแพร่ชุดคำสั่งเพื่อนำไปใช้กระทำความผิด
6. วิศวกรซอฟต์แวร์ ต้องไม่นำข้อมูลที่ผิด พ.ร.บ.เข้าสู่ระบบคอมพิวเตอร์
7. วิศวกรซอฟต์แวร์ ต้องไม่ให้ความร่วมมือ ยินยอม รู้เห็นเป็นใจกับผู้ร่วมกระทำความผิด
8. วิศวกรซอฟต์แวร์ ต้องไม่ละเมิดลิขสิทธิ์ ทรัพย์สินทางปัญญา นำผลงานของผู้อื่นมาเป็นของตน

โดยสรุปแล้ววิศวกรซอฟต์แวร์ควรจะประพฤติปฏิบัติตนตามกฎหมายและจริยธรรมอันดีโดยเริ่มจากตนเองและขยายไปยังเพื่อนร่วมทีมและองค์กรเพื่อส่งเสริมความเป็นวิศวกรซอฟต์แวร์อาชีพ

บทที่ 2 กระบวนการซอฟต์แวร์ Software process

วัตถุประสงค์

- เข้าใจกระบวนการซอฟต์แวร์
- เข้าใจแบบจำลองกระบวนการซอฟต์แวร์

กระบวนการซอฟต์แวร์คือแนวทางการพัฒนาซอฟต์แวร์อย่างเป็นระบบด้วยพื้นฐานวิศวกรรมซอฟต์แวร์โดยมีกลุ่มกิจกรรมต่างๆ บรรจุในการบวนการพัฒนาอย่างเป็นลำดับขั้นซึ่งไม่มีกระบวนการแบบใดแบบหนึ่งที่เป็นสามารถนำไปใช้ได้กับทุกรูปแบบการพัฒนาซอฟต์แวร์ เนื่องจากในการพัฒนาซอฟต์แวร์ใด ๆ นั้นแต่ละโครงการมีความแตกต่างกันทั้งปัญหา รูปแบบซอฟต์แวร์ ระบบกระบวนการทำงาน รูปแบบธุรกิจ นโยบาย และ ข้อกำหนดอื่น ๆ

แต่อย่างไรก็ตามกระบวนการซอฟต์แวร์ส่วนใหญ่มักจะประกอบด้วย 4 กิจกรรมหลักที่ใช้ในการพัฒนาซอฟต์แวร์เรียงลำดับดังต่อไปนี้ ข้อกำหนดซอฟต์แวร์ การพัฒนาซอฟต์แวร์ การตรวจสอบความถูกต้องของซอฟต์แวร์ และการวิวัฒนาการซอฟต์แวร์ ซึ่งเป็นกิจกรรมหลักที่พบได้ในกระบวนการซอฟต์แวร์ใด ๆ เพียงแต่บางกระบวนการอาจจะเรียกชื่อกิจกรรมแตกต่างออกไป

แบบจำลองกระบวนการซอฟต์แวร์ เป็นรูปแบบการนำเสนอกระบวนการพัฒนาซอฟต์แวร์แบบนามธรรมเพื่ออธิบายกระบวนการพัฒนาซอฟต์แวร์ได้ชัดเจนขึ้น จากหลายมุมมองผ่านแบบจำลองการพัฒนาซอฟต์แวร์ ยกตัวอย่างเช่นแบบจำลองน้ำตกที่อธิบายการพัฒนาเป็นลำดับขั้นไม่สามารภย้อนกลับไปขั้นก่อนหน้า เหมือนน้ำตกที่สายน้ำมิอาจไหลย้อนกลับโดยจะไหลจากขั้นสูงลงสู่ด้านล่างเสมอ

ภาพรวมกระบวนการซอฟต์แวร์

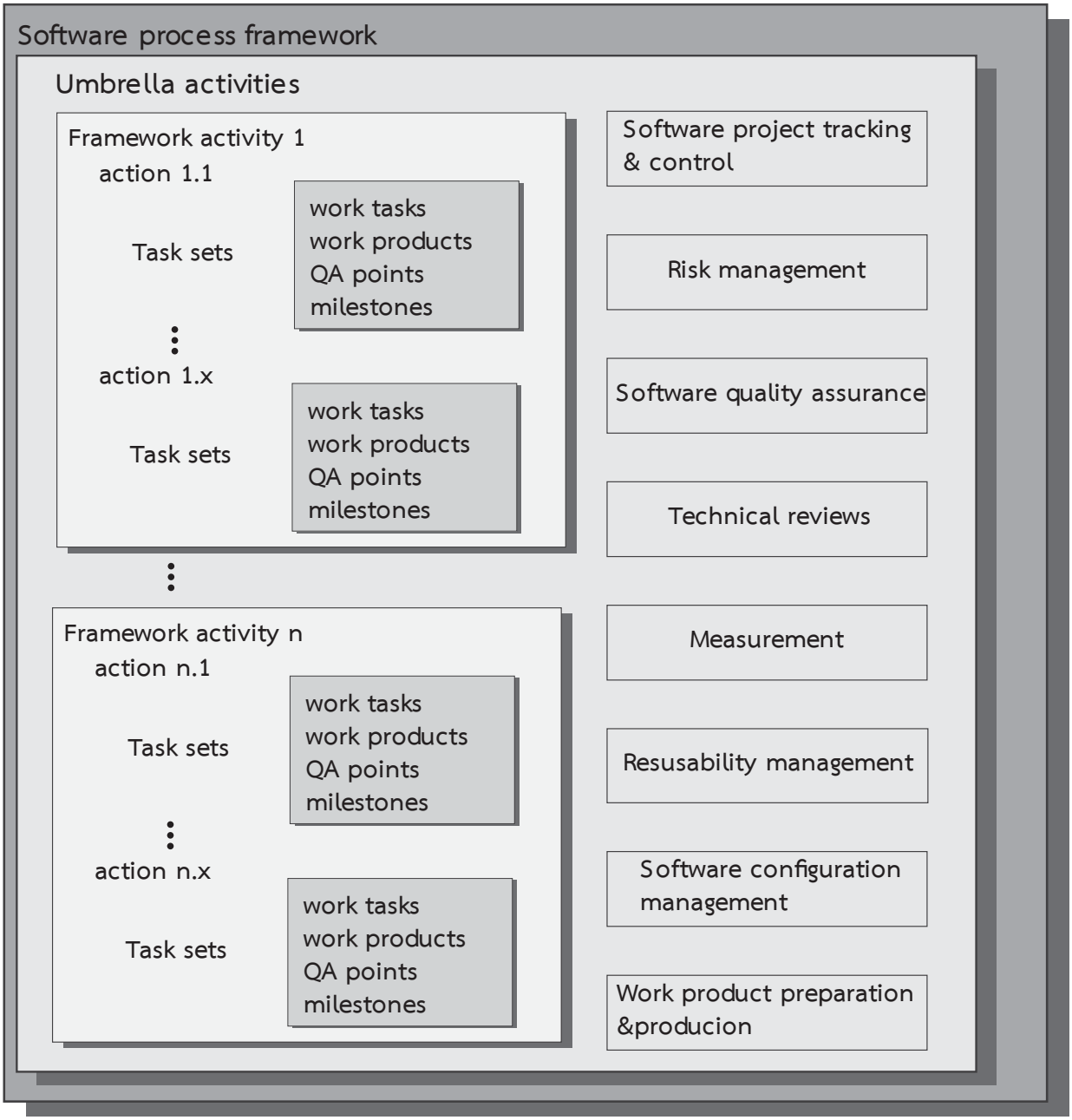
กระบวนการซอฟต์แวร์ คือกรอบงานการพัฒนาซอฟต์แวร์อย่างเป็นระบบบนพื้นฐานด้านวิศวกรรมซอฟต์แวร์ โดยกำหนดแนวทางการพัฒนาให้นำไปสู่ผลิตภัณฑ์ซอฟต์แวร์ที่มีคุณภาพและคุ้มค่า ซึ่งในกระบวนการซอฟต์แวร์อย่างที่กล่าวไว้ก่อนหน้านี้นั้นประกอบด้วยกิจกรรมหลักในการพัฒนา 4 กิจกรรม และอาจจะมีกิจกรรมเติม เพื่อเติมเต็มกระบวนการให้เหมาะกับการพัฒนาในแต่ละผลิตภัณฑ์ เช่น การประยุกต์ใช้กิจกรรมร่ม (umbrella activities) เป็นต้นและในแต่ละกิจกรรมก็จะมีส่วนอธิบายลายละเอียดกิจกรรมเป็นเสมือนคู่มือนำทางในการพัฒนาซอฟต์แวร์

การอธิบายกระบวนการซอฟต์แวร์ส่วนมากจะใช้การอธิบายกิจกรรมต่าง ๆ ที่ถูกบรรจุในกระบวนการซอฟต์แวร์ เพื่อให้เห็นภาพและลายละเอียดที่ชัดเจน และควรอธิบายกิจกรรมต่าง ๆ อย่างมีลำดับตามขั้นตอนในกระบวนการ เช่นอธิบายกิจกรรมการออกแบบโครงร้างข้อมูลและการไหลของงาน จากนั้นกิจกรรมการออกแบบประสบการณ์ผู้ใช้และส่วนติดต่อผู้ใช้ เมื่อเห็นภาพรวมกิจกรรมภายในของกระบวรนการซอฟต์แวร์ ทีมพัฒนาจะสามารถเข้าใจภาพรวมกระบวนการซอฟต์แวร์รวมทั้งรู้ว่าควรจะ

ทำกิจกรรมใดบ้าง ต้องทำงานอะไรบ้าง อะไรคือผลลัพธ์ในแต่ละกิจกรรม ใครมีหน้าที่รับผิดชอบอะไรบ้าง โดยเป้าหมายสำคัญของกระบวนการซอฟต์แวร์คือ สามารถส่งมอบผลิตภัณฑ์ที่มีคุณภาพคุ้มค่าตรงตามความต้องการของลูกค้าภายในระยะเวลาที่กำหนด

กรอบงานกระบวนการ (process framework)

กรอบงานกระบวนการ เป็นพื้นฐานสำคัญของกระบวนการทางซอฟต์แวร์ กรอบงานจะแยกออกได้เป็น กรอบงานของกิจกรรมประกอบด้วย งาน ชิ้นงาน จุดตรวจสอบคุณภาพ และหลักไมล์ในการวัดความก้าวหน้าโครงการ นอกจากนี้ยังมีชุดกิจกรรมร่มเพื่อช่วยเติมเต็มให้ครอบคลุมทั้งกระบวนการซอฟต์แวร์ ดังรูป 2.1



รูป 2.1 กรอบงานกระบวนการซอฟต์แวร์ (ดัดแปลงจาก) [2]

จากรูป 2.1 แสดงให้เห็นถึงกรอบงานกระบวนการซอฟต์แวร์ทั่วไปที่ประกอบไปด้วยกิจกรรมหลัก ๆ ในกระบวนการ โดยแต่ละกิจกรรมก็จะประกอบด้วยชุดของงาน ในส่วนของชุดกิจกรรมร่มนั้นเป็นชุดกิจกรรมเสริมหรือเติมเต็มกระบวนการซอฟต์แวร์ให้มีประสิทธิภาพโดยการประยุกต์ใช้กิจกรรมต่าง ๆ ในชุดกิจกรรมร่ม ตลอดในทุกกรอบงานกิจกรรม

ชุดกิจกรรมร่มประกอบด้วยการบริหารจัดการโครงการซอฟต์แวร์ การบริหารความเสี่ยง การประกันคุณภาพซอฟต์แวร์ การทบทวนเทคนิค การประเมินและการวัด การนำกลับมาใช้ใหม่ การจัดการโครงแบบซอฟต์แวร์ และการเตรียมและการผลิตซอฟต์แวร์ ซึ่งกิจกรรมเสริมเหล่านี้จะช่วยส่งเสริมประสิทธิภาพของกระบวนการซอฟต์แวร์

กิจกรรมหลักในกระบวนการซอฟต์แวร์

ในการกระบวนการพัฒนาซอฟต์แวร์นั้นประกอบไปด้วยกิจกรรมต่างๆมากมายที่ต้องทำเป็นลำดับหรือสามารถทำควบคู่กันได้ เป็นกิจกรรมด้านการพัฒนาด้านการบริหารจัดการด้านเทคนิคด้านการทำงานร่วมกันและกิจกรรมด้านอื่น ๆ ที่เกี่ยวข้องเพื่อให้การพัฒนาซอฟต์แวร์สำเร็จลุล่วงนั้นมีจุดประสงค์เดียวกันกับกิจกรรมหลักทั้ง 4 กลุ่มใหญ่ดังต่อไปนี้ ข้อกำหนดซอฟต์แวร์ การพัฒนาซอฟต์แวร์ การตรวจสอบความถูกต้องของซอฟต์แวร์ และการวิวัฒนาการซอฟต์แวร์ ซึ่งเป็นกิจกรรมหลักที่พบได้ในกระบวนการซอฟต์แวร์ใด ๆ

ข้อกำหนดซอฟต์แวร์ (software specification)

เป็นกิจกรรมที่ลูกค้าและวิศวกรซอฟต์แวร์ช่วยกันทำความเข้าใจระบบ ปัญหาและความต้องการของซอฟต์แวร์ เพื่อนำไปสู่การนิยามคุณลักษณะและข้อกำหนดข้อจำกัดและการทำงานของซอฟต์แวร์ที่กำลังจะถูกพัฒนาขึ้น โดยรวมถึงการระบุข้อจำกัดต่างๆในการดำเนินงานของระบบและข้อจำกัดในการพัฒนาซอฟต์แวร์ ซึ่งในกิจกรรมนี้จะว่าด้วยเรื่องการเก็บรวบรวมข้อมูล เพื่อนำมาวิเคราะห์หาความต้องการเพื่อสร้างข้อกำหนดความต้องการของซอฟต์แวร์ ด้วยการใช้เทคนิคและกระบวนการต่าง ๆ เพื่อให้ได้มาซึ่งผลลัพธ์เอกสารข้อกำหนดความต้องการนั่นเอง ในกิจกรรมแรกสุดนี้นับว่ามีความสำคัญมากหากเกิดข้อผิดพลาดหรือมีความคลาดเคลื่อนในขั้นตอนแรกนี้จะสร้างปัญหาส่งผลกระทบต่อกิจกรรมอื่น ๆ ที่ตามมาทั้งการออกแบบและพัฒนาซอฟต์แวร์อันอาจจะสร้างความเสียหายหรือนำไปสู่ความล้มเหลวในการพัฒนา ฉะนั้นกิจกรรมข้อกำหนดซอฟต์แวร์จึงมีความสำคัญมาก

สำหรับการตัดสินใจโครงการพัฒนาซอฟต์แวร์ ขั้นแรกควรรวบรวมข้อมูลเพื่อศึกษาความเป็นไปได้และการตลาดของโครงการด้วยการศึกษาและวิเคราะห์ปัจจัยต่าง ๆ ไม่ว่าจะเป็น มูลค่าปัจจัยทางเทคนิคสภาพการเงินสภาพตลาดเพื่อประเมินความคุ้มค่าในการพัฒนาซอฟต์แวร์ขึ้นมาหรือไม่ การศึกษาความเป็นไปได้ก่อนตัดสินใจนับว่าเป็นทางเลือกที่ดีและเป็นการลดความเสี่ยงและยังใช้เวลาไม่นาน เทียบกับระยะเวลาการพัฒนาทั้งหมด

วิศวกรรมความต้องการ (requirement engineering) เป็นศาสตร์หลักที่ใช้ในกิจกรรมนี้ ว่าด้วยกรรมวิธีและกลยุทธ์เพื่อให้ได้มาซึ่งข้อกำหนดความต้องการที่สนองความต้องการของลูกค้าและผู้มีส่วนเกี่ยวข้อง โดยปกติข้อกำหนดความต้องการจะมีสองระดับได้แก่ข้อกำหนดความต้องการของผู้ใช้และผู้มีเกี่ยวข้องจะอธิบายในระดับสูงมองเป็นมุมกว้างที่เข้าใจได้ง่าย และความต้องการของระบบในระดับลงรายละเอียดสำหรับทีมพัฒนาที่ต้องการรายละเอียดของระบบที่ลึกลงไป กระบวนการวิศวกรรมความต้องการนั้นทำหน้าที่สกัดความต้องการทั้งสองระดับผ่าน 3 กิจกรรมย่อยดังต่อไปนี้

1. **การสกัดและวิเคราะห์ความต้องการ (requirement elicitation and analysis)** คือกระบวนการที่ทำให้วิศวกรซอฟต์แวร์เข้าใจถึงความต้องการของลูกค้าและผู้มีส่วนเกี่ยวข้องด้วยการสกัดและวิเคราะห์ความต้องการจากกิจกรรมการเฝ้าสังเกต
2. **ข้อกำหนดความต้องการ (requirement specification)** เป็นกิจกรรมในการแปลงข้อมูลที่รวบรวมในช่วงการวิเคราะห์ความต้องการให้เป็นเอกสารที่นิยามความต้องการ ในสองระดับคือความต้องการของผู้ใช้หรือลูกค้า และความต้องการของระบบที่อธิบายลงรายละเอียดถึงฟังก์ชันการทำงานต่าง ๆ
3. **การตรวจสอบความต้องการ (requirement validation)** เป็นกิจกรรรเพื่อตรวจสอบให้มั่นใจว่าข้อกำหนดความต้องการที่สกัดออกมาได้นั้น เป็นความต้องการที่ถูกต้องแท้จริงและสนองความต้องการได้ ในขั้นตอนนี้อาจตรวจพบความผิดพลาดของข้อกำหนดความต้องการซึ่งนำไปสู่การปรับแก้ไขให้ถูกต้อง

การพัฒนาซอฟต์แวร์ (software development)

เป็นกิจกรรมในการออกแบบและพัฒนาซอฟต์แวร์ตามมาตรฐานหรือกระบวนการทางวิศวกรรมเพื่อส่งมอบซอฟต์แวร์ที่มีคุณภาพตรงตามความต้องการให้กับลูกค้าภายในระยะเวลาที่กำหนด โดยกิจกรรมย่อยหลักคือการออกแบบและการพัฒนาซึ่งสามารถทำสองกิจกรรมนี้ร่วมกันหรือแยกกันได้ขึ้นอยู่กับกระบวนการซอฟต์แวร์ที่ถูกนำมาประยุกต์ใช้ในการพัฒนา

การออกแบบซอฟต์แวร์คือการออกแบบและอธิบายโครงสร้างของซอฟต์แวร์ก่อนที่จะพัฒนา การออกแบบนั้นก็สามารถทำได้หลายระดับตั้งแต่การออกแบบสถาปัตยกรรมซอฟต์แวร์ออกแบบโครงสร้างออกแบบมอดูลส่วนประกอบ ฟังก์ชัน หรือขั้นตอนวิธี (algorithm) ถ้าเปรียบเทียบการออกแบบซอฟต์แวร์ก็เหมือนกับการให้สถาปนิกออกแบบบ้านตามความต้องการของผู้ใช้ว่าบ้านแบบไหนกี่ชั้นมีฟังก์ชันการใช้สอยอะไรในบ้านมีกี่ห้องรายละเอียดแต่ละส่วนของบ้านเป็นอย่างไรจะถูกเขียนลงรายละเอียดในพิมพ์เขียวบ้านเพื่อส่งต่อให้วิศวกรคำนวนและดำเนินการสร้างบ้านตามแบบที่ถูกออกแบบไว้นั่นเอง ในส่วนของวิศวกรรมซอฟต์แวร์เอกสารการออกแบบอาจจะใช้การเขียนบรรยายรูปแผนภาพหรือใช้ภาษายูเอ็มแอลในการอธิบายแล้วส่งต่อไปยังขั้นตอนการพัฒนาซอฟต์แวร์ต่อไป

การพัฒนาซอฟต์แวร์เป็นขั้นตอนในการสร้างซอฟต์แวร์ตามการออกแบบโดยการใช้เครื่องมือและเทคนิคในการพัฒนารูปแบบต่างๆโดยไม่จำเป็นต้องหมายถึงการเขียนรหัสต้นฉบับหรือการเขียนโปรแกรมเสมอ การพัฒนาซอฟต์แวร์นั้นอาจเป็นการนำเอาซอฟต์แวร์กลับมาใช้ใหม่ การประยุกต์ใช้ซอฟต์แวร์ส่วนประกอบ หรือการปรับแต่งซอฟต์แวร์เพื่อให้ได้ซอฟต์แวร์ที่ใช้งานได้ตรงความต้องการ และไม่มีกระบวนการที่ดีที่สุดสำหรับการพัฒนา นักพัฒนาแต่ละคนก็มีรูปแบบและความถนัดที่ต่างกันโดยเฉพาะการเขียนโปรแกรมนั้นมีความหลากหลายมากแต่วัตถุประสงค์นั้นเหมือนกัน คือสร้างซอฟต์แวร์ให้มีคุณภาพสูงถูกต้องและใช้งานได้ ซึ่งในการพัฒนาจะรวมส่วนทดสอบเข้าไปด้วยอย่างน้อยที่สุดก็ต้องผ่านการแปลภาษาให้ซอฟต์แวร์สามารถกระทำการได้ และทดสอบฟังก์ชันการทำงานต่าง ๆ เพื่อให้มั่นใจว่ารหัสต้นฉบับโปรแกรมถูกต้องในเบื้องต้นก่อนจะถูกตรวจสอบความถูกต้องของซอฟต์แวร์ในขั้นตอนต่อไป

การตรวจสอบความถูกต้องของซอฟต์แวร์ (software validation)

เป็นกิจกรรมว่าด้วยการทดสอบและตรวจสอบความถูกต้องของซอฟต์แวร์ว่าเป็นไปตามความต้องการของลูกค้าหรือไม่ หรืออาจเรียกว่ากระบวนการวีแอนด์วี (verification and validation: V & V)

การทดสอบหรือทวนสอบซอฟต์แวร์ (verification) นั้นอาจถูกดำเนินการควบคู่กับกิจกรรมการออกแบบและพัฒนาซอฟต์แวร์ เพื่อตรวจสอบความถูกต้องของซอฟต์แวร์ว่าสามารถทำงานตามข้อกำหนดคุณลักษณะของซอฟต์แวร์ที่ออกแบบไว้หรือไม่ ทำงานได้ถูกต้องมีประสิทธิภาพผ่านเกณฑ์มาตรฐานที่กำหนด และข้อผิดพลาดอยู่ในเกณฑ์ที่ยอมรับได้ เนื่องจากในความเป็นจริงนักพัฒนาไม่สามารถทำการทดสอบซอฟต์แวร์ได้ทุกกรณีจึงต้องมีการกำหนดเกณฑ์ที่ยอมรับได้ขึ้นโดยถ้าทดสอบผ่านเกณฑ์ที่กำหนดก็สามารถอนุมานได้ว่าส่วนที่เหลือมีระดับความมั่นใจในความถูกต้องของซอฟต์แวร์สูง

การตรวจสอบซอฟต์แวร์ (validation) คือการตรวจสอบความถูกต้องของซอฟต์แวร์ว่าสามารถทำงานได้ตรงความต้องการหรือไม่ โดยปกติจะทำการตรวจสอบเทียบกับเอกสารข้อกำหนดความต้องการซอฟต์แวร์ทั้งในส่วนของข้อกำหนดความต้องการผู้ใช้ และข้อกำหนดความต้องการของระบบ เพื่อทำการตรวจสอบไล่ไปเป็นข้อ ๆ เพื่อตรวจสอบว่าซอฟต์แวร์ทำงานได้ตรงและครบถ้วนทุกความต้องการหรือไม่

เมื่อซอฟต์แวร์ถูกทดสอบผ่านกระบวนการวีแอนด์วี แล้วพบว่าไม่มีปัญหานั่นหมายความว่ากระบวนการตรวจสอบได้ลุล่วงและลูกค้ายอมรับผลการทดสอบก็สามารถจัดทำเอกสารรายงานการทดสอบส่งมอบให้ลูกค้าเป็นอันจบกระบวนการทดสอบเมื่อผ่านการทดสอบเพื่อยอมรับ (acceptance test) จากลูกค้าก็พร้อมส่งมอบซอฟต์แวร์ได้

การวิวัฒนาการซอฟต์แวร์ (software evolution)

เป็นกิจกรรมที่เกี่ยวข้องกับการปรับเปลี่ยนแก้ไขซอฟต์แวร์เพื่อสนองการเปลี่ยนแปลงที่เกิดขึ้นจากความต้องการที่เปลี่ยนไปของลูกค้า รูปแบบธุรกิจหรือสภาพตลาดที่เปลี่ยนไป การเปลี่ยนแปลงที่ส่งผลกระทบกับซอฟต์แวร์ถ้าจำเป็นต้องปรับแก้ไข ก็ต้องดำเนินการนั่นก็คือกิจกรรมการวิวัฒนาการซอฟต์แวร์นั่นเอง เริ่มจากการรับคำร้องการเปลี่ยนแปลงมาวิเคราะห์และประเมินผลกระทบความเสี่ยงและความจำเป็นความคุ้มค่าในการดำเนินการ ถ้าเห็นว่าจำเป็นหรือมีผลกระทบก็ดำเนินการปรับเปลี่ยนแก้ไขซอฟต์แวร์เพื่อรองรับการเปลี่ยนแปลง รวมทั้งหมายถึงการบำรุงรักษาซอฟต์แวร์ให้ใช้งานได้การแก้ไขข้อผิดพลาดของซอฟต์แวร์ การปรับปรุงประสิทธิภาพซอฟต์แวร์ ซึ่งกิจกรรมนี้ส่วนใหญ่จะคงดำเนินไปจนกว่าซอฟต์แวร์จะสิ้นอายุ เลิกใช้งาน หรือไม่มีความสำคัญในธุรกิจ

เมื่อกล่าวถึงความเปลี่ยนแปลง ในปัจจุบันโลกเปลี่ยนแปลงอย่างรวดเร็วเช่นเดียวกับธุรกิจที่มีการเปลี่ยนแปลงตลอดเวลาซอฟต์แวร์เช่นกัน ยากที่หลีกเลี่ยงการเปลี่ยนแปลงไม่ได้จึงจำเป็นต้องหาวิธีรับมือการเปลี่ยนแปลงเพื่อให้ผู้ใช้หรือลูกค้ายังคงสามารถใช้งานซอฟต์แวร์เพื่อแข่งขันในธุรกิจได้หรือช่วยให้องค์กรปรับทันการเปลี่ยนแปลง โดยเฉพาะในโครงการซอฟต์แวร์ขนาดใหญ่การเปลี่ยนแปลงนั้นแทบจะหลีกเลี่ยงไม่ได้เลย กระบวนการซอฟต์แวร์ส่วนมากจึงออกแบบกลยุทธ์กรรมวิธีและกิจกรรมเพื่อรองรับการเปลี่ยนแปลง โดยส่งผลกระทบน้อยที่สุดเมื่อต้องเผชิญกับการเปลี่ยนแปลง ดังนั้นวิศวกรซอฟต์แวร์พึงระลึกไว้เสมอว่าการเปลี่ยนแปลงนั้นหลีกเลี่ยงไม่ได้ ให้คาดการณ์ไว้ว่าต้องเกิดแน่ ๆ และวางแผนรับมือหรือเตรียมการเผื่อไว้ในการพัฒนาซอฟต์แวร์ เช่นการทำซอฟต์แวร์รีแฟคทอริง (refactoring) เพื่อปรับให้รหัสต้นฉบับรองรับการเปลี่ยนแปลงได้ง่ายเป็นต้น

กิจกรรมทั้งสี่เป็นกิจกรรมแกนหลักของวิศวกรรมซอฟต์แวร์ที่สามารถพบได้กระบวนการซอฟต์แวร์ต่างๆ โดยกิจกรรมเหล่านี้จะอธิบายรายละเอียดในบทต่อๆไป

แบบจำลองกระบวนการซอฟต์แวร์ (software process models)

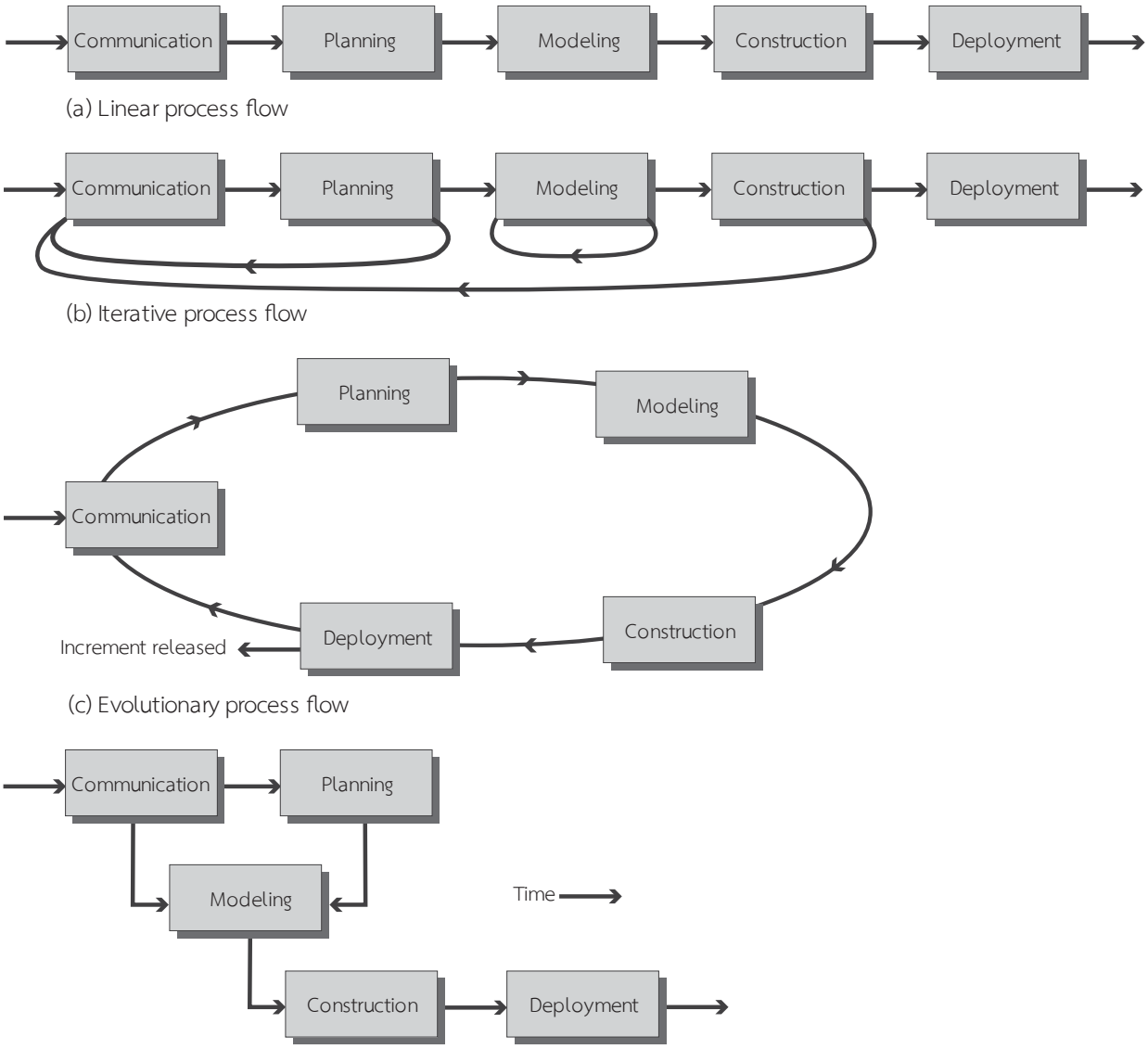
หลังจากทำความเข้าใจกระบวนการซอฟต์แวร์และกรอบงานกระบวนการซอฟต์แวร์ จะเห็นว่ากระบวนการซอฟต์แวร์เป็นแนวคิดที่เป็นนามธรรมสูง การทำความเข้าใจต้องใช้การอธิบายกิจกรรมต่าง ๆ เข้ามาช่วย และการใช้แบบจำลองเพื่อเป็นสื่อในการอธิบายกระบวนการซอฟต์แวร์นั้นทำให้เห็นภาพได้อย่างชัดเจนโดยเฉพาะขั้นตอนต่าง ๆ ภายในกระบวนการซอฟต์แวร์ ลำดับการทำงานของชุดกิจกรรม ใครรับผิดชอบหน้าที่อะไร ระบุชุดงานที่ต้องทำ ข้อมูลเข้าและออกเป็นอย่างไร มีเงื่อนไขอย่างไร ลำดับการไหลของงานเป็นอย่างไร ล้วนสามารถอธิบายได้ด้วยแบบจำลองกระบวนการซอฟต์แวร์

แบบจำลองกระบวนการซอฟต์แวร์สามารถแบ่งออกได้หลายประเภทขึ้นอยู่กับเกณฑ์ในการจำแนก สำหรับเอกสารคำสอนนี้จะใช้เกณฑ์ในการจำแนกแบบจำลองกระบวนการซอฟต์แวร์เป็นสามเกณฑ์คือ แบ่งตามลักษณะการไหลของกระบวนการซอฟต์แวร์ แบ่งตามรูปแบบระบบการพัฒนาซอฟต์แวร์ และแบ่งตามรูปแบบการเกณฑ์การดำเนินการกระบวนการซอฟต์แวร์

ลักษณะแบบจำลองแบ่งตามการไหลกระบวนการซอฟต์แวร์

แบบจำลองกระบวนการซอฟต์แวร์สามารถแบ่งตามการไหลกระบวนการซอฟต์แวร์ สามารถแบ่งออกเป็นสี่ประเภทตามลักษณะการไหลของกระบวนการดังนี้ การไหลกระบวนการแบบเชิงเส้น การไหลกระบวนการแบบวนซ้ำ การไหลกระบวนการแบบวิวัฒนาการ และการไหลกระบวนการแบบขนาน สำหรับการอิบายแบบจำลองในหัวข้อนี้จะแบ่งเป็น 5 กิจกรรมหลักดังนี้

1. **การสื่อสาร (communication)** การสื่อสารเพื่อทำความเข้าใจระหว่างลูกค้าและทีมพัฒนา เพื่อให้ได้มาซึ่งความต้องการนิยามคุณลักษณะและฟังก์ชันของซอฟต์แวร์
2. **การวางแผน (planning)** การสร้างแผนการพัฒนาซอฟต์แวร์เพื่อใช้ในการดำเนินงานพัฒนา ประกอบด้วย แผนงานที่ต้องทำ การประเมินความเสี่ยง การวางแผนทรัพยากร และการจัดตารางเวลาในการพัฒนา
3. **การสร้างแบบจำลอง (modeling)** การสร้างแบบจำลองเพื่อทำความเข้าใจปัญหาและความต้องการของลูกค้า โดยการวิเคราะห์และการออกแบบ และสร้างแบบจำลอง
4. **การสร้างซอฟต์แวร์ (construction)** คือการสร้างซอฟต์แวร์ตามที่ได้ออกแบบไว้ และรวมถึงการทดสอบซอฟต์แวร์
5. **การปรับใช้ซอฟต์แวร์ (deployment)** คือการนำเอาซอฟต์แวร์ที่สร้างไปติดตั้งพร้อมใช้งานบางส่วนหรือทั้งหมดโดยผู้ใช้หรือลูกค้า รวมถึงการส่งมอบผลิตภัณฑ์ซอฟต์แวร์ และการเก็บผลป้อนกลับมา สำหรับการบำรุงรักษา



รูปที่ 2.2 รูปแบบการไหลกระบวนการซอฟต์แวร์ [2]

1. **การไหลกระบวนการแบบเชิงเส้น (linear process flow)** ทำกิจกรรมเรียงตามลำดับเป็นขั้นตอนเริ่มจาก การสื่อสาร การวางแผน การสร้างแบบจำลอง การสร้างซอฟต์แวร์ การปรับใช้ซอฟต์แวร์ เพื่อส่งมอบ
2. **การไหลกระบวนการแบบวนซ้ำ (iterative process flow)** เป็นกระบวนการพัฒนาแบบวนซ้ำในบางกิจกรรมที่จำเป็น
3. **การไหลกระบวนการแบบวิวัฒนาการ (evolutionary process flow)** การพัฒนาการเชิงวิวัฒนาการแบบวนครบรอบโดยการทยอยส่งมอบให้ผู้ใช้ทีละส่วนในแต่ละรอบของกระบวนการพัฒนาซอฟต์แวร์
4. **การไหลกระบวนการแบบขนาน (parallel process flow)** รูปแบบนี้สามารถทำบางกิจกรรมควบคู่แบบขนานหรือเหลื่อมเวลาไปพร้อมๆกันได้เช่น การสร้างแบบจำลองสามารถทำควบคู่ไปกับการสร้างซอฟต์แวร์ได้

ลักษณะแบบจำลองแบ่งตามรูปแบบระบบการพัฒนาซอฟต์แวร์

รูปแบบการพัฒนาสามารถจำแนกแบบง่ายเป็นสองกลุ่มใหญ่โดยแบ่งตามระบบกระบวนการพัฒนา ว่ามีระบบแบบแผนหรือไม่ กลุ่มแรกคือแบบจำลองการพัฒนาซอฟต์แวร์ที่ไม่เป็นระบบ และ อีกกลุ่มคือแบบจำลองการพัฒนาซอฟต์แวร์แบบเป็นระบบ

1. **แบบจำลองกระบวนการซอฟต์แวร์แบบไม่เป็นระบบ** คือการพัฒนาที่ไม่มีระบบแบบแผนการพัฒนา ไม่มีการบริหารจัดการ ไม่มีการจัดทำเอกสาร เหมาะสำหรับการสร้างผลิตภัณฑ์ขนาดเล็กและไม่ซับซ้อน พัฒนาคนเดียว พัฒนาเพื่อใช้งานเองหรือระบบที่หากเกิดข้อผิดพลาดแล้วไม่ส่งผลกระทบมาก เป็นลักษณะเขียนโปรแกรมเพื่อแก้ปัญหา หรือตามความต้องการ เช่น แบบจำลองสร้างและแก้ปัญหา (build and fix model) เป็นแบบจำลองการพัฒนาซอฟต์แวร์ที่มีการเขียนโปรแกรม หรือการแก้ไขโปรแกรมเรื่อย ๆ โดยการลองผิดลองถูก จนกว่าจะได้ซอฟต์แวร์ตรงความต้องการของผู้ใช้
2. **แบบจำลองกระบวนการซอฟต์แวร์แบบเป็นระบบ** คือแบบจำลองที่มีการวางระบบแบบแผนในการพัฒนาซอฟต์แวร์ มีการบริหารจัดการที่ดี รองรับการพัฒนาแบบทีม เพื่อสร้างซอฟต์แวร์ที่มีคุณภาพ คุ้มค่า ตรงตามความต้องการของลูกค้าภายในระยะเวลาที่กำหนด ด้วยหลักการทางวิศวกรรมซอฟต์แวร์ ซึ่งในปัจจุบันซอฟต์แวร์ส่วนมากมีความซับซ้อนและขนาดใหญ่จึงจำเป็นต้องนำแบบจำลองการพัฒนาซอฟต์แวร์แบบเป็นระบบมาใช้ ยกตัวอย่างเช่น แบบจำลองแบบน้ำตก แบบจำลองเร่งรัด แบบจำลองต้นแบบ และแบบจำลองเวียนก้นหอย เป็นต้น

ลักษณะแบ่งตามรูปแบบการดำเนินการในกระบวนการซอฟต์แวร์

แบบจำลองกระบวนการซอฟต์แวร์สามารถแบ่งโดยใช้เกณฑ์การดำเนินการออกเป็นสองกลุ่ม แบบจำลองกระบวนการซอฟต์แวร์ขับเคลื่อนตามแผน และแบบจำลองกระบวนการซอฟต์แวร์ที่ดำเนินการแบบคล่องตัว (เอจายล์) สำหรับแบบจำลองกระบวนการซอฟต์แวร์ขับเคลื่อนตามแผนนั้นกระบวนการและกิจกรรมต่าง ๆ ถูกกำหนดไว้ล่วงหน้าในแผนเพื่อให้ดำเนินการไปตามแผนที่วางไว้โดยใช้แผนนี้ในการวัดและประเมินความก้าวหน้าเช่น แบบจำลองน้ำตก

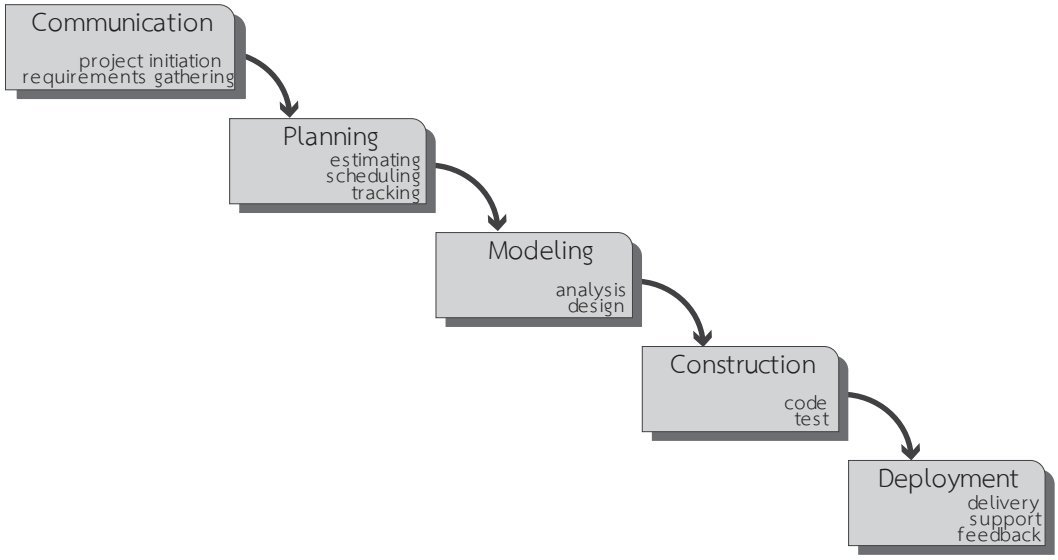
สำหรับแบบจำลองกระบวนการซอฟต์แวร์ในกลุ่มเอจายล์นั้น การวางแผนจะเป็นลักษณะค่อยๆเพิ่มขึ้นในระหว่างการดำเนินการพัฒนาเพื่อให้ง่ายต่อการเปลี่ยนแปลงกระบวนการรองรับความเปลี่ยนแปลงความต้องการของลูกค้า เช่นแบบจำลองการโปรแกรมสุดขีด (extreme programming: XP)

แต่อย่างไรก็ตามในทางปฏิบัติหลายๆกระบวนการซอฟต์แวร์ก็ใช้ทั้งสองแนวทางผสมผสานกันทั้งการดำเนินการขับเคลื่อนตามแผน และแบบเอจายล์ ตามความเหมาะสมในการพัฒนา ไม่มีหลักเกณฑ์ตายตัวเป็นหน้าที่วิศวกรซอฟต์แวร์ที่ต้องทำการวิเคราะห์ ประเมิน และเลือกให้เหมาะสมในการดำเนินการพัฒนาซอฟต์แวร์เป็นกรณี ๆ ไป เนื้อหาส่วนถัดไปจะกล่าวถึงแบบจำลองกระบวนการซอฟต์แวร์ที่เป็นที่นิยม

แบบจำลองน้ำตก (waterfall model)

แบบจำลองน้ำตกหรืออาจเรียกว่าแบบจำลองคลาสสิก (classic model) คิดค้นโดย Winston W. Royce ปีค.ศ. 1970 และมีการใช้แพร่หลายมาถึงปัจจุบัน โดย เป็นแบบจำลองกระบวนการซอฟต์แวร์ที่มีขั้นตอนการดำเนินงานที่ชัดเจนและง่ายต่อการนำไปใช้จริง ไม่ซับซ้อนเรียงลำดับกิจกรรมแบบไหลเชิงเส้น กระบวนการพัฒนาซอฟต์แวร์มีแบบแผนเป็นระบบ เริ่มจาก

การทำงานของแบบจำลองน้ำตกจะเริ่มกระบวนแต่ละกิจกรรม ไล่เรียงไปเป็นลำดับไม่สามารถข้ามกิจกรรมและไม่สามารถย้อนกลับไปทำกิจกรรมที่เสร็จสิ้นไปแล้วได้ เปรียบเสมือนการไหลของน้ำตกจากชั้นบนลงชั้นล่างลดหลั่นกันไปและสายน้ำไม่สามารถย้อนกลับไปได้ ผลจากกิจกรรมเมื่อเสร็จสิ้นจะส่งต่อเป็นตัวนำเข้าของกิจกรรมถัดไป ดังรูป 2.3

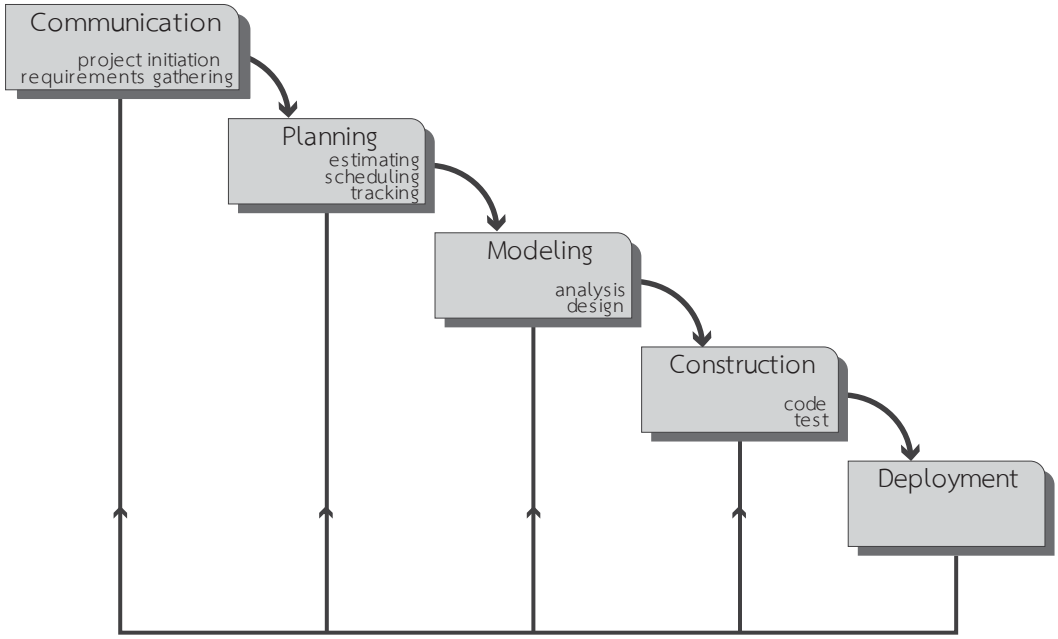


รูปที่ 2.3 แบบจำลองน้ำตก ดัดแปลงจาก [2]

1. **การสื่อสาร** การเริ่มโครงการพัฒนาซอฟต์แวร์ และการเก็บรวบรวมความต้องการจากลูกค้า
2. **การวางแผน** การประเมินและประมาณการใช้ทรัพยากรทั้งค่าใช้จ่ายกำลังคนและเวลาในการดำเนินโครงการ และทำการกำหนดตรารางเวลาในการพัฒนารวมถึงกำหนดวิธีการประเมินโครงการ
3. **การสร้างแบบจำลอง** การทำความเข้าใจระบบเพื่อทำการวิเคราะห์และออกอบบระบบเพื่อตอบสนองควาใมต้องการของลูกค้า
4. **การสร้างซอฟต์แวร์** กระเขียนโปรแกรมเพื่อสร้างซอฟต์แวร์ตามการออกแบบที่วางไว้ รวมทั้งต้องทำการทดสอบเพื่อให้มั้นใจว่าซอฟต์แวร์ทำงานถูกต้องอย่างมีประสิทธิภาพ
5. **การปรับใช้ซอฟต์แวร์** การส่งมอบชิ้นงานหรือซอฟต์แวร์ให้ลูกค้าเพื่อนำไปใช้ เก็บข้อมูลป้อนกลับมาปรับปรุงซอฟต์แวร์และส่งต่อให้ทีมบำรุงรักษาซอฟต์แวร์ดูแลต่อได้

แบบจำลองน้ำตกนับว่าเป็นแบบจำลองกระบวนการซอฟต์แวร์ที่เก่าแก่ เหมาะสำหรับโครงการใหญ่และซับซ้อน เนื่องจากต้องทำเอกสารให้ชัดเจนและรอบคลุมในทุก ๆ กิจกรรม ทำให้ทีมพัฒนาขนาดใหญ่สามารถขับเคลื่อนไปได้อย่างราบลื่น เนื่องจากมีผลในแต่ละกิจกรรมชัดเจน สามารถแบ่งงานออกเป็นงานย่อย ๆ

เนื่องจากข้อจำกัดของแบบจำลองน้ำตกที่ไม่สามารถย้อนกลับไปทำกิจกรรมที่เสร็จสิ้นไปแล้วได้ ทำให้การจัดการกับการเปลี่ยนแปลงเป็นไปได้ลำบาก และถ้ามีการเปลี่ยนแปลงต้องเสียเวลากลับไปเริ่มกระบวนการใหม่ตั้งแต่ต้น ดังนั้นจึงมีการเสนอแบบจำลองน้ำตกแบบวนซ้ำซึ่งเป็นรูปแบบการไหลกระบวนการแบบวนซ้ำ ดังรูปที่ 2.4 เพื่อให้สามารถวนกลับไปทำซ้ำในกิจกรรมที่จะเป็นเพื่อเพิ่มความยืดหยุ่นในการพัฒนาและเอื้อต่อการจัดการการเปลี่ยนแปลงได้ดีขึ้นกว่าแบบจำลองน้ำตกต้นฉบับ



รูปที่ 2.4 แบบจำลองน้ำตกแบบวนซ้ำ

ข้อดี

1. กระบวนการเข้าใจง่าย แบ่งกิจกรรมชัดเจน
2. การวิเคราะห์และทดสอบทำได้แบบตรงไปตรงมา
3. เอกสารชัดเจนทุกขั้นตอน เอกสารการพัฒนาซอฟต์แวร์ครบถ้วนส่งต่อให้ทีมบำรุงรักษาดูแลต่อง่าย
4. แต่ละกิจกรรมแยกจากกันทำให้บริหารจัดการและติดตามการพัฒนาได้ง่าย

ข้อเสีย

1. กระบวนการไม่เอื้อในการเปลี่ยนแปลงความต้องการลูกค้า การแก้ไขยาก ค่าใช้จ่ายสูง และ เสียเวลา
2. การทดสอบไปอยู่ช่วงท้ายกระบวนการ
3. ลูกค้าตรวจรับซอฟต์แวร์และทดลองใช้ครั้งเดียวหลังจบกระบวนการซึ่งอาจสายเกินไปหากซอฟต์แวร์ไม่ตรงความต้องการ

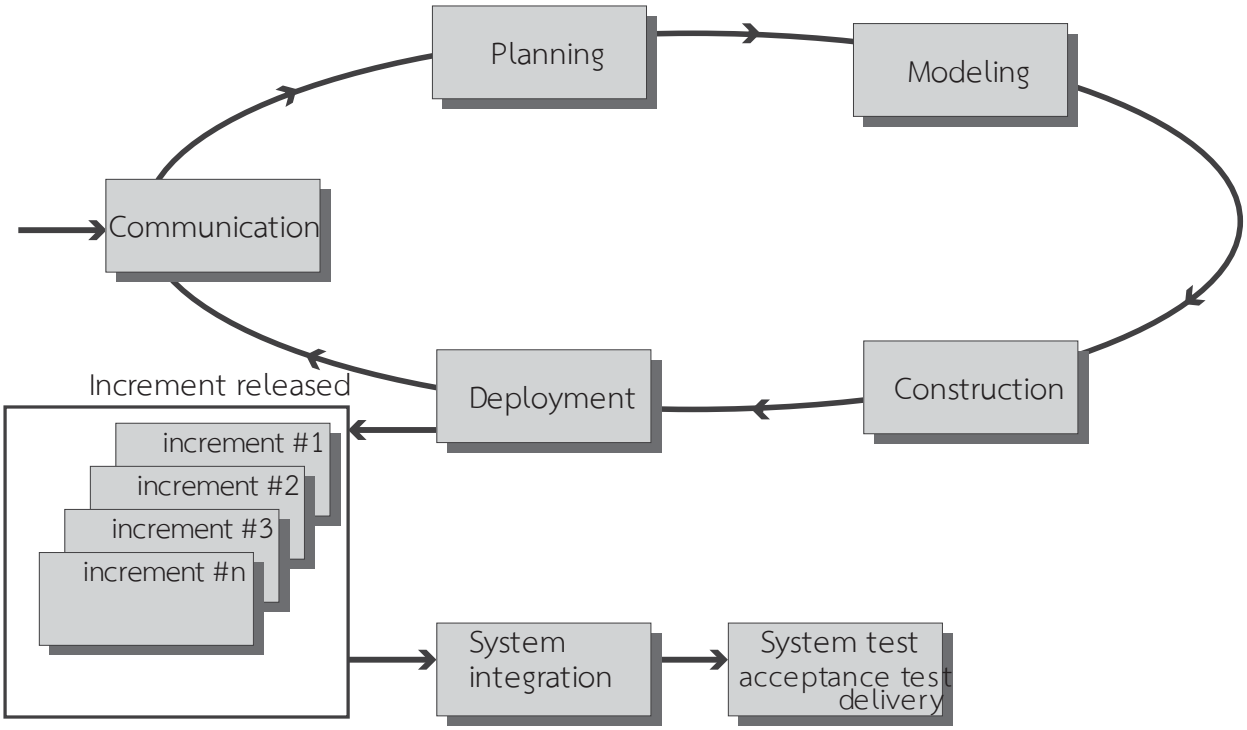
แบบจำลองเพิ่มเติม (incremental model)

แบบจำลองแบบเพิ่มเติม เป็นแบบจำลองที่มีลักษณะการไหลแบบกระบวนการวิวัฒนาการ มีหลักการคือแบ่งงานใหญ่ออกเป็นงานย่อย ๆที่มีขนาดพอเหมาะในการพัฒนาได้ในรอบระยะเวลาสั้นจากโครงการใหญ่แตกเป้นโครงการย่อยเล็ก ๆ และนำมาจัดลำดับ

ความสำคัญก่อนหลังในการพัฒนาแต่ละโครงการย่อย โดยที่ในแต่ละรอบของกระบวนการพัฒนา จะได้ซอฟต์แวร์เพิ่มทีละส่วน จนครบได้ซอฟต์แวร์ที่สมบูรณ์ดังรูปที่ 2.5

ขั้นตอนกิจกรรมในกระบวนการจะเป็นลักษณะวนรอบเพื่อค่อยๆเพิ่มซอฟต์แวร์ที่กำลังพัฒนาทีละส่วนโดยเริ่มขั้นตอนการเริ่มต้นโครงการ รวบรวมความต้องการและ วางแผนและแบ่งการพัฒนาจากข้อกำหนดความต้องการออกเป็นส่วนๆ แล้วออกแบบซอฟต์แวร์ในส่วนที่แบ่งมาทำงานในรอบ เพื่อส่งต่อไปสร้างและทดสอบซอฟต์แวร์ส่วนที่เพิ่มนี้ เพื่อส่งมอบซอฟต์แวร์ เป็นส่วนโดยจะทำกระบวนการวิวัฒนาการแบบนี้ไปจนซอฟต์แวร์ครบสมบูรณ์ตามความต้องการ ซึ่งจะนำซอฟต์แวร์ในแต่ละส่วนเพิ่มมารวมเข้าเป็นซอฟต์แวร์สมบูรณ์เพื่อทำการทดสอบและส่งมอบให้ลูกค้าในขั้นตอนสุดท้าย

การรวมระบบและทดสอบนั้นสามารถทำได้สองแนวทางคือ การทยอยส่งมอบซอฟต์แวร์ส่วนเพิ่มพร้อมการทดสอบทีละชุดในแต่ละรอบแล้วค่อยทำการรวมระบบสมบูรณ์พร้อมทดสอบระบบสมบูรณ์ในขั้นตอนสุดท้ายทีเดียว หรือจะทยอยส่งมอบซอฟต์แวร์ส่วนเพิ่มที่ทดสอบแล้วนำมารวมเข้ากับระบบที่ส่งมอบก่อนหน้าแล้วทดสอบเพื่อค่อยๆเพิ่มเติมระบบให้สมบูรณ์ไปอย่างต่อเนื่อง ซึ่งการเลือกวิธีรวมระบบก็ขึ้นอยู่รูปแบบในแต่ละโครงการ รูปแบบกาทำงานและความต้องการของลูกค้า



รูปที่ 2.5 แบบจำลองเพิ่มเติม

ข้อดี

1. ลดความเสี่ยงโดยการแบ่งงานเป็นส่วนย่อย ๆเพื่อพัฒนา
2. จัดการกับการเปลี่ยนแปลงความต้องการของลูกค้าได้ดี เนื่องจากกระบวนการในแต่ละรอบไม่ยาว

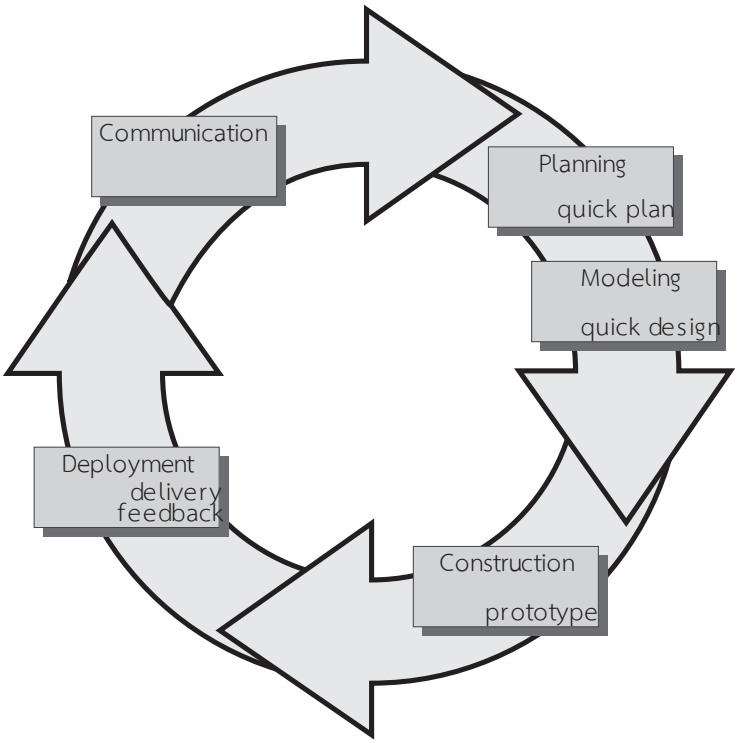
3. ลูกค้าสามารถทดลองใช้ได้แต่ช่วงแรกของการพัฒนาหรือซอฟต์แวร์ที่มีความจำเป็นบางส่วนสามารถนำไปใช้งานได้ก่อน
4. ได้ผลป้อนกลับจากลูกค้าแต่เนิ่น ๆ ลูกค้าเห็นภาพซอฟต์แวร์ได้เร็ว

ข้อเสีย

1. อาจมีความเสี่ยงเมื่อนำส่วนเพิ่มมารวมกันเป็นซอฟต์แวร์สมบูรณ์
2. การวางแผนและควบคุมกระบวนการพัฒนาค่อนข้างยาก

แบบจำลองกระบวนการต้นแบบ (prototyping process model)

เป็นแบบจำลองที่มีลักษณะการไหลแบบกระบวนการวิวัฒนาการ ที่มีการวนซ้ำโดยการสร้างซอฟต์แวร์ออกมาทีละรุ่น โดยค่อยๆ เพิ่มความสมบูรณ์ของซอฟต์แวร์ขึ้นเรื่อย ๆ จนได้ซอฟต์แวร์รุ่นที่สมบูรณ์ ซึ่งแบบจำลองกระบวนการต้นแบบช่วยให้สามารถสร้างต้นแบบขึ้นมาทดลองใช้ดูทำให้ลูกค้าและทีมพัฒนาเห็นภาพรวมของซอฟต์แวร์ร่วมกัน และเพิ่มความเข้าใจในการวิเคราะห์ระบบและเข้าใจความต้องการมากขึ้น จากความต้องการของลูกค้าช่วงแรกที่อาจยังไม่ชัดเจน การทำต้นแบบจะทยอยเพิ่มความต้องการเข้าไป ซึ่งภาพรวมกระบวนการจะมีลักษณะดังรูปที่ 2.6



รูปที่ 2.6 แบบจำลองกระบวนการต้นแบบ ดัดแปลงจาก [2]

ขั้นตอนแบบจำลองกระบวนการต้นแบบ เริ่มที่การสื่อการเพื่อเก็บรวบรวมปัญหาและความต้องการของลูกค้าและผู้เกี่ยวข้อง เพื่อให้ได้เค้าโครงของซอฟต์แวร์สำหรับการวางแผนพัฒนาในแต่ละรอบ โดยจะทำการวางแผนอย่างรวดเร็วสำหรับการพัฒนาในรอบ และจากกนั้นก็ทำการออกแบบอย่างรวดเร็วเฉพาะสำหรับการพัฒนาในรอบ โดยอาจเน้นที่การออกแบบเพื่อให้ผู้ใช้สามารถเห็นภาพและรูปแบบการทำงานซอฟต์แวร์ เช่น แสดงส่วนประสานผู้ใช้ และจากนั้นจำดำเนินการสร้างซอฟต์แวร์ต้นแบบ

ขึ้นมาให้ผู้ใช้หรือลูกค้าทดลองใช้เพื่อเก็บผลป้อนกลับสำหรับกระบวนการรอบถัดไป ซึ่งผลป้อนกลับอาจนำมาใช้ปรับเปลี่ยนความต้องการรวมทั้งวางแผนการพัฒนาในวงรอบถัดไป ซอฟต์แวร์ต้นแบบสามารถช่วยให้ผู้มีส่วนเกี่ยวข้องและทีมพัฒนาเข้าใจและเห็นภาพรวมของซอฟต์แวร์ได้ใกล้เคียงกัน โดยแนวทางการพัฒนาต้นแบบมีด้วยกันสองแนวทางดังนี้

1. ต้นแบบโยนทิ้ง (throw-away prototype) เป็นการพัฒนาต้นแบบเพื่อให้ผู้ใช้เห็นภาพรวมการทำงานของซอฟต์แวร์และทดสอบว่าตรงกับความต้องการหรือไม่ โดยเน้นการพัฒนาขึ้นมาให้รวดเร็วสร้างบางฟังก์ชันการทำงานหรือมีเพียงส่วนประสาน ต้นแบบโยนทิ้งอาจใช้เครื่องมือสร้างต้นแบบพัฒนาเพื่อความรวดเร็วได้เช่น อะโดบี เอ็กดี (Adobe XD) เป็นต้น หลังจากสร้างต้นแบบเสร็จแล้วจะไม่นำมาใช้งานจริงโดยจะทิ้งไปทันทีเนื่องจากส่วนมากโครงสร้างและการออกแบบและพัฒนาต้นแบบนั้น จะไม่ดีจึงควรทิ้งและพัฒนาซอฟต์แวร์จริงๆขึ้นมาใหม่
2. ต้นแบบวิวัฒนาการ (evolutionary prototype) เป็นการพัฒนาต้นแบบขึ้นโดยมีวัตถุประสงค์เพื่อขยายต้นแบบให้กลายไปเป็นซอฟต์แวร์ที่สมบูรณ์ ฉะนั้นการทำต้นแบบวิวัฒนาการ จะดำเนินงานอย่างรอบคอบเน้นการทยอยขยายฟังก์ชันหรือคุณสมบัติของต้นแบบไปทีละส่วนจนกลายไปเป็นซอฟต์แวร์ที่สมบูรณ์ ส่วนมากจะเริ่มจากการเลือกฟังก์ชันหรือลักษณะสำคัญของซอฟต์แวร์มาพัฒนาก่อน โดยร่วมกับผู้เกี่ยวข้องเพื่อกำหนดลำดับในการพัฒนาในแต่ละรอบ ผู้ใช้จะได้รับต้นแบบที่เป็นซอฟต์แวร์จริงใช้งานก่อนในบางฟังก์ชัน แล้วจึงค่อยๆเพิ่มจนสมบูรณ์

   ข้อดี

1. ลดความเสี่ยงโดยการใช้ต้นแบบเป็นตัวประเมินความก้าวหน้า
2. จัดการกับการเปลี่ยนแปลงความต้องการของลูกค้าได้ดี เนื่องจากกระบวนการในแต่ละรอบไม่ยาว
3. ลูกค้าสามารถทดลองใช้ได้แต่ต้นๆหรือซอฟต์แวร์ที่มีความจำเป็นบางส่วนสามารถนำไปใช้งานได้ก่อน
4. ได้ผลป้อนกลับจากลูกค้าแต่เนิ่น ๆ ลูกค้าและทีมพัฒนาเข้าใจซอฟต์แวร์ได้เร็ว

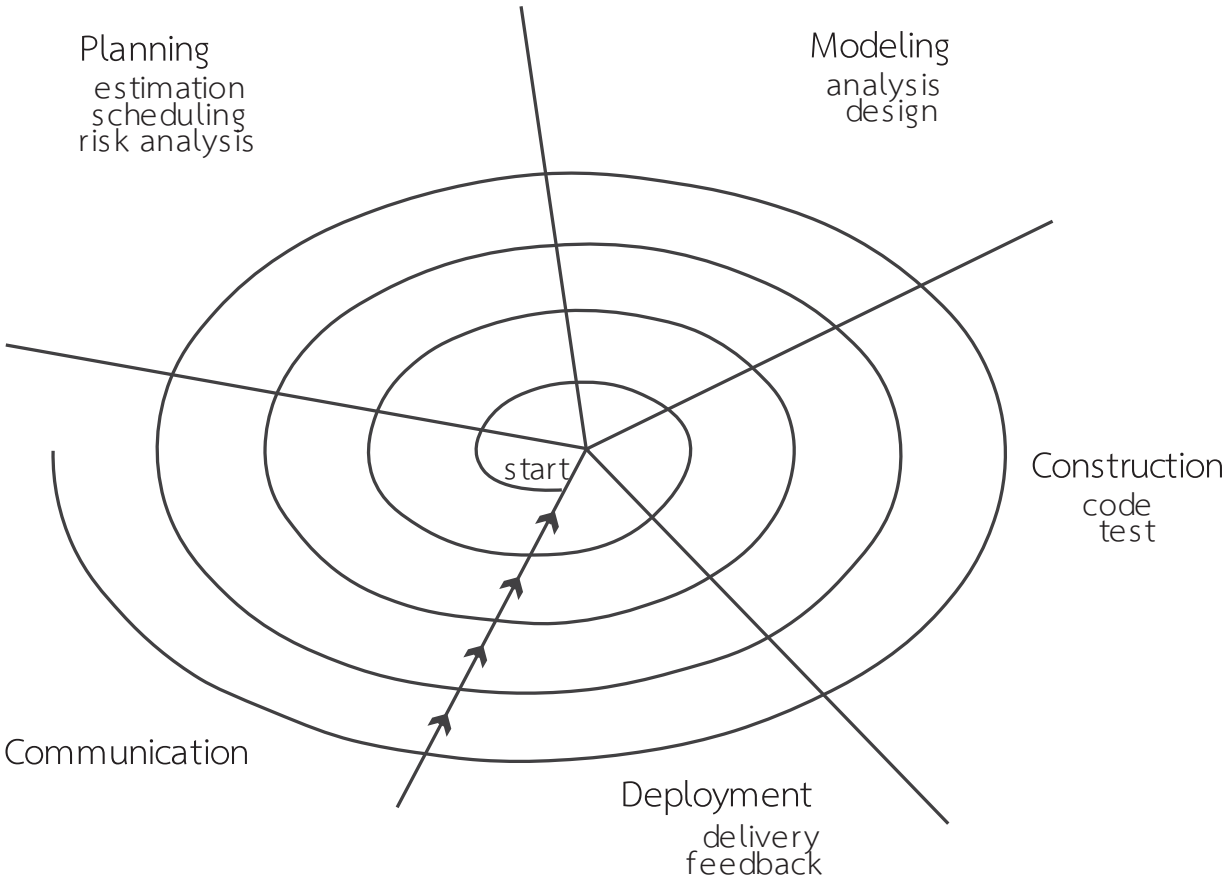
   ข้อเสีย

1. ต้องพัฒนาเร็วทีมพัฒนาต้องมีศักยภาพสูงและใช้ทรัพยากรมาก ค่าใช้จ่ายสูง
2. โครงสร้างของซอฟต์แวร์ต้นแบบจะถูกบั่นทอนลงในทุก ๆ รอบหรือมีการเปลี่ยนแปลง
3. การวางแผนและควบคุมกระบวนการพัฒนาค่อนข้างยาก
4. ไม่เหมาะสำหรับโครงการขนาดใหญ่
5. สิ้นเปลืองทรัพยากรในการจัดทำเอกสาร

แบบจำลองกระบวนการก้นหอย (spiral model)

แบบจำลองกระบวนการก้นหอยมีหลักการทำงานวนรอบคล้ายวงก้นหอย เป็นลักษณะการไหลแบบวนซ้ำผสมแบบวิวัฒนาการ เป็นการพัฒนาแบบค่อยเป็นค่อยไปที่มีจุดสำคัญคือการวิเคราะห์ความเสี่ยงหรืออาจเรียกแบบจำลองนี้อีกชื่อว่า แบบจำลองขับเคลื่อนความเสี่ยง กระบวนการจะแบ่งเป็นรอบๆเมื่อจบแต่ละรอบจะได้ส่วนซอฟต์แวร์ที่ใช้งานได้และดำเนินการวิเคราะห์และประเมินความเสี่ยงเพื่อวางแผนการพัฒนาในรอบถัดไปโดยขั้นตอนหลัก ๆ จะวนรอบ การสื่อสาร การวางแผน การสร้างแบบจำลอง การสร้างซอฟต์แวร์ และ การปรับใช้โดยจะมีกระบวนการวิเคราะห์และประเมินความเสี่ยงในทุกๆรอบกระบวนการพัฒนา ดังรูปที่ 2.7

การวิเคราะห์และประเมินความเสี่ยงช่วยให้การบริหารจัดการโครงการเป็นไปอย่างราบลื่นและหาวิธีลดผลกระทบจากความเสี่ยงระหว่างการพัฒนา โดยมีขั้นตอนเริ่มจากการ พิจารณาโครงการระบุข้อจำกัดต่างๆของโครงการในด้านต่าง ๆ ระบุความเสี่ยง หาวิธีลดหรือแก้ไขความเสี่ยง ประเมินผลกระทบ และ ทำการวางกลยุทธ์ หรือแนวทางจัดการความเสี่ยงในรอบการทำงานซึ่งจะกระทำซ้ำ ๆ ในทุก ๆ รอบการพัฒนาจนส่งซอฟต์แวร์สมบูรณ์



รูปที่ 2.7 แบบจำลองกระบวนการก้นหอย [2]

แบบจำลองการพัฒนาโปรแกรมประยุกต์อย่างรวดเร็ว (rapid application development model: RAD)

เป็นการพัฒนาโปรแกรมประยุกต์อย่างรวดเร็ว โดยอาศัยการใช้เครื่องมือสนับสนุนด้านวิศวกรรมซอฟต์แวร์ (computer-aided software engineering: CASE) หรือเรียกว่าเครื่องมือเคส การใช้เครื่องมือช่วยการพัฒนาซอฟต์แวร์ได้อย่างรวดเร็ว ลดต้นทุนและเวลาในการพัฒนาลงได้ สำหรับแบบจำลองนี้มีลักษณะการไหลแบบกระบวนการขนาน การแบ่ง และ การกระจายงานออกเป็นส่วน ๆ สำหรับทีมพัฒนาหลายทีมช่วยกันพัฒนาแบบขนานพร้อมกัน แล้วนำมารวมและทดสอบเป็นซอฟต์แวร์สมบูรณ์ ยิ่งใช้งานรวมกับเครื่องมือเคสก็จะสามารถลดระยะเวลาการพัฒนาให้สั้นลงได้

เครื่องมือเคสนั้น รองรับการทำงานแบบกึ่งหรืออัตโนมัติในกิจกรรมต่าง ๆ การพัฒนาซอฟต์แวร์ ครอบคุมตั้งแต่การเก็บข้อมูลความต้องการ การวิเคราะห์ การออกแบบ การพัฒนา การทดสอบ และ การปรับใช้ ทำให้สามารถสร้างซอฟต์แวร์คุณภาพสูง ความผิดพลาดต่ำ โดยเฉพาะฟังก์ชันสำคัญที่ช่วยลดระยะเวลาการพัฒนา คือเครื่องมือการสร้างรหัสต้นฉบับแบบอัตโนมัติ โดยการสร้างขึ้นจากการออกแบบนอกจากนั้น รหัสต้นฉบับที่ถูกสร้างขึ้นจากเครื่องมือนั้นมักจะถูกต้องและมีความน่าเชื่อถือสูง เพราะเกิดจากรูปแบบกระบวนการออกแบบและพัฒนาที่ถูกทดสอบมาอย่างดีแล้ว

ข้อดี

1. ลดระยะเวลาการพัฒนาซอฟต์แวร์ โดยใช้เครื่องมือสนับสนุนด้านวิศวกรรมซอฟต์แวร์
2. จัดการกับการเปลี่ยนแปลงความต้องการของลูกค้าได้
3. ลดต้นทุนประหยัดทรัพยากรในการพัฒนา โดยเฉพาะคนและเวลา
4. ได้ผลป้อนกลับจากลูกค้าแต่เนิ่น ๆ ลูกค้าและทีมพัฒนาเข้าใจซอฟต์แวร์ได้เร็ว
5. สนับสนุนการนำซอฟต์แวร์กลับมาใช้ใหม่ (software reuse)

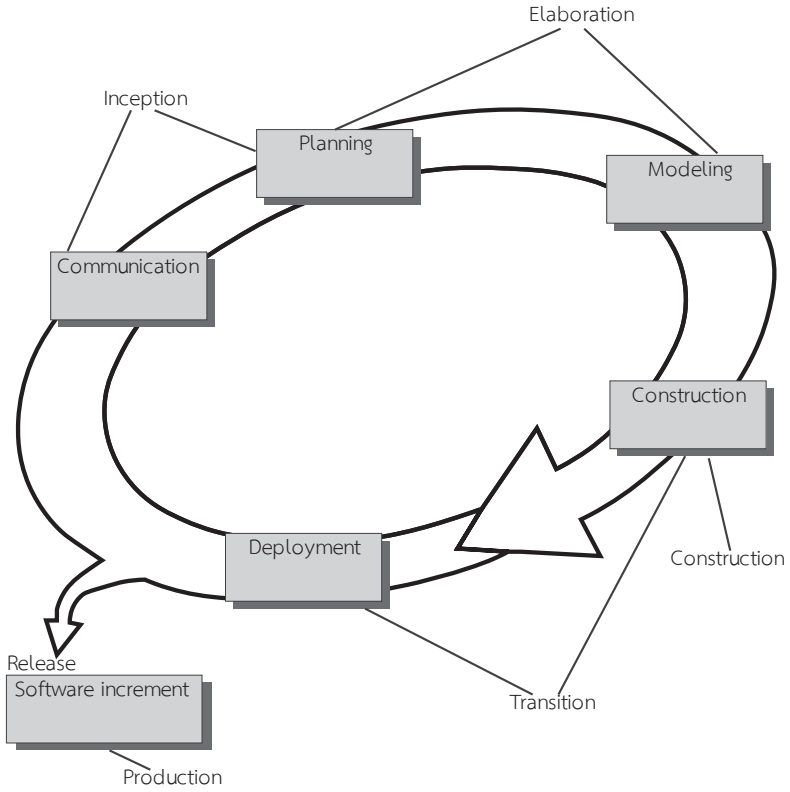
ข้อเสีย

1. ไม่เหมาะกับการพัฒนาขนาดใหญ่ เพราะต้องใช้ทีมพัฒนาขนาดใหญ่และใช้ทรัพยากรมาก ค่าใช้จ่ายสูง
2. ต้องการการมีส่วนร่วมของผู้ใช้หรือลูกค้าตลอดกระบวนการ
3. ทีมพัฒนาต้องมีศึกยภาพสูงโดยเฉพาะการออกแบบ
4. ไม่เหมาะสำหรับโครงการที่งบประมาณจำกัดเพราะเครื่องมือสนับสนุนด้านวิศวกรรมซอฟต์แวร์ราคาสูง

แบบจำลองกระบวนการรวม (unified process model: UP)

แบบจำลองกระบวนการรวมมีลักษณะการไหลกระบวนการแบบวนซ้ำผสานวิวัฒนาการโดยใช้วิธีการพัฒนาเชิงวัตถุเป็นแกน และใช้ภาษายูเอ็มแอล (unified modeling language) ในการอธิบายแบบจำลองกระบวนการรวม โดยมีวัตถุประสงค์เพื่อสร้างซอฟต์แวร์ที่มีคุณภาพสูง ตรงตามความต้องการลูกค้า ในกรอบเวลา และ งบประมาณที่กำหนด

การกำหนดความต้องการสามารถทยอยแบ่งย่อยเป็นส่วนๆกระทำควบคู่ไปกับการพัฒนาซอฟต์แวร์เพื่อให้สามารถรองรับการเปลี่ยนแปลงที่อาจเกิดขึ้นได้ตลอดการพัฒนา การที่แบบจำลองใช้ภาษายูเอ็มแอลและแผนภาพในการอธิบาย การสื่อสาร และการจัดการในกระบวนการพัฒนา รองรับการสร้างซอฟต์แวร์ขนาดใหญ่ได้เป็นอย่างดี นอกจากนั้นภาษายูเอ็มแอลในปัจจุบันเป็นมาตรฐานที่ใช้อย่างแพร่หลายโดยเฉพาะในการพัฒนาเชิงวัตถุ แบบจำลองกระบวนการรวมประกอบด้วย 5 ระยะในรูปแบบการวนรอบและวิวัฒนาการ ดังรูปที่ 2.8



ดังรูปที่ 2.8 แบบจำลองกระบวนการรวม [2]

1. **ระยะเริ่มต้น (inception phase)** เป็นการกำหนดขอบเขต เป้าหมาย คุณลักษณะ ฟังก์ชัน ของซอฟต์แวร์ รวมทั้งวางแผนทรัพยากร ประเมินความเสี่ยง และ เวลาในการพัฒนาในแต่ละรอบ
2. **ระยะเพิ่มรายละเอียด (elaboration phase)** เป็นกิจกรรมการวางแผนและการออกแบบ โดยการทำความเข้าใจปัญหาให้ชัดเจนว่าระบบทำงานอย่างไรโดยการวิเคราะห์และออกแบบ สร้างสถาปัตยกรรมพื้นฐานของระบบ ด้วยการใช้แบบจำลองยูสเคส แบบจำลองการวิเคราะห์ แบบจำลองการออกแบบ แบบจำลองการสร้าง และ แบบจำลองการปรับใช้ในการอธิบาย สำหรับการปรับเปลี่ยนแผนก็จะกระทำในระยะนี้เช่นกัน
3. **ระยะสร้าง (construction phase)** เป็นระยะที่เน้นกิจกรรมการสร้างซอฟต์แวร์ เพื่อสร้างรหัสต้นฉบับ ทำการทดสอบ ทำการรวม จัดทำเอกสาร ปรับปรุงแบบจำลองต่าง ๆ รวมทั้งการทำการทดสอบการยอมรับ (acceptance test) ก่อนการดำเนินการระยะถัดไป
4. **ระยะเปลี่ยนผ่าน (transition phase)** เป็นการปรับใช้ซอฟต์แวร์หรือส่วนของซอฟต์แวร์ที่จะส่งมอบในรอบการพัฒนาโดยการส่งมอบทั้งซอฟต์แวร์และเอกสารประกอบเพื่อให้ผู้ใช้สามารถทดสอบและนำไปใช้ แล้วจึงเก็บผลป้อนกลับมาใช้ปรับปรุงหรือเปลี่ยนแปลง สำหรับรอบถัดไป
5. **ระยะการผลิต (production phase)** เป็นระยะที่ซอฟต์แวร์ถูกนำไปปรับใช้โดยมีการคอยติดตามเฝ้าสังเกต การรายงานความผิดพลาด การร้องขอการเปลี่ยนแปลงถูกนำมาพิจารณา

หลังจากการดำเนินการพัฒนาในแต่ละรอบจะเห็นว่าระยะต่างๆสามารถทำแบบขนานเหลื่อมกันได้ เช่นในขณะที่ระยะสร้าง ระยะเปลี่ยนผ่าน ระยะการผลิตกำลังดำเนินการอยู่ งานรอบถัดไปก็สามารถดำเนินการไปได้

ข้อดี

1. เน้นคุณภาพของเอกสารโดยใช้ภาษายูเอ็มแอล
2. เหมาะกับโครงการที่ซับซ้อนและมีขนาดใหญ่
3. จัดการกับการเปลี่ยนแปลงความต้องการของลูกค้าได้ มีความยืดหยุ่นในการพัฒนา
4. รองรับการสนับสนุนและบำรุงรักษาซอฟต์แวร์อย่างดี ด้วยเอกสารที่ชัดเจนและมีมาตรฐาน
5. สนับสนุนการมีส่วนร่วมกับผู้ใช้หรือลูกค้าตลอดกระบวนการพัฒนา

ข้อเสีย

1. การทำงานแบบเหลื่อมของระยะต่างๆอาจก่อให้เกิดปัญหาได้
2. ทีมพัฒนาต้องมีศักยภาพสูง
3. การจัดทำเอกสารอาจมีปัญหาเช่นรหัสต้นฉบับไม่ตรงกับการออกแบบที่ปรับเปลี่ยนถ้าไม่มีกระบวนการจัดการเอกสารที่ดีโดยเฉพาะในโครงการขนาดใหญ่

กระบวนการซอฟต์แวร์ช่วยให้นักพัฒนาเข้าใจภาพรวมของกิจกรรมต่างๆที่ต้องทำเพื่อพัฒนาซอฟต์แวร์ให้มี คุณภาพ ตรงความต้องการของลูกค้า ภายใต้กรอบเวลาและทรัพยากรที่กำหนด การใช้แบบจำลองกระบวนการซอฟต์แวร์ช่วยให้การพัฒนาเป็นไปอย่างมีระบบมีแผนงานช่วยลดความเสี่ยง หลีกเลี่ยงความล้มเหลวในการพัฒนา และเป็นกรอบการดำเนินงานพัฒนาที่ประกอบไปด้วยกิจกรรมต่างที่ถูกนิยามและอธิบายไว้อย่างชัดเจน

ในระหว่างพัฒนาอาจมีการเปลี่ยนแปลงเกิดขึ้น ซึ่งกระบวนการซอฟต์แวร์สามารถช่วยจัดการกับการเปลี่ยนแปลงได้ โดยแบบจำลองซอฟต์แวร์ส่วนมากจะมีกลยุทธ์และกระบวนการในการจัดการการเปลี่ยนแปลงอยู่แล้ว ซึ่งหากกล่าวโดยสรุปจะได้ว่ากระบวนการซอฟต์แวร์คือแนวทางการพัฒนาซอฟต์แวร์อย่างเป็นระบบด้วยพื้นฐานวิศวกรรมซอฟต์แวร์โดยมี สี่ กลุ่มกิจกรรมหลัก ข้อกำหนดซอฟต์แวร์ การพัฒนาซอฟต์แวร์ การตรวจสอบความถูกต้องของซอฟต์แวร์ และการวิวัฒนาการซอฟต์แวร์ ซึ่งกิจกรรมเหล่านี้จะไปอยู่ในแบบจำลองกระบวนการซอฟต์แวร์แบบต่าง ๆ หน้าที่ของวิศวกรซอฟต์แวร์คือเลือกหรือประยุกต์ใช้แบบจำลองกระบวนการซอฟต์แวร์ให้เหมาะสมกับ ปัญหา รูปแบบซอฟต์แวร์ ระบบ กระบวนการทำงาน รูปแบบธุรกิจ นโยบาย และข้อกำหนดอื่น ๆ ของโครงการ

บทที่ 3 การพัฒนาซอฟต์แวร์แบบเอจายล์ Agile software development

วัตถุประสงค์

- *เข้าใจการพัฒนาซอฟต์แวร์แบบเอจายล์*

ซอฟต์แวร์ในปัจจุบันมีขนาดและความซับซ้อนมากขึ้นและที่สำคัญคือการเปลี่ยนแปลงที่อาจเกิดขึ้นได้ตลอดเวลา ซึ่งการเปลี่ยนแปลงส่วนใหญ่จะกระทบกับลูกค้าถ้าไม่สามารถจัดการกับความเปลี่ยนแปลงได้ โดยเฉพาะการเปลี่ยนแปลงที่กระทบโดยตรงกับธุรกิจ การปรับตัวเปลี่ยนแปลงช้าอาจสร้างความเสียหายอย่างมาก ดังนั้น การพัฒนาซอฟต์แวร์ที่สามารถรองรับการเปลี่ยนแปลงได้อย่างรวดเร็วและคุ้มค่าจึงเป็นที่นิยมนำมาประยุกต์ใช้ในการพัฒนาซอฟต์แวร์

การพัฒนาซอฟต์แวร์แบบเอจายล์ เป็นการพัฒนาซอฟต์แวร์ที่เน้นความคล่องตัวและความรวดเร็วในการพัฒนาพร้อมทั้งสามารถรับมือกับความเปลี่ยนแปลงได้อย่างดี กลุ่มผู้คิดค้นการพัฒนาซอฟต์แวร์แบบเอจายล์เลือกใช้ชื่อ เอจายล์เป็นคำหลักเพื่อให้สื่อถึงหลักความคิดในการพัฒนาซอฟต์แวร์โดยการปรับตัวและการตอบสนองต่อความเปลี่ยนแปลง ในบทนี้จะอธิบายการพัฒนาซอฟต์แวร์แบบเอจายล์

ภาพรวมการพัฒนาซอฟต์แวร์แบบเอจายล์

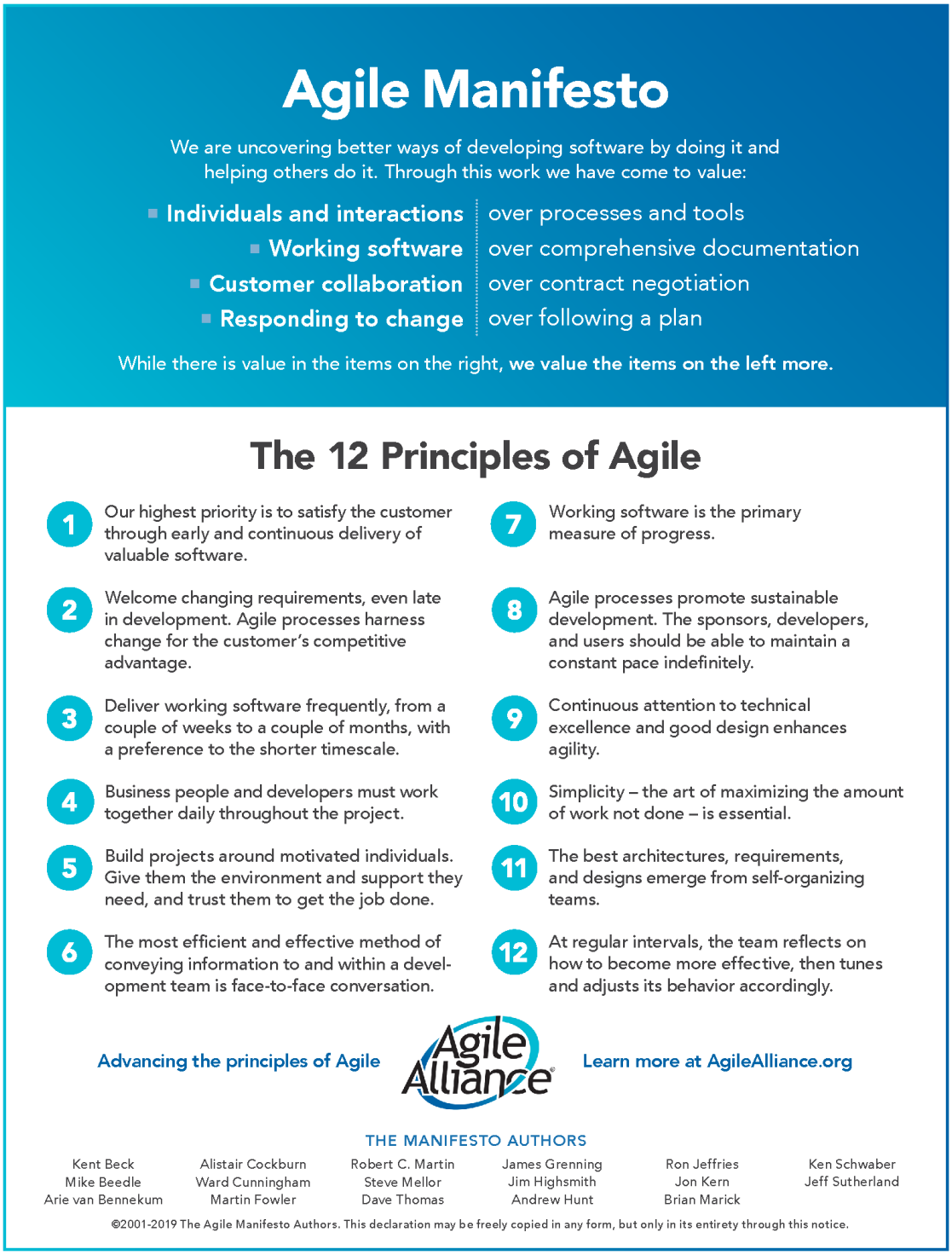
การพัฒนาซอฟต์แวร์แบบเอจายล์นั้น เพิ่งเริ่มมีมาไม่กี่ทศวรรษที่ผ่านมา โดยมีการนำไปใช้อย่างแพร่หลายและได้รับการยอมรับอย่างกว้างขวางและมีการปรับปรุงขัดเกลาเอจายล์เป็นระยะ โดยในปัจจุบันองค์กร Agile Alliance [5] ที่ก่อตั้งขึ้นโดยไม่แสวงหาผลกำไรทำหน้าที่หลักในการขับเคลื่อนเอจายล์ ตามแถลงการณ์แห่งเอจายล์ (agile manifesto) [6] ซึ่งเราสามารถเข้าไปศึกษาค้นหาข้อมูลรวมทั้งรับข่าวสารเกี่ยวกับเอจายล์ได้จากเว็บขององค์กร https://www.agilealliance.org/

เอจายล์คืออะไร

เอจายล์คือความสามารถในการสร้างและสนองต่อการเปลี่ยนแปลง ในปี 2001 ณ Snowbird รีสอร์ท สหรัฐอเมริกา กลุ่มนักพัฒนาซอฟต์แวร์ 17 ท่านมาประชุมและระดมสมองกันเรื่อง การพัฒนาซอฟต์แวร์แบบเบา (lightweight software development) จนนำไปสู่ข้อสรุปเป็นแนวทางการพัฒนาแบบเอจายล์ โดยกลุ่มผู้ก่อตั้งแถลงการณ์แห่งเอจายล์ ได้ออกแถลงการณ์แห่งเอจายล์และหลักการ 12 ข้อในการพัฒนาซอฟต์แวร์แบบเอจายล์ (12 principles of Agile) โดยความหมายของเอจายล์นั้นสามารถตีความได้กว้างๆเป็นหลักแนวคิดและหลักปฏิบัติ ในการพัฒนาซอฟต์แวร์หรือเรียกได้ว่าเป็นการปรับเปลี่ยนกระบวนทัศน์ (paradigm shift) จากการพัฒนารูปแบบเดิมเข้าสู่รูปแบบใหม่ที่คล่องตัวรวดเร็วและสามารถตอบสนองต่อความเปลี่ยนแปลงได้ดี ถึงแม้จะอยู่ภายใต้สภาพแวดล้อมที่ไม่แน่นอนหรือปั่นป่วน

การพัฒนาซอฟต์แวร์แบบเอจายล์เป็นมากกว่ากรอบการทำงาน (frame works) เช่น สกรัม (scrum) หรือการโปรแกรมสุดขีด (extreme programming) เป็นมากกว่าแนวปฏิบัติ (practices) เช่น การเขียนโปรแกรมคู่ (pair programming) การ

พัฒนาที่ขับเคลื่อนด้วยการทดสอบ (test-driven development) หรือการยืนประชุม (stand up meeting) แต่การพัฒนาซอฟต์แวร์แบบเอจายล์เป็นคำศัพท์เฉพาะที่ครอบคลุมชุดของกรอบงานและแนวปฏิบัติที่ตั้งอยู่บนคุณค่า (value) และหลักการดังที่แสดงไว้ในแถลงการณ์แห่งเอจายล์และหลัก 12 ประการแห่งเอจายล์ [7]



Agile Manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

While there is value in the items on the right, **we value the items on the left more.**

The 12 Principles of Agile

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Advancing the principles of Agile — Agile Alliance — Learn more at AgileAlliance.org

THE MANIFESTO AUTHORS

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, Robert C. Martin, Steve Mellor, Dave Thomas, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Ken Schwaber, Jeff Sutherland

รูปที่ 3.1 แถลงการณ์แห่งเอจายล์และหลัก 12 ประการแห่งเอจายล์ จาก [8]

เราค้นพบวิธีที่ดีกว่าในการพัฒนาซอฟต์แวร์
จากการลงมือทำจริงและช่วยเหลือผู้อื่น
นั่นคือ เราให้ความสำคัญกับ:

คนและการมีปฏิสัมพันธ์กัน มากกว่าการทำตามขั้นตอนและเครื่องมือ
ซอฟต์แวร์ที่นำไปใช้งานได้จริง มากกว่าเอกสารที่ครบถ้วนสมบูรณ์
ร่วมมือทำงานกับลูกค้า มากกว่าการต่อรองให้เป็นไปตามสัญญา
การตอบรับกับการเปลี่ยนแปลง มากกว่าการทำตามแผนที่วางไว้

ทั้งนี้ แม้เราจะเห็นความสำคัญในสิ่งที่กล่าวไว้ทางด้านขวา
แต่เราให้ความสำคัญกับสิ่งที่กล่าวไว้ทางด้านซ้ายมากกว่า [6]

จากแถลงการณ์แห่งเอจายล์ฉบับแปลเป็นภาษาไทยข้างต้นนั้นยังถูกแปลอีกกว่า 60 ภาษา โดยแถลงการณ์นี้ใช้คำที่กระชับและเข้าใจได้ง่ายแต่ลึกซึ้งและสะท้อนให้เห็นถึงการปรับเปลี่ยนกระบวนทัศน์จากรูปแบบการพัฒนาซอฟต์แวร์แบบเดิมอย่างชัดเจน

คนและการมีปฏิสัมพันธ์กัน มากกว่าการทำตามขั้นตอนและเครื่องมือ (individuals and interactions over processes and tools) ความสัมพันธ์ระหว่างบุคคลการทำงานเป็นทีมสำคัญมากในการทำงาน ความสัมพันธ์ที่ดีนำไปสู่การสื่อสารที่ดี และย่อมนำไปสู่ความสามัคคี ความไว้เนื้อเชื่อใจกันในทีม ถ้าทีมมีสิ่งเหล่านี้แล้วการทำงานร่วมกันเพื่อผ่านอุปสรรคไปก็คงไม่เกินความสามารถ การใช้การสื่อสารแบบเห็นหน้ากันในสถาณที่เดียวกันมีประสิทธิภาพมากเพราะสามารถสร้างความเข้าใจในระหว่างการสื่อสารได้ดี และควรจัดที่ทำงานให้ทุกคนในทีมทำงานด้วยกันอยู่ในห้องเดียวกันเพื่อสร้างความสัมพันธ์ที่ดีในทีมและการสื่อสารที่มีประสิทธิภาพอันเป็นกำลังขับเคลื่อนหลักในการทำงานเป็นทีม ในส่วนของการทำตามขั้นตอนและเครื่องมือนั้นก็ยังต้องใช้ในการทำงานเพียงแต่ให้น้ำหนักความสำคัญไปที่ทีมก่อนอันดับแรก ในส่วนการจะใช้กระบวนการหรือเครื่องมือใดมาใช้ให้พิจารณาเลือกที่เหมาะสมและสนับสนุนการทำงานของทีม หรืออีกนัยคือเลือกและปรับกระบวนการและเครื่องมือให้ตรงกับทีม

ซอฟต์แวร์ที่นำไปใช้งานได้จริง มากกว่าเอกสารที่ครบถ้วนสมบูรณ์ (working software over comprehensive documentation) มั่งเน้นสร้างซอฟต์แวร์ที่ใช้งานได้จริง มีคุณภาพ ข้อบกพร่องต่ำ ผ่านการทดสอบอย่างดี และลูกค้าสามารถใช้งานได้จริงอย่างมีประสิทธิภาพ รวมทั้งพัฒนาเสร็จได้ทันในเวลา ในส่วนของเอกสารที่ครบถ้วนสมบูรณ์นั้น ถ้ากล่าวถึงคู่มือ เอกสารการติดตั้ง เอกสารความช่วยเหลือ นั้นนับรวมเป็นส่วนหนึ่งของซอฟต์แวร์อยู่แล้ว ในมุมมองของเอจายล์ เอกสารที่ครบถ้วนสมบูรณ์หมายถึงเอกสารต่างๆ ที่ไม่ได้มีส่วนช่วยให้การใช้งานมีประสิทธิภาพขึ้นหรือเอกสารที่ไม่ช่วยให้การทำงานหรือการสื่อสารภายในทีมดีขึ้น เป็นเอกสารที่เกินความจำเป็น ถ่วงเวลาในการพัฒนา เช่นการทำเอกสารการออกแบบซอฟต์แวร์แบบละเอียดสมบูรณ์ในระหว่างพัฒนาอาจจะไม่จำเป็นเพราะซอฟต์แวร์มีการเปลี่ยนแปลงทั้งการออกแบบและการสร้าง

ร่วมมือทำงานกับลูกค้า มากกว่าการต่อรองให้เป็นไปตามสัญญา (customer collaboration over contract Negotiation) การมีส่วนร่วมและความร่วมมือทำงานกับลูกค้านับว่าสำคัญมากในเอจายล์ โดยลูกค้าต้องร่วมมือกันเพื่อสกัดความต้องการที่แท้จริงออกมา โดยพยายามหาว่าลูกค้าต้องการอะไร และลูกค้าถือว่าเป็นส่วนหนึ่งในทีมเพื่อร่วมพัฒนาอย่างใกล้ชิด ลูกค้าเห็นได้ลองใช้ทดสอบซอฟต์แวร์ได้ทันทีให้ผลป้อนกลับได้ทันทีและพิจารณาหรือตัดสินใจว่าชิ้นงานที่ได้ตรงกับความต้องการหรือไม่ ช่วยเพิ่มความพึงพอใจของลูกค้าและลดความเสี่ยงโดยรวมของโครงการ ในส่วนของการต่อรองให้เป็นตามสัญญา โดยส่วนใหญ่จะตีความกันตามข้อตกลง ผลประโยชน์ บทลงโทษ การปรับ กรอบจำกัดต่างๆ ตามสัญญาที่ตกลงกันไว้ โดยไม่ได้คำนึงถึงการทำงานและความสัมพันธ์ระหว่างทีมพัฒนากับลูกค้าเพื่อจะร่วมมือกับให้การดำเนินงานไปสู่ผลสำเร็จ สัญญาจึงกลายเป็นเครื่องมือในการปกป้องและโจมตีกันระหว่างคู่สัญญามากกว่า ซึ่งในเอจายล์จะเน้นความร่วมมือระหว่างทั้งสองฝ่ายเพื่อทำงานร่วมกันอย่างมีประสิทธิภาพมากกว่าการยึดสัญญาเป็นที่ตั้งและร่วมกันรับผิดชอบต่อผลลัพธ์ร่วมกัน

การตอบรับกับการเปลี่ยนแปลง มากกว่าการทำตามแผนที่วางไว้ (responding to change over following a plan) เอจายล์เน้นการตอบสนองความเปลี่ยนแปลง เพื่อให้เป็นไปตามความต้องการของลูกค้าและส่งมอบซอฟต์แวร์ที่มีคุณภาพ และประโยชน์ของลูกค้าเป็นสำคัญเช่นปรับเปลี่ยนซอฟต์แวร์เพื่อลดต้นทุน ประหยัดเวลา เพิ่มความสามารถในการแข่งขัน ปรับตามธุรกิจ กลไกตลาด ลดความเสี่ยง ทั้งหมดเพื่อประโยชน์และความพึงพอใจของลูกค้าเป็นสำคัญ ในขณะที่การวางแผนและทำตามแผนที่วางไว้ก็ยังคงจำเป็นเพียงแต่นำมาใช้เพื่อสนับสนุนการทำงานในทีมลดความเสี่ยงในการพัฒนาและช่วยให้ทีมทำงานอย่างมีประสิทธิภาพ เพียงแต่ให้ความสำคัญเรื่องความยืดหยุ่นในการแก้ไขเปลี่ยนแปลงเป็นหลักมากกว่าการทำตามแผนที่วางไว้

หลัก 12 ประการแห่งเอจายล์

หลัก 12 ประการแห่งเอจายล์ถูกใช้เป็นแนวทางในการประยุกต์ใช้เอจายล์เพื่อการพัฒนาซอฟต์แวร์เรื่อยมา โดยหลัก 12ประการมีรายละเอียดแปลเป็นภาษาไทยจาก [9] ได้ดังนี้

1. ความสำคัญสูงสุดของพวกเราคือความพึงพอใจของลูกค้าที่มีต่อการส่งมอบซอฟต์แวร์ที่มีคุณค่าต่อลูกค้า ตั้งแต่ต้นอย่างต่อเนื่อง
2. ยอมรับการเปลี่ยนแปลงความต้องการของลูกค้าแม้ในช่วงท้ายของการพัฒนาเพราะเอจายล์สามารถแปรเอาความเปลี่ยนแปลง มาเป็นความได้เปรียบในการแข่งขันของลูกค้า
3. ส่งมอบซอฟต์แวร์ที่ใช้งานได้จริงอย่างสม่ำเสมออาจเป็นทุกสองถึงสามสัปดาห์หรือทุกสองถึงสามเดือนโดยควรทำให้ระยะเวลาระหว่างการส่งมอบนั้นสั้นที่สุดเท่าที่เป็นไปได้
4. ตัวแทนจากฝ่ายธุรกิจและนักพัฒนาจะต้องทำงานร่วมกันเป็นประจำทุกวันตลอดโครงการ
5. ทำให้แน่ใจว่าสมาชิกในโครงการเข้าใจและมีจุดมุ่งหมายของโครงการร่วมกัน สร้างสภาวะแวดล้อมและให้การสนับสนุนในสิ่งที่พวกเขาต้องการและให้ความไว้วางใจแก่พวกเขาในการที่จะทำงานให้บรรลุเป้าหมายนั้น
6. วิธีที่มีประสิทธิภาพและประสิทธิผลสูงสุดในการถ่ายทอดข้อมูลต่าง ๆ ไปสู่ทีมพัฒนาและภายในทีมพัฒนาเองคือการพูดคุยแบบซึ่งหน้า
7. ซอฟต์แวร์ที่ใช้งานได้จริงเป็นตัวหลักในการวัดความก้าวหน้าของโครงการ

8. กระบวนการเอจายล์สนับสนุนให้เกิดการพัฒนาแบบยั่งยืน กล่าวคือผู้สนับสนุนนักพัฒนา และตัวแทนผู้ใช้ควรจะสามารถรักษาอัตราเร็วในการทำงานร่วมกันให้คงที่ได้ตลอดไป
9. การใส่ใจในความเป็นเลิศทางเทคนิคและงานออกแบบที่ดีอย่างต่อเนื่องจะช่วยเพิ่มความเป็นเอจายล์
10. ความเรียบง่ายหรือศิลปะในการทำงานอย่างพอเพียง นั้นสำคัญยิ่ง
11. สถาปัตยกรรมซอฟต์แวร์ ความต้องการของลูกค้า และ งานออกแบบที่ดีที่สุด เกิดจากทีมที่บริหารจัดการตัวเองได้
12. ทุกช่วงเวลาหนึ่งเป็นประจำ ทีมจะต้องย้อนกลับไปตรองดูสิ่งที่ผ่านมาเพื่อหาทางที่จะพัฒนาความมีประสิทธิผลของทีม แล้วนำสิ่งเหล่านั้นมาปรับปรุงและเปลี่ยนแปลงพฤติกรรมของทีม

จะเห็นได้ว่าหลัก 12 ประการนี้ถึงจะผ่านมากว่าสองทศวรรษแล้วก็ตาม ก็ยังคงถูกนำไปเป็นแนวทางในการประยุกต์ใช้เพื่อการพัฒนาซอฟต์แวร์เรื่อยมา โดยจากแนวคิดแถลงการณ์แห่งเอจายล์และหลัก 12 ประการ นั้นถูกนำไปใช้ในการสร้างกรอบการทำงาน กรรมวิธี กระบวนการ เทคนิค ในการพัฒนาซอฟต์แวร์แบบเอจายล์ในรูปแบบที่บริหารจัดการได้และเป็นระบบระเบียบ คล่องตัวรวดเร็วและตอบสนองความเปลี่ยนแปลงได้ดี

การโปรแกรมสุดขีด (extreme programming: XP)

การโปรแกรมสุดขีดหรือเรียกชื่อย่อว่าเอ็กพี (XP) เป็นกรรมวิธีในการพัฒนาซอฟต์แวร์แบบเอจายล์ ที่เป็นที่รู้จักกันเป็นอย่างดีด้วยจุดเด่นที่เอ็กพีมีหลักการแบบสุดขีดหลายๆด้านตามชื่อนั่นเอง โดยกรรมวิธีเอ็กพีนี้ถูกคิดค้นโดย Kent Beck ในปี ค.ศ. 1999 ในระหว่างที่เขาทำโครงการที่บริษัท Chrysler โดยมีนำแนวคิดการปฏิบัติสุดขีดมาประยุกต์ใช้กับการพัฒนาซอฟต์แวร์ เพื่อมุ่งสู่ความสำเร็จด้านคุณภาพของซอฟต์แวร์ภายในกรอบระยะเวลาที่หำหนด โดยกรรมวิธีนี้เหมาะสำหรับการพัฒนาซอฟต์แวร์ขนาดกลางถึงขนาดเล็กและทีมพัฒนาที่มีจำนวนคนไม่มาก

เอ็กพีมุ่งเน้นการมีส่วนร่วมของลูกค้าโดยการผนวกลูกค้าเข้ามาเป็นส่วนหนึ่งของทีมพัฒนา เพื่อให้สามารถสกัดความต้องการที่แท้จริงและสามารถได้รับผลตอบรับทันทีจากลูกค้าจากการทดสอบซอฟต์แวร์ เน้นไปที่การพัฒนาซอฟต์แวร์เป็นหลัก ไม่เน้นการทำเอกสารเพื่อลดขั้นตอนต่าง ๆ ให้กระชับแม้กระทั้งการออกแบบก็เน้นการออกแบบเฉพาะจุดที่จำเป็นไม่ต้องออกแบบเผื่อ ทำให้การโปรแกรมสุดขีดมีความคล่องตัวสูงและเน้นที่คุณภาพของซอฟต์แวร์เป็นหลัก เพียงแต่ว่าทีมพัฒนาต้องมีศักยภาพสูงด้วยเช่นกัน

แนวการปฏิบัติโปรแกรมสุดขีด

- **The Planning Game** การวางแผนร่วมกันระหว่างลูกค้าและทีมพัฒนาเพื่อให้ได้แผนงานที่ดีที่สุดตั้งอยู่บนความเป็นจริงมากที่สุดโดยที่ลูกค้าทำหน้าที่เขียนอธิบายคุณลักษณะของซอฟต์แวร์ที่ตนต้องการในรูปแบบเรื่องราวของผู้ใช้ (user stories) ใส่ไว้ใน แผ่นเรื่องราว (story card) หลังจากนั้นทีมพัฒนาจะประมาณการขนาดของแผ่นเรื่องราวว่าต้องใช้ความพยายาม (effort) และเวลาในการพัฒนารวมทั้งการทดสอบเป็นเท่าใดเพื่อวางแผนการดำเนินการพัฒนา โดยที่มีลูกค้าเป็นคนตัดสินใจในการจัดลำดับความสำคัญก่อนหลังเพื่อเลือกแผ่นเรื่องราวใดๆมาพัฒนาตามลำดับ โดยการพัฒนาจะดำเนินไปแบบวนรอบ เพื่อส่งมอบชิ้นงานได้ภายในระยะเวลาที่กำหนดซึ่งจะอยู่ในช่วง 1-4 สัปดาห์

- **Small Releases** ลูกค้าเลือกงานที่ต้องการให้พัฒนา โดยมักเลือกจากลักษณะที่สำคัญสุดก่อนเพื่อนำเข้ากระบวนการวนรอบ เพื่อให้ได้ซอฟต์แวร์ส่วนที่จำเป็นและสำคัญออกมาใช้ก่อนแล้วค่อยขยายต่อไปในรอบถัด ๆ ไป

- **Metaphor** แต่ละโครงการต้องมีการกำนดนิยามและความหมายต่าง ๆ ให้ตรงกันทั้งทีม เพื่อให้ทุกคนเห็นภาพตรงและเข้าใจสิ่งที่กำลังจะพัฒนาตรงกัน สื่อสารตรงกัน ลดความกำกวมต่าง ๆ

- **Simple Design** ออกแบบเน้นความง่ายเสมอ ลดความซับซ้อนปรับให้การออกแบบเรียบง่ายซึ่งจำนำมาสู่การออกแบบการทดสอบที่ง่าย การเขียนโปรแกรมที่ง่าย และ การปรับเปลี่ยนที่ง่ายด้วยเช่นกัน ไม่ออกแบบล่วงหน้า (upfront design) เพราะการเปลี่ยนแปลงเกิดขึ้นได้เสมอนั้นคือปรัชญาในการออกแบบของเอ็กพี

- **Testing** การทดสอบของเอ็กพีเน้นการทดสอบแบบต่อเนื่อง ทุกครั้งที่มีการเพิ่มส่วนที่พัฒนาใหม่เข้ามาต้องถูกทดสอบแยกและทดสอบรวมกับระบบเดิมเสมอ โดยเน้นการออกแบบชุดทดสอบการก่อนการพัฒนา (test first development) เพื่อให้มั้นใจว่าส่วนของซอฟต์แวร์มีคุณภาพและความหน้าเชื่อถือสูงโดยปกติ การทดสอบในเอ็กพีจะทำการทดสอบสองระดับคือ การทดสอบหน่วย (unit test) เป็นการทดสอบในระดับย่อยเพื่อทำการทดสอบรหัสต้นฉบับว่าทำงานได้ตามที่ออกแบบหรือไม่และสามารถผ่านการทดสอบในระดับย่อยนี้ได้หรือไม่ โดยที่เมื่อทุกหน่อยผ่านการทดสอบแล้วจะทำการทดสอบรวมเพื่อนำไปสู่การทดสอบเพื่อยอมรับ (acceptance test) จากลูกค้า โดยทำการทดสอบเทียบกับคุณลักษณะและความต้องการว่าถูกต้องและตรงกับสิ่งที่ลูกค้าต้องการและคาดหวังไว้หรือไม่

- **Refactoring** การปรับรหัสต้นฉบับให้มีประสิทธิภาพและเป็นไปตามมาตรฐาน หลักๆทำให้รหัสต้นฉบับเรียบง่าย มีประสิทธิภาพ และรองรับการเปลี่ยนแปลง เช่นการกำจักส่วนที่เป็นรหัสแข็ง (hard code) ให้เปลี่ยนแปลงได้ง่ายด้วยตัวแปรหรือการปรับรหัสด้นฉบับที่ยาวและซับซ้อนให้อยู่ในรูปของฟังก์ชันอย่างง่ายหลายฟังก์ชันทำงานร่วมกัน เป็นต้น

- **Pair Programming** การกำหนดให้การเขียนโปรแกรมที่ต้องใช้นักพัฒนาสองคนในการเขียนเสมอ โดยทำการเขียนรหัสต้นฉบับนั้นต้องมีนักพัฒนาสองคนเป็นผู้ดำเนินการพร้อมกันเพื่อช่วยกันเขียนรหัสต้นฉบับ ช่วยกันตรวจสอบข้อผิดพลาดในเครื่องคอมพิวเตอร์เครื่องเดียว เพื่อเพิ่มประสิทธิภาพและลดความผิดพลาดในการเขียนรหัสต้นฉบับ

- **Collective Ownership** ไม่มีใครคนใดคนหนึ่งเป็นเจ้าของรหัสต้นฉบับหรือส่วนของซอฟต์แวร์ ทีมพัฒนาสามารถเข้าถึงและจัดการรหัสต้นฉบับในโครงการได้ ทีมสามารถปรับและหมุนเวียนไปพัฒนาส่วนต่าง ๆ ได้เพื่อเรียนรู้ส่วนต่างๆทั้งโครงการ ทั้งนี้เพื่อหลีกเลี่ยงการเกิดบุคคลกุญแจ (key man) และลดความเสี่ยงเมื่อสมาชิกทีมพัฒนาลาออก

- **Continuous Integration** การพัฒนาซอฟต์แวร์จะถูกบันทึกไว้ในระบบจัดการรหัสต้นฉบับ และทุกครั้งที่มีการเพิ่มส่วนของซอฟต์แวร์เข้ามาใหม่จะต้องถูกทดสอบเสมอ เพื่อเพิ่มความมั่นใจว่าซอฟต์แวร์ยังคงทำงานได้อย่างถูกต้องทั้งก่อนและหลังการรวมซอฟต์แวร์

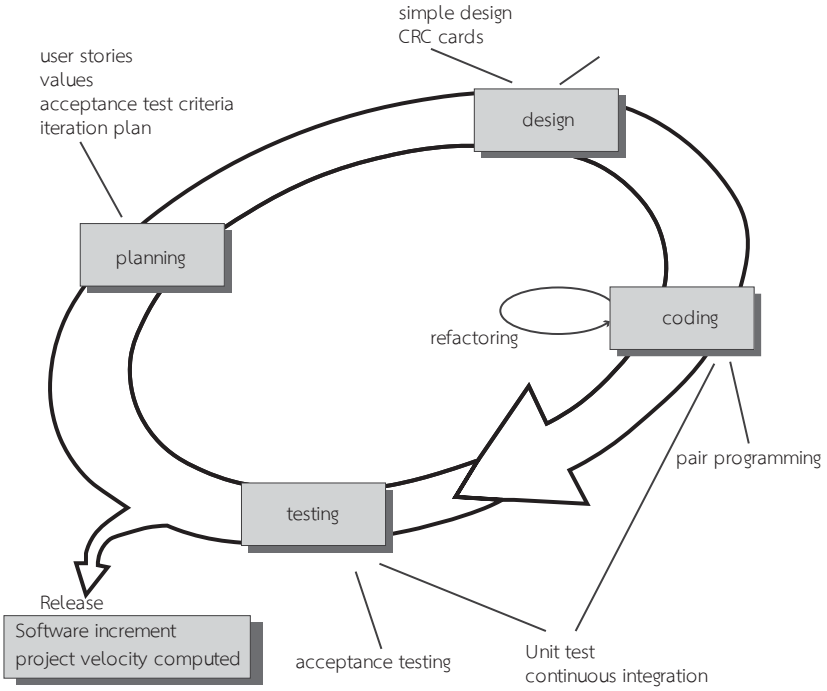
- **40-hour week** ทีมพัฒนาจะถูกจำกัดเวลาการพัฒนาไม่ให้มากเกินไปและไม่สนับสนุนการทำงานล่วงเวลา เพราะการทำงานมากเกินไปไม่ได้ส่งผลดีกับประสิทธิภาพการทำงานและผลลัพธ์ที่ได้ออกมาไม่ได้ดีกว่าการทำงานตามกรอบเวลาปกติ ดังนั้นสี่สิบชั่วโมงต่อสัปดาห์เพียงพอแล้วในการพัฒนา

- **On-site Customer** ลูกค้าต้องส่งตัวแทนมาร่วมเป็นส่วนหนึ่งในทีมพัฒนาแบบเต็มเวลา เพื่อให้ข้อมูลต่างๆเกี่ยวกับการพัฒนาทั้งการให้ความต้องการ การอธิบายการทำงาน ขยายตวามข้อสงสัยต่าง ๆ ทดสอบซอฟต์แวร์ และตัดสินใจร่วมกับทีมพัฒนา

- **Coding Standard** การเขียนรหัสต้นฉบับต้องเป็นไปตามมาตรฐานและรูปแบบที่ทีมพัฒนากำหนด เพื่อให้ทุกคนในทีมทำงานร่วมกันได้ง่ายขึ้น รหัสต้นฉบับควรถูกพัฒนาในรูปแบบที่เหมาะสมเป็นไปในแนวทางเดียวกัน อ่านง่าย แก้ไขง่าย และ สามารถเรียนรู้ได้ง่าย

จะเห็นว่าแนวการปฏิบัติทั้ง 12 ข้อที่กล่าวมานั้นสอดคล้องกับคำแถลงการณ์แห่งเอจายล์ โดยที่เน้นการพัฒนาแบบเพิ่มเติมค่อยๆวนรอบเพื่อพัฒนาส่วนของซอฟต์แวร์ขนาดเล็กแล้วจึงเพิ่มตามความต้องการขึ้นเรื่อง ๆ จนได้ซอฟต์แวร์ที่สมบูรณ์ โดยมีลูกค้ามาเป็นส่วนหนึ่งในทีมพัฒนาเพื่อทำหน้าที่นิยามความต้องการ ช่วยเลือกลำดับความสำคัญของงานที่จะต้องพัฒนาในแต่ละรอบ จนถึงการทดสอบเพื่อยอมรับในแต่ละรอบ ทีมเอ็กพีจะเน้นที่ทรัพยากรคนเป็นสำคัญ มุ่งเน้นการเสริมสร้างทักษะในการพัฒนาผ่านการเขียนโปรแกรมแบบคู่ การเป็นเจ้าของร่วมในรหัสต้นฉบับ และ การพัฒนาที่จำกัดที่ 40 ชั่วโมงต่อสัปดาห์ สำหรับการจัดการการเปลี่ยนแปลงนั้นเอ็กพี น้อมรับการเปลี่ยนแปลงได้อย่างดีผ่านรอบการพัฒนาที่สั้นมาก การออกแบบที่ง่าย การทำซอฟต์แวร์รีแฟคทอริง เพื่อรองรับการปรับเปลี่ยนได้ตลอดเวลา

ถึงแม้ว่ากรรมวิธีเอ็กพี จะมีประสิทธิภาพสูงสำหรับซอฟต์แวร์ขนาดกลางถึงขนาดเล็กก็ตาม ยังมีปัญหาสำคัญของเอ็กพี คือเรื่องศักยภาพของทีมพัฒนาต้องสูงเพื่อรองรับการโปรแกรมแบบสุดขีดและการบำรุงรักษาซอฟต์แวร์ในระยะยาว อันเนื่องมาจากเอ็กพีไม่เน้นการทำเอกสารทำให้ทีมที่มาบำรุงรักษาต่อที่ไม่ใช่ทีมเดิมหรือไม่คุ้นเคยกับกรรมวิธีเอ็กพีรับช่วงการดูแลซอฟต์แวร์ต่อได้ลำบาก แล้วทำไมทีมพัฒนาเอ็กพีส่วนใหญ่ไม่ทำหน้าดูแลต่อ ก็เนื่องจากธรรมชาติของทีมพัฒนาแบบเอ็กพีนั้นมีศักยภาพสูงต้องการความท้าทายในการทำโครงการ เมื่อพัฒนาโครงการสำเร็จก็มักจะย้ายไปทำโครงการอื่นต่อไป



รูปที่ 3.2 กระบวนการโปรแกรมสุดขีด [2]

กระบวนการโปรแกรมสุดขีดจะแบ่งขั้นตอนการทำงานหลักเป็น สี่ ขั้นตอนในหนึ่งรอบการพัฒนาซึ่งในหนึ่งรอบจะกินเวลาไม่เกิน สี่สัปดาห์โดยมีผลลัพธ์เป็นซอฟต์แวร์ที่สามารถใช้งานได้ตามความต้องการที่ระบุในตอนต้นของรอบการพัฒนาดังแสดงในรูปที่ 3.2

1. **การวางแผน** เป็นขั้นตอนในการเก็บความต้องการของลูกค้ามาวิเคราะห์และวางแผนการทำงานระยะสั้นสำหรับรอบการทำงานที่จะเกิดขึ้น โดยให้ผู้ใช้เขียนเรื่องราวผู้ใช้ (user stories) เพื่ออธิบายความต้องการในมุมมองของผู้ใช้และอธิบายรูปแบบการทำงานของซอฟต์แวร์ที่คาดหวังว่าจะได้เจอรวมทั้งสถานการณ์ที่จะเกิดขึ้นในรูปแบบเรื่องราว จากนั้นนักวิเคราะห์ระบบจะทำการวิเคราะห์จากเรื่องราวผู้ใช้ เพื่อนำมาแบ่งงานสร้างตารางงานสำหรับกำหนดงาน โดยจะให้ผู้ใช้เป็นผู้เลือกเรื่องราวที่มีความสำคัญลำดับต้น ๆ มาวางแผนการพัฒนาก่อนตามลำดับ เพื่อจะให้ได้ส่วนของซอฟต์แวร์ที่สำคัญนั้นมาใช้งานก่อน ซึ่งกระบวนการวางแผนนี้จะทำการวางแผนแบบวนรอบในแต่ละรอบการพัฒนา ในระยะเวลาไม่นานซึ่งในแต่ละรอบนั้นจะมีการวางแผนย่อยสำหรับรอบการพัฒนาสั้น ๆ โดยมีตัวชี้วัดหรือประเมินผลในแต่ละรอบคือข้อกำหนดการทดสอบการยอมรับไว้ในแผนด้วย ซึ่งทีมพัฒนาต้องดำเนินการเพื่อให้ผ่านการทดสอบการยอมรับในแต่ละรอบ ในเอ็กพียังมีการวางแผนรูปแบบเพิ่มเติมในระหว่างการพัฒนาเช่นการทำการประชุมทุกวันเพื่อติดตามงานรายงานปัญหาที่พบและแลกเปลี่ยนข้อมูลการพัฒนา เพื่อให้การพัฒนาดำเนินไปตามแผน รวมทั้งยังสามารถปรับเปลี่ยนแผนได้อย่างรวดเร็ว หรือมองว่าเป็นการวางแผนระยะสั้นรายวันก็ได้ ถ้าจำเป็นต้องปรับต้องเปลี่ยนทีมก็ต้องกล้าที่จะทำ
2. **การออกแบบ** นักพัฒนาพยายามออกแบบระบบเพื่อตอบสนองความต้องการของลูกค้าโดยคำนึงถึงความคุ้มค่าในการพัฒนา ภายใต้ปรัชญาการออกแบบที่เรียบง่ายที่สุดให้สามารถใช้งานได้ ไม่ต้องออกแบบเผื่ออนาคตเพราะอาจไม่ได้ใช้งานส่วนที่ออกแบบเผื่อไว้และการเปลี่ยนแปลงสามารถเกิดขึ้นได้เสมอ เน้นการออกแบบไปที่รอบการพัฒนาปัจจุบันสร้างวิธีแก้ไขปัญหาให้ลูกค้าหรือ (spike solution) เพื่อลดความเสี่ยงและได้ทดลองต้นแบบก่อนว่าสามารถสร้างได้จริงโดยการออกแบบจะใช้ class, responsibilities and collaborators card: CRC card ในการออกแบบและอธิบายระบบได้เป็นอย่างดี รวมทั้งรายละเอียดที่ช่วยให้การเขียนโปรแกรมทำได้สะดวกและในกระบวนการออกแบบก็สามารถวนรอบเพื่อปรับปรุงการออกแบบถ้าจำเป็น
3. **การเขียนโปรแกรม** เป็นอีกหนึ่งขั้นตอนที่เอ็กพีเน้นเพราะการเขียนโปรแกรมนั้นนำไปสู่การได้มาซึ่งซอฟต์แวร์ โดยเน้นที่ตัวงานที่ได้คุณภาพและมีความน่าเชื่อถือด้วยการเขียนโปรแกรมแบบสุดขีด เริ่มจากการใช้การเขียนรหัสต้นฉบับให้ได้ตามมาตรฐานของทีมเพื่อให้ทั้งทีมทำงานร่วมกันได้ง่าย การเขียนโปรแกรมคู่นับเป็นอีกหนึ่งจุดเด่นที่ต้องให้นักพัฒนาสองคนทำการเขียนโปรแกรมด้วยกันในเครื่องคอมพิวเตอร์เดียวกันเสมอ เพื่อคุณภาพของรหัสต้นฉบับที่ดีและอย่างน้อยการทดสอบด้วยนักพัฒนาอย่างน้อยสองคนเสมอ แน่นอนว่าการโปรแกรมสุดขีดต้องสมชื่อ ผู้ใช้ต้องร่วมทำงานแบบเต็มเวลาเพื่อให้นักพัฒนาสอบถามความต้องการได้ตลอดเวลาและการพัฒนาเป็นแบบผลักดันด้วยการทดสอบที่ให้ทำการทดสอบทุกครั้งที่สร้างในระดับหน่วยย่อยไปจนถึงการทดสอบรวม สามารถทำได้หลายครั้งต่อวันและชุดทดสอบต้องถูกสร้างก่อนการเขียนโปรแกรมอีกด้วย การประบปรุงต้นฉบับก็ถือว่าเป็นหลักการที่ต้องยึดปฏิบัติเสมอเพื่อให้รหัสต้นฉบับ ทำงานได้เรียบง่าย มีประสิทธิภาพ มีความน่าเชื่อถือสูง

4. **การทดสอบ** กระบวนการทดสอบในเอ็กพีนั้น เรียกได้ว่าเป็นการทดสอบที่สุดขีดควบคู่ไปกับการโปรแกรมสุดขีดเช่นกัน รหัสต้นฉบับต้องถูกทดสอบและชุดทดสอบต้องถูกออกแบบก่อนการเขียนรหัสต้นฉบับเสมอ การทดสอบครอบคลุมตั้งแต่การทดสอบหน่วย (unit test) ทดสอบมอดูล และ การทดสอบรวม โดยต้องทำการทดสอบทุกครั้งที่สร้าง การทดสอบสามารถทำได้หลายๆครั้งในหนึ่งวัน เรียกได้ว่าทดสอบทุกครั้งที่ทำการแปลและสร้างซอฟต์แวร์ และเนื่องจากการทดสอบเป็นขั้นตอนสุดท้ายของรอบการพัฒนา การทดสอบการยอมรับจะดำเนินการในขั้นตอนนี้เพื่อทดสอบการยอมรับโดยลูกค้าหรือผู้ใช้ก่อนจะดำเนินการต่อในรอบถัดไป

จากสี่ขั้นตอนหลักข้างต้นเห็นได้ว่าการโปรแกรมสุดขีดสามารถใช้ได้ดีกับทีมพัฒนาที่มีศักยภาพสูงและยังต้องการเครื่องมือช่วยในการพัฒนาที่เหมาะสมเช่น เครื่องมือทดสอบอัตโนมัติ (automated testing tool) มาช่วยในการทดสอบแบบสุดขีด การเขียนโปรแกรมคู่ช่วยลดความเป็นเจ้าของรหัสต้นฉบับ เพิ่มคุณภาพ และ ความน่าเชื่อถือของรหัสต้นฉบับ ตรงกับสำนวนไทยที่ว่าสองหัวดีกว่าหัวเดียว สี่ตาย่อมหาข้อผิดพลาดได้ดีกว่าสอง สำหรับรอบการพัฒนาที่สั้นในแต่ละรอบช่วยให้ทีมสามารถส่งมอบชิ้นงานได้เร็วได้รับผลป้อนกลับเร็วลดความเสี่ยงช่วยให้ลูกค้าสามารถใช้งานซอฟต์แวร์ส่วนสำคัญได้ก่อน

ข้อดี

1. จัดการกับการเปลี่ยนแปลงความต้องการของลูกค้าได้ มีความยืดหยุ่นในการพัฒนา
2. ลดต้นทุนในการพัฒนา ถ้ามีการเปลี่ยนแปลงไม่ต้องเสียเวลารื้อแก้ใหม่เพราะการออกแบบง่ายและไม่ออกแบบล่วงหน้า
3. เอกสารน้อยมากทำให้คล่องตัว
4. เน้นการทำงานเป็นทีมและการสื่อสาร มีลูกค้าอยู่ร่วมในทีมพัฒนาทำให้สร้างความสัมพันธ์ที่ดีในการพัฒนา
5. ลูกค้าและทีมพัฒนาเข้าใจสิ่งเดียวกันผ่านการใช้เรื่องราวผู้ใช้
6. ความน่าเชื่อถือสูงเนื่องจากการพัฒนาขับเคลื่อนด้วยการทดสอบแบบสุดขีด

ข้อเสีย

1. ไม่เหมาะกับโครงการที่ซับซ้อนและมีขนาดใหญ่
2. ทีมพัฒนาต้องมีศักยภาพสูง
3. การเปลี่ยนแปลงบ่อยอาจจะกระทบความสัมพันธ์ระหว่างนักพัฒนากับลูกค้ารูปแบบการว่าจ้างตามสัญญา
4. รองรับการสนับสนุนและบำรุงรักษาซอฟต์แวร์ได้ไม่ดีนัก เพราะเอกสารส่วนใหญ่ฝังในรหัสต้นฉบับ
5. ลูกค้าร่วมในทีมพัฒนาแบบเต็มเวลาอาจไม่สามารถทำได้สมบูรณ์ในทางปฏิบัติ
6. ทำงานได้ยากหากทีมพัฒนาอยู่ต่างสถานที่

การสกรัม (scrum)

Scrum เป็นแบบจำลองที่พัฒนาโดย Jeff Sutherland และ Ken Schwaber โดยมีนิยามของ Scrum “เป็นรูปแบบของการทำงานที่สามารถระบุปัญหาที่มีความซับซ้อนและมีการเปลี่ยนแปลงอยู่ตลอดเวลาได้ ในขณะที่ยังคงส่งมอบผลิตภัณฑ์ที่มีมูลค่าสูงสุดได้อย่างมีประสิทธิภาพและสร้างสรรค์” จากคู่มือ Scrum™ [10] โดยมีลักษณะเด่นที่สำคัญ สามประการ ได้แก่

1. ไม่ซับซ้อน
2. เข้าใจง่าย
3. แต่ยากที่จะนำไปใช้ได้อย่างชำนาญ

การสกรัมไม่ได้มีกรรมวิธีหรือเทคนิคที่ตายตัวสามารถผสมผสานกับการพัฒนาอื่นๆได้ โดยการสกรัมจะเน้นไปที่การบริหารจัดการการพัฒนาอย่างมีประสิทธิภาพ โดยสามารถผนวกเข้ากับหลักวิธีการพัฒนารูปแบบต่างๆเช่น การใช้การโปรแกรมคู่หรือการพัฒนาโดยการทดสอบก่อน เป็นต้น โดยการสกรัมถูกนำไปใช้อย่างแพร่หลายนับตั้งแต่ 1990 จนในปัจจุบันนักพัฒนาจำนวนมากนำสกรัมไปปรับใช้ในทีม นอกจากนั้นยังมีการนำสกรัมไปใช้ในการพัฒนาสื่อดิจิทัลด้วยโดยเฉพาะในอุตสาหกรรมเกมและในงานด้านอื่น ๆ เช่นการพัฒนาฮาร์ดแวร์ ซอฟต์แวร์ระบบสมองกลฝังตัว ระบบอัตโนมัติ อินเทอร์เน็ตของสรรพสิ่ง และเนื่องจากเทคโนโลยีและนวัตกรรมต่าง ๆ รอบ ๆ ตัวเราเปลี่ยนแปลงเร็ว สกรัมสามารถปรับและรองรับการเปลี่ยนแปลงได้ดีรวดเร็ว จึงทำให้สกรัมถูกนำมาใช้อย่างแพร่หลาย โดยทำเพื่อวัตถุประสงค์

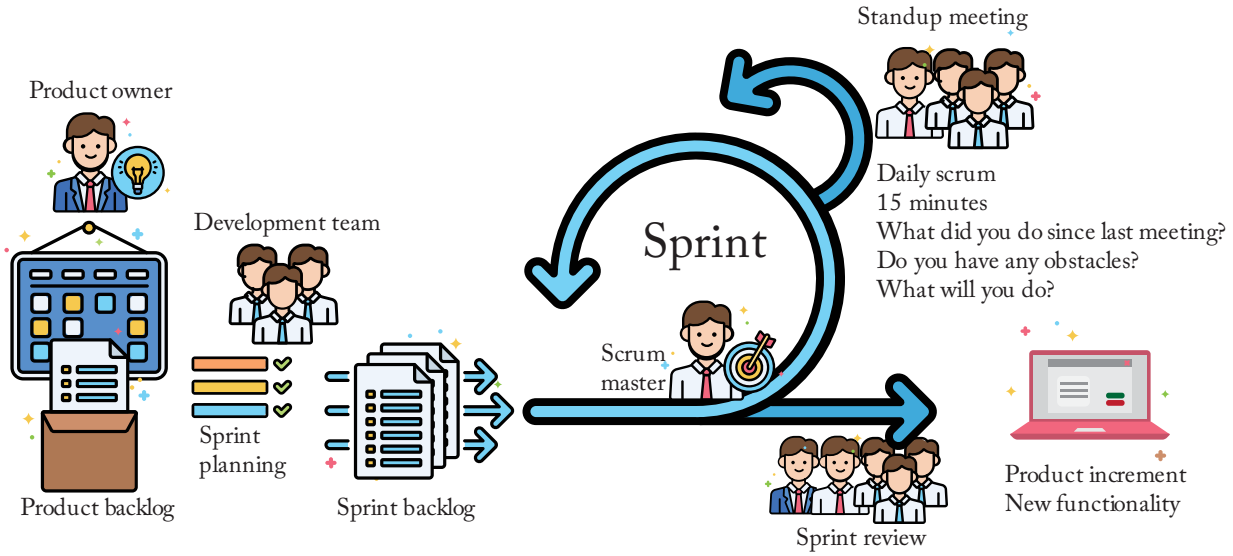
1. กำหนดทิศทางตลาด ประเมินความสามารถของผลิตภัณฑ์ และ เทคโนโลยีต่างๆ
2. พัฒนาเพื่อเพิ่มคุณค่าให้ผลิตภัณฑ์
3. ส่งมอบผลิตภัณฑ์ได้รวดเร็วและถี่มากขึ้น
4. ทำให้ผลิตภัณฑ์มีความยั่งยืนและการออกผลิตภัณฑ์ใหม่

ทฤษฎีสกรัม

ทฤษฎีสกรัมเน้นการนำความรู้จากประสบการณ์และการตัดสินใจ จากใช้งานจริงมาพัฒนาการดำเนินงาน ตามหลักการ Empiricism โดยมีหลัก สาม ประการดังนี้

1. **ความโปร่งใส (transparency)** สิ่งสำคัญคือ visible ที่ทีมต้องเห็นภาพและเข้าใจตรงกัน เช่น นิยามของคำว่างานเสร็จสมบูรณ์ ต้องตกลงให้เข้าใจตรงกัน ทำงานร่วมกันตามมาตรฐานเดียวกัน
2. **การตรวจสอบ (inspection)** ต้องนำผลลัพธ์การดำเนินกิจกรรมต่างๆ (scrum artifact) มาตรวจสอบและวัดผลเทียบกับวัตถุประสงค์สปรินท์ (sprint goal) อย่างสม่ำเสมอ ว่าเป็นไปตามที่กำหนดในแผนหรือไม่
3. **การปรับเปลี่ยน (adoption)** หากเกิดการเบี่ยงเบนจากข้อกำหนดและคาดว่าผลลัพธ์จะไม่เป็นตามที่กำหนด จึงจำเป็นต้องปรับเปลี่ยนการดำเนินงาน เพื่อให้ผลลัพธ์ออกมาใกล้เคียงมากที่สุด

สำหรับกิจกรรมหลักในการสกรัมจะประกอบไปด้วยกิจกรรมต่างๆในการพัฒนาแบบวนรอบ ที่จะมีรอบในช่วง 2-4 สัปดาห์โดยแบบจำลองสกรัมแสดงดังรูปที่ 3.3



รูปที่ 3.3 แบบจำลองสกรัม

แบบจำลองสกรัมประกอบด้วย สาม องค์ประกอบหลักคือ ทีมงาน กระบวนการ และ การสาธิตและประเมินผลงาน

ทีมสกรัม

ทีมงานในสกรัมส่วนมากจะเป็นทีมขนาดเล็กเพื่อความคล่องตัวในการทำงานโดยทีมที่ใช้สกรัมต้องมีความพร้อมในการปรับตัว การพัฒนา การเปลี่ยนแปลง และ การพึ่งพาตนเองในทีมโดยไม่ต้องพึ่งภายนอก ทีมที่ใช้สกรัมนั้นพึงมีค่านิยม ห้า ประการดังนี้

1. **ความมุ่งมั่น (commitment)** ทีมต้องมีความมุ่งมั่นที่จะประสบความสำเร็จ
2. **ความกล้า (courage)** ทีมต้องกล้าทำส่งที่ถูกต้อง กล้าเผชิญความท้าทาย ไม่หวั่นกับอุปสรรค
3. **เปิดใจ (openness)** ทีมและผู้มีส่วนเกี่ยวข้องต้อง เปิดใจยอมรับสถานะว่า กำลังทำอะไร มีปัญหาและอุปสรรคอะไร แล้วช่วยกันแก้ไขความกังวลใจไปด้วยกัน
4. **มุ่งเน้น (focus)** ทีมต้องมุ่งเน้นไปที่งานในแต่ละรอบสปรินท์และเป้าหมายของทีม เพื่อร่วมกันสร้างผลงานที่ยอดเยี่ยม
5. **เคารพ (respect)** ทีมต้องเคารพซึ่งกันและกัน และเห็นคุณค่าของการเคารพกัน

ทีมงานสกรัมประกอบด้วยเจ้าของผลิตภัณฑ์ (product owner) ทีมพัฒนา (development team) และสกรัมมาสเตอร์ (scrum master) โดยทั้งหมดคือทีมสกรัมที่จะขับเคลื่อนการพัฒนาให้ได้ผลิตภัณฑ์ที่มีคุณภาพและคุ้มค่า โดยทีมสกรัมจะมีจุดเด่นที่สามารถดูลแลบริหารจัดการทีมงานเองได้ และทีมงานมีศักยภาพพร้อมความสามารถที่หลากหลาย ยืดหยุ่น รวมทั้งทักษะที่จำเป็นในการพัฒนา ความคิดสร้างสรรค์ โดยไม่ต้องพึ่งพาภายนอก โดยทีมสกรัมมีลักษณะสำคัญคือ รู้หน้าที่ของตนเอง พึ่งพาตนเองได้ รู้ว่าจะต้องทำงานอะไรใน Product Backlog เพื่อสร้างงานในแต่ละรอบ พร้อมกับมีทักษะหลากหลายที่จำเป็นในการพัฒนา เป็นทีมหนึ่งเดียวไม่มีชื่อตำแหน่งไม่มีทีมย่อย ความรับผิดชอบต่องานเป็นของสมาชิกทุกคนในทีม สำหรับทีมสกรัมจะเป็นทีมขนาดเล็กคล่องตัวสูงศักยภาพสูง ขนาดที่เหมาะสมอยู่อาจอยู่ที่ 4-9 คนไม่นับรวม เจ้าของผลิตภัณฑ์และสกรัมมาสเตอร์

สกรัมมาสเตอร์ (scrum master) สกรัมมาสเตอร์มี สนับสนุนทีม ให้ใช้สกรัม อย่างถูกต้องทั้งทฤษฎีสกรัม ข้อกำหนด และ ข้อปฏิบัติต่าง ๆ ในการใช้สกรัม เป็นผู้ช่วยสนับสนุนการปฏิสัมพันธ์ภายในทีมและเป็นตัวแทนทีมในการปฏิสัมพันธ์กับบุคคลภายนอก เพื่อให้เกิดมูลค่าและประโยชน์สูงสุด โดยมีหน้าที่ต่อบุคคลอื่นดังนี้

สกรัมมาสเตอร์ต่อเจ้าของผลิตภัณฑ์

- ช่วยสร้างความเข้าใจในสกรัม เอจายล์ และ ขอบเขตงาน
- ช่วยหาเทคนิคต่าง ๆ ในการบริหารจัดการโครงการอย่างมีประสิทธิภาพ
- ช่วยให้เห็นความจำเป็นของสกรัม การพัฒนา และ Product Backlog
- ช่วยให้เจ้าของผลิตภัณฑ์เข้าใจการจัดเรียงความสำคัญใน Product Backlog
- สนับสนุนกิจกรรมต่าง ๆ ของสกรัม

สกรัมมาสเตอร์ต่อทีมพัฒนา

- ช่วยสร้างความเข้าใจสกรัม เอจายล์ และ ขอบเขตงาน
- ช่วยฝึกทีมให้สามารถทำงานแบบสกรัม และฝึกทักษะที่หลากหลาย
- สนับสนุนให้ทีมสามารถพัฒนาผลิตภัณฑ์ที่มีมูลค่าสูง
- ช่วยกำจัดอุปสรรคในการทำงานของทีม
- เผยแพร่และสอนการใช้สกรัมในองค์กร
- สนับสนุนกิจกรรมต่าง ๆ ของสกรัม

สกรัมมาสเตอร์ต่อองค์กร

- ผู้นำผู้สอนและผู้เผยแพร่การใช้สกรัมเพื่อสร้างความเข้าใจในองค์กร
- วางแผนการใช้สกรัมในองค์กร
- ช่วยกำจัดอุปสรรคในการทำงานของทีม
- ทำงานร่วมกับสกรัมมาสเตอร์คนอื่น ๆ

กิจกรรมสกรัม

กิจกรรมที่กำหนดในสกรัมนั้นออกแบบมาเพื่อให้ทีมนำไปใช้ให้เป็นปกตินิสัยภายในช่วงระยะเวลาที่กำหนดในแต่ละรอบที่เรียกว่า รอบ สปรินท์ (sprint) ที่จะถูกกำหนดไว้อย่างตายตัว เมื่อครบกำหนดเวลารอบสปรินท์ ทีมจะได้ส่วนเพิ่มเติมผลิตภัณฑ์และสามารถเริ่มรอบสปรินท์ใหม่ต่อไป ในระหว่างสปรินท์จะมีกิจกรรมต่าง ๆ ในการพัฒนา ที่ออกแบบมาเพื่อให้สามารถรองรับการเปลี่ยนแปลง โปร่งใส ตรวจสอบได้ตามหลักของสกรัม สำหรับกิจกรรมสกรัม สปรินท์ถือได้ว่าเป็นหัวใจสำคัญของการสกรัม

สปรินท์

หมายถึงกรอบเวลาที่กำหนดตายตัวไม่สามารถเปลี่ยนแปลงได้ตลอดช่วงการทำงานมีลักษณะเป็นรอบๆโดยรอบสปรินท์ใหม่จะเกิดทันทีหลังจากสิ้นสุดรอบสปรินท์ก่อนหน้า ซึ่งอาจถูกพิจารณาให้จำกัดไว้ไม่ให้เกิน 1 เดือน เพื่อช่วยลดความเสี่ยงและช่วยให้แต่ละรอบมีส่วนเพิ่มเติมผลิตภัณฑ์ออกมาใช้ในระยะเวลาไม่นาน โดยในสปรินท์จะประกอบด้วยกิจกรรมต่าง ๆ ดังต่อไปนี้ การวางแผนสปรินท์ (sprint planning) การทำสกรัมประจำวัน (daily scrum) การทำงาน การตรวจสอบสปรินท์ (sprint review) การทำสปรินท์ย้อนหลัง (sprint retrospective) โดยในระหว่างสปรินท์ สกรัมมาสเตอร์มีหน้าที่ในการบริหารจัดการให้สปรินท์ดำเนินไปอย่างราบลื่นทั้งการสนับสนุนทีมและการปกป้องสปรินท์ โดยในระหว่างสปรินท์จะต้อง

- ไม่มีการเปลี่ยนแปลงใดๆเกิดขึ้นที่อาจส่งผลกระทบต่อเป้าหมายสปรินท์
- ไม่มีการปรับลดเป้าหมายเชิงคุณภาพ
- ขอบเขตงานอาจมีการต่อรองใหม่ หรือชี้แจงระหว่างเจ้าของผลิตภัณฑ์กับทีมโดยมีสกรัมมาสเตอร์เป็นคนกลาง

การยกเลิกสปรินท์สามารถทำได้โดยเจ้าของผลิตภัณฑ์แต่มักจะไม่เกิดขึ้นบ่อยเนื่องจากแต่ละรอบสปรินท์มีระยะเวลาสั้น การยกเลิกสปรินท์ก่อนจึงมักไม่ค่อยเกิดและถ้าเกิดก็ไม่เสียหายมากนักเนื่องจากทรัพยากรที่ใช้ไปในระยะเวลาระหว่างรอบไม่ได้นานจึงไม่สูญเสียมาก หลังจากยกเลิกสปรินท์ทีมก็สามารถวางแผนสปรินท์ใหม่ได้ทันที

การวางแผนสปรินท์ (sprint planning)

สมาชิกในทีมร่วมกันวางแผนสปรินท์ที่จะทำในรอบสปรินท์ที่ปกติไม่เกิน 1 เดือน โดยสกรัมมาสเตอร์คอยช่วยสนับสนุนการดำเนินงานวางแผนให้แล้วเสร็จภายในเวลา 8 ชั่วโมง สาระสำคัญหลักในการวางแผนสปรินท์ คือการนิยามว่าจะทำงานอะไรในสปรินท์แล้วจะเลือกงานที่เลือกมาได้อย่างไร

ทีมทำการเลือกงานที่คาดว่าจะทำให้แล้วเสร็จในรอบสปรินท์โดยเจ้าของผลิตภัณฑ์ทำหน้าที่อภิปรายเป้าหมายของสปรินท์และงานไหนใน Product Backlog ที่สามารถทำให้บรรลุเป้าหมายได้ โดยทีมต้องทำความเข้าใจในงานที่จะต้องทำในรอบและพิจารณาเลือกงาน ใน Product Backlog ตามความเหมาะสมของปริมาณงาน ลำดับความสำคัญ ศักยภาพของทีม มาบรรจุในสปรินท์ด้วยความสมัครใจ และทำการเขียนเป้าหมายสปรินท์ที่ต้องทำให้ลุล่วงในรอบสปรินท์พร้อมส่งมอบผลิตภัณฑ์ส่วนเพิ่มเติม

สำหรับการเลือกงานที่จะมาบรรจุในสปรินท์นั้นต้องพิจารณาจากเป้าหมายสปรินท์และงานต่าง ๆ ที่ต้องทำจาก Product Backlog โดยต้องสามารถทำเสร็จในรอบและใช้ได้จริงตามนิยาม “Done” นำมาเขียนเป็น Sprint Backlog ที่ประกอบด้วยแผนการทำงานและส่งมอบงานในรอบสปรินท์ โดย Sprint Backlog จะแบ่งย่อยงานออกเป็นส่วนย่อย ๆ ที่ทีมสามาราถทำงานให้เสร็จในเวลาสั้น ๆ เช่น 1 วัน เพื่อให้แน่ใจว่างานทั้งหมด สามารถดำเนินกการให้บรรลุเป้าหมายสปรินท์ได้

ในช่วงท้ายของการวางแผนสปรินท์ทีมต้องสามารถอธิบายและให้ความมั่นใจกับเจ้าของผลิตภัณฑ์และสกรัมมาสเตอร์ได้ว่าทีมสามารถจัดการตนเองได้ เพื่อให้บรรลุเป้าหมายสปรินท์และส่งมอบผลิตภัณฑ์เพิ่มเติมได้ภายในรอบสปรินท์

การทำสกรัมประจำวัน (daily scrum)

การสกรัมในกีฬารักบี้เป็นกิจกรรมในการแข่งขันที่นักกีฬามารวมกัน เหมือนการสกรัมในการพัฒนาโดยในทีมจะร่วมกิจกรรม การทำสกรัมประจำวันในทุก ๆ วัน เป็นเวลา 15 นาทีตลอดระยะเวลาสปรินท์ เพื่อให้ทีมได้วางแผนในการทำงานในกรอบเวลา 24 ชั่วโมงโดยอาจเริ่มการประชุมแลกเปลี่ยนปัญหาที่ผ่านมาหลังจากการสกรัมประจำวันล่าสุด เพื่อทบทวนสิ่งที่ได้ทำ ปัญหา อุปสรรค และแนวทางแก้ไขให้ทุก ๆ คนในทีมรับรู้ ตรวจสอบความคืบหน้าของงานเทียบกับเป้าหมายสปรินท์ และ Sprint Backlog เพื่อประเมินความก้าวหน้าและวางแผนงานให้บรรลุเป้าหมาย โดยอาจใช้การตั้งคำถามในการอภิปรายเช่น เมื่อวานได้ทำอะไรจะช่วยให้ทีมบรรลุเป้าหมาสปรินท์ วันนี้จะทำอะไร อุปสรรคที่พบคืออะไร และ กระทบต่อเป้าหมายมากน้อยแค่ไหน

การสกรัมประจำวันช่วยส่งเสริมการสื่อสารในทีม การตัดสินใจ การเข้าแก้ไขปัญหา และ การรับมือการเปลี่ยนแปลงได้อย่างรวดเร็ว ที่สำคัญคือพยายามรักษาการสกรัมประจำวันให้อยู่ในกรอบเวลา 15 นาที โดยมีสกรัมมาสเตอร์คอยช่วยดูแลแนะนำการดำเนินการประชุม

การตรวจสอบสปรินท์ (sprint review)

การตรวจสอบสปรินท์จะดำเนินการตอนท้ายของสปรินท์เพื่อตรวจสอบผลิตภัณฑ์เพิ่มเติม ที่ถูกสร้างมาในรอบสปรินท์ เสร็จหรือไม่มีมูลค่าแค่ไหน เพื่อรับข้อเสนอแนะร่วมกันจากจากผู้มีส่วนเกี่ยวข้องและทีม โดยจะใช้เวลาไม่เกิน 4 ชั่วโมงต่อสปรินท์ 1 เดือน โดยดำเนินการในลักษณะดังต่อไปนี้

- เจ้าของผลิตภัณฑ์เชิญทีมและผู้มีส่วนเกี่ยวข้องทั้งหมดเข้าร่วมกิจกรรม
- เจ้าของผลิตภัณฑ์ชี้แจงงานใน Product Backlog ว่าส่วนไหนเสร็จ “Done” และ ส่วนไหนยังไม่เสร็จ
- ทีมพัฒนาอภิปรายงานในรอบสปรินท์ ทำอะไร ปัญหา และ แนวทางแก้ไข รวมทั้งสาธิตงานผลิตภัณฑ์เพิ่มเติมที่เสร็จ
- เจ้าของผลิตภัณฑ์ชี้แจงงานที่เหลือใน Product Backlog และกรอบเวลา
- ร่วมกันเลือกงานและวางแผนสำหรับสปรินท์ถัดไป
- ทบทวนสถานการณ์การเปลี่ยนแปลงต่างๆ โอกาสใหม่ๆ ทรัพยากรที่ใช้ และ แนวโน้วด้านการตลาด

การทำสปรินท์ย้อนหลัง (sprint retrospective)

เป็นกิจกรรมที่เปิดโอกาสให้สมาชิกได้ทบทวนและร่วมกันวางแผนปรับปรุงด้านต่าง ๆ จากสิ่งที่เรียนรู้มาเพื่อปรับแก้ไขให้ดีขึ้นสำหรับสปรินท์ถัดไป กิจกรรมนี้จะใช้เวลาไม่เกิน 3 ชั่วโมง มีสกรัมมาสเตอร์ช่วยดำเนินการประชุมให้เป็นไปอย่างราบลื่นภายในกรอบเวลา โดยเป้าหมายหลักของการทำสปรินท์ย้อนหลัง คือ การตรวจสอบสปรินท์ที่ผ่านมาจากหลาย ๆ มุมมองทั้งกระบวนการทำงาน การสื่อสาร ความสัมพันธ์ ตลอดจนเครื่องมือ และ เทคโนโลยีที่นำมาใช้เพื่อหาส่วนที่จะสามารถนำไปปรับปรุงได้และดำเนินการสร้างแผนการปรับปรุงตามความเห็นของทีม จากนั้นทีมจะทำการระบุแนวทางการปรับปรุงและพัฒนาที่จะถูกนำไปใช้ในสปรินท์ถัดไป กิจกรรมการทำสปรินท์ย้อนหลังเป็นการส่งเสริมและเปิดโอกาสให้ทีมได้ตรวจสอบและปรับเปลี่ยน

ผลลัพธ์จากกิจกรรมของกรัม (scrum artifacts)

ผลลัพธ์จากกิจกรรมของสกรัม เป็นส่วนที่ใช้แสดงถึงงานและคุณค่าของงานในแต่ละกิจกรรม โดยถูกออกแบบมาตามหลักความโปร่งใสตรวจสอบได้และให้ทุกคนมองเห็นภาพชัดเจนตรงกัน ประกอบด้วย Product Backlog Sprint Backlog และ ผลิตภัณฑ์เพิ่มเติม (increment)

Product Backlog เป็นงานทั้งหมดที่ต้องทำเพื่อสร้างผลิตภัณฑ์โดยจะถูกแบ่งย่อยเป็นส่วน ๆ และถูกจัดเรียงตามลำดับความสำคัญของงานใน Product Backlog ที่ใช้เป็นแกนหลักสำหรับสกรัม ถ้าจะมีการเปลี่ยนแปลงหมายถึงเปลี่ยนแปลงผลิตภัณฑ์ด้วยซึ่งมีเพียงเจ้าของผลิตภัณฑ์ที่มีหน้าที่รับผิดชอบต่อ Product Backlog ทั้งตัวเนื้องานและลำดับความสำคัญ โดยสกรัมใช้ Product Backlog ในการสื่อสารและเปรียบเหมือนตัวแทนผลิตภัณฑ์ที่สามารถถูกปรับเปลี่ยนได้ตลอดเวลา และถูกปรับปรุงตลอดการพัฒนา เพื่อรองรับการเปลี่ยนแปลงทางตลาด เทคโนโลยี และ การแข่งขัน

รายละเอียดงาน รายการของกลุ่มงานทั้งหมด ฟังก์ชันต่าง ๆ ทั้งหมดจะถูกบรรจุไว้ใน Product Backlog รวมถึงรายละเอียดการประมาณการงาน ลำดับความสำคัญ และ คำอธิบายการทดสอบงานที่เสร็จสมบูรณ์ เพื่อให้ทีมสกรัมสามารถนำไปใช้ในระหว่างการพัฒนา นอกจากนั้นข้อเสนอแนะจากผู้มีส่วนเกี่ยวข้อง ความต้องการ สถานการณ์ตลาด รูปแบบธุรกิจเหล่านี้สามารถนำใส่เพิ่มเข้าไปได้โดยส่งผลให้ Product Backlog ถูกปรับปรุงตลอดเวลาการสกรัม

เจ้าของผลิตภัณฑ์และทีมสกรัมมีหน้าที่จัดการลำดับการพัฒนาจาก กลุ่มงานใน Product Backlog เพื่อนำไปบรรจุในรอบสปรินท์เพื่อพัฒนา นอกจากนั้นยังใช้ Product Backlog ในการตรวจสอบความก้าวหน้าของโครงการและการตรวจสอบสปรินท์ สำหรับการบริหารจัดการโครงการให้ลุล่วงตามเป้าหมาย

Sprint Backlog เป็นชุดงานที่ถูกเลือกมาจาก Product Backlog และเป้าหมายสปรินท์ตามแผนที่วางไว้เพื่อส่งมอบผลิตภัณฑ์เพิ่มเติมเมื่อสิ้นสุดรอบสปรินท์ โดย Sprint Backlog จะถูกแบ่งงานออกเป็นส่วนย่อย ๆ เพื่อใช้ในการทำงานในระหว่างรอบสปรินท์โดยเป็นแผนงานที่ละเอียดเพียงพอต่อการทำงาน โดยจะถูกนำไปใช้ในการสกรัมประจำวันเพื่อตรวจสอบความก้าวหน้าของงานและปรับปรุงข้อมูลงานที่เหลือ โดยมีวัตถุประสงค์เพื่อให้ทีมทำงานตาม Sprint Backlog ให้บรรลุเป้าหมายสปรินท์และสามารถส่งมอบผลิตภัณฑ์เพิ่มเติมภายในรอบสปรินท์

ผลิตภัณฑ์เพิ่มเติม (increment) เป็นผลรวมของงานจาก Product Backlog ที่ได้มาจากการพัฒนาที่เสร็จในแต่ละสปรินท์ โดยเสร็จ “Done” คือผลิตภัณฑ์เพิ่มเติมที่นำไปใช้งานได้จริงตามข้อกำหนดในนิยาม “Done” และต้องสามารถตรวจสอบได้ โปร่งใส และมีมูลค่าสูง ตรงตามวัตถุประสงค์ของโครงการ และเมื่องานเสร็จสมบูรณ์ครบถ้วนแล้ว ผลิตภัณฑ์พร้อมใช้งานหรือพร้อมออกสู่ตลาด

การพัฒนาซอฟต์แวร์แบบเอจายล์เป็นกรอบการทำงาน หลักแนวคิด และ หลักปฏิบัติ บนแถลงการณ์แห่งเอจายล์ (agile manifesto) และ หลักการ 12 ข้อในการพัฒนาซอฟต์แวร์แบบเอจายล์ (12 principles of agile) ที่สามารถนำมาประยุกต์ใช้ในสถานการณ์ปัจจุบันที่เปลี่ยนแปลงเร็ว เนื่องด้วยความคล่องตัวรวดเร็ว และ สามารถตอบสนองต่อความเปลี่ยนแปลงได้ดีส่งผลให้เอจายล์ถูกนำไปใช้ในการพัฒนาซอฟต์แวร์อย่างแพร่หลายในปัจจุบัน

บทที่ 4 การกำหนดความต้องการและข้อกำหนดของซอฟต์แวร์ Software requirements and specifications

วัตถุประสงค์

- *เข้าใจการกำหนดความต้องการและข้อกำหนดของซอฟต์แวร์*
- *เข้าใจกระบวนการวิศวกรรมความต้องการ*

ในบทนี้จะกล่าวถึงการกำหนดความต้องการและข้อกำหนดของซอฟต์แวร์ โดยจะอธิบายถึงกระบวนการที่เกี่ยวข้องในการค้นหาความต้องการการจัดทำเอกสารเพื่อให้ความต้องการของซอฟต์แวร์รวมทั้งการจัดทำข้อกำหนดของซอฟต์แวร์ ทำความเข้าใจเกี่ยวกับลักษณะของความต้องการที่แตกต่างในหลายมุมมอง เข้าใจกระบวนการวิศวกรรมความต้องการ ทำความเข้าใจระบบปัญหา และ ความต้องการของซอฟต์แวร์ เพื่อนำไปสู่การนิยามคุณลักษณะ ข้อกำหนดข้อจำกัด และ การทำงานของซอฟต์แวร์ที่กำลังจะถูกพัฒนาขึ้น

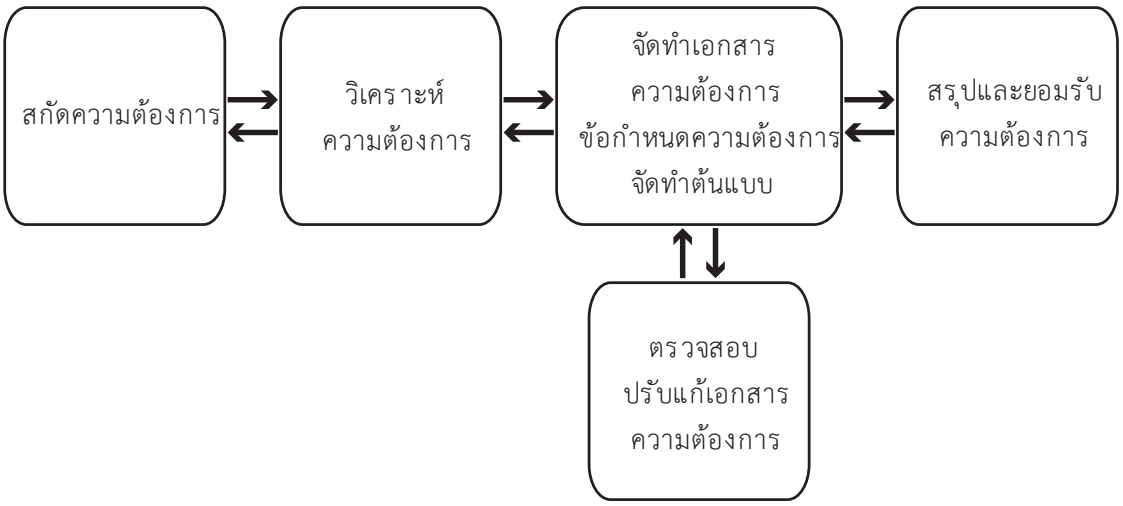
ความต้องการของระบบประกอบด้วยการอธิบายบริการต่าง ๆ ของระบบซอฟต์แวร์บนพื้นฐานข้อจำกัดและการดำเนินการของซอฟต์แวร์ โดยสะท้อนถึงความต้องการของผู้ใช้ในด้านต่างๆ เช่นการควบคุมอุปกรณ์ การจัดการธุรกรรม การค้นหาข้อมูล ซึ่งกระบวนการค้นหาความต้องการ การวิเคราะห์ และ กระบวนการนการจัดทำเอกสารความต้องการ ถูกจัดอยู่ในกระบวนการวิศวกรรมความต้องการ ความต้องการสามารถถูกมองได้จากหลายมุมมอง เช่น ความต้องการผู้ใช้ (user requirement) เป็นคำอธิบายนามธรรมระดับสูงผ่านมุมมองของผู้ใช้ ในขณะที่ความต้องการระบบ (system requirement) เป็นการอธิบายในรายละเอียดว่าระบบควรจะทำหรือสามารถทำอะไรได้บ้าง

1. ความต้องการผู้ใช้ (user requirement) คือความต้องการของผู้ใช้หรือความคาดหวังจากผู้ใช้ต่อระบบว่าสามารถที่จะ ทำอะไร ให้กับผู้ใช้ได้บ้าง มีข้อจำกัดอะไรในการดำเนินการ ส่วนมากจะเขียนอยู่ในลักษณะภาษาธรรมชาติประกอบกับการใช้แผนภาพ เพื่อใช้อธิบายละเอียด การทำงานฟังชันต่างๆของระบบซอฟต์แวร์
2. ความต้องการระบบ (system requirement) คือคำอธิบายความต้องการของระบบในรูปของฟังก์ชันการทำงานต่าง ๆ ของซอฟต์แวร์ จะสามารถทำอะไรให้กับได้บ้าง มีบริการอะไร มีข้อกำหนดหรือข้อจำกัดอะไร โดยเขียนออกมาเป็นรายละเอียด ซึ่งบางครั้งอาจเรียกว่าข้อกำหนดฟังก์ชัน

ความต้องการของซอฟต์แวร์ถือว่ามีความสำคัญมากในการพัฒนาซอฟต์แวร์ เพราะใช้ในการอธิบาย ความสามารถของซอฟต์แวร์ บริการต่าง ๆ ตลอดจนข้อจำกัดต่าง ๆ ทั้งผู้ใช้และผู้พัฒนาซอฟต์แวร์ควรทำความเข้าใจความต้องการของซอฟต์แวร์ให้ถูกต้องและตรงกัน เพื่อให้เกิดความชัดเจนในการอิบายความต้องการซอฟต์แวร์ สามารถอธิบายในรูปความต้องการเชิงฟังก์ชันและความต้องการไม่เชิงฟังก์ชัน

ภาพรวมของกระบวนการวิศวกรรมความต้องการจะดำเนินกิจกรรมเพื่อให้ได้มาซึ่งความต้องการและข้อกำหนดของซอฟต์แวร์ที่ทุกฝ่ายยอมรับและเข้าใจตรงกัน ประกอบด้วยการรวบรวมและสกัดความต้องการ การวิเคราะห์ความต้องการเพื่อ

จัดทำเอกสารความต้องการ ข้อกำหนดความต้องการ การตรวจสอบความต้องการ การสรุป และ ยอมรับความต้องการ ดังปรากฏในรูปที่ 4.1

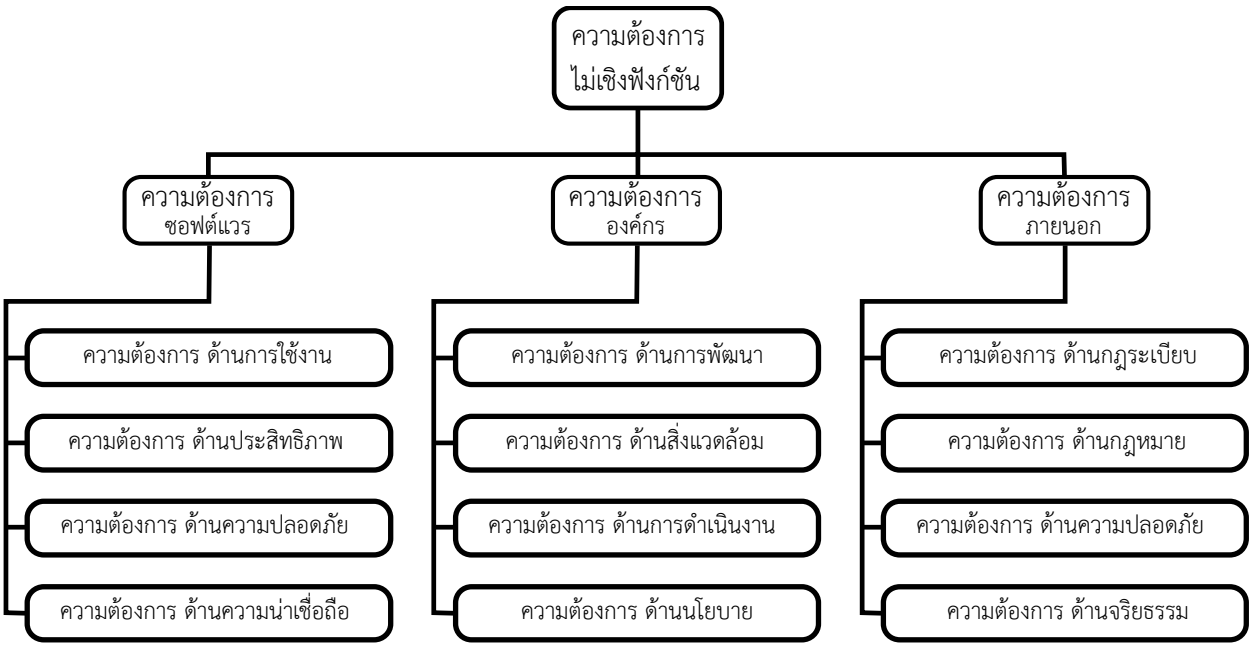


รูปที่ 4.1 ภาพรวมกระบวนการวิศวกรรมความต้องการ

ความต้องการเชิงฟังก์ชันและความต้องการไม่เชิงฟังก์ชัน (functional and nonfunctional requirements)

เพื่อให้เห็นภาพรวมของระบบ ความต้องการเชิงฟังก์ชัน และ ความต้องการไม่เชิงฟังก์ชันสามารถนำมาเขียนอธิบายความต้องการซอฟต์แวร์ ซึ่งทางผู้ใช้และผู้พัฒนาสามารถเข้าใจความต้องการได้ชัดเจนได้ตรงกัน

1. ความต้องการเชิงฟังก์ชัน คือความต้องการให้เขียนอธิบายในรูปแบบของฟังก์ชันหลักของซอฟต์แวร์ โดยมักจะเป็นส่วนที่มีความสำคัญกับการทำงานหลักหรือบริการที่ผู้ใช้ต้องการ ยกตัวอย่างความต้องการเชิงฟังก์ชัน ระบบต้องสามารถอ่านค่าและแสดงผลอัตราการเต้นของหัวใจของผู้ใช้ได้แบบเวลาจริง ระบบต้องสามารถแจ้งเตือนผู้ใช้ในกรณีอัตราการเต้นของหัวใจผิดปกติ
2. ความต้องการไม่เชิงฟังก์ชัน คือความต้องการที่อยู่ในรูปของข้อจำกัดของระบบหรือข้อจำกัดของบริการ ข้อจำกัดในด้านเวลา ประสิทธิภาพของระบบ จำกัดทางด้านมาตรฐาน และ ความต้องการอื่นๆที่ไม่สามารถเขียนในรูปความต้องการเชิงฟังก์ชันได้ รูปที่ 4.2 แสดงหมวดหมู่ความต้องการไม่เชิงฟังก์ชัน ความต้องการด้านการใช้งาน ความต้องการด้านประสิทธิภาพ ความต้องการด้านความน่าเชื่อถือ ความต้องการด้านความปลอดภัย ความต้องการตามนโยบาย ความต้องการตามมาตรฐาน ความต้องการด้านกฎหมาย ความต้องการการเข้ากันได้ และ ความต้องการด้านจริยธรรม ยกตัวอย่างความต้องการไม่เชิงฟังก์ชัน ระบบต้องสามารถประมวลผลอย่างน้อยหนึ่งร้อยชุดข้อมูลต่อวินาที ระบบต้องมีการเข้ารหัสและจัดเก็บข้อมูลเพื่อความปลอดภัยของข้อมูลผู้ป่วยตามมาตรฐานสากล



รูปที่ 4.2 ความต้องการไม่เชิงฟังก์ชัน

การสกัดและวิเคราะห์ความต้องการ (requirement elicitation)

คือกระบวนการที่ทำให้วิศวกรซอฟต์แวร์เข้าใจถึงความต้องการของลูกค้าและผู้มีส่วนเกี่ยวข้องด้วยการสกัดและวิเคราะห์ความต้องการจากกิจกรรมการเฝ้าสังเกต ถือว่าเป็นกิจกรรมที่สำคัญเพื่อทำความเข้าใจปัญหา เข้าใจระบบ และ ข้อจำกัดต่างๆ เพื่อนำไปสู่การได้มาซึ่งความต้องการที่แท้จริง สามารถดำเนินการได้หลากหลายวิธีทั้งแบบเป็นทางการและไม่เป็นทางการ โดยมีวิธีการสกัดในรูปแบบหลัก สาม วิธีคือ การสัมภาษณ์ การสังเกต และ การบรรยายเรื่องราว

1. การสัมภาษณ์ สามารถดำเนินการได้ทั้งแบบเป็นทางการและไม่เป็นทางการโดยการพูดคุยสอบถามผู้ที่มีส่วนเกี่ยวข้องด้วยการถามคำถามเพื่อเก็บรวบรวมข้อมูลที่เกี่ยวของกับระบบการทำงานระบบเดิมและระบบที่กำลังจะพัฒนาโดยความต้องการจะถูกสกัดออกมาจากคำตอบที่ได้ทำการรวบรวมมา กระบวนการสัมภาษณ์สามารถใช้วิธีการสัมภาษณ์แบบปิดโดยให้ผู้มีส่วนเกี่ยวข้องเลือกตอบคำถามจากคำตอบที่เตรียมไว้ และการสัมภาษณ์แบบปลายเปิดที่ไม่มีการกำหนดคำถามไว้ล่วงหน้าเพื่อให้สามารถเก็บข้อมูลได้กว้างและลึกแต่ทั้งนี้ต้องขึ้นอยู่กับข้อจำกัดด้านเวลาในการสัมภาษณ์ด้วย
2. การสังเกต สามารถดำเนินการสังเกตการณ์การทำงานและดำเนินการของระบบทั้งการเยี่ยมชมสถานที่ และการสังเกตการทำงาน และ กระบวนการต่าง ๆ ที่เกิดขึ้นในระบบเพื่อให้เข้าใจระบบ พบปัญหาและแนวทางแก้ปัญหา หากแต่วิธีนี้ใช้ระยะเวลานานในการดำเนินการ หลังจากได้ข้อมูลมาแล้ว ก็จะเข้าสู่การนำมาวิเคราะห์เพื่อให้ได้ความต้องการ
3. การบรรยายเรื่องราวและสถานการณ์ เป็นกรรมวิธีในการสกัดข้อมูลโดยมุ่งเน้นการบรรยาย ระบบและความต้องการผ่านการเขียนเล่าเรื่องราวและสถานการณ์ โดยผู้มีส่วนเกี่ยวข้องทำการเขียนบอกเล่าเรื่องราวในภาษาและบริบทการทำงานจริงที่สามารถถ่ายทอดออกมาได้อย่างชัดเจนและเข้าใจง่าย มักจะเลือกใช้ภาษาธรรมชาติในการบรรยาย นอกจากการบอกเล่าเรื่องราวเพื่ออธิบายระบบแล้ว ผู้มีส่วนเกี่ยวข้องสามารถเขียนบรรยายสถานการณ์จากจินตนาการ โดยคิดถึงภาพสถานการณ์ที่จะเกิดขึ้นกับระบบใหม่ โดยความต้องการจะถูกบันทึกในเรื่องราวเหล่านี้ในรูปแบบที่อ่านแล้วเข้าใจง่ายเห็นภาพได้ชัดเจน เพื่อนำไปสู่การสกัดออกมาเป็นความต้องการ

ข้อกำหนดความต้องการ (requirement specification)

เป็นกิจกรรมในการแปลงข้อมูลที่รวบรวมในช่วงการวิเคราะห์ความต้องการให้เป็นเอกสารสำหรับนิยามความต้องการ ในสองรูปแบบคือความต้องการของผู้ใช้หรือลูกค้าและความต้องการของระบบที่อธิบายลงรายละเอียดถึงฟังก์ชันการทำงานต่าง ๆ โดยข้อกำหนดความต้องการ ควรมีความชัดเจน ไม่กำกวม เข้าใจง่าย และ มีความสมบูรณ์

โดยปกติความต้องการผู้ใช้จะถูกเขียนในรูปแบบภาษาธรรมชาติ แผนภาพ แผนภูมิมาประกอบ เพื่ออธิบายให้ชัดเจน โดยความต้องการผู้ใช้นั้นจะถูกเขียนอธิบายความต้องการเชิงฟังก์ชันและไม่เชิงฟังก์ชัน ควรเขียนให้อยู่ในรูปแบบของภาษาธรรมชาติให้มากที่สุด หลีกเลี่ยงศัพท์เทคนิค หรือแบบจำลองทางคณิตศาสตร์ที่ซับซ้อน ถ้าจำเป็นควรเลือกใช้แผนภูมิหรือตารางอย่างง่ายในการอธิบาย

ในส่วนข้อกำหนดความต้องการระบบ อาจเขียนอยู่ในรูปแบบที่วิศวกรซอฟต์แวร์สามารถเข้าใจได้และสามารถนำไปสู่การพัฒนาระบบต่อไป ซึ่งรวมถึงการเขียนข้อกำหนดสำหรับการสร้างซอฟต์แวร์ลงไปด้วย โดยปกติข้อกำหนดความต้องตามระบบจะเขียนอธิบายพฤติกรรมของระบบ รูปแบบการปฏิสัมพันธ์ ฟังชันการทำงานโดยรวม และ ข้อจำกัดต่าง ๆ ของระบบ เพื่อให้เข้าใจระบบชัดเจนมากขึ้นวิศวกรซอฟต์แวร์สามารถใส่รายละเอียดการออกแบบสถาปัตยกรรมซอฟต์แวร์เบื้องต้นลงไปได้ เพื่อให้เห็นภาพรวมของระบบพร้อมความต้องการที่สอดคล้องกันทั้งความต้องการระบบและความต้องการของผู้ใช้

การเขียนอธิบายข้อกำหนดความต้องการสามารถเขียนได้หลากหลายรูปแบบเช่น การเขียนข้อกำหนดด้วยภาษาธรรมชาติ การเขียนข้อกำหนดเชิงโครงสร้าง การใช้แผนภูมิยูสเคส การเขียนเอกสารตามมาตรฐาน ข้อกำหนดความต้องการซอฟต์แวร์ (SRS)

การตรวจสอบความต้องการ (requirement validation)

เป็นกิจกรรมเพื่อตรวจสอบให้มั่นใจว่าข้อกำหนดความต้องการที่สกัดออกมาได้นั้น เป็นความต้องการที่ถูกต้องแท้จริงและสนองความต้องการได้ ในขั้นตอนนี้อาจตรวจพบความผิดพลาดของข้อกำหนดความต้องการซึ่งนำไปสู่การปรับแก้ไขให้ถูกต้อง กระบวนการตรวจสอบความต้องการนี้นับว่ามีความสำคัญมากในการพัฒนาซอฟต์แวร์ เนื่องจากหากมีข้อผิดพลาดเกิดขึ้นที่ความต้องการ จะนำไปสู่การเพิ่มงานในการแก้ไขพร้อมทั้งการมีค่าใช้จ่ายในการแก้ไขให้ถูกต้องตามมาด้วย ยิ่งไปกว่านั้นหากข้อผิดพลาดในความต้องการถูกตรวจพบในขณะพัฒนาหรือหลังจากที่สร้างซอฟต์แวร์เสร็จแล้ว จะทำให้เกิดความเสียหายมากมาย ทั้งเวลาและทรัพยากรที่ต้องใช้ในการแก้ไขข้อผิดพลาดของทั้งระบบ ฉะนั้นวิศวกรซอฟต์แวร์ควรให้ความใส่ใจในการตรวจสอบความต้องการเพื่อลดความเสี่ยง ในกระบวนการพัฒนาซอฟต์แวร์

ในกระบวนการตรวจสอบความต้องการ จะนำเอกสารความต้องการมาทำการตรวจสอบในแต่ละประเด็นแยกเป็นหัวข้อเพื่อตรวจหาข้อผิดพลาดอย่างละเอียดแนะนำไปสู่การปรับปรุงแก้ไขเอกสารความต้องการให้ถูกต้อง ดังประเด็นต่อไปนี้

1. ตรวจสอบความถูกต้อง ทำการตรวจสอบความต้องการว่าสะท้อนความถึงต้องการที่แท้จริงของผู้ใช้หรือไม่ และความต้องการยังคงเหมือนเดิมหรือไม่ เนื่องจากความต้องการอาจเปลี่ยนไปจากการสกัดข้อมูลในช่วงแรก
2. ตรวจสอบความสมบูรณ์ ตรวจความสมบูรณ์ถูกต้องครบถ้วนของเอกสาร มีการนิยามทุกฟังก์ชันการทำงานของระบบตลอดจนข้อจำกัด ต่าง ๆ ของระบบ

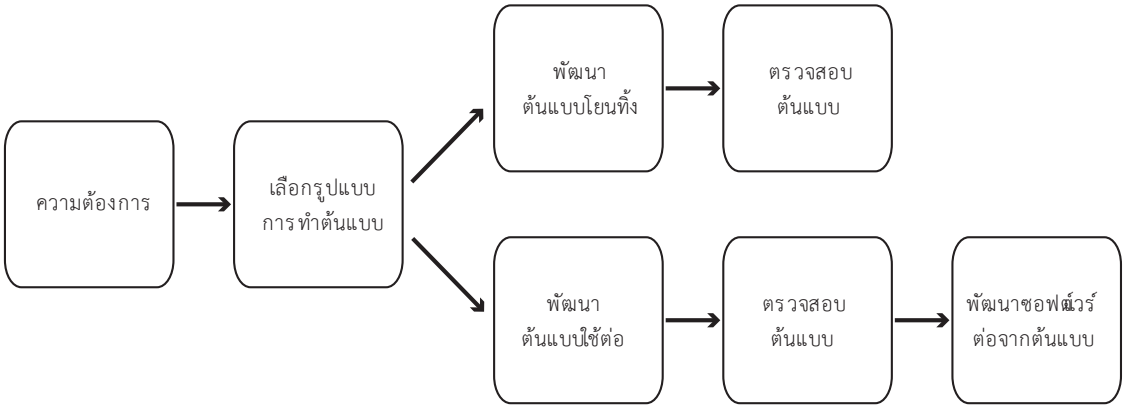
3. ตรวจสอบความเป็นไปได้ ตรวจสอบขีดความสามารถของทีมพัฒนา เทคโนโลยีที่มี และ ข้อจำกัดด้านต่าง ๆ เพื่อให้แน่ใจว่าสามารถพัฒนาซอฟต์แวร์ให้แล้วเสร็จและสามารถทำได้จริงภายใต้กรอบเวลาและทรัพยากรที่กำหนด
4. ตรวจสอบความขัดแย้ง เอกสารความต้องการควรมีความชัดเจน ไม่มีข้อขัดแย้งระหว่างฟังชันการทำงานและข้อกำหนดต่าง ๆ
5. ตรวจสอบการทดสอบได้ เอกสารความต้องการต้องเขียนความต้องการในรูปแบบที่สามารถทำการทดสอบได้ในแต่ละข้อ เพื่อลดโอกาสของการเกิดความขัดแย้งระหว่างผู้พัฒนาและลูกค้าในกระบวนการส่งมอบซอฟต์แวร์ เนื่องจากซอฟต์แวร์จะต้องถูกทดสอบเทียบกับความต้องการในการส่งมอบ

นอกจากการตรวจสอบในแต่ละประเด็นแล้ว วิศวกรซอฟต์แวร์ควรจะอ่านตรวจทานเอกสารความต้องการเพื่อตรวจสอบความถูกต้อง ความคงเส้นคงวาของเอกสาร และ ภาษาที่ใช้เขียน โดยกระบวนการตรวจทานนี้สามารถทำแบบรายบุคคลหรือประชุมกลุ่ม โดยอาจทำเป็นรายงานสรุปเพื่อแจ้งผลการตรวจทานเอกสารความต้องการ นอกจากนั้นยังมีกระบวนการออกแบบและจัดทำต้นแบบ (prototyping) ที่สามารถให้ทุกคนเห็นภาพรวมของระบบได้อย่างชัดเจน พื่อใช้ในการตรวจสอบความต้องการ

การทำต้นแบบ (prototyping)

การสร้างต้นแบบสำหรับอธิบายความต้องการสามารถช่วยให้ผู้มีส่วนเกี่ยวข้องเข้าใจและเห็นภาพของระบบมากขึ้น และยังเป็นการสร้างความเข้าใจที่ตรงกันระหว่างลูกค้าและผู้พัฒนา โดยทั้งทีมและลูกค้าจะเห็นภาพรวมที่เหมือนกันเกิดความเข้าใจตรงกัน ผู้พัฒนาสามารถใช้ต้นแบบในการแสดงให้เห็นถึงศักยภาพที่จะสามารถพัฒนาซอฟต์แวร์ให้แล้วเสร็จได้ ต้นแบบยังสามารถนำมาใช้ในการทดสอบให้เห็นว่าสามารถที่จะตอบสนองความต้องการของลูกค้าได้

วิศวกรซอฟต์แวร์สามารถสร้างต้นแบบขึ้นมาเพื่อตรวจสอบความต้องการและใช้สำหรับการเก็บข้อมูลป้อนกลับจากลูกค้า โดยต้นแบบนั้นต้องตอบคำถามสำคัญให้ได้ว่า ต้นแบบสามารถตอบสนองความต้องการของลูกค้าได้หรือไม่ เครื่องมือในการสร้างต้นแบบมีให้เลือกมากมาย หน้าที่ของวิศวกรซอฟท์แวร์คือเลือกเครื่องมือที่เหมาะสม เพื่อใช้สำหรับสร้างต้นแบบออกมาในระยะเวลาอันสั้นและใช้ทรัพยากรให้น้อยที่สุด การพัฒนาต้นแบบสามารถเริ่มสร้างจากกันการออกแบบบนกระดาษ หรือกระดานในห้องประชุม จนนำไปสู่การพัฒนาซอฟต์แวร์ต้นแบบเพื่อใช้ในการทดสอบความต้องการของลูกค้าและทดสอบความเป็นได้ในการพัฒนาซอฟต์แวร์

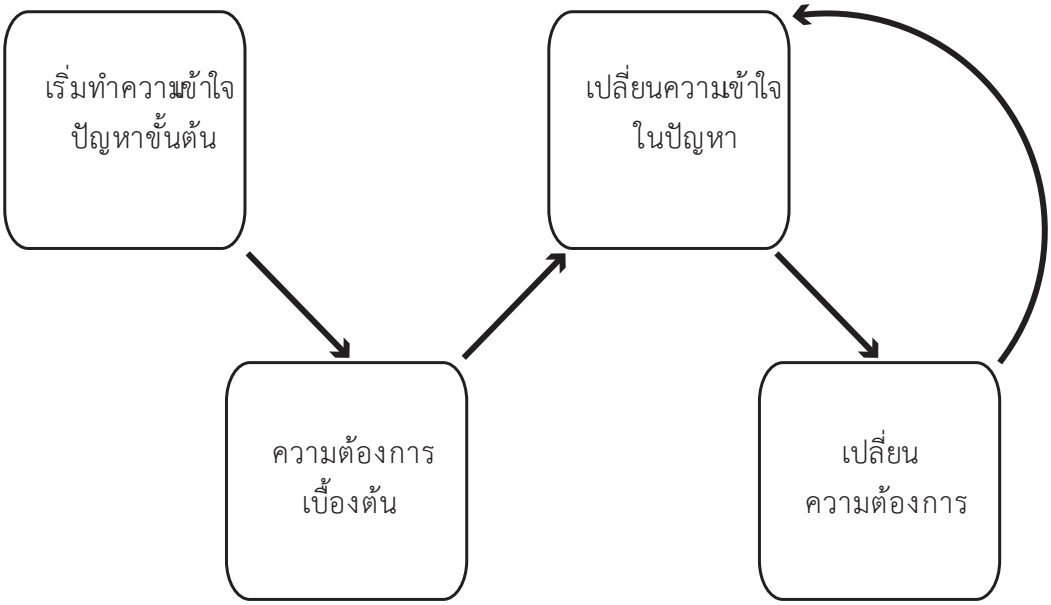


รูปที่ 4.3 การพัฒนาต้นแบบ

การพัฒนาต้นแบบมี สอง ลักษณะคือ ต้นแบบโยนทิ้ง (throwaway prototype) และ การทำต้นแบบ แล้วใช้ต่อ ดังรูปที่ 4.3 ในทางวิศวกรรมซอฟต์แวร์จะแนะนำให้ผู้พัฒนาเลือกใช้การพัฒนาต้นแบบโยนทิ้งเพื่อทำการทดสอบแนวคิดและนำเสนอต้นแบบให้ผู้มีส่วนเกี่ยวข้องตรวจสอบตามความต้องการ ใช้เพื่อนำเสนอวิสัยทัศน์ของซอฟต์แวร์ให้กับทีมพัฒนาและลูกค้าเห็นภาพเดียวกันและเข้าใจตรงกัน หลังจากนั้นในขั้นตอนการพัฒนาจริงจะไม่ได้นำต้นแบบโยนทิ้งมาใช้ในการพัฒนาต่อ จะเริ่มดำเนินการพัฒนาซอฟต์แวร์ขึ้นมาใหม่ แต่ในทางกลับกันต้นแบบใช้ต่อนั้นหลังจากตรวจสอบตรงตามความต้องการเรียบร้อยแล้ว จะถูกนำไปพัฒนาต่อยอดเพื่อเป็นซอฟต์แวร์ที่ใช้งานจริง ซึ่งแม้กรณีนี้จะมีความเสี่ยงเกิดขึ้นเนื่องจากซอฟต์แวร์ต้นแบบอาจไม่ได้มีการวางแผนออกแบบมาเป็นอย่างดี มีโครงสร้างไม่เสถียรเนื่องจากเกิดการเปลี่ยนแปลงโครงสร้างบ่อย นั่นคือสาเหตุว่าทำไมจึงควรใช้ต้นแบบเพียงเพื่อตรวจสอบความต้องการของลูกค้าและไม่เหมาะนำไปใช้ต่อในขั้นตอนการพัฒนาซอฟต์แวร์จริง

การเปลี่ยนแปลงความต้องการ (requirement change)

ความต้องการของลูกค้าสามารถเปลี่ยนแปลงได้อยู่เสมอโดยพาะในการพัฒนาซอฟต์แวร์ในระบบขนาดใหญ่ ทั้งสาเหตุจากการเปลี่ยนแปลงโดยสภาพตลาด ธุรกิจ หรือเกิดจากการนิยามปัญหาไม่ครอบคลุมหรือชัดเจนพอทำให้การเปลี่ยนแปลงความต้องการจึงเป็นเหตุที่ไม่สามารถหลีกเลี่ยงได้ การจัดการกับการเปลี่ยนแปลงความต้องการจึงเป็นกระบวนการที่สำคัญ



รูปที่ 4.4 การวิวัฒนาการความต้องการ

จากรูปที่ 4.4 จะเห็นได้ว่าในระหว่างการพัฒนาความต้องการสามารถปรับเปลี่ยนได้เมื่อมีความเข้าใจปัญหาที่ชัดเจนมากขึ้นเกิดการเปลี่ยนแปลงความต้องการได้เสมอ ถึงแม้ผลิตภัณฑ์จะถูกส่งมอบแล้วหรือนำไปใช้งานแล้วก็ตาม การเปลี่ยนแปลงสามารถเกิดขึ้นได้ อาจเพราะผู้มีส่วนเกี่ยวข้องได้เข้าใจปัญหามากขึ้นและทราบความต้องการที่ชัดเจนหรือมีการตรวจพบข้อผิดพลาดหรือปัญหา ซึ่งสิ่งเหล่านี้จะถูกนำมาร่วมพิจารณาเพื่อดำเนินการเปลี่ยนแปลงความต้องการต่อไป

สิ่งที่วิศวกรซอฟต์แวร์ต้องรับมือกับการเปลี่ยนแปลงความต้องการคือการวิเคราะห์และตัดสินใจในการเปลี่ยนแปลงบนพื้นฐานความถูกต้องลำดับความสำคัญความคุ้มค่าตลอดจน การดำเนินการติดตามปรับปรุงความต้องการและการพัฒนาที่เกี่ยวข้องทั้งหมดเพื่อให้ทุกส่วนสอดคล้องกับการเปลี่ยนแปลงความต้องการที่เกิดขึ้น โดยการดำเนินการบริหารจัดการความต้องการเพื่อบริหารจัดการการเปลี่ยนแปลงความต้องการได้อย่างมีประสิทธิภาพ สามารถใช้เครื่องมือทางวิศวกรรมซอฟต์แวร์มาช่วยในการจัดการบริหารการเปลี่ยนแปลง ช่วยจัดเก็บความต้องการอย่างเป็นระบบ ระบุความสัมพันธ์ความเชื่อมโยงต่าง ๆ ในความต้องการการจัดการ และ ติดตามการเปลี่ยนแปลง รวมถึงสามารถสืบย้อนความต้องการที่เกิดขึ้นได้ สำหรับโครงการขนาดเล็กอาจจะเลือกประยุกต์ใช้เครื่องมือพื้นฐานเพื่อบริหารจัดการเช่นการแชร์เอกสารในทีมหรือการใช้ตารางคำนวณก็เพียงพอ

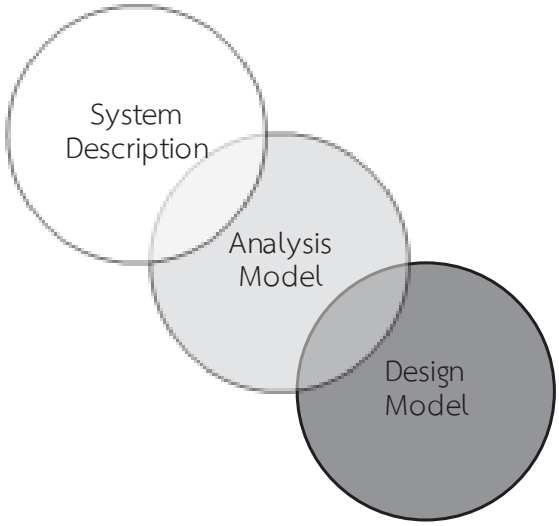
สำหรับกิจกรรมการจัดการการเปลี่ยนแปลงความต้องการนั้น ทีมสามารถดำเนินการได้หลังจากได้รับความเห็นชอบให้จัดการการเปลี่ยนแปลง เริ่มจากกระบวนการระบุปัญหาและการเปลี่ยนแปลงเพื่อสร้างข้อกำหนดการเปลี่ยนแปลง ทำการวิเคราะห์ผลกระทบต่าง ๆ ที่จะเกิดขึ้นในการเปลี่ยนแปลงและพิจารณาข้อกำหนดการเปลี่ยนแปลงว่าถูกต้องหรือไม่ ทำการหารือร่วมกับผู้ร้องขอการเปลี่ยนแปลงเพื่อดูว่าต้องเพิ่มเติมรายละเอียดหรือควรจะยกเลิกการดำเนินการ จากนั้นหากต้องการดำเนินการเปลี่ยนแปลง ทางทีมต้องทำการวิเคราะห์ค่าใช้จ่ายที่จะเกิดขึ้นเพื่อคำนวณความคุ้มค่าในการดำเนินการเปลี่ยนแปลง สมควรดำเนินการต่อหรือไม่ หากสมควรต้องดำเนินการต่อ สิ่งที่จะต้องปรับอันดับแรกคือเอกสารความต้องการเพื่อส่งต่อให้ทีมพัฒนาดำเนินการพัฒนาแก้ไขตามเอกสารความต้องการล่าสุดนี้รวมทั้งทำการปรับปรุงเอกสารที่เกี่ยวข้องกับการเปลี่ยนแปลงทั้งหมด ในกรณีการเปลี่ยนแปลงนั้นมีความเร่งด่วนหรือฉุกเฉิน ทีมสามารถดำเนินการแก้ไขก่อนได้ในทันทีและจัดการแก้ไขเอกสารตามภายหลังโดยยึดหลักการ รักษาผลประโยชน์สูงสุดให้กับลูกค้าเป็นสำคัญ

วัตถุประสงค์

- *เข้าใจการวิเคราะห์ซอฟต์แวร์*
- *เข้าใจแบบจำลองที่ได้จากการวิเคราะห์*

ในกระบวนการวิศวกรรมซอฟต์แวร์การวิเคราะห์ซอฟต์แวร์เป็นกิจกรรมที่สำคัญในการทำความเข้าใจระบบและความต้องการของลูกค้าเพื่อสร้างออกมาเป็นแบบจำลองที่ใช้เป็นสื่อกลางระหว่างทีมพัฒนาลูกค้าและผู้มีส่วนเกี่ยวข้อง โดยแบบจำลองที่ได้จากการวิเคราะห์นี้จะสะท้อนระบบซอฟต์แวร์ทั้งหมด โดยแบบจำลองที่ได้จากการวิเคราะห์ จะถูกนำไปใช้ในกระบวนการออกแบบและพัฒนาซอฟต์แวร์ในลำดับต่อไป เนื่องจากซอฟต์แวร์อยู่ในรูปนามธรรม การวิเคราะห์และสร้างแบบจำลองมีการใช้สัญลักษณ์แผนภาพ เพื่อแสดงให้เห็นในแต่ละมุมมองของระบบที่สามารถเห็นภาพ เข้าใจตรงกัน และ สื่อสารกันง่ายขึ้น

ทางวิศวกรรมซอฟต์แวร์ มีการใช้แบบจำลองหลากหลายรูปแบบขึ้นอยู่กับวัตถุประสงค์และรูปแบบระบบที่จะถูกจำลอง แบบจำลองพื้นฐานที่ใช้ในกระบวนการวิเคราะห์ซอฟต์แวร์คือ แบบจำลองระบบ ที่สามารถอธิบายความต้องการของลูกค้าและภาพรวมการทำงานของระบบ ช่วยให้สามารถมองเป็นองค์รวมเริ่มจากการอธิบายระบบ แบบจำลองวิเคราะห์ และ แบบจำลองการออกแบบ ดังแสดงในรูปที่ 5.1 โดยแบบจำลองการวิเคราะห์จะทำหน้าที่เป็นตัวเชื่อมกลางระหว่างการอธิบายระบบและแบบจำลองการออกแบบ



รูปที่ 5.1 แบบจำลองการวิเคราะห์ [2]

สำหรับการวิเคราะห์ซอฟต์แวร์นั้นจะเน้นไปที่การใช้แบบจำลองการวิเคราะห์ที่อธิบายระบบและความต้องการตลอดจนมุ่งไปที่การแก้ไขปัญหาให้กับลูกค้า โดยแบบจำลองการวิเคราะห์ควรมีความชัดเจน เข้าใจง่าย และ เป็นไปตามหลักการสร้างแบบจำลองการวิเคราะห์ ดังต่อไปนี้

1. โดเมนของปัญหาต้องถูกนำเสนอและอธิบายให้ชัดเจนและเข้าใจง่าย
2. ทุกฟังก์ชันของซอฟต์แวร์ต้องถูกนิยาม
3. พฤติกรรมของซอฟต์แวร์ต้องถูกนำเสนอในรูปแบบที่เข้าใจง่ายทั้ง ข้อมูล การปฏิสัมพันธ์กับผู้ใช้ และ สภาพแวดล้อม
4. ถ้าแบบจำลองควรถูกแบ่งย่อยลงเพื่อลดความซับซ้อนและสามารถทำความเข้าใจได้ง่าย
5. แบบจำลองควรมุ่งเน้นการอธิบายปัญหาในมุมมองของผู้ใช้ก่อนแล้วจึงลงรายละเอียดในการสร้างซอฟต์แวร์

แบบจำลองการวิเคราะห์มีหลายรูปแบบให้เลือกใช้งาน ฉะนั้นวิศวกรซอฟต์แวร์มีหน้าที่ในการเลือกใช้แบบจำลองการวิเคราะห์ที่เหมาะสมกับระบบและปัญหาที่ต้องการวิเคราะห์ โดยตัวอย่างแบบจำลองต่าง ๆ ประกอบด้วย แบบจำลองเชิงสถานการณ์ แบบจำลองเชิงคลาส แบบจำลองเชิงฟังก์ชัน และ แบบจำลองเชิงพฤติกรรม

แบบจำลอง เชิงสถานการณ์

แบบจำลองเชิงสถานการณ์สร้างขึ้นเพื่ออธิบายสถานการณ์ที่จะเกิดขึ้นในระหว่างการใช้งานระบบซอฟต์แวร์ เพื่อให้เห็นภาพรวมการใช้งานของซอฟต์แวร์อย่างละเอียด รวมถึงสิ่งที่จะเกิดขึ้นในระหว่างการทำงานของซอฟต์แวร์ สามารถใช้ยูเอ็มแอลมาสร้างแบบจำลองเชิงสถานการณ์ ด้วยการใช้แผนภูมิยูสเคส แผนภูมิกิจกรรม และ แผนภูมิเรียงลำดับ

แผนภูมิยูสเคส ถูกนำมาใช้ในการอธิบายระบบในมุมมองของผู้มีส่วนเกี่ยวข้องในการปฏิสัมพันธ์กับระบบ แสดงผ่านทางผู้ใช้ในแต่ละบทบาทที่ต่างกัน โดยในแผนภูมิจะแสดงผู้ที่มีส่วนเกี่ยวข้อง ปฏิสัมพันธ์กับระบบซอฟต์แวร์อย่างไร มีกระบวนการปฏิสัมพันธ์และประมวลผลอะไรบ้าง สำหรับแผนภูมินี้สามารถช่วยให้ทีมพัฒนาเห็นภาพรวมของกระบวนการปฏิสัมพันธ์ระหว่างระบบกับ ผู้ใช้โดยจะเน้นความต้องการเชิงฟังชันและพฤติกรรมของระบบ แต่อย่างไรก็ตามวิศวกรซอฟต์แวร์สามารถเขียนอธิบายแบบจำลองเชิงสถานการณ์ด้วยภาษาธรรมชาติอธิบายเป็นข้อ ๆ เพื่อให้เห็นภาพรวมของระบบซอฟต์แวร์ประกอบกับแผนภูมิยูสเคส ดังตัวอย่างการเขียนอธิบายแบบจำลองเชิงสถานการณ์ ดังต่อไปนี้

ระบบกลอนประตูดิจิทัล

ยูสเคส การปลดกลอนประตูดิจิทัล

ผู้กระทำ ผู้ใช้งานหรือเจ้าของ

วัตถุประสงค์ ปฏิสัมพันธ์กับระบบกลอนดิจิทัล เพื่อปลดกลอนประตู

เงื่อนไขก่อนหน้า กลอนประตูถูกตั้งรหัสปลดกลอนไว้ล่วงหน้าและอยู่ในสถานะพร้อมใช้งาน

สิ่งกระตุ้น การกดปุ่มบนกลอนดิจิทัลเพื่อปลุกระบบเพื่อรับรหัสปลดกลอนจากผู้ใช้

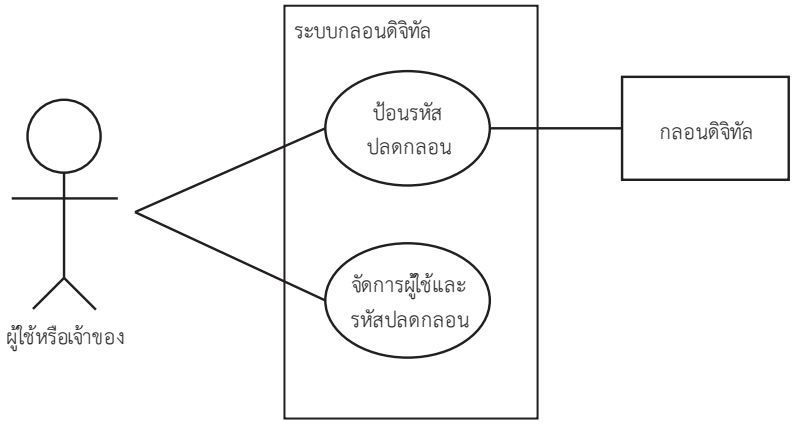
สถานการณ์

1. ผู้ใช้กดปุ่มใดๆบนแป้นกลอนประตูดิจิทัล
2. ผู้ใช้กดปุ่มเพื่อป้อนรหัสปลดกลอนประตูดิจิทัล
3. ระบบเปรียบเทียบรหัสที่ผู้ใช้ป้อนเทียบกับรหัสปลดกลอน
4. ปลดกลอนเมื่อรหัสตรงหรือแจ้งให้ป้อนรหัสใหม่หากรหัสไม่ตรง

ความถี่ในการใช้งาน ทุกครั้งที่ต้องการปลดกลอนดิจิทัล

ปัญหา

1. แหล่งพลังงานระบบไม่สามารถรองรับการทำงานของระบบ เช่นพลังงานหมด
2. การรั่วไหลของรหัสอาจถูกผู้ไม่หวังดีขโมยรหัสปลดกลอน



รูปที่ 5.2 แผนภูมิยูสเคสกลอนดิจิทัล

จากตัวอย่างข้างต้น จะเห็นได้ว่าแบบจำลองเชิงสถานการณ์สามารถนำเสนอออกมาในรูปแบบของสถานการณ์ต่าง ๆ ที่จะเกิดขึ้นในระบบ โดยเน้นการนำเสนอในรูปแบบของการปฏิสัมพันธ์และพฤติกรรมของระบบกับผู้ใช้ แผนภาพยูสเคสสามารถใช้อธิบายสถานการณ์และความสัมพันธ์ระหว่างผู้ใช้กับระบบได้เป็นอย่างดี ช่วยให้ทั้งผู้ใช้และทีมผู้พัฒนาสามารถเข้าใจง่ายและเห็นภาพตรงกัน นอกจากนั้นอาจจะใช้แผนภูมิกิจกรรมและแผนภูมิเรียงลำดับเพื่ออธิบายแบบจำลองให้ชัดเจนมากยิ่งขึ้น

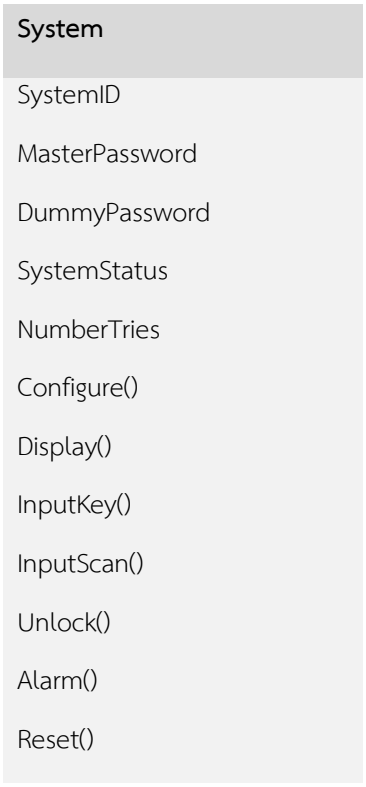
แบบจำลอง เชิงคลาส

แบบจำลองเชิงคลาส ใช้หลักการโปรแกรมเชิงวัตถุในการวิเคราะห์และนำเสนอในรูปแบบจำลองเชิงคลาส วิศวกรซอฟต์แวร์และนักวิเคราะห์ระบบจะเริ่มทำความเข้าใจปัญหา ความต้องการ และ ระบบเพื่อนำมาระบุคลาสและวัตถุในการสร้างแบบจำลอง การระบุคลาสส่วนมากจะสกัดออกมาจากระบบ ในส่วนของกระบวนการและสถานการณ์เช่น เอนทิตีภายนอก วัตถุ เหตุการณ์ บทบาท การจัดการองค์กร สถานที่ โครงสร้าง และ อื่น ๆ ที่เกี่ยวข้องในระบบสามารถนำมาสกัดและวิเคราะห์เชิงคลาสเพื่อสร้างแบบจำลองให้สมบูรณ์ ดังแสดงในตัวอย่างระบบกลอนดิจิทัล แบบจำลองเชิงคลาสสามารถส่งต่อไปยังกระบวนการออกแบบเพื่อสร้างเป็นแบบจำลองการออกแบบและพัฒนาต่อด้วยกระบวนการโปรแกรมเชิงวัตถุ

Potential Class	General Classification
Owner	Role or external entity
User	Role or external entity
Lock	External entity
Keypad	External entity
Finger scan sensor	External entity
Display	External entity
Configuration	Event
System	Thing
Number, fingerprint	Not object, attribute of sensor
Master password	Thing
User password	Thing
Sensor event	Event
Unlock event	Event
Alarm	External entity

ตัวอย่างการสกัดคลาสและวัตถุจากระบบและความต้องการของระบบกลอนดิจิทัล จะเห็นได้ว่าส่วนประกอบต่าง ๆ จะถูกนำมาวิเคราะห์และระบุออกมาเป็นคลาส วัตถุ และ ลักษณะประจำ (attribute) สำหรับสร้างแบบจำลองเชิงคลาส โดยที่นักวิเคราะห์จะทำการเลือกตัดหรือจัดกลุ่มคลาสพร้อมการระบุความสัมพันธ์เพื่อสร้างเป็นแบบจำลองเชิงคลาสที่สมบูรณ์ แต่ละคลาสถูกนิยามลักษณะและการดำเนินการให้สอดคล้องกับระบบและความต้องการ ดังตัวอย่างแผนภูมิคลาสระบบกลอนดิจิทัลในรูปที่ 5.3

แบบจำลองเชิงคลาสส่วนมากจะใช้แบบจำลองยูเอ็มแอลคลาส เพื่อใช้ในการนิยามและอธิบายเนื่องจากเป็นแบบจำลองมาตรฐานที่นิยมใช้กันอย่างสากลและรองรับการทำงานกับเครื่องมือทางวิศวกรรมซอฟต์แวร์จำนวนมาก นักพัฒนาสามารถเข้าใจได้ตรงกันสื่อสารกันได้ดีเนื่องจากนิยามและสัญลักษณ์ต่างๆถูกกำหนดมาเป็นมาตรฐาน

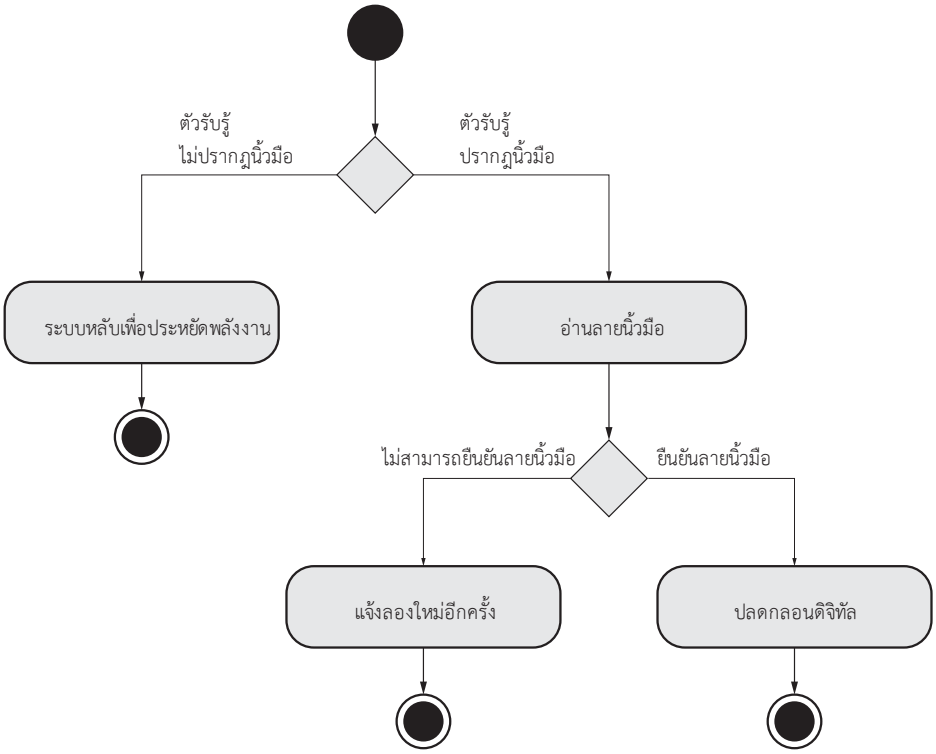


รูปที่ 5.3 แผนภูมิคลาส system ของระบบกลอนดิจิทัล

แบบจำลอง เชิงฟังก์ชัน

แบบจำลองเชิงฟังก์ชันสามารถมองได้ สอง รูปแบบจาก สอง มุมมองเพื่ออธิบายความต้องการและการทำงานของระบบ กระบวนการทำงานและเอาพุท มุมมองแรกเป็นมุมมองของผู้ใช้ในแง่การปฏิสัมพันธ์กับระบบเพื่ออธิบายการทำงานระหว่างผู้ใช้กับระบบและบริการที่จะได้รับ ยกตัวอย่างเช่นซอฟต์แวร์จองบัตรเข้าชมภาพยนตร์บนอุปกรณ์เคลื่อนที่ในมุมมองของผู้ใช้คือฟังก์ชันการเลือกรอบภาพยนต์และระบุที่นั่งชมโดยที่ผลลัพธ์คือบัตรเข้าชมภาพยนต์ดิจิทัล ในส่วนของมุมมองที่สองเป็นมุมมองที่ใช้อธิบายกระบวนการและปฏิสัมพันธ์ภายในระหว่างคลาสเพื่อตอบสนองฟังก์ชันการทำงานที่ผู้ใช้ต้องการจากระบบ หรืออาจกล่าวได้ว่าเป็นมุมมองพื้นฐานระดับล่างเพื่อใช้วิเคราะห์และอธิบายฟังก์ชันการทำงานของระบบ เช่นฟังก์ชันการตรวจสอบจำนวนและตำแหน่งที่นั่งชมภาพยนต์ที่คงเหลือในแต่ละรอบและฟังก์ชันการทำธุรกรรมการเงินเพื่อรับชำระและออกบัตรเข้าชมภาพยนต์ดิจิทัล

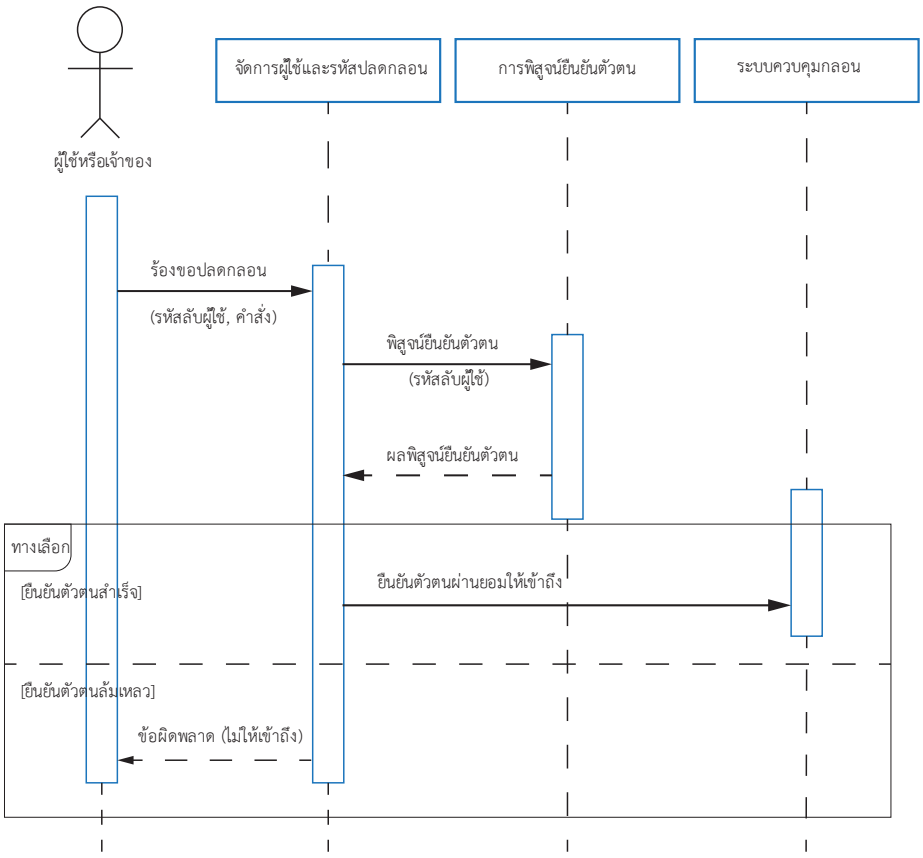
แบบจำลองเชิงฟังก์ชันสามารถใช้แผนภูมิยูเอ็มแอล ในการอธิบายฟังก์ชันการทำงานได้ทั้งสองมุมมอง จากหลายระดับเพื่อฉายภาพของแบบจำลองออกมาให้ชัดเจนและเข้าใจได้ง่าย เช่นการใช้แผนภาพกิจกรรม (activity diagram) ในการแสดงภาพกระบวนการทำงานและการไหลของการปฏิสัมพันธ์ในแต่ละสถานการณ์ ซึ่งการเขียนอธิบายแบบจำลองด้วยแผนภาพกิจกรรมนั้นจะคล้ายกับการเขียนผังงาน (flowchart) โดยใช้สัญลักษณ์สี่เหลี่ยมมนในการระบุฟังก์ชันและใช้ลูกศรแสดงการไหล รูปสี่เหลี่ยมขนมเปียกปูนแทนการเลือกตัดสินใจ ดังตัวอย่างรูปที่ 5.4 แสดงแผนภาพกิจกรรมการปลดกลอนดิจิทัลด้วยการตรวจสอบลายนิ้วมือ



รูปที่ 5.4 แผนภาพกิจกรรมการปลดรหัสกลอนดิจิทัล

จากรูปแผนภาพกิจกกรมแสดงให้เห็นการทำงานของระบบปลดกลอนดิจิทัล เป็นขั้นตอนตามผังงาน โดยเริ่มจากระบบจะทำการตรวจสอบการมีอยู่ของนิ้วมือผู้ใช้ หากไม่พบนิ้วมือระบบจะหลับไปสักครู่เพื่อประหยัดพลังงาน แต่พบการมีอยู่ที่ตัวรับรู้ระบบจะสั่งให้ดำเนินการอ่านค่าลายนิ้วมือเข้ามาเพื่อระบุตัวตนของผู้ใช้ ในกรณีที่ไม่สามารถระบุยืนยันลายนิ้วมือได้ ระบบจะแจ้งให้ผู้ใช้ลองแสกนอีกครั้ง ถ้าการยืนยันตัวตนสำเร็จระบบจะปลดกลอนดิจิทัล

แบบจำลองเชิงฟังก์ชันยังสามารถใช้ ยูเอ็มแอลแผนภาพลำดับ (UML sequence diagram) เพื่ออธิบายพฤติกรรมของแบบจำลอง โดยแสดงลำดับเหตุการณ์ต่าง ๆ ที่เกิดขึ้น เชื่อมความสัมพันธ์ และ การปฏิสัมพันธ์จากวัตถุไปยังอีกวัตถุ แผนภาพสามารถแสดงทั้งการไหลและลำดับเวลาโดยการเริ่มลำดับเหตุการณ์จากบนลงล่าง และใช้ลูกศรแสดงการไหล ดังรูปที่ 5.5



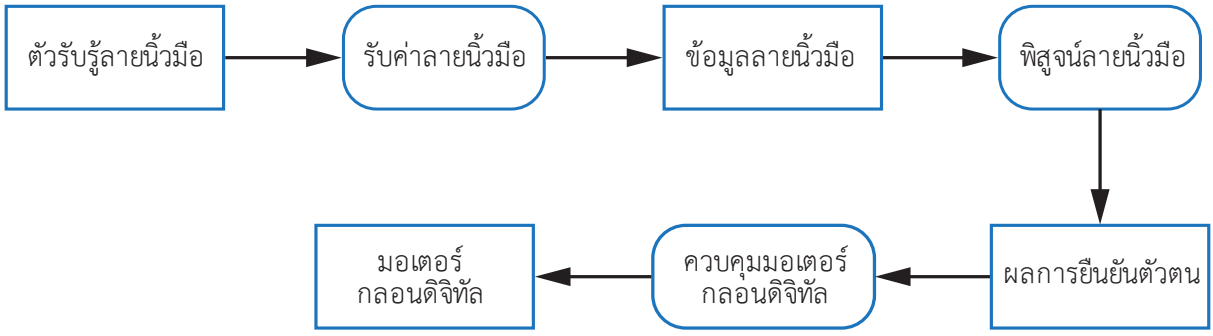
รูปที่ 5.5 แผนภาพลำดับ ระบบปลดกลอนดิจิทัล

แบบจำลอง เชิงพฤติกรรม

แบบจำลองเชิงพฤติกรรมแสดงให้เห็นพฤติกรรมการทำงานของระบบว่าเกิดเหตุการณ์อะไรขึ้นบ้างและระบบตอบสนองอย่างไรกับเหตุการณ์หรือสิ่งเร้า โดยสามารถแบ่งสิ่งเร้าที่กระตุ้นระบบออกเป็น สอง รูปแบบ คือ สิ่งเร้าจากข้อมูลและสิ่งเร้าจากเหตุการณ์ ดังนั้นแบบจำลองเชิงพฤติกรรมจึงแบ่งออกเป็น สอง กลุ่มตามสิ่งเร้าคือแบบจำลองขับเคลื่อนด้วยข้อมูล (data-driven model) และแบบจำลองขับเคลื่อนด้วยเหตุการณ์ (event-driven model)

แบบจำลองขับเคลื่อนด้วยข้อมูล

แบบจำลองขับเคลื่อนด้วยข้อมูลเหมาะสำหรับแสดงระบบที่เน้นการประมวลผลข้อมูลเป็นหลัก โดยระบบจะใช้ข้อมูลเป็นสำหรับป้อนเข้าระบบเพื่อประมวลผลตามกระบวนการโดยส่วนใหญ่จะไม่ขึ้นกับเหตุการณ์ต่าง ๆ กระบวนการประมวลผลข้อมูลมักจะเป็นแบบอนุกรมเรียงลำดับเป็นสายจนเสร็จได้ผลลัพธ์ที่เป็นงานออกมา เช่น ระบบประมวลผลค่าใช้ค่าบริการโทรศัพท์เคลื่อนที่โดยที่ระบบจะรับข้อมูลค่าใช้ง่ายของลูกค้าทุก ๆ รายการมาประมวลผลคิดค่าใช้จ่ายรายเดือนแล้วทำการสรุปยอดเพื่อออกใบแจ้งหนี้เรียกเก็บค่าบริการ ซึ่งลักษณะการทำงานจะเป็นขั้นตอน แบบจำลองการไหลข้อมูลจึงเหมาะสำหรับการอธิบายระบบในลักษณะนี้ แสดงด้วยแผนภาพการไหลข้อมูล (data-flow diagrams: DFD) ด้วยการใช้แผนภาพกิจกรรมจากยูเอ็มแอลดังตัวอย่างรูปที่ 5.6



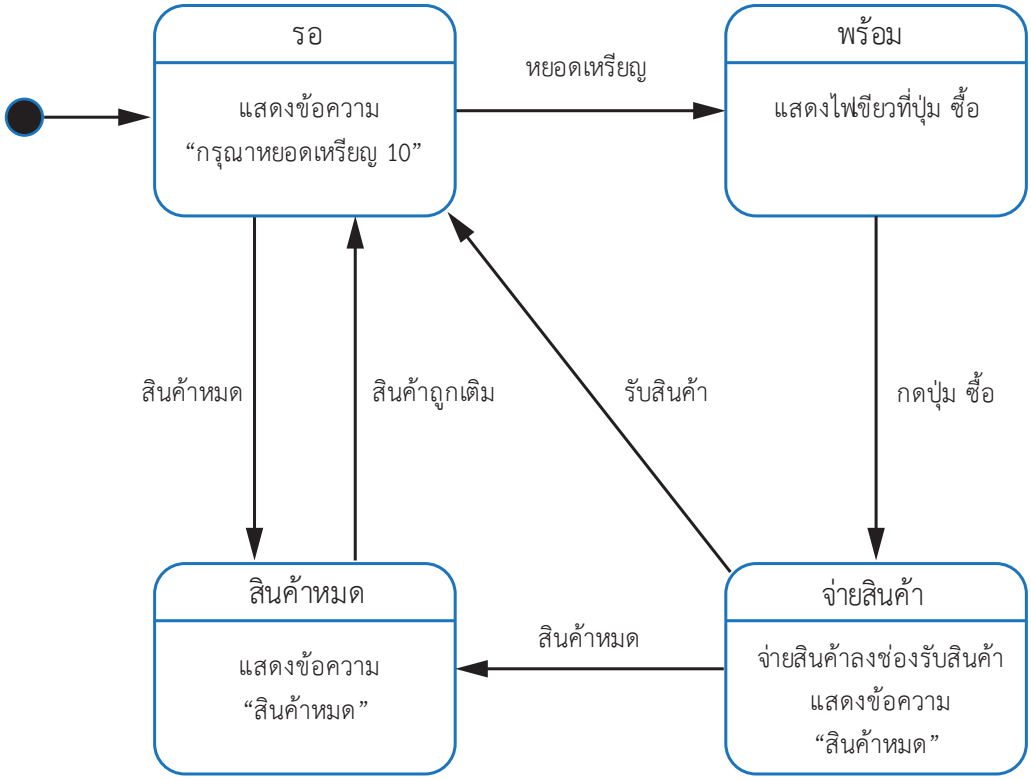
รูปที่ 5.6 แผนภาพกิจกรรมแสดงการไหลข้อมูลระบบปลดกลอนดิจิทัลด้วยลายนิ้วมือ

การนำเสนอแบบจำลองขับเคลื่อนด้วยข้อมูลด้วยแผนภาพกิจกรรมดังรูปที่ 5.6 ช่วยให้ทีมเห็นภาพกระบวนการทำงานในระบบได้อย่างชัดเจนเข้าใจลำดับการประมวลผลข้อมูลผ่านส่วนประกอบต่าง ๆ แบบจำลองขับเคลื่อนด้วยข้อมูลยังสามารถใช้แผนภาพลำดับ เพื่อแสดงลำดับและการปฏิสัมพันธ์ที่ชัดเจนหรืออาจใช้ทั้งสองแผนภาพเพื่อนำเสนอแบบจำลองในทั้งสองมุมมอง

แบบจำลองขับเคลื่อนด้วยเหตุการณ์

แบบจำลองขับเคลื่อนด้วยเหตุการณ์เหมาะสำหรับการนำเสนอระบบในรูปแบบการตอบสนองต่อเหตุการณ์ทั้งภายในและภายนอก โดยระบบจะถูกจำลองในรูปของสถานะและเหตุการณ์ต่าง ๆ ที่สามารถเกิดขึ้นได้ โดยเมื่อมีการเปลี่ยนแปลงเหตุการณ์จะส่งผลให้สถานะเปลี่ยน เช่น เปลี่ยนสถานะล็อกกลอนหรือปลดกลอนโดยขึ้นอยู่กับเหตุการณ์ที่จะส่งคำสั่งมาควบคุม ส่วนมากแบบจำลองขับเคลื่อนด้วยเหตุการณ์จะนิยมใช้ในระบบเวลาจริงและระบบฝังตัว

แผนภาพสถานะ (state diagram) มักถูกนำมาใช้นำเสนอแบบจำลองขับเคลื่อนด้วยเหตุการณ์ โดยประกอบด้วยสถานะและเหตุการณ์ที่มีการแสดงการเปลี่ยนแปลงสถานะจากสถานะใดไปยังอีกสถานะตามเหตุการณ์ที่กระตุ้น และอาจจะเพิ่มรายละเอียดในแต่ละสถานะ เช่น การประมวลผลได้ ดังตัวอย่างรูปที่ 5.7 แสดงแผนภาพสถานะเครื่องขายกาแฟกระป๋องอัตโนมัติ



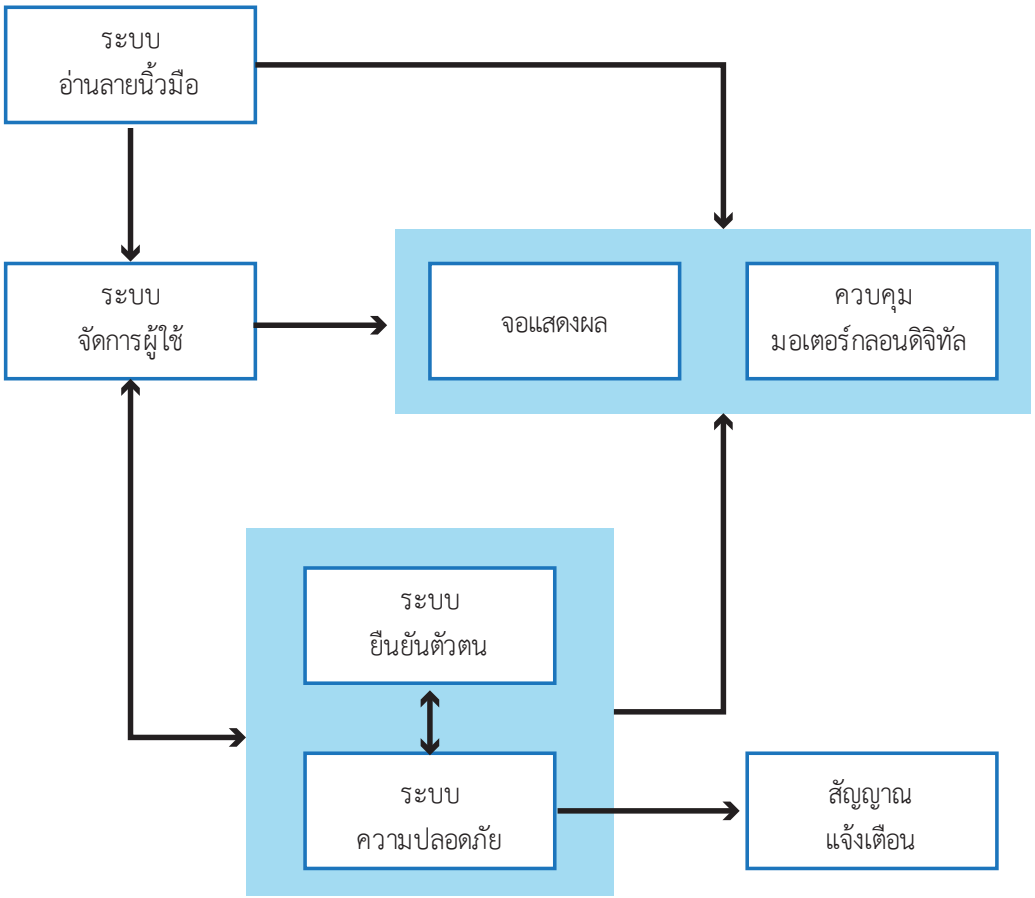
รูปที่ 5.7แสดงแผนภาพสถานะเครื่องขายกาแฟกระป๋องอัตโนมัติ

จากรูปแผนภาพสถานะเครื่องขายกาแฟกระป๋องอัตโนมัติจะเห็นว่าเมื่อระบบเริ่มการทำงานจะไปอยู่ที่สถานะรอเพื่อรอลูกค้ามาปฏิสัมพันธ์ในเหตุการณ์หยอดเหรียญ 10 บาทเพื่อเปลี่ยนไปสถานะพร้อมเพื่อทำการรอการยืนยันการซื้อเมื่อลูกค้ากดปุ่มซื้อเพื่อเปลี่ยนไปสถานะจ่ายสินค้า จะเห็นว่าสถานะจะถูกเปลี่ยนเมื่อมีเหตุการณ์มาเป็นสิ่งเร้า และในแต่ละสถานะสามารถเพิ่มรายละเอียดการทำงานลงไปได้เช่นที่สถานะจ่ายสินค้าจะดำเนินการแสดงผลข้อความ “กรุณารับสินค้า” และจ่ายสินค้าออกไปที่ช่องรับสินค้า จากตัวอย่างนี้เป็นเพียงการแสดงให้เห็นถึงแบบจำลองอย่างง่ายการทำงานจริงของเครื่องขายอัตโนมัติจะมีความซับซ้อนมากกว่านี้ตามฟังก์ชันการทำงาน ดังนั้นอาจมีการใช้แผนภาพอื่นหรือตารางอธิบายการทำงานเพื่อช่วยขยายรายละเอียดของแบบจำลองให้สมบูรณ์ขึ้นได้

วัตถุประสงค์

- *เข้าใจความสำคัญของการออกแบบสถาปัตยกรรม*
- *สามารถการเลือกสถาปัตยกรรมที่เหมาะสมกับการพัฒนาซอฟต์แวร์*

สถาปัตยกรรมซอฟต์แวร์ใช้อธิบายภาพรวมและโครงสร้างของระบบ โดยการออกแบบสถาปัตยกรรมถือว่าเป็นกิจกรรมต้น ๆ เพื่อทำความเข้าใจและออกแบบระบบ ที่เน้นโครงสร้างและภาพรวมโดยการแสดงส่วนประกอบหลักและความสัมพันธ์ภายในระบบ รูปที่ 6.1 แสดงตัวอย่างสถาปัตยกรรมระบบปลดกลอนดิจิทัลเพื่อแสดงให้เห็นตัวอย่างการออกแบบสถาปัตยกรรมส่วนประกอบและความสัมพันธ์ต่าง ๆ ซึ่งกระบวนการออกแบบส่วนใหญ่จะทำควบคู่กับกระบวนการวิศวกรรมความต้องการ



รูปที่ 6.1 สถาปัตยกรรมระบบกลอนดิจิทัล

สำหรับสถาปัตยกรรมซอฟต์แวร์ระบบอาจแบ่งออกได้สองระดับคือสถาปัตยกรรมขนาดเล็กและสถาปัตยกรรมขนาดใหญ่ โดยสถาปัตยกรรมขนาดเล็กจะใช้แสดงซอฟต์แวร์ในระดับโปรแกรมเดี่ยว ๆ และแบ่งย่อยออกมาแสดงส่วนประกอบภายใน ดังตัวอย่างรูปที่ 6. ในส่วนของสถาปัตยกรรมขนาดใหญ่จะมองในระบบที่ใหญ่และซับซ้อน เช่น ในองค์กรใหญ่

สถาปัตยกรรมซอฟต์แวร์นับว่ามีความสำคัญมากในการพัฒนาระบบ โดยมักจะส่งผลกระทบด้านประสิทธิภาพ ความเสถียร ความน่าเชื่อถือ และ การบำรุงรักษา วิศวกรซอฟต์แวร์จึงควรระมัดระวังในการออกแบบและตัดสินใจเลือกสถาปัตยกรรมให้เหมาะสมกับระบบซอฟต์แวร์

การออกแบบสถาปัตยกรรม

สำหรับการออกแบบสถาปัตยกรรมเป็นกระบวนการที่ใช้ความคิดสร้างสรรค์ในการออกแบบระบบเพื่อแก้ไขปัญหาให้สามารถตอบสนองความต้องการของลูกค้าอย่างมีประสิทธิภาพและคุ้มค่าตามหลักวิศวกรรมซอฟต์แวร์ โดยสถาปัตยกรรมนั้นต้องเป็นไปตามความต้องการทั้งแบบเชิงฟังก์ชันและไม่เชิงฟังก์ชัน การออกแบบนั้นไม่ได้มีรูปแบบตายตัวไม่มีสูตรสำเร็จ การออกแบบจะต้องพิจารณาปัจจัยหลาย ๆ ด้านประกอบทั้งระบบที่กำลังจะพัฒนา ประสบการณ์ผู้ออกแบบ ความต้องการและข้อจำกัดของระบบ จึงไม่มีรูปแบบตายตัวในการออกแบบแต่จะออกมาในรูปแบบกระบวนการตัดสินใจในการออกแบบส่วนต่าง ๆ เสียมากกว่า

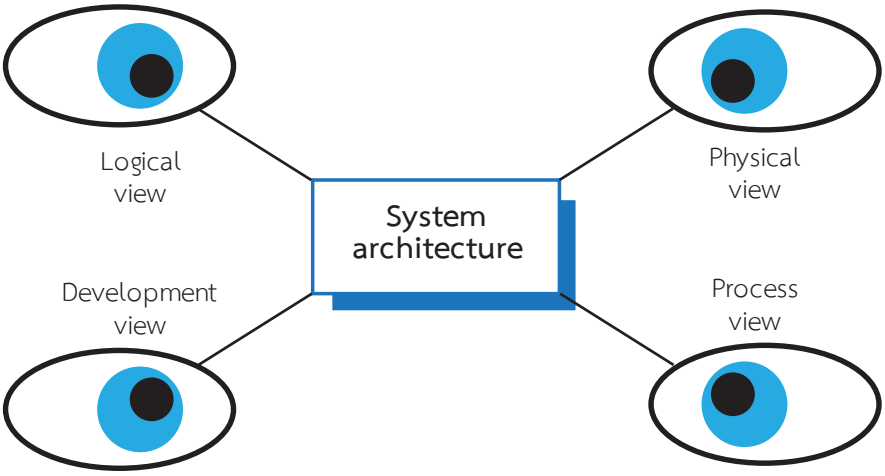
ถึงแม้ว่าระบบและปัญหาต่าง ๆ จะมีความเป็นเอกลักษณ์หรือเฉพาะเจาะจง แต่ในภาพรวมนั้นซอฟต์แวร์ต่าง ๆ อาจมีความใกล้เคียงในกลุ่มเดียวกันหรือในสภาพแวดล้อมที่คล้ายคลึงกัน มักจะมีสถาปัตยกรรมซอฟต์แวร์ที่มีแกนใกล้เคียงกัน เช่น ระบบขนส่งสินค้าของผู้ให้บริการรายต่าง ๆ ส่วนใหญ่จะมีสถาปัตยกรรมร่วมที่คล้ายกันเพราะอยู่ในโดเมนเดียวกัน โดยรูปแบบสถาปัตยกรรมสามารถจัดกลุ่มตามรูปแบบเรียกว่าแบบแผนสถาปัตยกรรม (architectural patterns) ที่ใช้ในการอธิบายระบบในแต่ละลักษณะโดเมนหรือตามรูปแบบโครงสร้าง เช่น สถาปัตยกรรม client-server สถาปัตยกรรม layered และ สถาปัตยกรรม repository เป็นต้น การเลือกใช้แบบแผนสถาปัตยกรรมนั้นอาจพิจารณาประกอบกับความต้องการไม่เชิงฟังก์ชันร่วมด้วยเพื่อให้สามารถตอบสนองความต้องการได้อย่างเหมาะสม ดังเช่น

1. ***ด้านประสิทธิภาพ*** การออกแบบสถาปัตยกรรมควรมุ่งเน้นที่การจัดการด้านประสิทธิภาพ ออกแบบเป็นส่วนย่อยๆเพื่อการประมวลผลภายในเครื่องมากกว่าการกระจายออกไปประมวลผลในเครือข่าย ลดการสื่อสารระหว่างส่วนประกอบโดยยึดการออกแบบเพื่อประสิทธิภาพ
2. ***ความน่าเชื่อถือ*** อาจจะเลือกสถาปัตยกรรมที่มีความน่าเชื่อถือสูงลดความผิดพลาดหรือล้มเหลวในการดำเนินการ เช่น การออกแบบด้วย ส่วนประกอบซ้ำซ้อนเพื่อเพิ่มความทนทานต่อการผิดพลาด หรือการทำงานคู่ขนาน การปรับปรุงส่วนประกอบได้ทันทีโดยไม่กระทบกับการทำงาน เป็นต้น
3. ***ความปลอดภัย*** ถ้าความปลอดภัยมาก่อนอาจเลือกใช้ สถาปัตยกรรม layered เพื่อการปกป้องเป็นชั้น ๆ ห่อหุ้มส่วนสำคัญไว้ภายใน

จากตัวอย่างจะเห็นได้ว่าการเลือกสถาปัตยกรรมมาใช้ในการออกแบบเพื่อตอบสนองความต้องการไม่เชิงฟังก์ชันนั้นอาจมาความขัดแย้งในตัวจึงต้องทำการตัดสินใจเลือกโดยดูจากลำดับความสำคัญของความต้องการเพื่อปรับการออกแบบสถาปัตยกรรมให้เหมาะสมกับระบบที่จะพัฒนา

มุมมองสถาปัตยกรรม

เนื่องจากสถาปัตยกรรมซอฟต์แวร์นั้นมีความเป็นนามธรรมสูงการจะอธิบายให้เห็นภาพและเข้าใจตรงกันนั้นจะเป็นต้องหาวิธีในการสื่อสารที่เหมาะสมเพื่ออธิบายสถาปัตยกรรมระบบ การใช้แผนภาพเป็นอีกวิธีที่มีประสิทธิภาพในการอธิบายแต่อาจจะไม่สามารถใช้เพื่ออธิบายให้ครอบคลุมได้ จึงควรใช้การอธิบายจากหลากหลายมุมมองด้วยแผนภาพที่แตกต่างกัน ดังรูปที่ 6.2



รูปที่ 6.2 มุมมองสถาปัตยกรรม ดัดแปลงจาก [11]

1. มุมมอง logical เป็นการมองส่วนประกอบหลัก ๆ ในระบบผ่านมุมมองเชิงวัตถุหรือคลาส
2. มุมมอง process แสดงให้เห็นกระบวนการทำงานและการปฏิสัมพันธ์ต่าง ๆ ในระบบ ซึ่งมุมมองนี้สามารถสะท้อนด้านประสิทธิภาพและความน่าเชื่อถือของระบบได้
3. มุมมอง development เป็นมุมมองที่แสดงส่วนประกอบต่าง ๆ ในมุมมองของการพัฒนาซอฟต์แวร์การแบ่งย่อยเป็นชิ้นส่วนต่าง ๆ เพื่อนำไปพัฒนา
4. มุมมอง physical แสดงให้เห็นส่วนประกอบในระบบด้านฮาร์ดแวร์และส่วนประกอบต่าง ๆ ที่กระจายอยู่ในระบบ

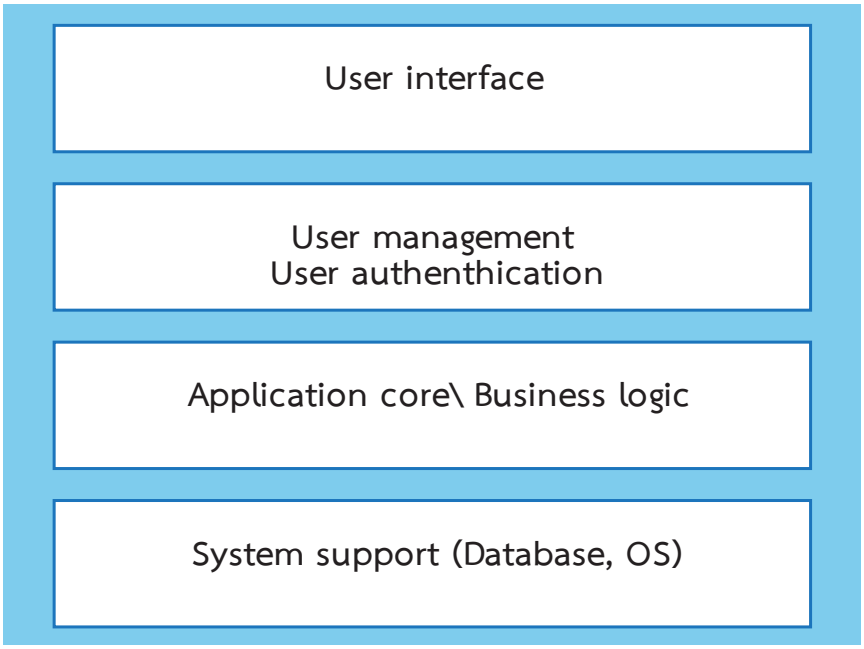
การเขียนแผนภาพเพื่ออธิบายสถาปัตยกรรมในมุมมองต่าง ๆ สามารถเขียนแบบไม่เป็นทางการได้ โดยใช้สัญลักษณ์อย่างง่ายในการเขียนเพื่ออธิบายระบบในมุมมองต่าง ๆ ไม่มีมาตรฐานในการเขียน แต่วิศวกรซอฟต์แวร์อาจนำยูเอ็มแอลมาใช้ก็ย่อมได้ หรือจะออกแบบรูปแบบการนำเสนอเองในมุมมองที่เข้าใจง่าย ปรับแก้ได้ง่าย ที่สำคัญคือทำให้ทีมเห็นและเข้าใจตรงกัน

แบบแผนสถาปัตยกรรม (architectural pattern)

แบบแผนสถาปัตยกรรมเปรียบเสมือนตำราพิชัยสงครามที่บันทึกกลยุทธ์แบบแผนการทำศึกสงครามในสถานการณ์ต่าง ๆ เพื่อให้ชนรุ่นหลังนำไปศึกษาเพื่อนำไปประยุกต์ใช้ แบบแผนสถาปัตยกรรมก็ได้มาจากการออกแบบเพื่อแก้ไขปัญหาในโดเมนต่าง ๆ ซ้ำ ๆ จนเกิดเป็นรูปแบบที่เหมาะสมในการแก้ปัญหาในด้านนั้น ๆ ส่งต่อเป็นองค์ความรู้เพื่อนำกลับมาใช้ใหม่ โดยในหัวข้อนี้จะยกตัวอย่างแบบแผนสถาปัตยกรรมพร้อมคำอธิบายประกอบ

สถาปัตยกรรมแบบชั้น (layered)

สถาปัตยกรรมแบบชั้น จะแบ่งส่วนประกอบต่าง ๆ จัดเรียงและจัดการแบ่งเป็นชั้น ๆ โดยในแต่ละชั้นก็จะมีการนิยามคุณสมบัติและการเข้าถึงบริการต่าง ๆ ที่อยู่ในชั้นล่างถัดลงไปดังรูปที่ 6.2 ในแต่ละชั้นจะประกอบด้วยส่วนประกอบจัดเรียงเป็นชั้น ๆ สามารถเพิ่มปรับเปลี่ยนแก้ไขส่วนต่าง ๆ ในชั้นได้โดยไม่กระทบชั้นอื่น ๆ เช่นถ้าต้องการเพิ่มหรือปรับปรุงชั้นส่วนติดต่อผู้ใช้ ก็สามารถปรับได้โดยไม่กระทบส่วนอื่น ๆ หรือจะเพิ่มให้รองรับระบบปฏิบัติการอื่นก็เพียงแก้ที่ชั้นล่างสุด (system support)

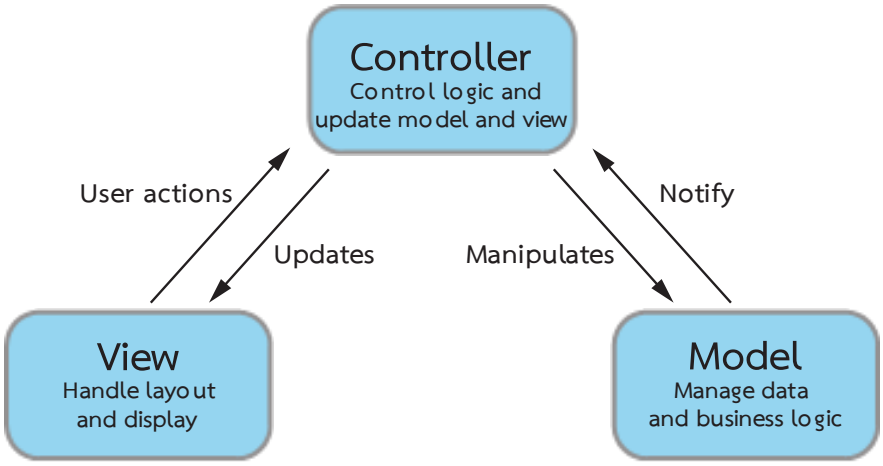


รูปที่ 6.2 สถาปัตยกรรมแบบชั้น

สถาปัตยกรรมแบบชั้นมีการจัดเรียงเป็นชั้น ๆ โดยชั้นล่างสนับสนุนบริการให้แก่ชั้นที่อยู่สูงขึ้นไป ยกตัวอย่างระบบปลดกลอนดิจิทัล ชั้นล่างสุดเก็บข้อมูลผู้ใช้รหัสลับลายนิ้วมือ เพื่อบริการให้กับชั้นพิสูจน์ยืนยันตัวตน โดยข้อดีของสถาปัตยกรรมแบบชั้นความปลอดภัยเพราะมีการทำงานเป็นชั้น ๆ เพื่อปกป้องข้อมูลชั้นล่าง ๆ และความคล่องตัวในการปรับเปลี่ยนซอฟต์แวร์เช่นต้องการปรับเปลี่ยนส่วนประสานผู้ใช้ (UI) จากหน้าจอปกติเป็นหน้าจอสัมผัสก็เพียงปรับเปลี่ยนที่ชั้นบนสุด แต่อย่างไรก็ตามการแบ่งชั้นอย่างเด็ดขาดนั้นทำได้ค่อนข้างยากและอาจจะมีปัญหาด้านประสิทธิภาพถ้าต้องมีการเรียกบริการต่อ ๆ กันหลายลำดับชั้น

สถาปัตยกรรมแบบโมเดลวิวคอนโทรลเลอร์ (MVC)

สถาปัตยกรรมแบบเอ็มวีซีเป็นสถาปัตยกรรมที่แยกโครงสร้างซอฟต์แวร์ออกจากกันเป็นสามส่วนคือโมเดล วิว และคอนโทรลเลอร์ สำหรับแบ่งซอฟต์แวร์เป็นส่วนการนำเสนอ การปฏิสัมพันธ์ และข้อมูล โดยโครงสร้างหลักทั้ง สาม ทำงานประสานกันดังแสดงในรูปที่ 6.3 โมเดลเป็นส่วนประกอบหลักในการจัดการตรรกะทางธุรกิจ (business logic) และดำเนินการจัดการกับข้อมูล วิวเป็นส่วนที่นำเสนอและติดต่อกับผู้ใช้ ในส่วนของคอนโทรลเลอร์ทำหน้าที่จัดการการปฏิสัมพันธ์กับผู้ใช้และเป็นตัวกลางในการผสานการทำงานระหว่างโมเดลและวิว

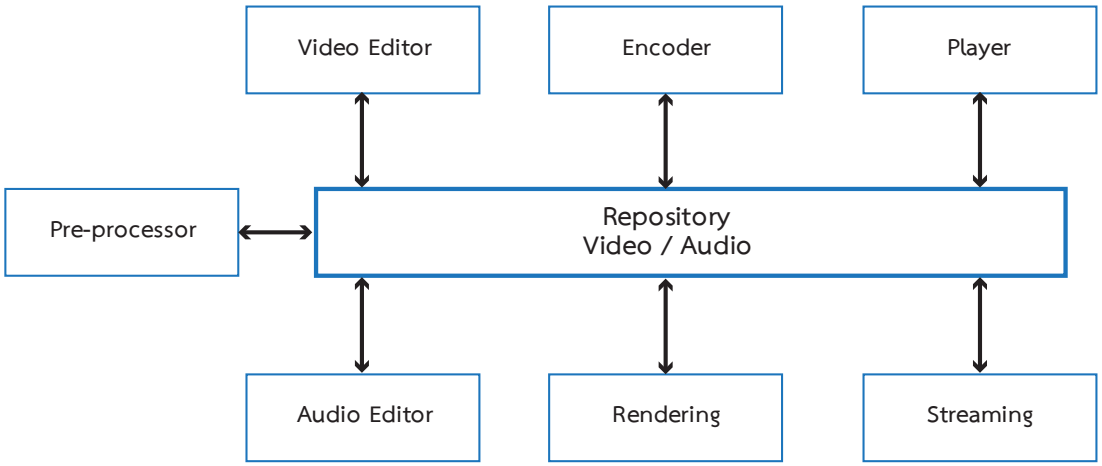


รูปที่ 6.3 สถาปัตยกรรม MVC

จะเห็นได้อย่างชัดเจนว่าสถาปัตยกรรม MVC นั้นแยกส่วนประกอบอย่างชัดเจนเพื่อแยกการนำเสนอกับตรรกะทางธุรกิจอย่างชัดเจนเพื่อให้รองรับการพัฒนาซอฟต์แวร์ในรูปแบบปัจจุบันที่มี การนำเสนอหลากหลายรูปแบบผ่านวิว เช่นซอฟต์แวร์บนอุปกรณ์เคลื่อนที่หากเราใช้สถาปัตยกรรม MVC จะทำให้การใช้พัฒนาสะดวกขึ้นในกรณีการรองรับหลายแพลตฟอร์มเช่นการพัฒนาให้รองรับทั้งแท็ปเลตหน้าจอขนาดใหญ่และสมาร์ทโฟน ซึ่งมีความแตกต่างกันที่ข้อจำกัดโดยเฉพาะเรื่องพื้นที่หน้าจอ แต่สถาปัตยกรรม MVC เพียงแค่เพิ่มวิวที่เหมาะสมสำหรับแต่ละขนาดหน้าจอ และให้คอนโทรลเลอร์เลือกที่เหมาะสมในการใช้งานโดยไม่ต้องยุ่งเกี่ยวกับโมเดลเลย เป็นการแยกส่วนที่สนับสนุนการนำซอฟต์แวร์กลับมาใช้ใหม่เป็นอย่างดี เช่นเดียวกับการใช้งานเว็บเทคโนโลยีที่นิยมใช้สถาปัตยกรรมแบบ MVC เพื่อรองรับการใช้งานจาก client (web browser) ที่มีความหลากหลาย

สถาปัตยกรรมแบบคลัง (repository)

สถาปัตยกรรมแบบคลังออกแบบมาเพื่อรองรับการเก็บรวบรวมข้อมูลไว้ที่ส่วนกลางเพื่อเป็นคลังในการให้บริการข้อมูลแก่ส่วนประกอบต่าง ๆ ของระบบโดยลักษณะโครงสร้างของสถาปัตยกรรมนั้นจะมีคลังข้อมูลเป็นศูนย์กลางเชื่อมต่อกับส่วนประกอบต่าง ๆ ตัวอย่างดังรูปที่ 6.4

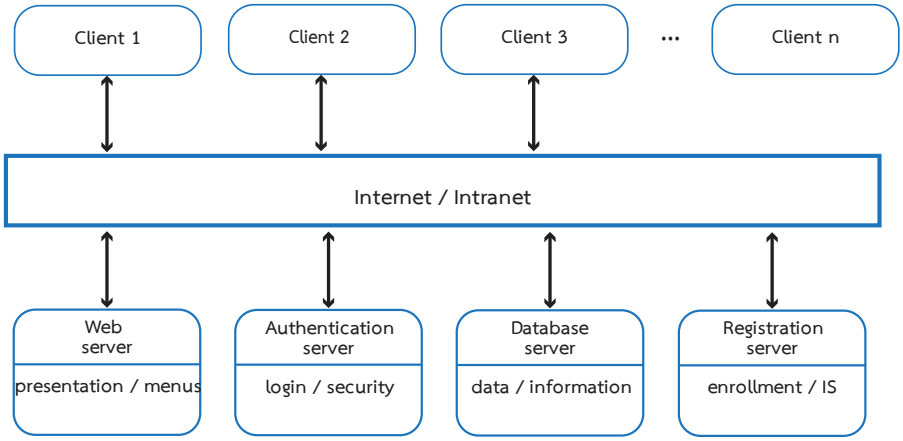


รูปที่ 6.4 สถาปัตยกรรมคลัง

จากตัวอย่างสถาปัตยกรรมแบบคลังที่ใช้คลังข้อมูลเป็นแกนของระบบโดยเน้นการแชร์ข้อมูลระหว่างส่วนประกอบต่าง ๆ ตัวอย่างนำเสนอสถาปัตยกรรมคลังสำหรับซอฟต์แวร์ระบบสร้างสื่อดิจิทัลที่มีข้อมูลกลางคือสื่อดิจิทัลที่แชร์ให้ส่วนประกอบต่างๆ เข้าถึงเพื่อประมวลผลหรือนำไปใช้งานหรืออาจเรียกได้ว่าเป็นการรวมศูนย์ด้วยข้อมูล ยกตัวอย่างเพิ่มเติม ซอฟต์แวร์ชุดพัฒนาโปรแกรมส่วนใหญ่ใช้สถาปัตยกรรมคลัง โดยในส่วนของคลังข้อมูลจะไว้เก็บรหัสต้นฉบับ เอกสาร และ การทดสอบโดยที่ส่วนประกอบต่าง ๆ สามารถเข้าถึงข้อมูลส่วนกลางเพื่อใช้ประกอบการทำงานและประมวลผลในแต่ละส่วน

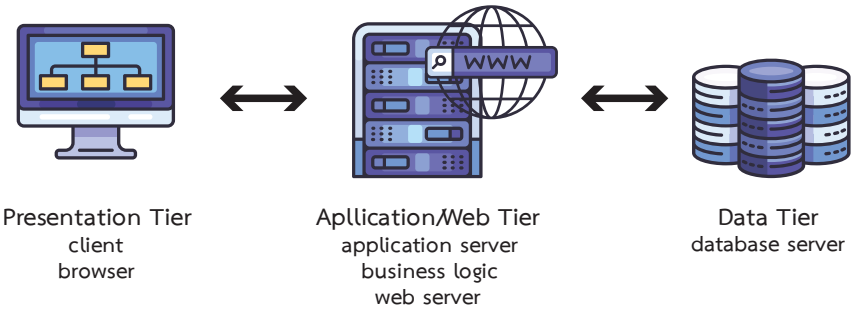
สถาปัตยกรรมแบบไคลเอ็นเซิร์ฟเวอร์ (client-server)

สถาปัตยกรรมแบบไคลเอ็นเซิร์ฟเวอร์ประกอบด้วย สาม ส่วนหลักคือไคลเอ็น เซิร์ฟเวอร์ และ เครือข่ายโดยเน้นการนำเสนอบริการต่าง ๆ ที่ฝั่งเซิร์ฟเวอร์มีให้บริการโดยแต่ละบริการก็จะเชื่อมต่อไปยังเซิร์ฟเวอร์ที่รับผิดชอบบริการ เช่น ระบบงานทะเบียนของมหาวิทยาลัยสามารถแสดงสถาปัตยกรรมได้ดังรูปที่ 6.5



รูปที่ 6.5 สถาปัตยกรรมไคลเอ็นเซิร์ฟเวอร์

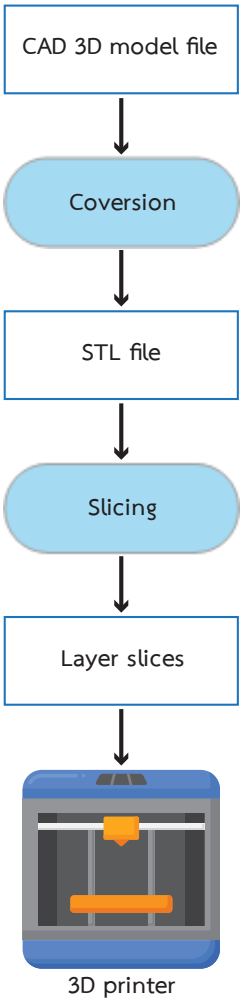
จากตัวอย่าง เซิร์ฟเวอร์ให้บริการไคลเอ็นที่มาร้องขอการให้บริการโดยทำงานในลักษณะที่ไคลเอ็นร้องขอ เซิร์ฟเวอร์ตอบสนองโดยการให้บริการตามการร้องขอกลับไปยังไคลเอ็น โดยเป็นรูปแบบสถาปัตยกรรมที่เรียบง่ายและมีการใช้กันมาอย่างยาวนาน สถาปัตยกรรมไคลเอ็นเซิร์ฟเวอร์ยังมีอีกหลายรูปแบบที่ออกแบบมาเพื่อเพิ่มประสิทธิภาพ เพิ่มความปลอดภัย และ ความคล่องตัวในการปรับเปลี่ยน เช่นสถาปัตยกรรมแบบ 3-Tier ที่มีการแบ่งแยกหน้าที่การทำงานในแต่ละ Tier ออกจากกันดังรูปที่ 6.6 จะเห็นว่า Data ถูกแยกออกจาก การบริการหลัก และการแสดงผล เพื่อความปลอดภัยประสิทธิภาพ และ ความยืดหยุ่น



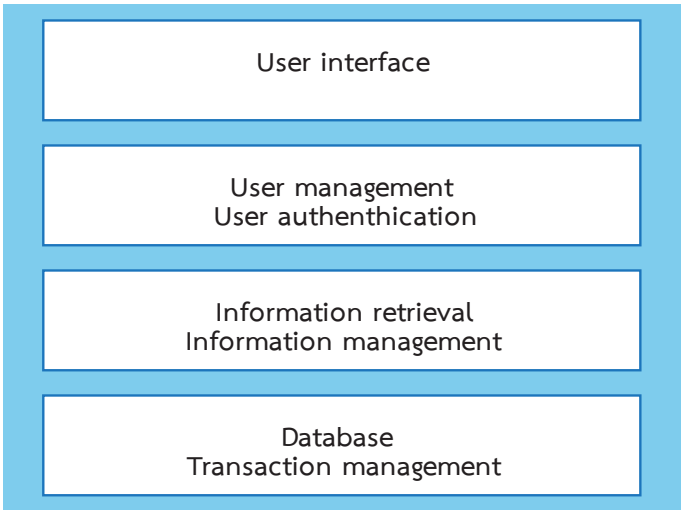
รูปที่ 6.6 สถาปัตยกรรม 3-Tier

สถาปัตยกรรมแบบท่อและตัวกรอง (pipes and filters)

สถาปัตยกรรมแบบท่อและตัวกรองมีโครงสร้างสำหรับการประมวลผลกระแสข้อมูล โดยแต่ละขั้นตอนของการประมวลผลจะถูกห่อหุ้มไว้ในส่วนประกอบแบบตัวกรอง ส่งต่อกันไปเป็นทอด ๆ จนได้ผลลัพธ์ออกมาในที่สุด สถาปัตยกรรมประเภทนี้เริ่มมีในระบบปฏิบัติการ Unix ในการประมวลผลแบบท่อ โดยสถาปัตยกรรมนี้เหมาะกับระบบที่มีการทำงานเป็นสายหรือการประมวลผลข้อมูลในลักษณะก้อน (batch processing) ดังแสดงในรูปที่ 6.7



รูปที่ 6.7 สถาปัตยกรรมแบบท่อและตัวกรอง



รูปที่ 6.8 สถาปัตยกรรมระบบสารสนเทศ

ระบบสารสนเทศ (information system)

ระบบสารสนเทศสามารถประยุกต์ใช้ได้หลากหลายธุรกิจหลากหลายองค์กร โดยเน้นที่การบูรณาการข้อมูลเพื่อประมวลผลและเข้าถึงสารสนเทศ ซึ่งรูปแบบสถาปัตยกรรมก็จะคล้าย ๆ กันโดยส่วนใหญ่เลือกใช้สถาปัตยกรรมแบบชั้นและแบบไคลเอ็นเซิร์ฟเวอร์ เช่นระบบงานทะเบียนมหาวิทยาลัย ระบบสืบค้นข้อมูลห้องสมุด ระบบการศึกษาอิเล็กทรอนิกส์ ระบบบริหารจัดการองค์กร โดยจะมีสถาปัตยกรรมที่คล้ายกันดังรูปที่ 6.8

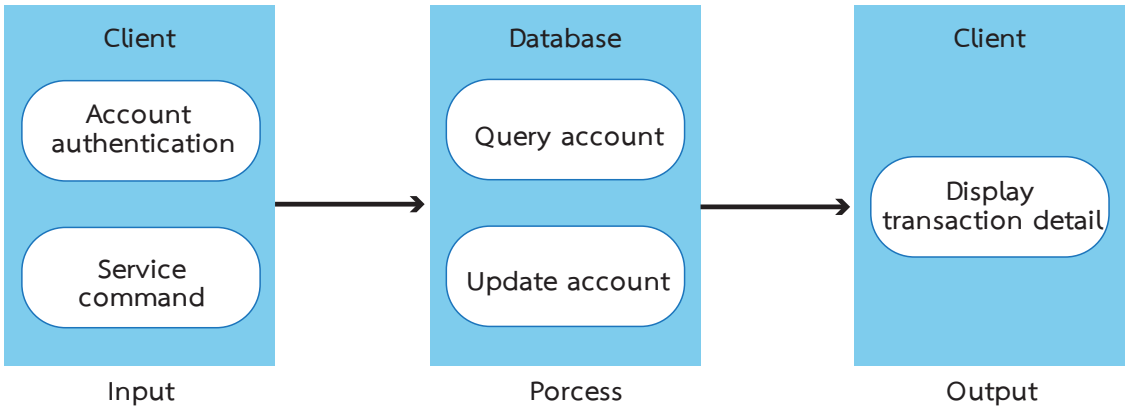
สถาปัตยกรรมตามการประยุกต์

ในปัจจุบันซอฟต์แวร์สามารถช่วยขับเคลื่อนธุรกิจและองค์กรได้ ซึ่งธุรกิจในแต่ละกลุ่มก็มักจะมีรูปแบบการดำเนินงานรูปแบบธุรกิจที่คล้าย ๆ กันอย่างเช่นกลุ่มธุรกิจขนส่งสินค้าก็จะมีรูปแบบการดำเนินธุรกิจรูปแบบการให้บริการที่คล้าย ๆ กัน ซอฟต์แวร์ที่ใช้สำหรับธุรกิจขนส่งก็จะมีหลายส่วนที่เหมือนกัน รูปแบบโครงสร้างและสถาปัตยกรรมที่เหมือนกันแบบนี้เรียกว่าสถาปัตยกรรมตามการประยุกต์เช่น สถาปัตยกรรมกลุ่ม ระบบสารสนเทศ ระบบพาณิชย์อิเล็กทรอนิกส์ ระบบจัดการขนส่งสินค้า

รูปแบบธุรกิจที่คล้าย ๆ กันนำไปสู่การออกแบบและพัฒนาเป็นสถาปัตยกรรมเพื่อตอบสนองกับการประยุกต์ใช้ในลักษณะดังกล่าว โดยอาจเริ่มต้นจากการประยุกต์กับระบบที่ตรงหรือใกล้เคียงกันเพื่อนำมาพัฒนา หรือเลือกระบบทั่วไปมาปรับให้เข้ากับองค์กรณ์หรือธุรกิจได้ เช่น การปรับแต่งระบบ Enterprise Resource Planning (ERP) มาใช้หรือเลือกใช้ส่วนประกอบที่มีขายในท้องตลาดมาปรับแต่งใช้งาน เช่น ระบบจัดการโรงแรม ไม่จำเป็นต้องออกแบบใหม่สามารถเลือกซื้อส่วนประกอบด้านการบริหารหรือซอฟต์แวร์ระบบบริหารโรงแรมมาปรับใช้ให้เข้ากับธุรกิจโรงแรมได้ หรือพัฒนาขึ้นมาใหม่บางส่วนให้ตอบสนองความต้องการได้ครบถ้วน เป็นการสนับสนุนการพัฒนาซอฟต์แวร์แบบนำกลับมาใช้ใหม่

ระบบประมวลผลธุรกรรม (transaction processing system)

ระบบประมวลผลธุรกรรมเป็นระบบที่ใช้กันมากในธุรกิจที่ขึ้นอยู่กับการทำธุรกรรมเช่นการทำธุรกรรมการเงินที่ธนาคารโดยหัวใจสำคัญของระบบประเภทนี้คือการประมวลผลข้อมูลธุรกรรมร่วมกับการจัดเก็บข้อมูลอย่างมีประสิทธิภาพและมีความปลอดภัย โดยเฉพาะธุรกรรมด้านการเงินที่จำเป็นต้องมีความปลอดภัยสูงและความผิดพลาดต่ำ รูปแบบระบบประมวลผลธุรกรรมโดยมากจะเป็นระบบแบบปฏิสัมพันธ์โดยผู้ใช้ร้องขอบริการในการทำธุรกรรมซึ่งจะไม่ขึ้นอยู่กับเวลาและปริมาณจำนวนธุรกรรม สถาปัตยกรรมที่ใช้อาจจะผสมผสานให้เหมาะสมขึ้นอยู่กับรูปแบบธุรกรรมเช่นการใช้สถาปัตยกรรมไคลเอ็นเซิร์ฟเวอร์เพื่อความสะดวกในการเข้าถึงระบบเพื่อขอใช้บริการในขณะที่ระบบหลังสำนักงาน (back office) ในส่วนของตรรกะทางธุรกิจ (business logic) ใช้สถาปัตยกรรมแบบท่อและตัวกรองในการประมวลผล เช่น ในระบบธนาคารลูกค้าใช้ web application ของธนาคารเพื่อชำระค่าสินค้าหรือขอรายงานรายการการเดินบัญชีดังรูปที่ 6.9



รูปที่ 6.9 สถาปัตยกรรมระบบประมวลผลธุรกรรม

บทที่ 7 การออกแบบและการพัฒนา Design and implementation

วัตถุประสงค์

- *เข้าใจการออกแบบซอฟต์แวร์*
- *เข้าใจการพัฒนาซอฟต์แวร์*
- *สามารถเลือกรูปแบบการพัฒนาให้เหมาะสมกับระบบ*

กิจกรรมการออกแบบและพัฒนาซอฟต์แวร์เป็นกิจกรรมสำคัญในกระบวนการพัฒนาซอฟต์แวร์ เพื่อส่งมอบซอฟต์แวร์ที่มีคุณภาพตรงตามความต้องการให้กับลูกค้าภายในระยะเวลาที่กำหนด การออกแบบซอฟต์แวร์คือการนิยามและอธิบายโครงสร้างของซอฟต์แวร์ก่อนจะพัฒนาสามารถทำได้หลายระดับตั้งแต่การออกแบบสถาปัตยกรรมซอฟต์แวร์ออกแบบโครงสร้างออกแบบมอดูลส่วนประกอบ ฟังก์ชัน หรือ ขั้นตอนวิธี (algorithm) ในส่วนของวิศวกรรมซอฟต์แวร์เอกสารการออกแบบอาจจะใช้การเขียนบรรยายรูปแผนภาพหรือใช้ภาษายูเอ็มแอลในการอธิบายแล้วส่งต่อไปพัฒนาซอฟต์แวร์

การพัฒนาซอฟต์แวร์คือขั้นตอนในการสร้างซอฟต์แวร์ตามการออกแบบด้วยเครื่องมือและเทคนิคในการพัฒนารูปแบบต่าง ๆ การเขียนรหัสต้นฉบับ การนำเอาซอฟต์แวร์กลับมาใช้ใหม่ การประยุกต์ใช้ซอฟต์แวร์ส่วนประกอบ หรือการปรับแต่งซอฟต์แวร์เพื่อให้ได้ซอฟต์แวร์ที่ใช้งานได้ตรงความต้องการ ไม่มีกระบวนการที่ดีที่สุดสำหรับการพัฒนา เป็นหน้าที่ของวิศวกรซอฟต์แวร์ที่ต้องเลือกรูปแบบหรือกระบวนการพัฒนาซอฟต์แวร์ที่เหมาะสมเพื่อสร้างซอฟต์แวร์ให้มีคุณภาพสูงมีความถูกต้องและใช้งานได้ตรงตามความต้องการ

แนวคิดและหลักการในการออกแบบและพัฒนาซอฟต์แวร์

กระบวนการออกแบบและพัฒนาซอฟต์แวร์เป็นกิจกรรมที่นักพัฒนาดำเนินการเพื่อให้ได้มาซึ่งซอฟต์แวร์ที่ใช้งานได้ตรงตามความต้องการของลูกค้า โดยถือว่าเป็นหัวใจหลักของกระบวนการวิศวกรรมซอฟต์แวร์เนื่องจากเป็นขั้นตอนการผลิตซอฟต์แวร์นั่นเอง ในบางระบบที่ไม่ใหญ่และไม่ซับซ้อน กระบวนการวิศวกรรมซอฟต์แวร์อาจนำทุกกิจกรรมผนวกรวมเข้ากับกิจกรรมการออกแบบและพัฒนาซอฟต์แวร์ก็เพียงพอ ในทางกลับกลับกันระบบที่ใหญ่และซับซ้อนยังคงจำเป็นต้องดำเนินการกิจกรรมทางวิศวกรรมซอฟต์แวร์อื่น ๆ โดยที่กิจรรมการออกแบบและพัฒนาเป็นเพียงหนึ่งในกิจกรรมเหล่านั้น

การออกแบบและพัฒนาส่วนใหญ่จะดำเนินการควบคู่กันไป โดยการออกแบบนั้นคือการคิดวิเคราะห์เพื่อหาวิธีแก้ปัญหาให้ตรงกับความต้องการและส่งต่อการออกแบบนี้ไปพัฒนาเพื่อให้ได้ซอฟต์แวร์มาใช้งาน ส่วนมากมักจะไม่แยกกิจกรรมนี้ออกจากกันอย่างชัดเจน ยกเว้นบางกระบวนการที่จำเป็นต้องมีการออกแบบในเสร็จสิ้นก่อนนำไปดำเนินการพัฒนาต่อ อย่างไรก็ตามจะเห็นว่าการออกแบบนั้นเป็นกระบวนการที่ต้องใช้จินตาการและความคิดสร้างสรรค์ในการออกแบบซอฟต์แวร์โดยอาจจะออกแบบในสมอง เขียนลงกระดาษ กระดาน หรือ เขียนแผนภาพเก็บไว้ในคอมพิวเตอร์ ซึ่งการออกแบบนั้นไม่ได้มีมาตรฐานหรือกระบวนการที่กำหนดไว้อย่างชัดเจน

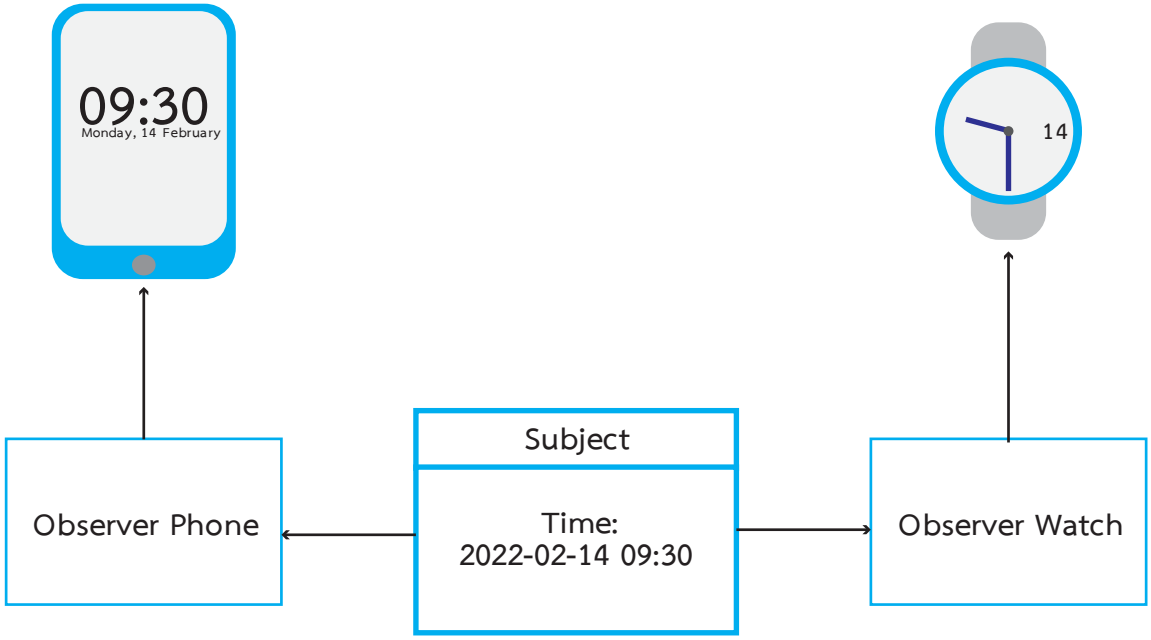
แนวคิดการออกแบบซอฟต์แวร์นั้นไม่ควรถูกจำกัดด้วยมาตรฐานหรือกฎเกณฑ์ นักออกแบบไม่จำเป็นต้องออกแบบและเขียนแผนภาพตามยูเอ็มแอลหรือต้องทำเอกสารตามมาตรฐานใด ๆ เว้นแต่องค์กรหรือทีมได้กำหนดไว้ เนื่องจากการออกแบบและพัฒนานั้นมีความแนบแน่นมาก เอจายล์จึงมีแนวคิดเรื่องการออกแบบซอฟต์แวร์ให้เป็นหน้าที่นักพัฒนาดำเนินการพร้อมการพัฒนาในส่วนการจัดทำเอกสารเป็นเรื่องรองลงมาโดยการออกแบบจะเน้นไปที่การออกแบบอย่างไม่เป็นทางการมากกว่าเพื่อความคล่องตัวและสะดวกเมื่อมีการเปลี่ยนแปลง สำหรับแนวคิดในการพัฒนานั้นจะเป็นไปในแนวทางเดียวกันกับการออกแบบโดยการพัฒนานั้นจะอิงตามกระบวนการออกแบบเพื่อเลือกกรรมวิธีการพัฒนาให้เหมาะสมกับการออกแบบโดยคำนึงถึงความคุ้มค่าในการพัฒนาและความต้องการเป็นสำคัญ

วิศวกรซอฟต์แวร์ควรพิจารณาเพื่อตัดสินใจเลือกแนวทางออกแบบและพัฒนาซอฟต์แวร์แต่เนิ่น ๆ เลือกว่าจะใช้การพัฒนาซอฟต์แวร์จากการเลือกซื้อส่วนประกอบจากชั้น (off-the-shelf) การซื้อซอฟต์แวร์มาปรับใช้ให้ตรงความต้องการ หรือดำเนินการพัฒนาขึ้นมาเองทั้งหมด ซึ่งวิศวกรซอฟต์แวร์ต้องทำหน้าที่วิเคราะห์และตัดสินใจเลือกรูปแบบการออกแบบและพัฒนา เช่น การพัฒนาซอฟต์แวร์จัดการโรงแรมจากการเลือกซื้อซอฟต์แวร์ที่มีอยู่ธุรกิจการให้บริการโรงแรมนำมาปรับและติดตั้งให้เหมาะสมกับความต้องการของลูกค้าน่าจะเป็นแนวทางที่เหมาะสม อาจจะมีค่าใช้จ่ายที่ต่ำกว่ารูปแบบอื่น ใช้ระยะเวลาในการพัฒนาสั้นกว่ารูปแบบอื่น ๆ ในทางกลับกันถ้ารูปแบบธุรกิจของลูกค้าไม่มีซอฟต์แวร์ที่จะนำมาปรับใช้ได้ การพัฒนาขึ้นมาเองน่าจะเป็นวิธีที่ดีและตอบสนองความต้องการของลูกค้าได้เต็มที่ ยกตัวอย่างธุรกิจแพลตฟอร์มให้เช่าและแชร์พาหนะรูปแบบต่าง ๆ เป็นระบบที่ใหม่และมีความต้องการเฉพาะที่แตกต่างจากรูปแบบธุรกิจอื่น ๆ การพัฒนาขึ้นมาใหม่ทั้งหมดหรือดำเนินการแบบผสมผสานน่าจะเป็นวิธีที่เหมาะสม ดังนั้นวิศวกรซอฟต์แวร์ควรให้ความสำคัญในการตัดสินใจเลือกรูปแบบการออกแบบและพัฒนาซอฟต์แวร์

แบบแผนการออกแบบ (design patterns)

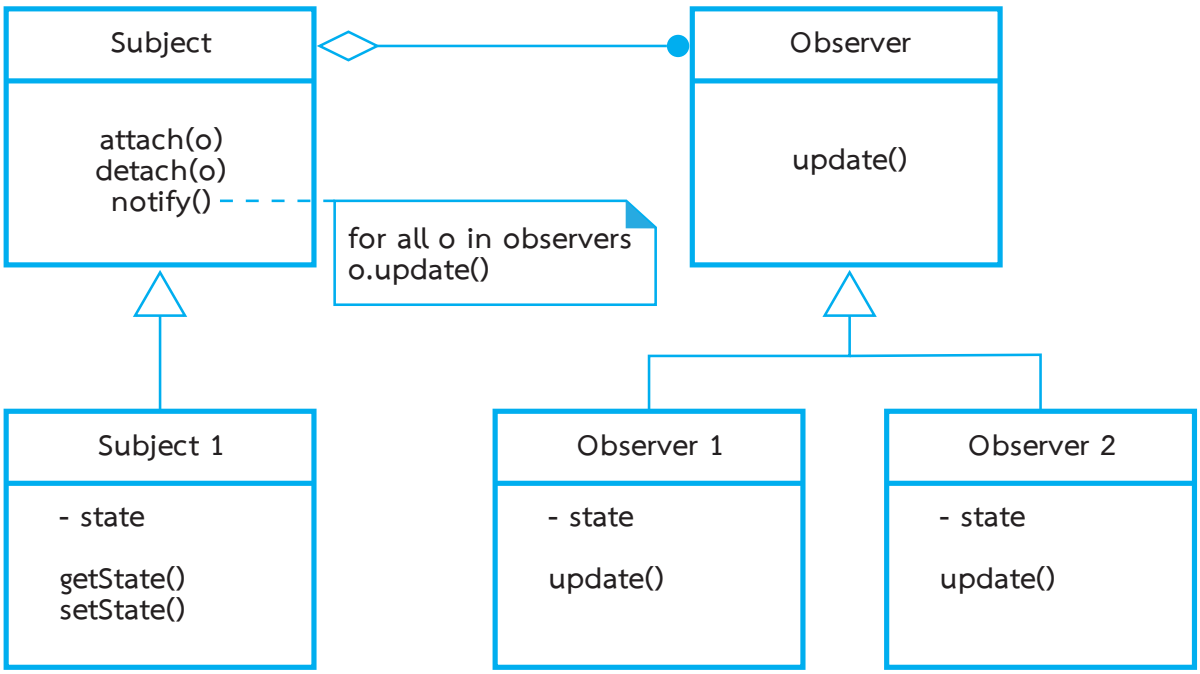
ย้อนไปปีคศ. 1979 Christopher Alexander นำเสนอแนวความคิดเรื่องการออกแบบที่มีลักษณะเหมือน ๆ กันสามารถจัดกลุ่มออกมาเป็นรูปแบบการออกแบบ (design pattern) ที่นำมาใช้แก้ไขปัญหาที่ซ้ำๆ หรือใกล้เคียงกัน โดยเกิดจากการออกแบบทดสอบและปรับปรุง เพื่อส่งต่อ สืบทอดรูปแบบที่ใช้งานได้ต่อไป เป็นการประหยัดเวลาในการออกแบบโดยการเลือกรูปแบบโดยเทียบเคียงกับปัญหาหรือระบบที่ต้องการจะพัฒนา โดยจะอธิบายรูปแบบกว้าง ๆ ไว้ไม่ได้ระบุชัดเจนหรือเคร่งครัด นักพัฒนาสามารถนำรูปแบบไปประยุกต์ใช้เพื่อแก้ปัญหาตามแนวทางการนำกลับมาใช้ใหม่

การออกแบบเชิงวัตถุนิยมนำรูปแบบการออกแบบมาประยุกต์ใช้ เนื่องจากรูปแบบส่วนใหญ่ถูกพัฒนาและทดสอบเป็นอย่างดีแล้วว่าสามารถนำไปแก้ไขปัญหาในลักษณะใกล้เคียงกันได้มีการนิยามในรูปแบบเชิงวัตถุไว้ทำให้ง่ายต่อการนำไปประยุกต์ใช้ในการออกแบบ สำหรับรูปแบบการออกแบบนั้นสามารถศึกษาเพิ่มเติมจากหนังสือ โดยเฉพาะรูปแบบการอออกแบบที่ถูกนิยามโดยวิศวกรซอฟต์แวร์รุ่นบุกเบิก (gang of four) โดยรูปแบบการออกแบบที่นำเสนอนั้น ได้ถูกสกัดเรียบเรียงมาจากประสบการณ์และองค์ความรู้ของนักออกแบบโดยส่วนใหญ่จะเขียนเพื่ออธิบายด้วยรูปแบบของการโปรแกรมเชิงวัตถุ สามารถศึกษาเพิ่มเติมจากหนังสือ [12] ตัวอย่างรูปแบบผู้สังเกต (bbserver pattern) ตามรูป 7.1



รูปที่ 7.1 รูปแบบผู้สังเกต

จากรูปจะเห็นได้ว่ารูปแบบผู้สังเกตสามารถนำมาใช้ในการแก้ปัญหาการแสดงผลข้อมูลเดียวกันแต่มีความแตกต่างในการนำเสนอ โดยในรูปข้อมูลใน subject คือเวลาตามมาตรฐาน ISO 8601 ซึ่งแต่ละผู้สังเกต (observer) จะนำข้อมูลนี้ไปแสดงผลในรูปแบบของตนเช่น โดยหลักการพื้นฐานคือเมื่อ subject มีการเปลี่ยนแปลงจะแจ้งให้ observer ทุกตัวทราบว่ามีการเปลี่ยนแปลงทำให้ทุก observer ทำการปรับปรุงตามการเปลี่ยนแปลง ในกรณีตัวอย่างข้างต้น subject มีการเปลี่ยนแปลงทุก ๆ หนึ่งนาทีและเมื่อเวลามีการเปลี่ยนแปลงจะแจ้งไปยัง observer เพื่อดำเนินการปรับปรุงการแสดงผลเวลาบนหน้าจอให้ถูกต้อง รูปแบบผู้สังเกตสามารถเขียนในรูปยูเอ็มแอลดังรูปที่ 7.2



รูปที่ 7.2 แผนภาพยูเอ็มแอลแสดงรูปแบบผู้สังเกต

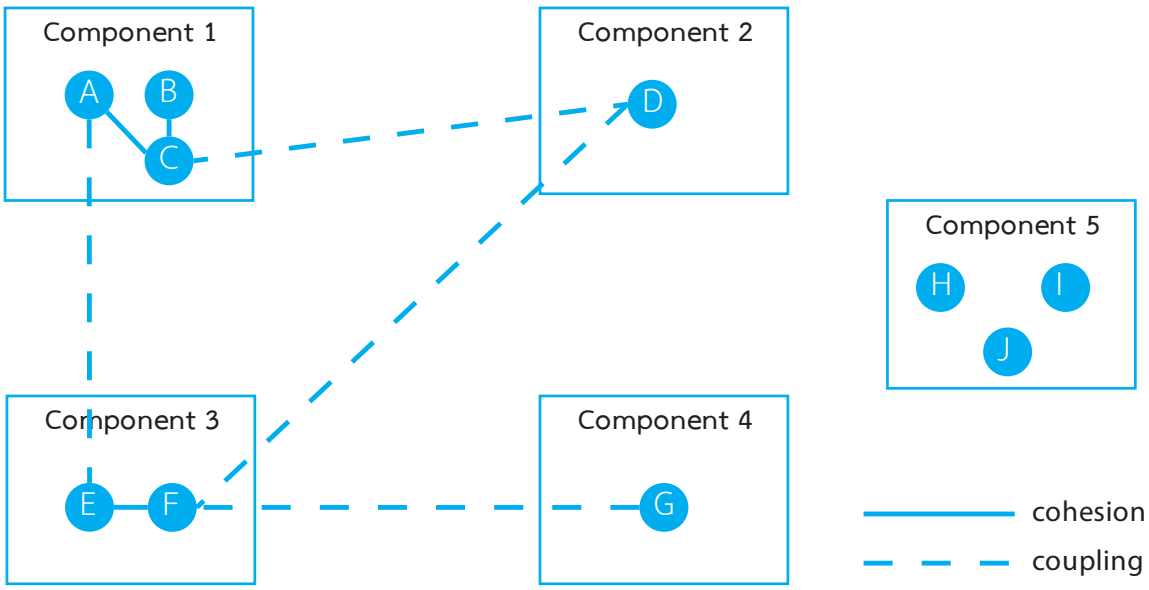
การออกแบบโครงสร้าง (structure design)

การออกแบบโครงสร้างของซอฟต์แวร์ อิงตามการออกแบบสถาปัตยกรรมซอฟต์แวร์ที่ได้นำเสนอในบทที่ 6 โดยทำการออกแบบโครงสร้างโดยรวมของระบบเพื่อให้เห็นภาพรวมของระบบ ระบุส่วนประกอบสำคัญและความสัมพันธ์ เพื่อฉายภาพรวมให้เห็นว่าระบบจะทำงานเพื่อตอบสนองความต้องการได้อย่างไร นอกจากนั้นยังสะท้อนให้เห็นถึงแนวคิดในการออกแบบเพื่อแก้ปัญหา

การออกแบบโครงสร้างควรเริ่มด้วยออกแบบอย่างง่ายเพื่ออธิบายโครงสร้างระบบผ่านส่วนประกอบหลักและการปฏิสัมพันธ์ระหว่างกันเพื่อแสดงให้เห็นภาพรวมและแนวทางการแก้ไขปัญหาและการทำงานของระบบซอฟต์แวร์ที่กำลังจะพัฒนา นักออกแบบควรนำเสนอโครงสร้างอย่างง่ายและไม่ซับซ้อน หากระบบมีความซับซ้อนก็ควรออกแบบให้ง่ายลงด้วยการแตกย่อยระบบเพื่อให้ง่ายต่อการออกแบบและการจัดการ โดยสามารถออกแบบในรูปแบบของมอดูลที่ง่ายต่อการจัดการเพื่อช่วยลดความซับซ้อนในจัดการให้อยู่ในรูปแบบที่ง่ายต่อการทำความเข้าใจเหมาะจะนำไปพัฒนาต่อ

การออกแบบโครงสร้างซอฟต์แวร์เป็นมอดูลนั้นควรคำนึงถึงจำนวนมอดูลที่ออกแบบเนื่องจาก การออกแบบและพัฒนาแต่ละมอดูลมีแนวโน้วด้านค่าใช้จ่ายลดลงเมื่อจำนวนมอดูลเพิ่มขึ้น ในทางกลับกันค่าใช้จ่ายในการรวมประกอบมอดูลต่าง ๆ เข้าด้วยกันจะเพิ่มขึ้นเมื่อจำนวนมอดูลเพิ่มมากขึ้น นักออกแบบจึงต้องพิจารณาหาจุดสมดุลของจำนวนมอดูลที่เหมาะสมบนพื้นฐานความคุ้มค่าที่สุดในด้านค่าใช้จ่ายในการพัฒนา

สำหรับการออกแบบโครงสร้างสิ่งที่ต้องคำนึงอีกเรื่องคือความอิสระต่อกันทั้งด้านฟังก์ชันและทั้งด้านข้อมูลโดยโดยการออกแบบมักจะใช้ตัวชี้วัดสองค่าในการวัดคือ การเชื่อมติด (cohesion) การคู่ควบ (coupling) ดังรูปที่ 7.3 โดยการเชื่อมติดหมายถึงการมีความสอดคล้องกันภายในมอดูลที่มีการทำงานไปในทิศทางเดียวกัน ส่วนการควบคู่เป็นตัวชี้วัดความเป็นอิสระต่อกันระหว่างมอดูลโดยการออกแบบระบบที่ดีควรมีการเชื่อมติดสูงและการควบคู่ต่ำหรืออาจมองว่าแต่ละมอดูลมีฟังก์ชันการทำงานเฉพาะทาง และมีการอ้างอิงเชื่อมถึงกันต่ำ ทำให้แต่ละมอดูลมีความเป็นอิสระต่อกันสูง การออกแบบการพัฒนาและการทดสอบสามารถทำได้ง่ายนับว่าเป็นการออกแบบโครงสร้างที่ดี แต่อย่างรก็ตามในความเป็นจริงอาจไม่สามารถทำได้ในแบบอุดมคติที่กล่าวไว้ข้างต้น นักออกแบบก็ควรจะออกแบบและปรับปรุงตลอดจนทำการรีแฟคเตอริงโครงสร้างเพื่อให้ที่เหมาะสมกับปัญหา

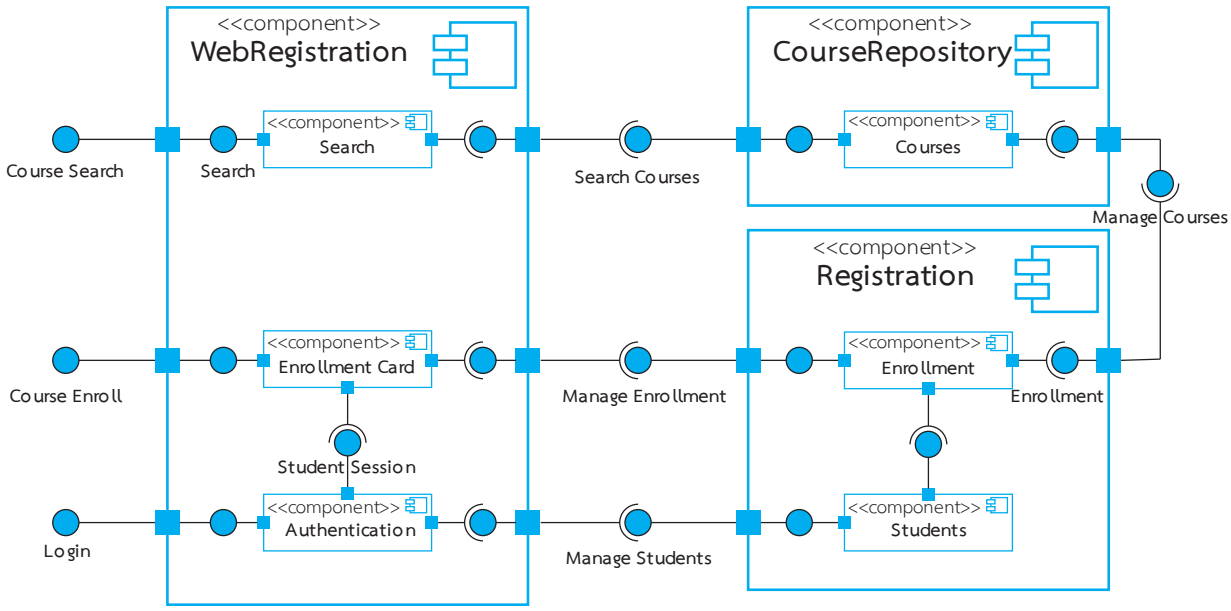


รูปที่ 7.3 การเชื่อมติด (cohesion) และการคู่ควบ (coupling)

การออกแบบระดับส่วนประกอบ (component-level design)

การออกแบบโครงสร้างเป็นการออกแบบระบบโดยรวมในระดับส่วนประกอบหรือมอดูลซึ่งไม่ได้ลงในรายละเอียด นักออกแบบจึงต้องดำเนินการออกแบบระดับส่วนประกอบเพื่อขยายความแสดงการทำงานของส่วนประกอบต่าง ๆ ในระบบ โดยการออกแบบนี้จะลงในรายละเอียดเพื่อให้สามารถอธิบายส่วนประกอบได้อย่างชัดเจนมีข้อมูลเพียงพอต่อการทำความเข้าใจและนำไปพัฒนาในขั้นตอนต่อไปโดยการออกแบบจะรวมถึง การอธิบายการทำงานส่วนประกอบ การใช้อัลกอริทึมในการแก้ไขปัญหา โครงสร้างข้อมูล การประมวลผลต่าง ๆ และ การเชื่อมต่อส่วนประสานของส่วนประกอบ

การอธิบายการออกแบบระดับส่วนประกอบ สามารถใช้แผนภาพยูเอ็มแอลมาประกอบการอธิบาย สามารถใช้รหัสเทียมหรือผังกระแสในการบรรยายการทำงานหรืออัลกอริทึมเพื่อให้เข้าใจการออกแบบได้ชัดเจน ในส่วนของด้านการออกแบบข้อมูลจะใช้การอธิบายโครงสร้างข้อมูลที่นำมาใช้กับส่วนประกอบ นักออกแบบควรเน้นการออกแบบที่ชัดเจนมีรายละเอียดที่เพียงพอสามารถนำไปใช้ในกระบวนการพัฒนาต่อได้ ยกตัวอย่างแผนภาพส่วนประกอบระบบลงทะเบียนดังรูปที่ 7.4



รูปที่ 7.4 แผนภาพส่วนประกอบระบบลงทะเบียน

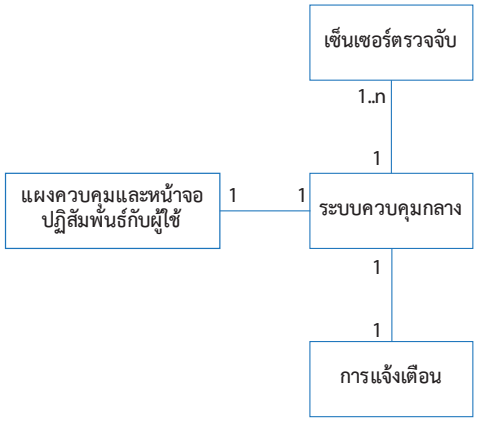
การวิเคราะห์และออกแบบเชิงวัตถุ (object-oriented analysis and design)

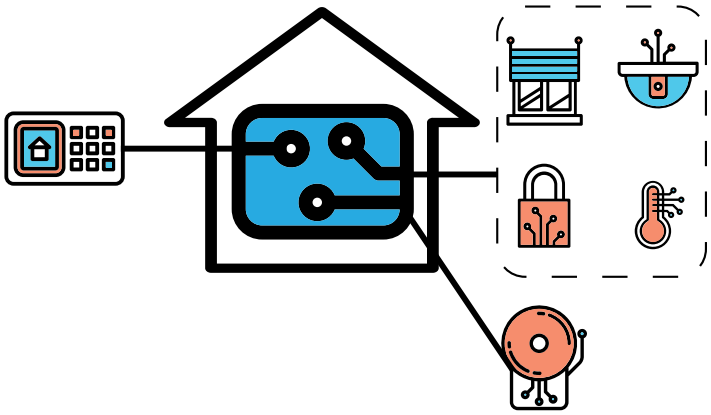
การออกแบบและพัฒนาซอฟต์แวร์แบบเชิงวัตถุเป็นอีกหนึ่งรูปแบบที่ได้รับความนิยมและใช้กันอย่างแพร่หลายอันเนื่องมาจากแนวความคิดที่ใช้วัตถุในการแสดงถึงส่วนต่าง ๆ ความสัมพันธ์ในระบบ การวิเคราะห์ และ การออกแบบก็สามารถใช้ยูเอ็มแอลเป็นสื่อกลาง มีภาษาโปรแกรมหลายภาษารองรับการพัฒนาในรูปแบบนี้ นอกจากนั้นยังรองรับจากเครื่องมือการออกแบบและพัฒนา

การวิเคราะห์และออกแบบเชิงวัตถุนักพัฒนาจำเป็นต้องมีความรู้และเข้าใจในแนวคิดและหลักการพื้นฐานของการโปรแกรมเชิงวัตถุ หลังจากดำเนินการวิเคราะห์และออกแบบเชิงวัตถุเสร็จเรียบร้อย กระบวนการพัฒนาจะราบลื่นโดยเพียงนำการออกแบบไปพัฒนาด้วยภาษาและเครื่องมือพัฒนาเชิงวัตถุ เครื่องมือทางวิศวกรรมซอฟต์แวร์บางชิ้นสามารถดำเนินการสร้างรหัสต้นฉบับให้โดยอัตโนมัติจากการออกแบบ (ยูเอ็มแอล หรือ ผังการไหล) แต่สิ่งที่สำคัญคือนักออกแบบจำเป็นต้องเข้าใจหลักการและแนวเชิงวัตถุเป็นอย่างดี เพื่อนำมาวิเคราะห์และออกแบบซอฟต์แวร์ให้ได้ตรงกับความต้องการและเหมาะแก่การนำไปพัฒนา

สิ่งที่นักพัฒนาควรรู้และเข้าใจในการออกแบบเชิงวัตถุคือ การเข้าใจการปฏิสัมพันธ์และความสัมพันธ์ทั้งภายในและภายนอกระบบ เข้าใจการออกแบบสถาปัตยกรรมซอฟต์แวร์ สามารถระบุและนิยามวัตถุที่สำคัญในระบบได้ รวมทั้งการเชื่อมต่อระหว่างวัตถุ โดยหลักการเหล่านี้เมื่อนำมาประกอบกันจะทำให้สามารถออกแบบเชิงวัตถุ ร่วมกับการใช้ความคิดสร้างสรรค์ในการออกแบบเพื่อให้ได้มาซึ่งคำตอบสำหรับปัญหา

โดยทั่วไปกระบวนการออกแบบจะเริ่ม จากการหาแนวความคิดต่าง ๆ ในการแก้ปัญหา นำเสนอแนวทางการแก้ปัญหา วิเคราะห์และกลั่นกรองแนวทางการแก้ปัญหา โดยสามารถประเมินร่วมกับข้อมูลอื่น ๆ สามารถใช้แผนภาพ สัญญลักษณ์ หรือ สื่อต่างในการนำเสนอการออกแบบ เพื่อใช้เป็นข้อมูลอ้างอิง ใช้เป็นสื่อกลางในการทำงานในทีม และ เพื่อแปลงจากการออกแบบในสมองของนักออกแบบมาอยู่ในรูปแบบที่เป็นรูปธรรมมากขึ้นทีมสามารถเห็นตรงกันและเข้าใจในสิ่งที่ออกแบบร่วมกัน โดยแนะนำให้ยูเอ็มแอลซึ่งเป็นมาตรฐานที่นิยมใช้กัน หรืออาจจะใช้รูปแบบอื่น ๆ ก็ย่อมทำได้ดังรูปที่ 7.5





รูปที่ 7.5 แผนภาพแสดงระบบสัญญาณกันขโมย

การออกแบบเพื่อนำกลับมาใช้ (design for reuse)

การออกแบบเพื่อนำกลับมาใช้เป็นกรรมวิธีการนำซอฟต์แวร์ทั้งการออกแบบและพัฒนานำกลับมาใช้ใหม่ เพื่อลดเวลา ลดค่าใช้จ่ายในการพัฒนา และ เพิ่มความน่าเชื่อถือโดยอยู่บนพื้นฐานความเชื่อว่าการนำกลับมาใช้นั้นเคยการผ่านการใช้งานการทดสอบและการปรับปรุงมาแล้วในระดับหนึ่งก่อนการนำกลับมาประยุกต์ใช้ใหม่ และแนวทางการออกแบบและพัฒนาซอฟต์แวร์ในปัจจุบันสนับสนุนให้ออกแบบเพื่อนำกลับมาใช้เพื่อลดการใช้ทรัพยากรในการพัฒนา เพื่อความรวดเร็วและรองรับการเปลี่ยนแปลงได้อย่างมีประสิทธิภาพ โดยการออกแบบเพื่อนำกลับมาใช้สามารถจำแนกได้ ดังนี้

1. **การนำกลับมาใช้ในระดับระบบ** เป็นการนำกลับมาใช้ทั้งระบบโดยเลือกระบบที่มีความใกล้เคียงกับระบบที่ต้องการพัฒนา
2. **การนำกลับมาใช้ในระดับแอปพลิเคชัน** โดยนำซอฟต์แวร์มาปรับใช้กับลูกค้าไม่มีการปรับเปลี่ยนระบบ เพียงดำเนินการตั้งค่าต่าง ๆ ให้เหมาะสมกับความต้องการของลูกค้า เช่น การปรับแต่งค่าให้เหมาะสมหรือเลือกจากสายผลิตภัณฑ์มาปรับแต่งให้เหมาะสมกับลูกค้า โดยอาจจะพิจารณาจากความคล้ายคลึงของสถาปัตยกรรม นอกจากนั้นเพื่อเติมเต็มความต้องการของลูกค้าบางส่วนสามารถดำเนินการออกแบบและพัฒนาเพิ่มเพื่อมาใช้งานร่วมกันได้
3. **การนำกลับมาใช้ในระดับส่วนประกอบ** การออกแบบลักษณะนี้เป็นการนำกลับมาใช้ในส่วนประกอบภายในซอฟต์แวร์ ไม่ว่าจะเป็นส่วนประกอบจากชั้น ทั้งในระดับวัตถุไปจนถึงระบบย่อย เช่น การเลือกส่วนประกอบสำหรับการทำธุรกรรมพาณิชย์อิเล็กทรอนิกส์มาประกอบใช้ในระบบร้านค้าออนไลน์หรือ การเข้าถึงบริการจากทางไกลหรือจากคลาวน์ก็ถือว่าเป็นการนำกลับมาใช้ใหม่ในระดับส่วนประกอบ เช่นการเข้าถึงบริการ API ต่าง ๆ เป็นต้น (Google map, speech regcognition, AI)
4. **การนำกลับมาใช้ในระดับวัตถุหรือฟังก์ชัน** การออกแบบที่เล็กในระดับวัตถุหรือฟังก์ชันสามารถใช้ซ้ำได้โดยมักจะอยู่ในรูปแบบของไลบรารีหรือคลาส โดยการนำกลับมาใช้มีมานานแล้วในการออกแบบและพัฒนาซอฟต์แวร์ เช่น การใช้งานไลบรารีมาตรฐาน อัลกอริทึมต่าง ๆ โดยส่วนใหญ่จะสามารถลดค่าใช้จ่ายได้มากกว่าการพัฒนาเองขึ้นมาใหม่

การออกแบบเพื่อนำกลับมาใช้นับว่ามีความสำคัญและมีประโยชน์มากสำหรับการพัฒนาซอฟต์แวร์ เป็นการส่งผ่านความรู้และประสบการณ์รูปแบบหนึ่งโดยจะเก็บส่วนที่ดีและมีประสิทธิภาพไว้ใช้งานต่อไป สามารถปรับปรุงและเพิ่มเติมให้ดียิ่งขึ้นได้โดยประโยชน์จากการนำกลับมาใช้ใหม่ที่เห็นได้ชัดเจนคือ ช่วยเร่งกระบวนการออกแบบและพัฒนาให้เร็วขึ้น ลดค่าใช้จ่ายในการพัฒนาและยังช่วยลดความเสี่ยงในการพัฒนาอีกด้วย นอกจากนั้นจะเห็นว่าการนำกลับมาใช้ช่วยส่งเสริมการสร้างมาตรฐานของซอฟต์แวร์ เช่น จากการนำกลับมาใช้ใหม่กลายเป็นมาตรฐานเฉพาะแล้วนำไปสู่เป็นมาตรฐานสากล หรือส่วนประสานผู้ใช้รูปแบบเมนูถูกนำมาใช้ซ้ำ ๆ จนกลายเป็นรูปแบบมาตรฐาน

ในส่วนของข้อจำกัดและปัญหาของการนำกลับมาใช้นั้นจะมีประเด็นของการ เข้าถึงเพื่อแก้ไขหรือดูแลส่วนที่มำกลับมาใช้โดยทางผู้ผลิตหรือผู้ให้บริการส่วนที่นำกลับมาใช้ไม่ให้นักพัฒนาเข้าถึงรหัสต้นฉบับหรือขาดการสนับสนุนเพียงพอ จะทำให้ระบบมีข้อจำกัดและเพิ่มค่าใช้จ่ายในส่วนการบำรุงรักษา การออกแบบอาจถูกจำกัด ความไม่เข้ากันของส่วนประกอบ นอกจากนั้นการนำกลับมาใช้ใหม่ วิศวกรซอฟต์แวร์มีหน้าที่ศึกษาทำความเข้าใจและตัดสินใจเลือกส่วนที่นำกลับมาใช้ให้เหมาะสมกับระบบ

นอกจากการนำกลับมาใช้จาก สี่ ระดับเบื้องต้นการออกแบบเพื่อนำกลับมาใช้สามารถออกแบบในระดับแนวคิด เพื่อนำมาออกแบบระบบโดยดูแบบอย่างจากแนวคิดการแก้ไขปัญหาที่มีอยู่แล้ว ซึ่งการออกแบบนี้เป็นลักษณะนามธรรมโดยออกแบบเพื่อกลับมาใช้อย่างกว้าง ๆ เช่น การประยุกต์การออกแบบร่วมกับการใช้ รูปแบบการออกแบบ (design pattern) หรือระดับสถาปัตยกรรม ซึ่งโดยรวมแล้วสามารถใช้ประกอบเป็นแนวทางการออกแบบ ช่วยลดระยะเวลา และ ค่าใช้จ่ายในการพัฒนาลงได้

ในส่วนของการพัฒนาซอฟต์แวร์นั้น มีหลายแนวทางที่สนับสนุนการนำกลับมาใช้ดัง เช่น กรอบการทำงาน รูปแบบสถาปัตยกรรม รูปแบบการออกแบบ การใช้ส่วนประกอบ ระบบบริหารจัดการทรัพยากรภายในองค์กร (ERP) ไลบรารี ระบบเชิงบริการ สายผลิตภัณฑ์ ระบบช่วยสร้างโปรแกรม เป็นต้น ซึ่งในการออกแบบเพื่อนำกลับมาใช้นั้นอาจจะมีข้อจำกัดที่ต้องนำมาพิจารณาเพื่อให้สามารถตอบสนองความต้องการของลูกค้า โดยหากจำเป็นต้องปรับเปลี่ยนความต้องการ สามารถทำได้โดยวิศวกรซอฟต์แวร์ต้องชี้ให้เห็นถึงความจำเป็น ข้อจำกัดบนพื้นฐาน คุณภาพของซอฟต์แวร์ และ ความคุ้มค่าเป็นสำคัญ

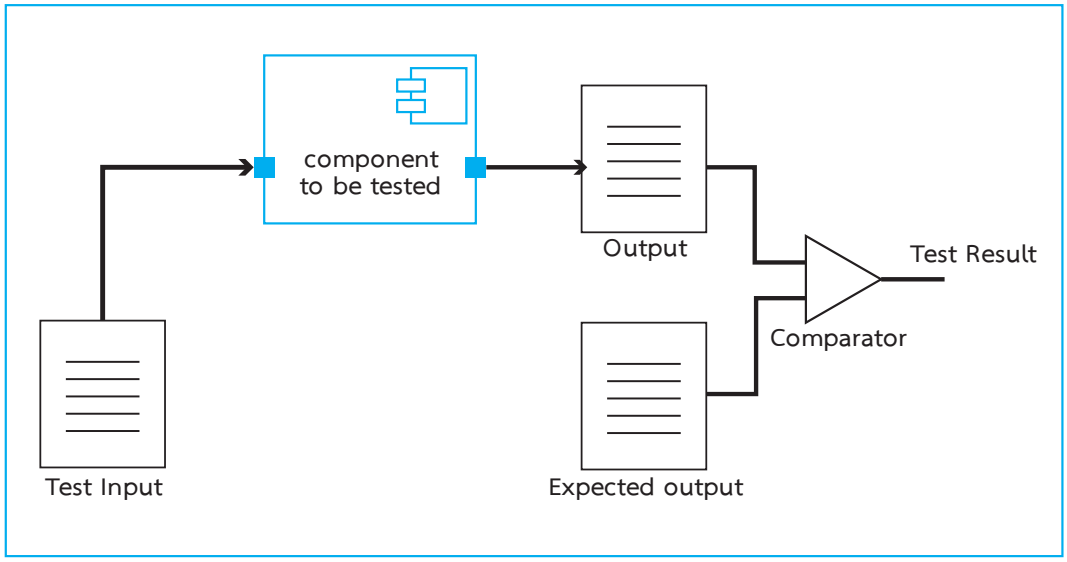
วัตถุประสงค์

- *เข้าใจการทดสอบและการตรวจสอบซอฟต์แวร์*
- *เข้าใจกระบวนการออกแบบชุดทดสอบซอฟต์แวร์*
- *สามารถเลือกรูปแบบการทดสอบให้เหมาะสมกับระบบ*

หลักการทดสอบ

หลักการพื้นฐานในการทดสอบซอฟต์แวร์คือการตรวจสอบให้แน่ใจและเชื่อมั่นได้ว่าซอฟต์แวร์ทำงานถูกต้องตามความต้องการของลูกค้า โดยมีคุณภาพในระดับที่ยอมรับได้ ซึ่งกระบวนการทดสอบซอฟต์แวร์สามารถดำเนินการควบคู่กับกิจกรรมการออกแบบและพัฒนาได้ โดยกิจกรรมเหล่านี้สามารถดำเนินการควบคู่กับกิจกรรมการทดสอบ

หลักพื้นฐานสำหรับการทดสอบซอฟต์แวร์ การตรวจสอบประเมินซอฟต์แวร์ทำงานได้ตรงความต้องการของลูกค้าหรือไม่ ผลลัพธ์จากการทดสอบเป็นไปตามผลลัพธ์ที่คาดหวังไว้หรือไม่ถ้าตรงกันแสดงว่าการทำงานไม่พบข้อผิดพลาด ดังรูปที่ 8.1 ซึ่งเป็นหลักพื้นฐานในการทดสอบระบบซอฟต์แวร์ ทั้งนี้เพื่อให้ทั้งทีมพัฒนาและผู้มีส่วนเกี่ยวของมั่นใจว่าซอฟต์แวร์ที่พัฒนานั้นถูกต้องโดยพยายามมุ่งหาข้อผิดพลาดของซอฟต์แวร์ ด้วยการออกแบบชุดทดสอบที่ดีมาดำเนินการทดสอบ โดยชุดทดสอบที่ดีมีโอกาสที่จะตรวจพบข้อผิดพลาดของซอฟต์แวร์ได้สูง และการทดสอบที่ประสบความสำเร็จอาจหมายถึงการที่สามารถตรวจพบข้อผิดพลาดที่ซ่อนอยู่ได้นั่นเอง เมื่อสิ้นสุดกระบวนการทดสอบที่ดีแล้ว อาจอนุมานได้ว่าข้อผิดพลาดต่าง ๆ ถูกตรวจพบเกือบหมดแล้วซอฟต์แวร์มีความผิดพลาดน้อยและน่าจะนำไปสู่ระดับความน่าเชื่อถือสูง



รูปที่ 8.1 หลักการทดสอบพื้นฐาน

การทดสอบซอฟต์แวร์มีหลักการสำคัญดังนี้ ทำการทดสอบให้ตรงตามความต้องการของลูกค้าเป็นแกนหลักในการทดสอบเพื่อให้มั่นใจว่าซอฟต์แวร์นั้นสามารถสนองความต้องการของลูกค้าได้ครบถ้วน มีการวางแผนการทดสอบล่วงหน้าโดยเฉพาะด้านทรัพยากรที่ต้องใช้ในการทดสอบ ควรเริ่มจากการทดสอบขนาดเล็กแล้วจึงขยายให้ใหญ่ขึ้น เริ่มจากง่ายไปซับซ้อน การทดสอบแบบครบถ้วนทุกกรณีนั้นดำเนินการได้ยากจึงควรมีการกำหนดนิยามระดับการยอมรับได้ของการทดสอบ และการทดสอบนั้นควรจ้างทีมทดสอบอิสระภายนอกมาทำการทดสอบเพื่อลดความขัดแย้งหรือความลำเอียงในการทดสอบ

การทดสอบที่ดีหมายถึงการมีโอกาสในการค้นพบข้อผิดพลาดได้สูง การทดสอบไม่ซ้ำซ้อน การทดสอบไม่ง่ายไป และ ไม่ซับซ้อนจนเกินไป ควรพิจารณาถึงค่าใช้จ่ายในการทดสอบให้มีความคุ้มค่า

กระบวนการวีแอนด์วี (validation and verification)

การทดสอบและตรวจสอบความถูกต้องของซอฟต์แวร์ว่าเป็นไปตามความต้องการของลูกค้าหรือไม่ หรืออาจเรียกว่ากระบวนการวีแอนด์วี (verification and validation: V & V)

การทดสอบหรือทวนสอบซอฟต์แวร์ (verification) ตรวจสอบความถูกต้องของซอฟต์แวร์ว่าสามารถทำงานตามข้อกำหนดคุณลักษณะของซอฟต์แวร์ที่ออกแบบไว้หรือไม่ ทำงานได้ถูกต้องมีประสิทธิภาพผ่านเกณฑ์มาตรฐานที่กำหนด และข้อผิดพลาดอยู่ในเกณฑ์ที่ยอมรับได้ โดยถ้าทดสอบผ่านเกณฑ์ที่กำหนดก็สามารถอนุมานได้ว่าส่วนที่เหลือมีระดับความมั่นใจในความถูกต้องของซอฟต์แวร์สูง

การตรวจสอบซอฟต์แวร์ (validation) เพื่อตรวจสอบความถูกต้องของซอฟต์แวร์ตรงความต้องการหรือไม่เทียบกับเอกสารข้อกำหนดความต้องการซอฟต์แวร์ทั้งในส่วนของข้อกำหนดความต้องการผู้ใช้และข้อกำหนดความต้องการของระบบครบถ้วนทุกความต้องการหรือไม่

แนวทางการทดสอบซอฟต์แวร์

แนวทางการทดสอบซอฟต์แวร์สามารถแบ่งอออกได้เป็นสามกลุ่มใหญ่ ๆ ตามระดับของซอฟต์แวร์ตั้งแต่การทดสอบระดับหน่วย ระดับรวม และ ทั้งระบบ

1. **การทดสอบระดับหน่วย** การทดสอบในระดับหน่วยย่อยพื้นฐานของระบบ ขนาดเล็กโดยทำการทดสอบในระดับ ฟังก์ชัน ระดับคลาส และระดับมอดูล เพื่อทำการค้นหาข้อผิดพลาดของซอฟต์แวร์โดยทำการทดสอบแยกเป็นส่วน ๆ ตามหน่วยต่าง ๆ ด้วยกรรมวิธีกล่องขาวที่ออกแบบและทดสอบโดยนักพัฒนารู้และเข้าใจการทำงานภายในหน่วยอย่างชัดเจน สามารถเข้าถึงรหัสต้นฉบับได้และทำให้สามารถออกแบบชุดทดสอบให้ครอบคลุมทุกส่วนที่ต้องการทดสอบ อีกแนวทางคือการทดสอบแบบก ล่องดำที่ไม่จำเป็นต้องรู้การทำงานภายในหน่วยเพียงแค่ทำการทดสอบและเปรียบเทียบผลลัพธ์ว่าตรงกับผลที่คาดหวังหรือไม่

2. **การทดสอบระดับรวม** การทดสอบโดยการประกอบส่วนต่าง ๆ เข้าด้วยกันแล้วทดสอบ จากเล็กไปใหญ่เป็นการทดสอบในขั้นตอนการประกอบหรือรวมซอฟต์แวร์จากส่วนประกอบต่าง ๆ เข้าด้วยกัน ทดสอบการไหล การทำงานร่วมกันระหว่างส่วนประกอบ ส่วนประสาน ควรทดสอบทุกครั้งที่เพิ่มส่วนประกอบเข้ามาใหม่ โดยในระดับนี้จะมุ่งเน้นที่การทวนสอบและการประกอบสร้างซอฟต์แวร์
3. **การทดสอบทั้งระบบ** ขั้นตอนการทดสอบในระดับสูงโดยจะเน้นที่การตรวจสอบซอฟต์แวร์เทียบกับความต้องการของลูกค้าเป็นหลัก ประกอบกับการทดสอบประสิทธิภาพการทำงานและความน่าเชื่อถือของระบบซอฟต์แวร์

การออกแบบกรณีทดสอบ (test cases design)

การออกแบบกรณีทดสอบเป็นการสร้างชุดทดสอบอย่างมีกลยุทธ์เพื่อให้สามารถทดสอบซอฟต์แวร์ได้อย่างมีประสิทธิภาพ สามารถทดสอบได้อย่างครอบคลุมทุกคุณสมบัติในประมาณชุดที่เหมาะสม ไม่มากหรือน้อยเกินไป โดยการออกแบบกรณีทดสอบสามารถทำได้ทั้งแบบกล่องขาวและกล่องดำ

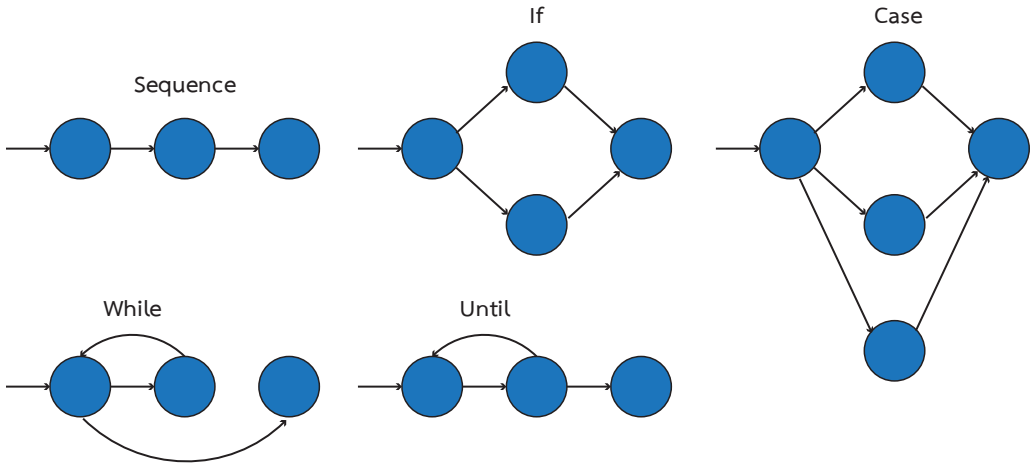
การทดสอบแบบกล่องขาว หรืออาจเรียกว่าการทดสอบแบบกล่องแก้ว ที่ใสสามารถมองเห็นวัตถุที่อยู่ภายในกล่องได้อย่างชัดเจนเช่นเดียวกับการมองการทำงานและรหัสต้นฉบับภายของส่วนที่ต้องการทดสอบอย่างทะลุปรุโปร่ง โดยจะสามารถรับประกันได้ว่าทุก ๆ ส่วนจะถูกดำเนินการในการทดสอบและการตัดสินใจต่าง ๆ ในโครงสร้างของซอฟต์แวร์จะถูกทดสอบทุกกรณีการวนรอบได้รับการทดสอบ โดยชุดทดสอบจะถูกถอดมาจากการวิเคราะห์โครงสร้างของซอฟต์แวร์ การทดสอบแบบกล่องขาวทำได้หลายวิธี เช่น การทดสอบเส้นทางพื้นฐาน Basis path เป็นต้น

การทดสอบแบบกล่องดำ เป็นการทดสอบเชิงพฤติกรรมของซอฟต์แวร์โดยส่วนใหญ่จะดำเนินการในช่วงกลางหรือท้ายของการพัฒนา มีหลักการทดสอบโดยไม่จำเป็นต้องรู้รหัสต้นฉบับหรือการทำงานภายใน เพียงรู้ว่าส่วนของโปรแกรมทำงานอย่างไร ส่วนรับเข้า (input) และผลลัพธ์ (output) จะเป็นอย่างไร ซึ่งจะเน้นการทดสอบการทำงานว่าได้ผลลัพธ์ออกมาตรงกับความต้องการและคุณลักษณะที่กำหนดไว้หรือไม่ การทดสอบแบบกล่องดำนั้นมิได้เป็นทางเลือกในการทดสอบแทนกล่องขาวแต่อย่างใด ควรทดสอบร่วมกันทั้งสองวิธีเพื่อให้สามารถสกัดและตรวจจับความผิดพลาดได้มากที่สุด

การทดสอบเส้นทางพื้นฐาน (basis path testing)

เป็นการทดสอบแบบกล่องขาวที่สามารถรับประกันว่าคำสั่งทั้งหมดในรหัสต้นฉบับจะถูกดำเนินการ อย่างน้อยหนึ่งครั้งเสมอเพื่อให้ครอบคลุมการทดสอบ โดยการวิเคราะห์และสร้างการทดสอบเส้นทางพื้นฐานจะใช้กราฟไหลมาช่วยวิเคราะห์ดังรูปที่ 8.2

กราฟไหล (flow graph) เป็นแบบจำลองที่แสดงโครงสร้างการทำงานของซอฟต์แวร์ผ่านโครงสร้างและการไหล โดยใช้สัญลักษณ์วงกลมแทนจุดต่อ (node) และลูกศรในการแสดงทิศทางการไหลของการทำงาน โดยมีรูปแบบพื้นฐานโครงสร้างกราฟไหล ห้า รูปแบบที่สามารถนำมาแสดงการทำงานใดๆของซอฟต์แวร์ได้ จากโครงสร้างการทำงานตามลำดับ การตัดสินใจ การเลือกจากหลายทางเลือก การวนแบบการตรวจสอบต้นลูป และการวนแบบการตรวจสอบท้ายลูป

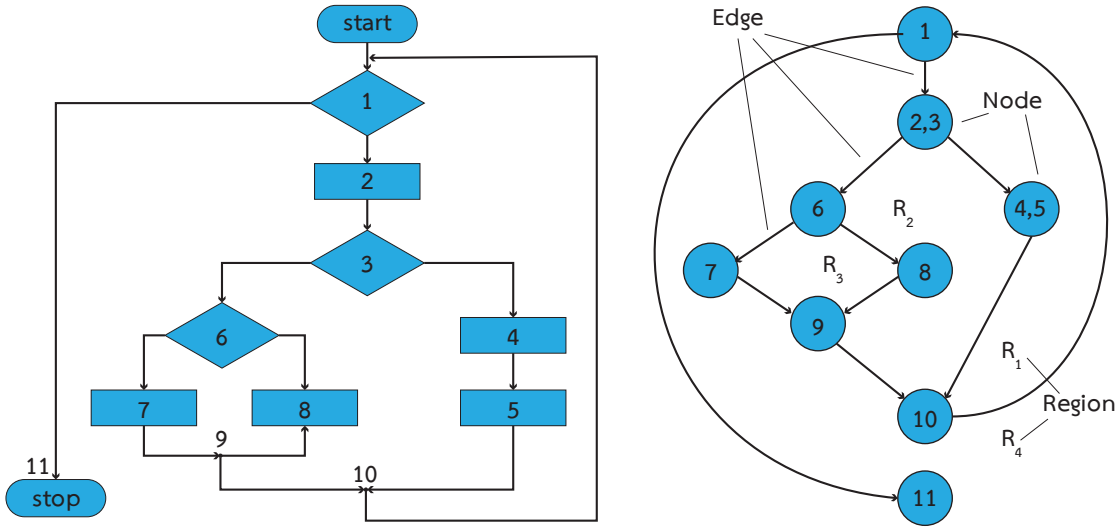


รูปที่ 8.2 กราฟไหลตามรูปแบบโครงสร้าง

ขั้นตอนการสร้างเส้นทางพื้นฐานเริ่มจาก

1. การวิเคราะห์รหัสต้นฉบับเพื่อแปลงให้เป็นผังงาน
2. แปลงผังงานเป็นกราฟไหล
3. คำนวณหาค่าความซับซ้อน (ค่าขอบเขตบนของกรณีทดสอบ)
4. เขียนเส้นทางพื้นฐานจากการเดินในกราฟไหล

การหาการทดสอบเส้นทางพื้นฐานเริ่มจากการนำรหัสต้นฉบับที่ต้องการทดสอบมาเขียนเป็นผังงาน จากนั้นดำเนินการแปลงจากผังงานมาเป็นกราฟไหลตามโครงสร้างของซอฟต์แวร์ที่ปรากฏในผังงาน ดังรูปที่ 8.3



รูปที่ 8.3 ตัวอย่างการแปลงผังงานไปเป็นกราฟไหล [2]

คำนวณหาค่าความซับซ้อนของซอฟต์แวร์ (cyclomatic complexity) เป็นค่าที่ใช้ชี้วัดความซับซ้อนของซอฟต์แวร์ด้วยการคำนวนจาก การนับจำนวนของพื้นที่ (region) ของกราฟไหล การคำนวณจากเส้นเชื่อมและจุดต่อ โดยค่าความซับซ้อนของซอฟต์แวร์เป็นขอบเขตบนของกรณีทดสอบหรือกล่าวคือจำนวนเส้นทางที่จะเดินในกราฟให้ครบทุกเส้นทางเป็นการรับประกันว่าทุกส่วนของซอฟต์แวร์จะถูกดำเนินการเพื่อทดสอบ

ค่าความซับซ้อน = จำนวนของพื้นที่

ค่าความซับซ้อน = เส้นเชื่อม – จุดต่อ + 2

ค่าความซับซ้อน = ขอบเขตบนของกรณีทดสอบ

จากรูปที่ 8.3 การคำนวณความซับซ้อนเพื่อหาเส้นทางพื้นฐานที่จะเดินให้ครบในกราฟไหลเพื่อนำไปสร้างกรณีทดสอบให้ครอบคลุม ตามเส้นทางทั้งหมด

ค่าความซับซ้อน = 4 พื้นที่

ค่าความซับซ้อน = 11 – 9 + 2

ค่าความซับซ้อน = 4

เส้นที่ 1: 1-11

เส้นที่ 2: 1-2-3-4-5-10-1-11

เส้นที่ 3: 1-2-3-6-8-9-10-1-11

เส้นที่ 4: 1-2-3-6-7-9-10-1-11

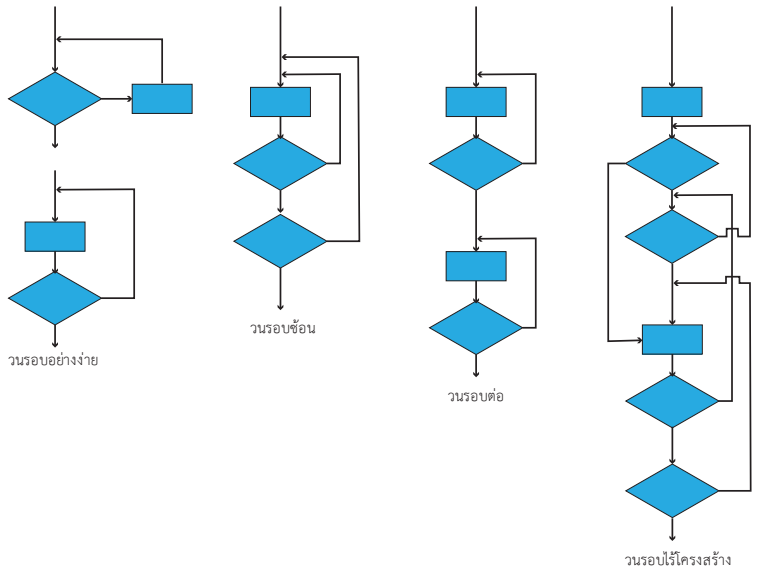
เมื่อได้เส้นทางครบจะดำเนินการวิเคราะห์เพื่อสร้างกรณีทดสอบให้ดำเนินการตามเส้นทางพื้นฐานทั้ง สี่ โดยจะหาส่วนรับเข้า ที่เหมาะสมในแต่ละเส้นทาง ในท้ายที่สุดจะเห็นได้ว่ากรณีทดสอบครอบคลุมทุกเส้นทางและรับประกันได้ว่าทุกๆส่วนของโปรแกรมจะถูกดำเนินการเพื่อทดสอบอย่างน้อยหนึ่งครั้ง ตามหลักการทดสอบแบบกล่องขาวจะเห็นได้ว่าการนำรหัสต้นฉบับมาวิเคราะห์โครงสร้างเพื่อหากรณีทดสอบที่ครอบคลุมนั้นทำให้สามารถทดสอบได้อย่างละเอียดทุก ๆ ส่วนของโปรแกรม ซึ่งเหมาะกับการทดสอบระดับย่อย

การทดสอบตามโครงสร้างควบคุม

โครงสร้างควบคุมของซอฟต์แวร์เป็นส่วนสำคัญที่จะต้องดำเนินการทดสอบในการทดสอบแบบกล่องขาวเนื่องจากเป็นส่วนควบคุมการทำงานหลักซึ่งประกอบด้วย การทดสอบเงื่อนไขการตัดสินใจ และการทดสอบการวนรอบ ดังนั้นการออกแบบกรณีทดสอบจึงควรให้ความสำคัญในการตรวจสอบที่โครงสร้างเหล่านี้

1. **การทดสอบเงื่อนไข** เป็นการทดสอบที่โครงสร้างการตัดสินใจซึ่งเป็นการดำเนินการทางตรรกศาสตร์และคณิตศาสตร์ โดยการทดสอบจะเน้นไปที่การทดสอบเงื่อนไขรูปแบบต่าง ๆ ให้ครอบคลุมทุกกรณีที่จะเป็นไปได้ทั้งการทดสอบนิพจน์ต่าง ๆ การทดสอบตัวดำเนินการ เงื่อนไขตรรกศาสตร์แบบเดี่ยวแบบผสม เพื่อให้มั่นใจว่าได้ทำการทดสอบในทุกความเป็นไปได้ ซึ่งข้อผิดพลาดที่มักจะตรวจพบเสมอ ได้แก่ การผิดพลาดที่ตัวแปรตรรกะ ผิดพลาดที่ตัวดำเนินการ ผิดพลาดที่นิพจน์คณิตศาสตร์
2. **การทดสอบการวนรอบ** การทดสอบโครงสร้างการวนรอบเพื่อตรวจสอบการทำงานให้มั่นใจว่ารอบการทำงานนั้นทำงานถูกต้องในสถานการณ์ต่าง ๆ และในโครงสร้างการวนรอบรูปแบบต่าง ๆ โดยมีแนวทางการทดสอบดังนี้สำหรับการวนรอบอย่างง่าย เมื่อ n คือจำนวนรอบที่มากที่สุดในการวนรอบ
    a. ข้ามลูป
    b. เข้ามาในลูป หนึ่งรอบ
    c. เข้ามาในลูปวน สองรอบ
    d. เข้ามาในลูป m รอบ เมื่อ m<n
    e. ข้ามาในลูป n-1, n, n+1 รอบ

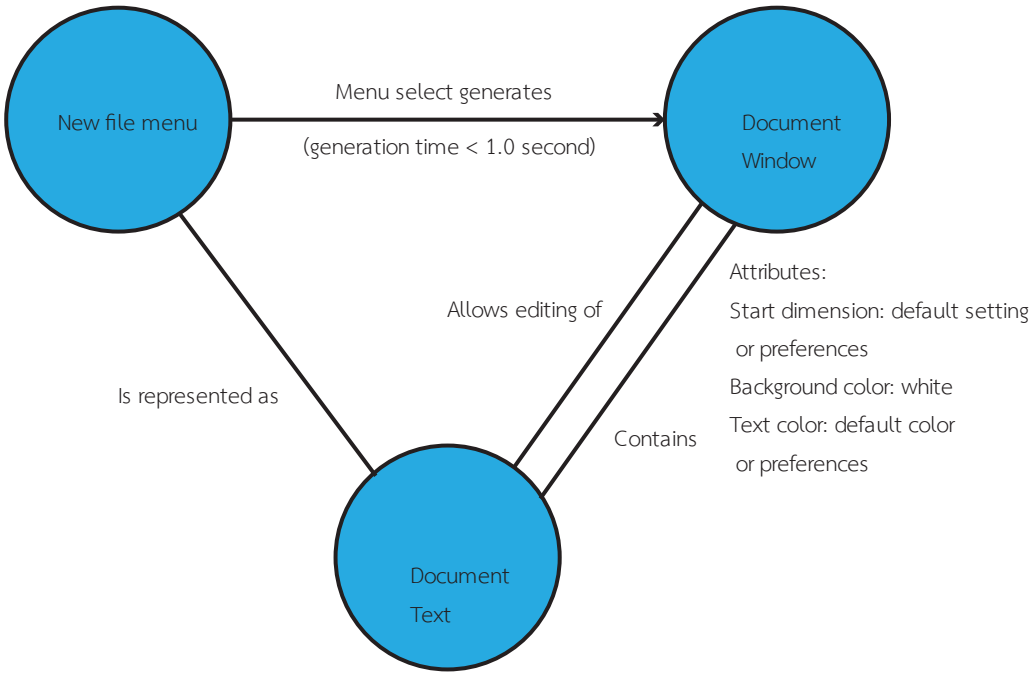
    สำหรับการวนรอบแบบซ้อนก็ดำเนินการทดสอบแบบการวนรอบอย่างง่ายจากลูปในสุดก่อนแล้วจึงขยายออกมาเป็นลำดับจนกว่าจะดำเนินการทดสอบครับทุกลูป การทดสอบลูปแบบวนรอบต่อให้ดำเนินการทดสอบลูปอย่างง่ายแยกกัน และในท้ายที่สุดถ้าการวนรอบแบบไร้โครงสร้างไม่จำเป็นต้องทดสอบเพราะการทดสอบจะซับซ้อนและโครงสร้างเองก็ไม่ดีจึง ควรให้นักพัฒนานำกลับไปปรับปรุงโครงสร้างให้อยู่ในรูปแบบโครงสร้างพื้นฐานก่อนนำมาทดสอบ ดังรูปที่ 8.4



รูปที่ 8.4 โครงสร้างการวนรอบ [2]

การทดสอบด้วยกราฟ

การทดสอบซอฟต์แวร์เริ่มต้นด้วยการสร้างกราฟของวัตถุที่สำคัญและนิยามความสัมพันธ์ จากนั้นวิเคราะห์สร้างชุดการทดสอบที่จะครอบคลุมกราฟเพื่อให้แต่ละวัตถุและความสัมพันธ์ถูกนำมาใช้เพื่อเผยข้อผิดพลาด โดยการวางแผนการทดสอบจะใช้กราฟเป็นแนวทางสำหรับการทดสอบให้ดำเนินการทดสอบตามความสัมพันและคุณลักษณะของกราฟดังตัวอย่างรูปที่ 8.5 แสดงการทดสอบฟังก์ชันการทำงานของซอฟต์แวร์ประเภทการประมวลผลคำ (word processing) สำหรับฟังก์ชันการเริ่มเอกสารใหม่จากเมนูเพื่อสร้างหน้าต่างการทำงานที่มีเอกสารข้อความอยู่ภายในที่สามารถแก้ไขได้ โดยมีคุณลักษณะและการตั้งค่าต่าง ๆ กำหนดไว้ในกราฟตลอดจนข้อกำหนดในการทดสอบที่ต้องนำมาตรวจสอบในขณะดำเนินการทดสอบ



รูปที่ 8.5 การทดสอบด้วยกราฟ

การทดสอบระดับหน่วย (unit testing)

การทดสอบขนาดเล็กในแต่ละหน่วยในระดับ ฟังก์ชัน ระดับคลาส และ ระดับมอดูล เพื่อทำการค้นหาข้อผิดพลาดของซอฟต์แวร์ โดยจะใช้การทดสอบแบบอัตโนมัติเป็นหลักเพื่อตรวจสอบว่าส่วนที่ทำการทดสอบนั้นเป็นไปตามการออกแบบหรือไม่ทำงานได้ตามที่คาดไว้หรือไมโดยการทดสอบระดับหน่วยมักจะเริ่มต้นจากหน่วยเล็ก ๆ ก่อนและค่อย ๆ ขยายการทดสอบในระดับที่ใหญ่และซับซ้อนขึ้น ข้อดีของการทดสอบแบบนี้คือการแยกส่วนต่าง ๆ ออกมาทดสอบ ทำให้ง่ายต่อการแยกปัญหา การทดสอบส่วนย่อยเพิ่มความมั่นใจว่าเมื่อผ่านการทดสอบแล้วนำไปประกอบรวมกันก็จะน่าจะยังคงถูกต้อง ทำให้การรวมส่วนต่าง ๆ และการทดสอบในระดับรวมหน่วยเป็นไปได้ง่าย การทดสอบแบบนี้ยังช่วยโน้มน้าวให้นักพัฒนาทำการรีเฟคเตอริงรหัสต้นฉบับไปในตัว

การทดสอบระดับหน่วยนิยมใช้กรอบการทำงานมาช่วยในการสร้างสภาวะการทดสอบที่เหมาะสมในรูปแบบการทดสอบอัตโนมัติเช่นการใช้ Unit test framework ใน Java ด้วย JUnit เป็นต้น หรือในหลาย ๆ ภาษาก็มีการผนวก การทดสอบระดับหน่วยเข้าไปในตัวภาษาให้นักพัฒนาเรียกใช้งานหรือเปิดการใช้งานได้ เช่น GO Pyhon และ LabVIEW เป็นต้น

ในส่วนของกระบวนการพัฒนานั้น เอ็กพีเป็นกระบวนการที่เน้นการทดสอบระดับหน่วยแบบเข้มข้นด้วยการทดสอบแบบอัตโนมัติร่วมกับหลักการพัฒนาแบบ (test first development) โดยเน้นการออกแบบชุดทดสอบก่อนการพัฒนารหัสต้นฉบับเพื่อให้มั่นใจได้ว่าโปรแกรมจะถูกทดสอบอย่างเข้มข้นและออกมาตรงกับการออกแบบตัวอย่างการเขียนการทดสอบระดับหน่วยแสดงในตาราง 8.1 [13]

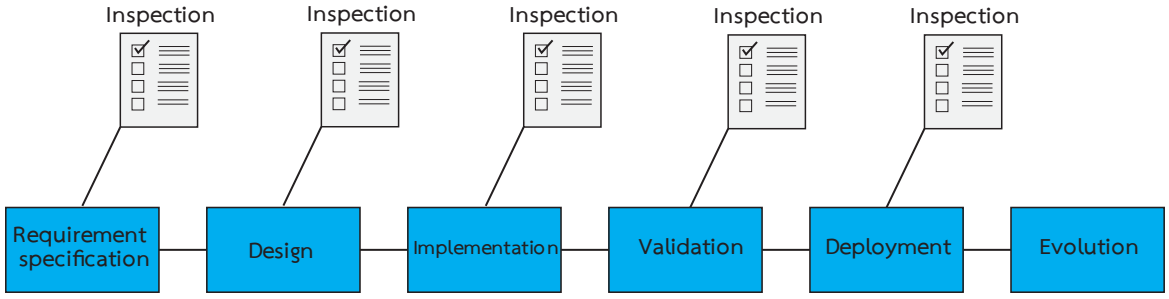
ตาราง 8.1 ตัวอย่าง unit test ด้วย Junit5 framework

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import example.util.Calculator;
import org.junit.jupiter.api.Test;
class MyFirstJUnitJupiterTests {
    private final Calculator calculator = new Calculator();
    @Test
    @DisplayName("Test simple addition  2 = 1+1")
    void addition() {
        assertEquals(2, calculator.add(1, 1), "Simple addition should work");
    }
    @Test
    @DisplayName("Test addition with zero 1 = 1+0")
    void additionWithZero() {
        assertEquals(1, calculator.add(1, 0), "Addition one with zero should be one");
        assertEquals(1, calculator.add(0, 1), "Addition zero with one should be one");
    }
    @Test
    @DisplayName("Test  addition negative and positive 0 = -1+1")
    void additionNegativeAndPositive() {
        assertEquals(0, calculator.add(-1, 1), "Addition should be zero");
    }
⋮
}
```

```
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }
}
```

การตรวจสอบซอฟต์แวร์ (software inspection)

การตรวจสอบซอฟต์แวร์เป็นกระบวนการทดสอบซอฟต์แวร์ในรูปแบบที่เน้นการตรวจสอบซอฟต์แวร์จากเอกสารที่ในการพัฒนาซอฟต์แวร์ตั้งแต่เอกสารความต้องการ คุณสมบัติของซอฟต์แวร์ เอกสารการออกแบบ สถาปัตยกรรมซอฟต์แวร์ รหัสต้นฉบับ และเอกสารการทดสอบ โดยการตรวจสอบนั้นส่วนมากจะกระทำโดยนักตรวจสอบที่มีประสบการณ์มาทำการตรวจสอบโดยสามารถทำการตรวจสอบได้ในทุก ๆ กระบวนการพัฒนาซอฟต์แวร์ดังรูปที่ 8.6



รูปที่ 8.6 การตรวจสอบซอฟต์แวร์

การตรวจสอบความต้องการและคุณสมบัติของซอฟต์แวร์เป็นขั้นตอนในการตรวจสอบเอกสารความต้องการว่าถูกต้องตรงกับความต้องการของลูกค้าจริงหรือไม่ ข้อความในเอกสารถูกต้องชัดเจนสามารถเข้าใจได้ตรงกันทุกฝ่ายหรือไม่ ซึ่งในการตรวจสอบขั้นตอนนี้ถือว่ามีความสำคัญมากเนื่องจากเป็นการตรวจสอบอีกรอบว่าสิ่งที่กำลังจะพัฒนานั้นจะตอบสนองความต้องการได้หรือไม่ นอกจากนั้นยังต้องตรวจสอบไปอีกว่าความต้องการและคุณลักษณะของซอฟต์แวร์อยู่ในวิสัยที่จะดำเนินการพัฒนาภายใต้ข้อกำหนดและงบประมาณหรือไม่ ความต้องการต่าง ๆ อยู่บนความเป็นไปได้หรือไม่

การตรวจสอบในขั้นตอนการออกแบบ นักตรวจสอบสามารถตรวจสอบซอฟต์แวร์ในส่วนของการออกแบบได้จากเอกสารการออกแบบเทียบเคียงกับเอกสารความต้องการ สามารถตรวจสอบหาข้อผิดพลาดในการออกแบบได้ด้วยการวิเคราะห์จากโครงสร้าง สถาปัตยกรรม และ แนวทางแก้ปัญหาที่แสดงในเอกสารการออกแบบ ในการตรวจสอบนั้นจะมุ่งเน้นไปที่การตรวจสอบความสามารถในการตอบสนองความต้องการของลูกค้า ภายใต้ข้อกำหนดต่าง ๆ และเป็นไปตามคุณลักษณะของซอฟต์แวร์หรือไม่ การออกแบบมีความชัดเจนหรือไม่ ขนาด ความซับซ้อน และ แนวทางการทดสอบอยู่ในเกณฑ์ที่เหมาะสมหรือไม่

การตรวจสอบในขั้นตอนการพัฒนาเป็นการตรวจสอบเพื่อดูว่าการพัฒนานั้นเป็นไปตามการออกแบบ ตรงกับการออกแบบ รหัสต้นฉบับอยู่ในเกณฑ์มาตรฐานที่เหมาะสม เอกสารการพัฒนาครบถ้วน สมบูรณ์ และ ถูกต้องหรือไม่

การตรวจสอบในกระบวนการทดสอบจะทำการตรวจสอบกรรมวิธีที่ใช้ทดสอบตามเอกสารการทดสอบโดยตรวจความถูกต้องในการทดสอบ กรณีทดสอบ ข้อมูลที่ใช้ในการทดสอบ ตลอดจนกรรมวิธีในการทดสอบเพื่อให้มั่นใจว่ากระบวนการทดสอบจะมีความถูกต้อง เที่ยงตรง และ โปร่งใส

การตรวจสอบเพื่อการส่งมอบสามารถดำเนินการเพื่อตรวจเช็คความสมบูรณ์ถูกต้องของผลิตภัณฑ์ที่จะถูกส่งมอบให้ลูกค้าว่ามีความครบถ้วนตรงที่ตกลงกันไว้ในสัญญา ตรงตามความต้องการ และ เป็นไปตามข้อกำหนดคุณลักษณะซอฟต์แวร์หรือไม่ ผลิตภัณฑ์ เอกสารต่าง ๆ คู่มือการติดตั้ง คู่มือการใช้งาน และ เอกสารในการบำรุงรักษาซอฟต์แวร์มีความถูกต้อง

การตรวจสอบซอฟต์แวร์นั้นส่วนใหญ่จะดำเนินการผ่านการตรวจจากเอกสารเป็นหลัก โดยที่ผู้ตรวจไม่จำเป็นต้องปฏิสัมพันธ์กับผู้มีส่วนเกี่ยวข้อง ว่าไปตามเอกสาร โดยผลลัพธ์จากการตรวจสอบจะอยู่ในรูปของรายงานการตรวจสอบที่จะระบุถึงข้อผิดพลาด ข้อสงสัยในจุดที่ไม่ชัดเจน ข้อแนะนำ เพื่อใช้ในการปรับปรุงการพัฒนาให้ถูกต้องและเป็นไปตามมาตรฐาน

แนวทางการทดสอบซอฟต์แวร์สำหรับอุปกรณ์เคลื่อนที่

ซอฟต์แวร์บนอุปกรณ์เคลื่อนที่มีอัตราการเติบโตสูงและมีการใช้งานอย่างแพร่หลายบนทั้งบนสมาร์ทโฟนและแทปเล็ต ซึ่งสามารถใช้การทดสอบเช่นเดียวกับการทดสอบซอฟต์แวร์อื่น ๆ เพียงแต่ในบางสถานการณ์สำหรับการใช้งานที่อาจจะต่างออกไปกับการใช้งานซอฟต์แวร์คอมพิวเตอร์เช่น การทดสอบฟังก์ชันในการรับรู้สภาวะแวดล้อมของผู้ใช้ การเข้าถึงบริการต่าง ๆ การรับรู้ตำแหน่งและตัวรับรู้แบบต่าง ๆ และรูปแบบการปฏิสัมพันธ์ทำให้การออกแบบการทดสอบซอฟต์แวร์สำหรับอุปกรณ์เคลื่อนที่จึงมีความแตกต่างไปบ้าง โดยมีแนวทางการทดสอบ ดังต่อไปนี้

- ทดสอบทั้งบนโปรแกรมเลียนแบบ (emulator) และบนอุปกรณ์จริง
- ทดสอบกับสภาพแวดล้อมจริง เครือข่ายจริง ตามเงื่อนไข และ สถานการณ์รูปแบบต่างๆ
- ทดสอบกับอุปกรณ์ที่หลากหลาย เพื่อให้ครอบคลุม และความเข้ากันได้
- ใช้บริการห้องปฏิบัติการทดสอบ เพื่อทดสอบอุปกรณ์ได้อย่างครอบคลุม
- ทดสอบประสิทธิภาพการทำงานของซอฟต์แวร์
- ทดสอบ GUI และการปฏิสัมพันธ์
- ทดสอบด้านข้อมูล
- ทดสอบด้านความปลอดภัย
- ทดสอบประสบการณ์ผู้ใช้
- ทดสอบการเชื่อมต่อ เครือข่ายไร้สายทั้ง WIFI และ Cellular
- ทดสอบให้ผ่านตามมาตรฐานของแพลตฟอร์มเพื่อให้สามารถ เผยแพร่บนร้านค้าของแพลตฟอร์มได้

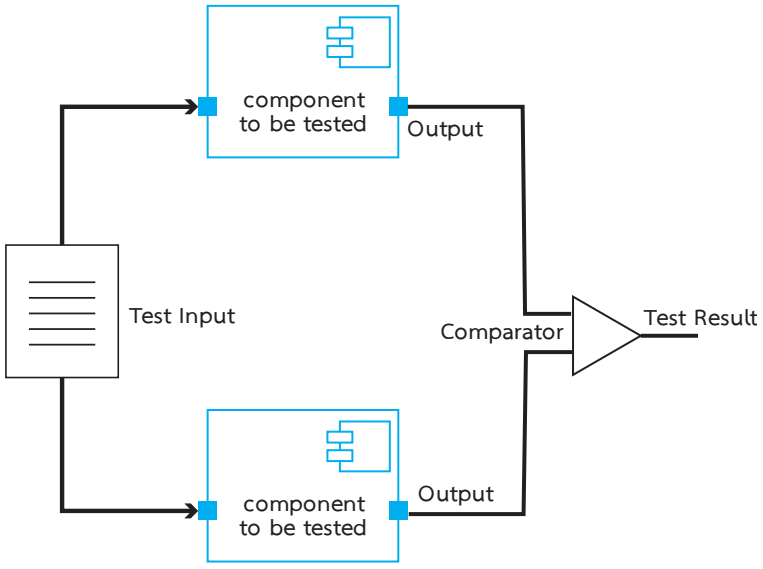
การทดสอบซอฟต์แวร์สำหรับอุปกรณ์เคลื่อนที่นั้นจะเน้นที่การทดสอบการใช้งานจริงสภาพแวดล้อมจริงประกอบกับประสบการณ์ผู้ใช้ ประสิทธิภาพและความเข้ากันได้กับอุปกรณ์ เพื่อให้มั่นใจได้ว่าเมื่อนำไปเผยแพร่บนร้านค้าแล้วผู้ใช้จะได้รับประสบการณ์ที่ดี ได้ใช้งานที่ผ่านการทดสอบมาแล้วมีข้อผิดพลาดต่ำ และผลิตออกมาผ่านมาตรฐานของแต่ละแพลตฟอร์ม ซึ่งในแต่ละแพลตฟอร์มจะมีแนวทางการออกแบบและพัฒนาซอฟต์แวร์สำหรับแต่ละแพลตฟอร์มเผยแพร่เป็นแนวทางให้กับนักพัฒนาในสร้างซอฟต์แวร์ให้ผ่านมาตรฐานของแต่ละแพลตฟอร์มนั่นเอง การตรวจสอบขั้นสุดท้ายก่อนการเผยแพร่จะถูกดำเนินการโดยแพลตฟอร์มร้านค้า เพื่อให้มั่นใจว่าซอฟต์แวร์ที่จะถูกเผยแพร่นั้นมีคุณภาพ ประสิทธิภาพ และ ความปลอดภัยดีพอตามมาตรฐาน

แนวทางการทดสอบระบบเวลาจริง

ซอฟต์แวร์ระบบเวลาจริงนั้นแตกต่างจากซอฟต์แวร์ประเภทอื่น ๆ ซอฟต์แวร์ส่วนใหญ่ทำงานบนระบบฝังตัวแบบเวลาจริงทำงานตลอดเวลา โดยมักจะเริ่มทำงานเมื่อจ่ายกระแสไฟเลี้ยงเข้าไปยังระบบ เป็นการทำงานแบบอัตโนมัติ บางระบบก็สามารถควบคุมการทดสอบได้ สามารถเข้าถึงได้ แต่ส่วนมากจะเข้าถึงการควบคุมเพื่อทดสอบได้ยาก เช่น นักพัฒนาอาจจะไม่สามารถหยุดสัญญาณนาฬิกาเพื่อตรวจสอบค่าตัวแปร หรือหยุดระบบเพื่อตรวจดูการทำงานได้ ดังนั้นแนวทางการทดสอบระบบเวลาจริง ดังนี้

- ทดสอบทั้งบนโปรแกรมเลียนแบบ (emulator) หรือระบบจำลอง (simulator) ถ้ามี และบนระบบจริง
- ทดสอบกับสภาพแวดล้อมจริง การทำงานจริงโดยการสังเกตจากผลการทำงาน
- ใช้การบันทึกล็อก เพื่อนำมาใช้ประกอบการทดสอบ
- ตรวจสอบสัญญาณต่าง ๆ เพื่อประกอบการทดสอบ เช่นการใช้เครื่องวิเคราะห์สัญญาณตรรกะ (logic analyzer)
- ติดตั้งอุปกรณ์เพื่อใช้ในการทดสอบ เช่นการเพิ่มปุ่มเพื่อควบคุมการทำงาน
- ทดสอบประสิทธิภาพการทำงานของซอฟต์แวร์
- ทดสอบด้วยวิธีกล่องดำ
- ทดสอบจากข้อมูล

การทดสอบซอฟต์แวร์ในระบบเวลาจริงนั้นจะเน้นการทดสอบในสภาพการทำงานจริงเวลาจริงเพื่อทดสอบการทำงานของทั้งฮาร์ดแวร์และซอฟต์แวร์ โดยส่วนใหญ่จะใช้การบันทึกล็อกและนำมาวิเคราะห์เพื่อทดสอบการทำงาน ประกอบกับการทดสอบการทำงานจริงโดยเปรียบเทียบผลลัพธ์จากการทำงานที่ได้เทียบกับผลลัพธ์ที่คาดหวังไว้จากกรณีทดสอบ หรือในระบบที่ต้องการความน่าเชื่อถือสูงอาจใช้การทดสอบแบบเปรียบเทียบ (comparision testing) เพื่อตรวจสอบการทำงานจากสองระบบที่จะต้องได้ผลลัพธ์ที่ตรงกันออกมาดังรูปที่ 8.7 เช่นในระบบที่ต้องการความน่าเชื่อถือสูงเช่นระบบควบคุมวิถีการโคจรยานอวกาศเป็นต้น



รูปที่ 8.7 การทดสอบแบบเปรียบเทียบ

บทที 9 ความทนต่อความผิดพร่องของซอฟต์แวร์ Software fault tolerance

วัตถุประสงค์

- *เข้าใจการความน่าเชื่อถือของซอฟต์แวร์*
- *เข้าใจความทนทานต่อข้อผิดพลาด*
- *สามารถเลือกรูปแบบการวัดและประเมินความน่าเชื่อถือของซอฟต์แวร์ได้*

ความเชื่อถือได้ของซอฟต์แวร์ (software reliability)

ความน่าเชื่อถือของซอฟต์แวร์โดยทั่วไปจะหมายถึงความคาดหวังด้านความเชื่อถือจากผู้ใช้ต่อซอฟต์แวร์ คาดหวังว่าจะใช้งานได้ สามารถรับบริการจากซอฟต์แวร์เมื่อต้องการและยอมรับในความผิดพลาดในระดับที่ผู้ใช้ยอมรับได้ แต่ในระบบที่ต้องการความน่าเชื่อถือสูง ความคาดหวังมักจะสูงตามไปด้วย เช่น ระบบซอฟต์แวร์ทางด้านการแพทย์ การทหาร และ ระบบการบินเป็นต้น

เมื่อกล่าวถึงความน่าเชื่อถือจึงต้องทำความเข้าใจประเภทของความผิดพลาดได้แก่ ข้อผิดพลาดจากมนุษย์ซึ่งเกิดได้จากผู้พัฒนาและผู้ใช้งานเช่นระบบการป้อนวันเดือนปีที่ผู้ใช้อาจกรอกรูปแบบที่แตกต่างกัน ข้อผิดพร่องจากระบบ (system fault) เป็นส่วนที่สามารถนำไปสู่ข้อผิดพลาดของระบบ (system error) เช่นไม่ได้ตรวจสอบปีว่าอยู่ในช่วงคศ. ในระบบสากลหรือไม่ หากผู้ใช้ใส่เป็นพศ. ทำให้ข้อผิดพร่องนั้นสามารถนำไปสู่ข้อผิดพลาดของระบบและเมื่อระบบดำเนินการข้อผิดพลาดจะสามารถนำไปสู่ความล้มเหลวของระบบ (system failure) ได้ซึ่งนำไปสู่การที่ซอฟต์แวร์ไม่สามารถให้บริการตามการร้องขอได้ ไม่สามารถตอบสนองความต้องการตามความคาดหวังของผู้ใช้ได้ เช่น ไม่สามารถคำนวณอายุที่ถูกต้องให้ได้ระบบล้มเหลวเนื่องจากการใช้ปีพศ. ในการคำนวนอายุในระบบการคำนวนด้วยปีคศ. ทำให้อายุที่คำนวณออกมาติดลบส่งผลต่อกระบบต่อการคำนวณอื่น ๆ ทำให้ระบบล้มเหลว

ความล้มเหลว (failure) มักจะเกิดจากข้อผิดพลาดต่าง ๆ ในระบบซึ่งก็เกิดจากข้อผิดพร่องอีกทีต่อเนื่องกัน ซึ่งหากพิจารณาจากระดับความรุ่นแรงจากน้อยไปมากตามลำดับดังนี้ ข้อผิดพร่อง ข้อผิดพลาด และความล้มเหลว แต่อย่างไรก็ตามในบางข้อผิดพร่องก็ไม่นำไปสู่ข้อผิดพลาด เช่นเดียวกับข้อผิดพลาดก็ไม่ได้นำไปสู่ความล้มเหลวเสมอ แนวทางจัดการปัญหาเหล่านี้สามารถทำได้โดยการเริ่มจัดการตั้งแต่ข้อผิดพร่อง โดยการจัดการตามหลัก สาม ประการดังต่อไปนี้

1. การหลีกเลี่ยงข้อผิดพร่อง (fault avoidance) สามารถทำได้โดยการพัฒนาระบบที่ลดและหลีกเลี่ยงข้อผิดพลาดจากมนุษย์ให้ได้มากที่สุด มีกระบวนการพัฒนาที่รองรับการจัดการกับช้อผิดพร่องเพื่อให้มั่นใจได้ว่าจะสามารถจัดการข้อผิดพร่องได้ก่อนการส่งมอบซอฟต์แวร์ให้กับลูกค้า โดยทีมพัฒนาต้องใช้กลยุทธ์ในการพัฒนาไม่ว่าจะเป็นการลดความผิดพลาดต่าง ๆ ที่อาจเกิดขึ้นให้มากที่สุหรือการดักจับความผิดพลาดต่าง ๆ ที่อาจจะนำไปสู่ข้อผิดพร่อง
2. การตรวจหาข้อผิดพร่อง (fault detection) เป็นส่วนหนึ่งในกระบวนการทดสอบซอฟต์แวร์ซึ่งการตรวจหาข้อผิดพร่องจะดำเนินการในกระบวนการวีแอนวีอยู่แล้ว เพื่อตรวจหาและจัดการแก้ไขข้อผิดพร่องที่พบในระหว่างกระบวนการพัฒนา โดยเมื่อตรวจพบข้อผิดพร่องทีมพัฒนาก็ควรจัดการแก้ไขเพื่อกำจัดข้อผิดพร่องให้ได้มากที่สุด

3. ความทนต่อความผิดพร่อง (fault tolerance) เป็นการออกแบบและพัฒนาซอฟต์แวร์ให้สามารถทนต่อความผิดพร่องได้เพื่อหลีกเลี่ยงความล้มเหลวของระบบเมื่อลูกค้านำไปใช้งานจริง

การวัดความเชื่อถือได้ของซอฟต์แวร์สามารถพิจารณาได้จากสองมุมมองหลักคือจากสภาพพร้อมใช้งาน (availability) และความเชื่อถือได้ (reliability) โดยสามารถนำเสนอมาในเชิงปริมาณได้เช่นสภาพพร้อมใช้งานที่ 99.9% ของเวลาการทำงาน โดยความเชื่อถือได้ของซอฟต์แวร์จะถูกระบุไว้ในข้อกำหนดคุณลักษณะของซอฟต์แวร์เพื่อใช้อ้างอิงในการทบสอบซอฟต์แวร์ให้เป็นไปตามความต้องการและข้อกำหนด

ความต้องการความเชื่อถือได้ของซอฟต์แวร์ จะนิยามถึงความต้องการของระบบที่สามารถหลีกเลี่ยงตรวจหาและทนต่อความผิดพร่องในรูปแบบเชิงฟังก์ชันและไม่เชิงฟังก์ชันโดยจะครอบคลุมไปส่วนความล้มเหลวของฮาร์ดแวร์และข้อผิดพลาดในการดำเนินการ เพื่อให้ครอบคลุมทั้งระบบ โดยนิยมใช้ตัวชี้วัดเชิงปริมาณในการกำหนดขอบเขตและระดับความล้มเหลวที่ยอมรับได้ เพื่อให้ผู้มีส่วนเกี่ยวข้องเข้าใจตรงกัน ใช้เป็นเกณฑ์พื้นฐานในการทดสอบ หรือเป็นไปตามข้อกำหนดจากภายนอกเช่นตามข้อบังคับของหน่วยงานเช่น ข้อบังคับความปลอดภัยมาตรฐานการบินพลเรือน โดยการใช้หน่วยวัดพื้นฐานดังต่อไปนี้

1. ความน่าจะเป็นของความล้มเหลวตามความต้องการ (probability of failure on demand: POFOF) เป็นค่าความน่าจะเป็นที่ใช้บ่งชี้โอกาสที่ระบบจะล้มเหลวเมื่อมีการร้องขอการให้บริการในหน่วยของความถี่ ซึ่งจะเป็นตัวกำหนดคุณลักษณะว่าจะมีความน่าจะเป็นของความถี่ในการล้มเหลวเท่าไร ส่วนมากจะใช้หน่วยวัดนี้ในระบบที่เน้นในเรื่องของความเชื่อถือได้ของระบบโดยเฉพาะในส่วนที่วิกฤตและในส่วนของความปลอดภัย
2. อัตราการเกิดข้อผิดพร่อง (rate of fault occurrence: ROCOF) เป็นค่าที่สะท้อนอัตราการเกิดข้อผิดพร่องต่อหน่วยเวลา เช่น ระบบสัญญาณกันขโมยมีค่า ROCOF เป็น 0.001 หมายความว่าล้มเหลวได้ 1 ใน 1000 ชั่วโมง
3. สภาพพร้อมใช้งาน (availability) เป็นค่าหน่วยวัดสภาพการพร้อมใช้งานในหน่วยเวลาเช่น 0.999 หมายถึงระบบจะอยู่ในสภาพพร้อมใช้งานพร้อมให้บริการ 999 หน่วยจาก 1000 หน่วยเวลา โดยส่วนใหญ่มักจะใช้หน่วยชม. ซึ่ง 0.999 คำนวณได้ว่าระบบจะไม่อยู่ในสภาพพร้อมใช้งานเป็นเวลา 84 วินาทีในรอบ 24 ชั่วโมง โดยหน่วยวัดสภาพพร้อมใช้งานนิยมใช้กับระบบที่ทำงานต่อเนื่องไปตลอด เช่น ระบบรักษษความปลอดภัยหรือระบบการสื่อสาร

ตาราง 9.1 ตัวอย่าง ความต้องการความเชื่อถือได้ของซอฟต์แวร์

ความต้องการความเชื่อถือ1	ระบบต้องทำการตรวจสอบอินพุททุกค่าก่อนการดำเนินการ โดยการตรวจสอบเทียบกับค่าที่กำหนดไว้ตามรูปแบบของอินพุทประเภทต่างๆ
ความต้องการความเชื่อถือ2	ระบบต้องดำเนินการแบบขนานด้วยฮาร์ดแวร์ที่แตกต่างกันเพื่อให้ทำงานได้อย่างต่อเนื่องในกรณีเกิดความล้มเหลวด้านฮาร์ดแวร์
ความต้องการความเชื่อถือ3	ระบบต้องพัฒนา ซอฟต์แวร์ N รุ่นสำหรับส่วนที่วิกฤตของซอฟต์แวร์ เช่นระบบ Auto pilot
ความต้องการความเชื่อถือ4	ระบบต้องมีการสำรองข้อมูลเก็บไว้มากกว่า 1 สถานที่

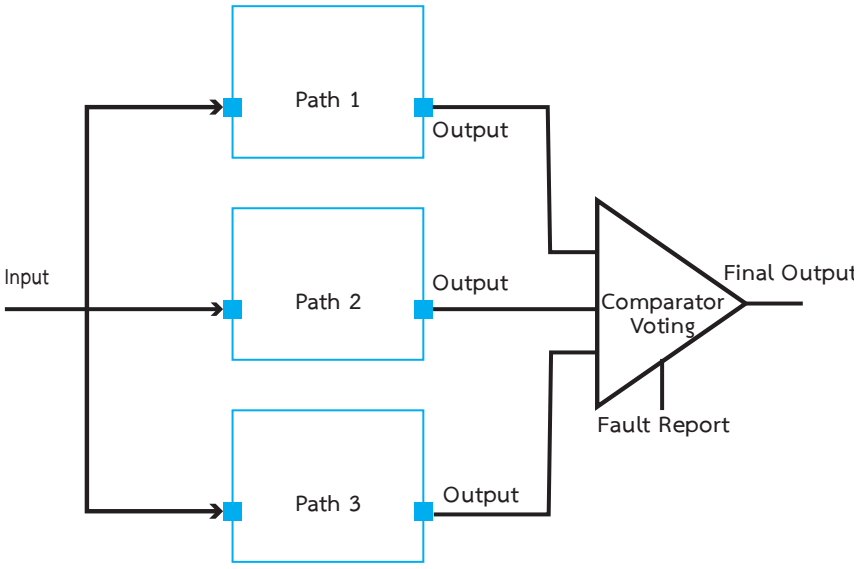
ความทนต่อความผิดพร่อง (fault tolerance)

ความทนต่อข้อผิดพร่องมักเป็นคุณสมบัติหลักที่ต้องมีในระบบที่มีส่วนที่วิกฤตต้องการความน่าเชื่อถือสูงทนต่อข้อผิดพลาดและความผิดพร่องเพื่อให้ระบบนั้นยังสามารถให้บริการได้อย่างต่อเนื่องเช่น ระบบตรวจสัญญาณชีพในห้องฉุกเฉิน ระบบควบคุมการบิน ระบบโทรคมนาคม โดยส่วนใหญ่ต้องการทั้งความเชื่อถือได้สูง และสภาพพร้อมใช้งานที่สูงด้วย ส่วนมากการออกแบบและพัฒนาระบบที่มีความทนต่อความผิดพร่องจะอยู่บนพื้นฐานความซ้ำซ้อน (redundancy) และความแตกต่าง (diversity)

สถาปัตยกรรมที่มีความทนต่อความผิดพร่องมีหลายรูปแบบขึ้นอยู่กับการออกแบบและการนำไปประยุกต์ใช้งานโดยจะตั้งอยู่บนพื้นฐานความซำซ้อนและความแตกต่างเป็นหลัก วิศวกรซอฟต์แวร์ต้องเลือกประยุกต์ใช้สถาปัตยกรรมให้เหมาะสมกับปัญหาและความต้องการของระบบเลือกพิจารณาจากลำดับความสำคัญจากส่วนที่วิกฤตมากที่สุดและลดลงไปตามลำดับ

สถาปัตยกรรมแบบปกป้อง เป็นสถาปัตยกรรมที่เน้นการเข้าควบคุมระบบในกรณีฉุกเฉินแบบอัตโนมัติ เช่น ระบบป้องกันลิฟท์ตกที่จะทำการหยุดการทำงานและล็อกลิฟท์ค้างไว้ในกรณีเกิดเหตุฉุกเฉินหรือเกิดเหตุขัดข้อง ระบบคอมพิวเตอร์ปรับลดความเร็วลงหรือปิดตัวเองลงเมื่อซอฟต์แวร์ระบบตรวจพบอุณภูมิอุปกรณ์ภายในที่สูงจนอาจก่อให้เกิดความเสียหาย โดยระบบปกป้องจะทำเฝ้าสังเกตการทำงานและสภาพแวดล้อมของระบบและเมื่อตรวจพบปัญหาจะดำเนินการตามแผนที่เตรียมไว้ในกรณีฉุกเฉิน โดยระบบปกป้องจะมีความซ้ำซ้อนด้วยการเพิ่มระบบเฝ้าสังเกตการทำงานควบคู่กับระบบควบคุมและควรใช้เทคโนโลยีที่แตกต่างกัน

สถาปัตยกรรมเฝ้าสังเกตเอง เป็นสถาปัตยกรรมที่มีช่องการทำงานมากกว่าหนึ่งทาง และตรวจสอบตัวเองจากการเปรียบเทียบผลลัพธ์ที่ได้ออกมาโดยหากผลลัพธ์ตรงกันก็อนุมานได้ว่าการดำเนินการถูกต้องหากไม่ตรงกันแสดงว่ามีคความล้มเหลวเกิดขึ้น ดังนั้นการจึงนิยมใช้ช่องการดำเนินการมากกว่า สอง ทางและเป็นเลขคี่เพื่อหากเกิดผลลพัธ์ไม่ตรงกันระบบจะเลือกใช้ผลลัพธ์ที่ตรงกันจากช่วงทางส่วนใหญ่ โดยใช้ สาม ช่องทางเป็นที่นิยม ดังรูป 9.1 โดยในแต่ละทางควรมีความแตกต่างทั้งฮาร์ดแวร์และซอฟต์แวร์ โดยสามารถประยุกต์ใช้ชุดระบบเฝ้าสังเกตเองหลาย ๆ ชุดพร้อม ๆ กันในกรณีที่ระบบต้องการสภาพพร้อมใช้งานที่สูง โดยจะเห็นว่าระบบจะเป็นการทำงานแบบขนานที่มีความซ้ำซ้อนสำรองสำหรับการทำงานที่สูง



ดังรูปที่ 9.1 ระบบเฝ้าสังเกตเอง

ระบบการพัฒนาด้วย N รุ่น มีหลักการคล้ายกับระบบเฝ้าสังเกตเองคือการทำงานของซอฟต์แวร์แบบขนานหลาย ๆ รุ่นและนำผลลัพธ์มาเปรียบเทียบหาความผิดพร่อง นิยมใช้ตรวจสอบจากความเหมือนข้างมากเป็นหลัก ด้วยการใช้สามส่วนเพื่อเพิ่มความซ้ำซ้อนให้ทนต่อข้อผิดพร่องได้มากขึ้น โดยซอฟต์แวร์แต่ละรุ่นที่นำมาดำเนินการควรมีความแตกต่างทั้งทีมพัฒนา ภาษาและเครื่องมือที่ใช้พัฒนา อัลกอริทึมที่ใช้ และ กระบวนการพัฒนา

การนำความทนต่อความผิดพร่องมาใช้สามารถช่วยเพิ่มความเชื่อถือและสภาพพร้อมใช้งานของซอฟต์แวร์ ซึ่งเป็นส่วนสำคัญในระบบที่มีส่วนวิกฤต แต่การดำเนินการเหล่านี้มีค่าใช้จ่ายที่เพิ่มขึ้นมาซึ่งวิศวกรซอฟต์แวร์ต้องสามารถวิเคราะห์ความคุ้มค่าที่ต้องแลกมาด้วยค่าใช้จ่ายและความจำเป็นนั้นคุ้มค่าหรือไม่

การจำแนกข้อบกพร่อง (defect classification)

ข้อบกพร่องที่เกิดขึ้นในระบบซอฟต์แวร์นั้นมีสาเหตุมาจากหลายส่วนด้วยกัน โอกาสการเกิดก็กระจายไปตามส่วนต่าง ๆ ซึ่งสามารถจำแนกความบกพร่องจากสาเหตุได้ สาม ส่วนหลักคือความบกพร่องจากมนุษย์ ความบกพร่องจากฮาร์ดแวร์ และ ความบกพร่องจากซอฟต์แวร์ นอกจากนี้ยังสามารถจำแนกประเภทข้อบกพร่องของซอฟต์แวร์ออกเป็นกลุ่ม ดังนี้

จำแนกข้อบกพร่องทั่วไป เป็นข้อบกพร่องพื้นฐานที่มักพบได้บ่อยในทุก ๆ กระบวนการพัฒนาซอฟต์แวร์ตั้งแต่ความต้องการไปจนถึงการส่งมอบซอฟต์แวร์

- ความต้องการไม่ครบถ้วนหรือไม่ชัดเจน
- ความต้องการกำกวมไม่คงเส้นคงวา
- ความต้องการไม่อยู่บนพิสัยที่จะพัฒนาได้
- ฟังก์ชันการทำงานบกพร่อง
- บกพร่องด้านข้อมูลและสารสนเทศ
- บกพร่องทางด้านประสิทธิภาพ
- บกพร่องในด้านการใช้งาน
- บกพร่องในความเข้ากันได้
- บกพร่องด้านความปลอดภัย

จำแนกข้อบกพร่องตามความรุ่นแรง ข้อบกพร่องสามารถจำแนกตามระดับความรุนแรงเทียบเคียงได้จากมาตฐานซอฟต์แวร์และสามารถนิยามระดับความรุนแรงโดยวิศวกรซอฟต์แวร์ สามารถดูจากกรณีทดสอบและฟังก์ชันต่าง ๆ พิจารณาร่วมกับข้อผิดพลาดต่าง ๆ กระทบมากน้อยกับระบบเพียงใด และฟังก์ชันที่กระทบนั้นสำคัญต่อระบบแค่ไหน ขนาดเป็นอย่างไร เรียงระดับความรุ่นแรงจากมากไปน้อย ดังนี้ ระดับวิกฤต ระดับสำคัญ ระดับหลัก ระดับรอง และ ระดับไม่สำคัญ

- ระดับวิกฤต อยู่ในระดับที่ระบบไม่สามารถดำเนินการต่อได้ ไม่สามารถทดสอบได้ โดยกระทบในส่วนหลักที่วิกฤต
- ระดับหลัก ข้อบกพร่องทำให้ฟังก์ชันหลักไม่สามารถทำงานได้ แต่ยังสามารถทดสอบได้
- ระดับรอง ข้อบกพร่องพบในบางส่วนของฟังก์ชันหรือส่วนย่อยของมอดูล โดยที่ฟังก์ชันการทำงานหลักยังทำงานได้
- ระดับไม่สำคัญ เป็นข้อบกพร่องเล็กๆน้อย ๆ ที่ไม่กระทบกับการทำงาน เช่น การจัดวางส่วนประสานผู้ใช้ การเลือกใช้สี

จำแนกข้อบกพร่องตามความสำคัญ ข้อบกพร่องสามารถจำแนกตามตวามสำคัญไล่เรียงจากลำดับสำคัญสูงลงไปถึงลำดับความต่ำ โดยจะนำมาใช้กำหนดลำดับข้อบกพร่องที่ควรแก้ไข ซึ่งโดยปกติจะถูกจัดลำดับโดยทีมประกันคุณภาพและผู้ใช้ เพื่อเป็นแนวทางแก่ทีมพัฒนาในการจัดการข้อบกพร่อง

- ดำเนินการทันที ข้อบกพร่องต้องได้รับการแก้ไขทันที โดยข้อบกพร่องอื่นๆรอไว้ก่อน
- ฉุกเฉิน ข้อบกพร่องถูกจัดลำดับการแก้ไขก่อนลำดับอื่นๆ
- ระดับสูง ข้อบกพร่องควรถูกแก้ไขโดยเร็วเท่าที่จะทำได้
- ระดับปกติ สามารถแก้ไขหลังจากปล่อยซอฟต์แวร์ หรือรอแก้ในรอบการพัฒนาถัดไป
- ระดับต่ำ ดำเนินการแก้ไขหลังจากข้อบกพร่องอื่นๆถูกแก้ไขแล้ว

จำแนกข้อบกพร่องตามข้อผิดพลาดทั่วไป ข้อบกพร่องจำแนกจากข้อผิดพลาดทั่วไปจาก กระบวนการพัฒนา เครื่องมือพัฒนา เทคโนโลยี และ มนุษย์ ซึ่งทั้งนักพัฒนาและนักทดสอบจะตรวจจากข้อบกพร่องที่มักจะพบได้เสมอ ๆ เหล่านี้ โดยใช้การตรวจหาแบบย้อนกระบวนการจากการทดสอบการพัฒนาไปการออกแบบและความต้องการซอฟต์แวร์

- ผิดพลาดด้านเอกสาร
- ผิดพลาดการทดสอบไม่ถูกต้อง
- ผิดพลาดการทดสอบไม่ครอบคลุม
- ผิดพลาดด้านประสิทธิภาพ
- ผิดพลาดในระบบ
- ผิดพลาดด้านมาตรฐาน
- ผิดพลาดการกำหนดตัวแปร
- ผิดพลาดจากการคำนวณ สูตรที่ใช้
- ผิดพลาดจากอัลกอริทึม
- ผิดพลาดจากตรรกะ
- ผิดพลาดจากสื่อสาร
- ผิดพลาดจากข้อมูลหรือฐานข้อมูล
- ผิดพลาดจากการคอมเมนต์
- ผิดพลาดจากส่วนประสาน
- ผิดพลาดจากการออกแบบไม่ถูกต้อง
- ผิดพลาดจากการออกแบบไม่ครบถ้วนหรือไม่ละเอียดพอ
- ผิดพลาดจากการออกแบบกำกวมไม่ชัดเจน
- ผิดพลาดจากความต้องการไม่ชัดเจน
- ผิดพลาดจากความต้องการไม่ถูกต้อง
- ผิดพลาดจากความต้องการไม่ครบถ้วน

การประกันคุณภาพ (quality assurance)

การประกันคุณภาพเป็นอีกกระบวนการสำคัญในวิศวกรรมซอฟต์แวร์เพื่อเพิ่มความมั่นใจในคุณภาพของกระบวนการพัฒนาซอฟต์แวร์และผลิตภัณฑ์ที่ส่งมอบให้กับลูกค้าที่มีคุณภาพและความคุ้มค่า โดยตรวจสอบในทุกกระบวนการพัฒนา การบริหารจัดการโครงการ การดำเนินการควบคุมคุณภาพโดยประกอบด้วยการตรวจสอบ (audit) รายงานผลการประเมินความสมบูรณ์ผลิตภัณฑ์และประสิทธิภาพ เพื่อเป็นการรับประกันว่าซอฟต์แวร์เป็นไปตามความต้องการและตามมาตรฐานต่าง ๆ ที่ได้กำหนดไว้

วัตถุประสงค์

- *เข้าใจการวิวัฒนาการซอฟต์แวร์*
- *เข้าใจการบำรุงรักษาซอฟต์แวร์*
- *เข้าใจรูปแบบการใช้ซอฟต์แวร์ซ้ำ*

การวิวัฒนาการเป็นหนึ่งในสี่กระบวนการพัฒนาซอฟต์แวร์หลัก โดยกระบวนการวิวัฒนาการมักจะขึ้นอยู่กับการเปลี่ยนแปลงเป็นสำคัญ โดยวิศวกรซอฟต์แวร์พึงระลึกไว้เสมอว่าการเปลี่ยนแปลงนั้นเป็นสิ่งที่หลีกเลี่ยงได้ยาก จำเป็นต้องจัดการเพื่อสนองความต้องการที่สามารถเปลี่ยนแปลงได้เสมออันเนื่องมาจากสาเหตุต่าง ๆ เช่นการเพิ่มความต้องการ รูปแบบธุรกิจที่เปลี่ยน สภาพแวดล้อมที่เปลี่ยน เกิดข้อผิดพลาด ความเชื่อถือได้ หรือ การปรับปรุงประสิทธิภาพที่จะนำไปสู่การวิวัฒนาการซอฟต์แวร์

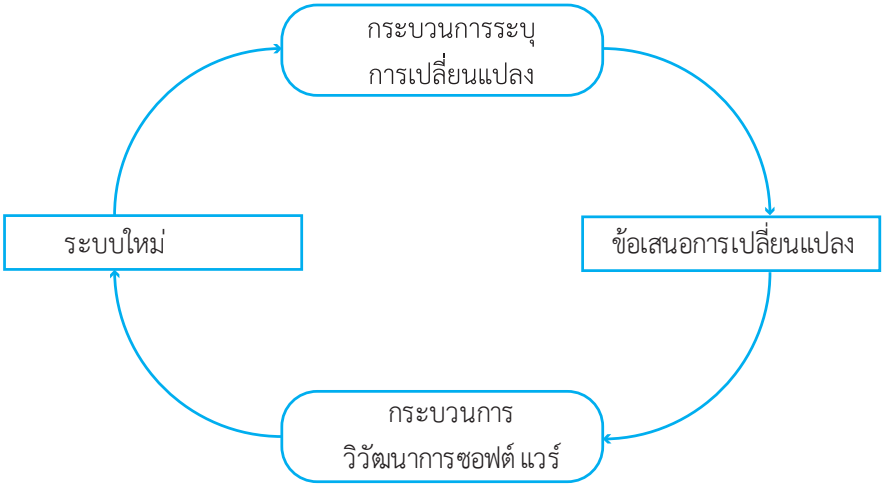
กุญแจสำคัญในส่วนของการวิวัฒนาการซอฟต์แวร์คือการบริหารจัดการการเปลี่ยนแปลงอย่างมีประสิทธิภาพและคุ้มค่าในการดำเนินการ เนื่องจากปัจจุบันซอฟต์แวร์ถือว่าเป็นทรัพย์สินและปัจจัยในการดำเนินธุรกิจในองค์กรส่วนใหญ่ซึ่งมีการลงทุนด้านซอฟต์แวร์สูงโดยเฉพาะในส่วนที่สำคัญต่อธุรกิจ ดังนั้นการบำรุงรักษาซอฟต์แวร์ให้สามารถให้บริการได้อย่างต่อเนื่องจึงเป็นสิ่งจำเป็นที่ต้องดูแล โดยปกติองค์กรมักจะใช้งบประมาณในการบำรุงรักษาซอฟต์แวร์ที่มีใช้งานอยู่แล้วมากกว่าการพัฒนาระบบขึ้นมาใหม่

โดยภาพรวมของซอฟต์แวร์ที่ทำการส่งมอบเพื่อใช้งานแล้วจะประกอบไปด้วยส่วนช่วงหลัก ๆ ด้วยกันคือช่วงระยะการวิวัฒนาการ ช่วงระยะให้บริการและสุดท้ายช่วงระยะเตรียมปลดจากการให้บริการ สำหรับช่วงการวิวัฒนาการนั้นซอฟต์แวร์จะมีการปรับเปลี่ยนตามความต้องการเพื่อให้สามารถตอบสนองความต้องการได้จนระบบมีความเสถียรทั้งด้านการทำงานและความต้องการ หลังจากนั้นจะเข้าสู่ระยะให้บริการซึ่งเป็นระยะที่มีการเปลี่ยนแปลงความต้องการน้อยมากทำเพียงแค่การปรับแก้ข้อผิดพลาดของซอฟต์แวร์ เพื่อให้ระบบสามารถให้บริการได้จะไม่มีการเพิ่มฟังก์ชันใหม่เข้ามา โดยจะใช้งานจนเข้าสู่ระยะเตรียมปลดการให้บริการ โดยในระยะนี้ซอฟต์แวร์ยังคงใช้อยู่แต่ไม่มีการเปลี่ยนแปลงใด ๆ จนยุติการใช้งานไปในที่สุด

ระยะการวิวัฒนาการ➔ ระยะให้บริการ ➔ ระยะเตรียมปลดจากการให้บริการ ➔ สิ้นสุดการให้บริการ

การวิวัฒนาการซอฟต์แวร์ (software evolution)

การเปลี่ยนแปลงเป็นสาเหตุหลักที่นำไปสู่การวิวัฒนาการซึ่งการเปลี่ยนแปลงนั้นมักจะหลีกเลี่ยงไม่ได้ โดยจุดเริ่มในการทำการวิวัฒนาการแต่ละรอบจะมาจากการเปลี่ยนแปลงและผลักดันให้เกิดการเปลี่ยนแปลง ดังรูปที่ 10.1



รูปที่ 10.1 การวิวัฒนาการ ดัดแปลงจาก [11]

ในส่วนของกระบวนการวิวัฒนาการจะใกล้เคียงกับรูปแบบในแบบจำลองก้นหอยที่มีการวนรอบในการพัฒนาโดยการวิวัฒนาการนั้นจะขึ้นอยู่กับปัจจัยหลาย ๆ ด้านประกอบกัน เช่น ประเภทของซอฟต์แวร์ กระบวนการพัฒนาซอฟต์แวร์ที่ใช้ และศักยภาพของทีมที่รับผิดชอบในการดูแล ตามที่ได้กล่าวมาในขั้นต้น โดยเมื่อมีการระบุการเปลี่ยนแปลงจะทำการสร้างข้อเสนอเพื่อทำการเปลี่ยนแปลง ทีมพัฒนาต้องทำการวิเคราะห์ผลกระทบ พิจารณาความสำคัญ และ ความคุ้มค่าในการดำเนินการเปลี่ยนแปลง ก่อนจะทำการเปลี่ยนแปลง

การดำเนินการเปลี่ยนแปลงจะเริ่มจากการนำข้อเสนอการเปลี่ยนแปลงมาวิเคราะห์ความต้องการให้เรียบร้อย ก่อนดำเนินการปรับปรุงความต้องการซอฟต์แวร์เดิมให้ทันสมัยและตรงกับการเปลี่ยนแปลง จากนั้นทำการพัฒนาซอฟต์แวร์ตามความต้องการล่าสุดและทดสอบตามกระบวนการ เพื่อให้มั่นใจว่าซอฟต์แวร์ถูกเปลี่ยนแปลงไปตามความต้องการใหม่เรียบร้อย โดยในขั้นตอนการพัฒนานิยมใช้การพัฒนาแบบวนรอบ

ในกรณีเร่งด่วนมีการร้องขอการเปลี่ยนแปลงอย่างเร่งด่วน เช่นเกิดข้อผิดพลาดที่มีผลกระทบต่อธุรกิจ หรือทำให้ระบบไม่สามารถดำเนินการได้ หรือมีข้อผิดพร่องรุนแรง ดังนั้นทีมพัฒนาควรจะรีบดำเนินการโดยเปลี่ยนแปลงโดยทันที สามารถข้ามขั้นตอนต่าง ๆ ในกระบวนการทางวิศวกรรมซอฟต์แวร์ โดยสามารถเข้าถึงรหัสต้นฉบับเพื่อวิเคราะห์แล้วดำเนินการแก้ไขปัญหาจนสามารถส่งมอบซอฟต์แวร์ที่ผ่านการแก้ไขทันที หลังจากระบบกลับคืนสู่ภาวะปกติทีมพัฒนาค่อยทำการปรับปรุงส่วนต่าง ๆ ที่ดำเนินการเปลี่ยนแปลงทั้งความต้องการและเอกสารที่เกี่ยวข้อง

การวิเคราะห์ผลกระทบ (impact analysis)

การวิเคราะห์ผลกระทบของการเปลี่ยนแปลงที่จะเกิดขึ้นต่อซอฟต์แวร์ในกระบวนการวิวัฒนาการ จะขึ้นอยู่กับวัตถุประสงค์และเป้าหมายของซอฟต์แวร์เพื่อให้วิศวกรซอฟต์แวร์สามารถวิเคราะห์ตามน้ำหนักความสำคัญของการเปลี่ยนแปลง โดยจะทำการวิเคราะห์และตัดสินใจบนพื้นฐานของคุณภาพและความคุ้มค่าในการดำเนินการต่อความเปลี่ยนแปลง โดยภาพรวมจะทำการวิเคราะห์ผลกระทบต่อการดำเนินการของซอฟต์แวร์ก่อนในมุมมองด้านคุณภาพของซอฟต์แวร์และอีกปัจจัยที่สำคัญคือผลกระทบต่อคุณค่าทางธุรกิจซึ่งทั้งสองเป็นปัจจัยหลักที่ใช้ในการวิเคราะห์ผลกระทบ

ผลกระทบด้านคุณภาพของซอฟต์แวร์ ปัจจัยต่าง ๆ ที่ส่งผลกระทบต่อคุณภาพของซอฟต์แวร์โดยพิจารณาในมุมมองการนำไปใช้งาน ความคุ้มค่าในการบำรุงรักษา ประสิทธิภาพของซอฟต์แวร์ ซึ่งหากมีการเปลี่ยนแปลงเกิดขึ้นผลกระทบด้านคุณภาพของซอฟต์แวร์จะเป็น เช่น คุ้มค่ากับการลงทุนหรือไม่ ความเสี่ยงต่าง ๆ มีมากน้อยเพียงใด

ผลกระทบด้านคุณค่าทางธุรกิจ ปัจจัยที่ส่งผลกระทบต่อคุณค่าทางธุรกิจ ทั้งทางตรงและทางอ้อมโดยวิเคราะห์หลากหลายมุมมองจากผู้มีส่วนเกี่ยวข้องที่แตกต่างเพื่อรวบรวมข้อมูลคุณค่าทางธุรกิจ จัดลำดับความสำคัญที่กระทบกับธุรกิจ วิศวกรซอฟต์แวร์ต้องทำการประเมินการเปลี่ยนแปลงและผลกระทบที่จะเกิดทางด้านคุณค่าทางธุรกิจและนำมาชั่งน้ำหนักเทียบกับความคุ้มค่าในการที่จะดำเนินการวิวัฒนาการซอฟต์แวร์หรือไม่ นอกจากนั้นยังต้องพิจารณาอีกว่าหากดำเนินการเปลี่ยนแปลงแล้วผลกระทบที่จะเกิดขึ้นจะส่งผลกระทบทั้งทางบวกและทางลบต่อผู้ที่มีส่วนเกี่ยวข้องกลุ่มใดบ้าง

ผลการวิเคราะห์ผลกระทบจะถูกนำไปพิจารณาประกอบการตัดสินใจในการเปลี่ยนแปลงตามคำเสนอขอ โดยตัดสินใจจากปัจจัยต่าง ๆ ที่จะส่งผลกระทบต่อซอฟต์แวร์และธุรกิจบนพื้นฐานของความจำเป็นและความคุ้มค่าในการดำเนินการวิวัฒนาการเป็นสำคัญ

การบำรุงรักษาซอฟต์แวร์ (software maintenance)

เป็นกิจกรรมหลักในกระบวนการวิวัฒนาการซอฟต์แวร์โดยมีเป้าหมายหลักคือการทำให้ซอฟต์แวร์ยังคงสามารถให้บริการตอบสนองความต้องการได้อย่างมีประสิทธิภาพและคุ้มค่า ซึ่งโดยปกติการบำรุงรักษาซอฟต์แวร์จะไม่ปรับเปลี่ยนสถาปัตยกรรมหรือโครงสร้างหลักของซอฟต์แวร์ จะเน้นการทำให้ซอฟต์แวร์ยังคงความสามารถให้บริการได้ เช่น การปรับเปลี่ยนส่วนของซอฟต์แวร์ที่มีอยู่หรือการเพิ่มเติมส่วนประกอบเข้าไปในระบบ โดยรูปแบบการบำรุงรักษาประกอบด้วยการแก้ไขข้อผิดพร่อง การปรับเปลี่ยนสภาพแวดล้อมและการปรับเปลี่ยนหรือเพิ่มฟังก์ชันตามความต้องการที่เปลี่ยนแปลง

ค่าใช้จ่ายในการบำรุงรักษาซอฟต์แวร์ส่วนใหญ่จะมากกว่าค่าพัฒนาหลายเท่า เนื่องจากค่าใช้จ่ายในการดูแลแก้ไขซอฟต์แวร์นั้นสูงมากโดยเฉพาะในระยะยาว ยิ่งปรับเปลี่ยนมากก็ยิ่งยากในการดำเนินการ ยิ่งซอฟต์แวร์เก่าค่าใช้จ่ายยิ่งสูง เช่น ภาษาที่ใช้ในการพัฒนาเก่านักพัฒนาที่เชี่ยวชาญหายากค่าแรงสูง ทีมพัฒนาที่จะมาดูแลต้องทำการศึกษาซอฟต์แวร์ ระบบ และรหัสต้นฉบับให้เข้าใจก่อนจะสามารถบำรุงรักษาได้ซึ่งเป็นค่าใช้จ่ายที่สูง ทำให้องค์กรต้องมีการวางแผนงบประมาณในส่วนของการบำรุงรักษาซอฟต์แวร์ไว้ให้อย่างเหมาะสม โดยเฉพาะซอฟต์แวร์ที่มีคุณค่าทางธุรกิจสูง

ลักษณะของซอฟต์แวร์ที่บำรุงรักษาได้ (characteristics of maintainable software)

ซอฟต์แวร์ที่พัฒนาขึ้นมาส่วนใหญ่ถูกออกแบบมาเพื่อให้สามารถบำรุงรักษาได้ สามารถใช้งานได้นาน โดยเป็นการส่งผ่านความรับผิดชอบในการดูแลจากทีมพัฒนาไปให้กับทีมดูแลบำรุงรักษาซอฟต์แวร์ โดยซอฟต์แวร์ที่สามารถบำรุงรักษานั้นควรที่จะประกอบไปด้วยคุณลักษณะต่างๆดังนี้

1. ซอฟต์แวร์ทำงานได้ถูกต้องสมบูรณ์
2. คู่มือการใช้งานถูกต้องและครบถ้วน
3. มีเอกสารครบถ้วนและถูกต้อง
4. สามารถเข้าถึงรหัสต้นฉบับและเอกสารการออกแบบได้
5. สามารถเข้าถึงชุดทดสอบและเอกสารการทดสอบได้
6. ซอฟต์แวร์รองรับการเปลี่ยนแปลง
7. โครงสร้างและสถาปัตยกรรมซอฟต์แวร์ง่ายและไม่ซับซ้อน
8. ใช้ภาษาโปรแกรมที่ไม่เก่าเกินไป
9. ส่วนประกอบต่าง ๆ ที่มาจากผู้ผลิตภายนอก ต้องสามารถเข้าถึงหรือขอรับการสนับสนุนได้
10. ซอฟต์แวร์ถูกพัฒนาตามมาตรฐาน

ทีมพัฒนาควรจะทำการส่งมอบซอฟต์แวร์ที่มีคุณลักษณะที่สามารถบำรุงรักษาได้ เพื่อให้ทีมบำรุงรักษาซึ่งส่วนใหญ่จะเป็นคนละทีมกับทีมพัฒนาสามารถนำไปบำรุงรักษาต่อ โดยเฉพาะการใช้แก้ไขรหัสต้นฉบับ ดังนั้นการรีแฟคเตอริงจึงเป็นสิ่งสำคัญสำหรับนักพัฒนาที่ควรกระทำอย่างสม่ำเสมอตลอดการพัฒนาเพื่อให้รหัสต้นฉบับเข้าใจง่าย มีประสิทธิภาพ และ ง่ายต่อการบำรุงรักษา ซึ่งทีมพัฒนาควรระลึกเสมอในการพัฒนาซอฟต์แวร์ให้เป็นไปตามมาตรฐาน มีคุณภาพสูง ปรับเปลี่ยนได้ง่าย การทดสอบที่ครอบคลุม รีแฟคเตอริงรหัสต้นฉบับ ซึ่งจะช่วยสนับสนุนการบำรุงรักษาซอฟต์แวร์

การปรับรื้อระบบเดิม (reengineering legacy system)

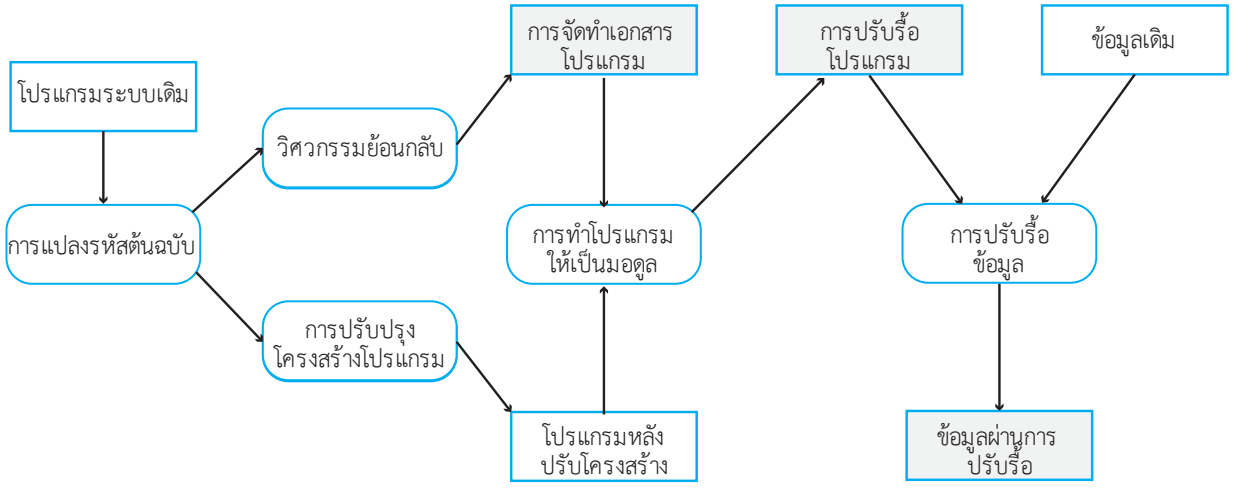
การปรับรื้อระบบเดิมคือการปรับโครงสร้างหรือการพัฒนาซอฟต์แวร์ทั้งบางส่วนหรือทั้งระบบเพื่อทดแทนการทำงานของระบบเดิม (legacy system) โดยคงไว้ซึ่งฟังก์ชันการทำงานที่เหมือนเดิม อาจใช้การปรับปรุงโครงสร้างใหม่พร้อมปรับปรุงเอกสารใหม่ ทำให้การบำรุงรักษาเป็นไปได้ง่ายขึ้น หรือค่าใช้จ่ายน้อยลง นอกจากนั้นการปรับรื้อระบบเดิมยังช่วยลดความเสี่ยงในการพัฒนาซอฟต์แวร์ใหม่ที่ต้องมีการเก็บความต้องการ การกำหนดคุณลักษณะของซอฟต์แวร์ รูปแบบการดำเนินการที่มีความเสี่ยงสูงกว่าการปรับรื้อระบบเดิมการดำเนินการต่าง ๆ เหมือนเดิม สิ่งที่ได้เพิ่มเติมขึ้นมาคือระบบใหม่ที่มีประสิทธิภาพ บำรุงรักษาได้ และค่าใช้จ่ายในการพัฒนาที่ต่ำกว่าการพัฒนาระบบขึ้นมาใหม่

กระบวนการปรับรื้อระบบเดิมสามารถพิจารณาวิธีที่จะเลือกใช้ในการปรับรื้อจากลักษณะของระบบเดิมใน สอง มุมมอง คือคุณภาพของระบบและคุณค่าทางธุรกิจ เช่นเดียวกับการประเมินผลกระทบ ทั้งนี้หลักความคุ้มค่ายังคงใช้เป็นแกนในการพิจารณาร่วมด้วยเสมอ โดยมีแนวทางหลักการเลือกรูปแบบการปรับรื้อ ดังนี้

- คุณค่าทางธุรกิจต่ำ และคุณภาพซอฟต์แวร์ต่ำ ควรเลิกใช้งานระบบซอฟต์แวร์ไปเลย
- คุณค่าทางธุรกิจสูง และคุณภาพซอฟต์แวร์สูง ระบบดีอยู่แล้วไม่ควรเปลี่ยนแปลงให้ใช้ต่อไป การบำรุงรักษาตามปกติ
- คุณค่าทางธุรกิจต่ำ และคุณภาพซอฟต์แวร์สูง เลิกใช้งาน ใช้งานต่อบำรุงรักษาตามปกติหรือ เปลี่ยนด้วย COTS
- คุณค่าทางธุรกิจสูง และคุณภาพซอฟต์แวร์ต่ำ เป็นระบบที่ควรทำการปรับรื้อระบบ หรือพัฒนาระบบใหม่มาทดแทน

จากแนวทางการเลือกรูปแบบการปรับรื้อระบบจะเห็นว่าวิศวกรซอฟต์แวร์ควรพิจารณาคุณค่าทางธุรกิจเป็นอันดับแรก เนื่องเป็นปัจจัยเชื่อมโยงโดยตรงไปยังความคุ้มค่าเพื่อประกอบการตัดสินใจ โดยในกรณีที่มีคุณค่าทางธุรกิจสูงแปลว่าสำคัญกับองค์กรส่งผลกระทบต่อองค์กรจึงมีน้ำหนักในการเลือกเพื่อเข้าสู่กระบวนการปรับรื้อโดยเฉพาะหากซอฟต์แวร์คุณภาพไม่สูงการใช้งานต่อไปอาจจะไม่คุ้มค่าสู้ทำการปรับรื้อระบบเพื่อให้สามารถบำรุงรักษาได้ในค่าใช้จ่ายที่ต่ำน่าจะคุ้มค่ากว่า ทั้งนี้วิศวกรซอฟต์แวร์อาจจะต้องคำนวนชั่งน้ำหนักจากปัจจัยต่าง ๆ รวมทั้งการคาดการณ์ล่วงหน้าเพื่อเลือกแนวทางที่เหมาะสมที่สุด

กระบวนการปรับรื้อระบบจะขึ้นอยู่กับการวิเคราะห์ระบบเดิมและเพื่อเลือกกรรมวิธีและเครื่องมือที่จะใช้ในการปรับรื้อระบบโดย ดูจากภาษาโปรแกรมที่ใช้ เครื่องมือการพัฒนา กระบวนการพัฒนา ข้อมูล และ เทคโนโลยีที่จะใช้โดยกระบวนการส่วนใหญ่จะขึ้นอยู่กับเครื่องมือที่ช่วยในการปรับรื้อและการเลียนแบบการทำงานและฟังก์ชันต่าง ๆ ของระบบเดิม ดังรูปที่ 10.2



รูปที่ 10.2 กระบวนการปรับรื้อระบบ ดัดแปลงจาก [11]

จากรูปที่ 10.2 จะเห็นได้ว่ากระบวนการปรับรื้อระบบนั้นเน้นที่การพัฒนาระบบจากระบบเดิมโดยไม่มีการปรับเปลี่ยนเพิ่มเติมฟังก์ชันใด ๆ โดยเป็นการนำรหัสต้นฉบับเดิมมาวิเคราะห์เพื่อแปลงให้เป็นภาษาโปรแกรมที่จะใช้สำหรับการปรับรื้อและการบำรุงรักษาต่อไป จากนั้นทำกระบวนการวิศวกรรมย้อนกลับเพื่อสร้างซอฟต์แวร์เลียนแบบการทำงานตามฟังก์ชันเดิม พร้อมการปรับปรุงโครงสร้างซอฟต์แวร์ให้เหมาะสมเพื่อนำสู่กระบวนการปรับเปลี่ยนให้เป็นมอดูลจนได้ผลลัพธ์สุดท้ายเป็นโปรแกรมที่ปรับรื้อเรียบร้อย ในส่วนของข้อมูลก็ควรจะทำการปรับรื้อให้เหมาะสมกับซอฟต์แวร์ใหม่โดยข้อมูลเดิมยังคงอยู่ในบริบทเดิม

การใช้ซอฟต์แวร์ซ้ำ (software Reuse)

การใช้ซอฟต์แวร์ซ้ำเป็นแนวทางที่ได้รับความนิยมมามากกว่าทศวรรษในทางวิศวกรรมซอฟต์แวร์ โดยคำนึงถึงความคุ้มค่าโดยเฉพาะด้านเวลา แทนที่จะต้องเริ่มพัฒนาส่วนของซอฟต์แวร์ขึ้นมาใหม่ทุกครั้ง การใช้ซอฟต์แวร์ซ้ำจึงเป็นอีกหนึ่งทางเลือกที่ช่วยลดเวลาการพัฒนา เพิ่มความน่าเชื่อถือ และ ลดความเสี่ยงลงได้ จากแนวความคิดการพัฒนามาจากการนำเอาส่วนของซอฟต์แวร์ที่เคยผ่านการพัฒนาทดสอบและใช้งานแล้วนำกลับมาประยุกต์ใช้ซ้ำในโครงการอื่น ๆ ตามรูปแบบหลัก ๆ ดังต่อไปนี้

- การใช้ซ้ำในระดับระบบ เป็นการนำระบบทั้งระบบมาใช้ในโครงการอื่น ๆ โดยเฉพาะระบบที่มีคุณลักษณะคล้ายคลึงกัน
- การใช้ซ้ำในระดับโปรแกรม เป็นการนำกลับมาใช้ใหม่โดยการคัดเลือกโปรแกรมที่มีอยู่แล้วไปใช้งานตามความต้องการ
- การใช้ซ้ำในระดับส่วนประกอบ เป็นการนำเอาส่วนประกอบของซอฟต์แวร์ไปรวมเข้ากับซอฟต์แวร์ที่พัฒนา
- การใช้ซ้ำในระดับวัตถุหรือฟังก์ชัน เป็นการใช้ซ้ำในระดับการรหัสต้นฉบับเพื่อไปประยุกต์ใช้ในการพัฒนา

ข้อดีของการใช้ซอฟต์แวร์ซ้ำสามารถเร่งกระบวนการพัฒนาให้เร็วขึ้นไม่ต้องมาพัฒนาใหม่เองทั้งหมด และบางส่วนของซอฟต์แวร์ถูกพัฒนาโดยมืออาชีพผ่านการทดสอบการใช้งานมาแล้วจึงเป็นการเพิ่มความน่าเชื่อถือ ช่วยเพิ่มในเรื่องของความอิสระต่อกันในซอฟต์แวร์เนื่องจากการใช้ซ้ำส่วนมากจะมาในรูปแบบส่วนประกอบที่แยกออกจากกันชัดเจน ช่วยลดค่าใช้จ่ายในการพัฒนา ลดความเสี่ยง และ เพิ่มระดับความเป็นมาตรฐาน ซึ่งวิศวกรซอฟต์แวร์ต้องทำการตัดสินใจและเลือกแนวทางการใช้ซอฟต์แวร์ซ้ำให้เหมาะสมกับความต้องการและข้อจำกัดของซอฟต์แวร์

รูปแบบการใช้ซ้ำมีได้หลากหลายรูปแบบ วิศวกรซอฟต์แวร์สามารถเลือกรูปแบบให้เหมาะสมกับซอฟต์แวร์ที่จะทำการพัฒนาให้ตรงกับความต้องการและคุ้มค่าโดยตัวอย่างรูปแบบการใช้ซ้ำ ดังนี้

1. กรอบการทำงาน (framework)
2. การเลือกจากสายผลิตภัณฑ์ (product line)
3. ระบบอีอาร์พี (ERP)
4. ระบบแอปพลิเคชันการกำหนดค่า (configuration application system)
5. แบบแผนการออกแบบ (design pattern)
6. แบบแผนสถาปัตยกรรม (architecture pattern)
7. ระบบเชิงบริการ (service-oriented system)
8. วิศวกรรมซอฟต์แวร์เชิงส่วนประกอบ (component-based software engineering)
9. โปรแกรมไลบรารี (program libraries)

บทที่ 11 การบริหารโครงการพัฒนาซอฟต์แวร์ Software project management

วัตถุประสงค์

- *เข้าใจการบริหารโครงการพัฒนาซอฟต์แวร์*
- *เข้าใจการวัดและประเมินซอฟต์แวร์*

การบริหารโครงการพัฒนาซอฟต์แวร์เป็นกิจกรรมที่ผู้บริหารควรจะดำเนินการเพื่อให้โครงการดำเนินไปได้จนสำเร็จภายในกรอบเวลาที่กำหนด ภายในงบประมาณและทรัพยากรที่กำหนด และสามารถส่งมอบซอฟต์แวร์ที่มีคุณภาพ ตรงตามความต้องการ โดยส่วนมากการบริหารโครงการจะมุ่งเน้นที่การดำเนินงานให้เสร็จตามกรอบเวลาที่วางไว้และไม่เกินงบประมาณที่กำหนด มีการบริหารจัดการความเสี่ยง การบริหารทีมให้มีความสามัคคีและเหนียวแน่น

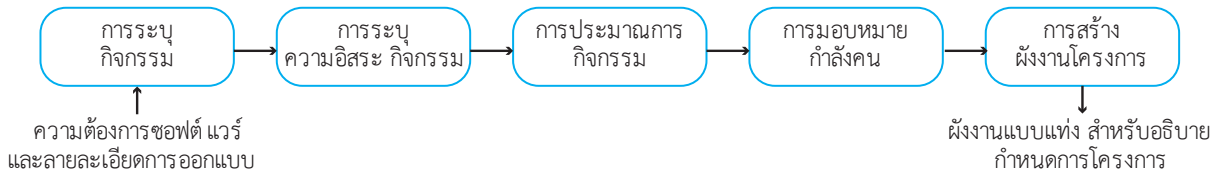
ซอฟต์แวร์เป็นผลิตภัณฑ์ที่เป็นนามธรรมไม่สามารถจับต้องได้ ทำให้ในการบริหารโครงการนั้นยากที่จะวัดและประเมินความก้าวหน้า ทำให้การบริหารจัดการโครงการพัฒนาซอฟต์แวร์นั้นแตกต่างจากระบบการบริหารโครงการอื่น ๆ และนอกจากนั้นส่วนใหญ่โครงการที่ดำเนินการพัฒนาจะเป็นโครงการที่มีเอกลักษณ์เฉพาะตัวมีปัญหาเฉพาะ ซึ่งทำให้การบริหารโครงการมีความยาก ถึงแม้ผู้บริหารจะมีประสบการณ์ก็ตาม มีปัจจัยอื่น ๆ ที่ส่งผลกระทบในการบริหารที่ต้องนำมาคิดและพิจารณาร่วมในการบริหารโครงการ เช่น เรื่องของรูปแบบทีม วัฒนธรรมองค์กร ขนาดของโครงการ ประเภทของซอฟต์แวร์ กระบวนการพัฒนาซอฟต์แวร์ และ ผู้มีส่วนเกี่ยวข้อง ซึ่งเป็นหน้าที่ของผู้บริหารที่จะต้องนำมาคิดและปรับการบริหารให้เหมาะกับโครงการ

กิจกรรมหลักของการบริหารโครงการคือการวางแผนการดำเนินโครงการ การตรวจสอบ และ ควบคุมให้โครงการดำเนินไปตามแผนที่วางไว้ โดยมีกิจกรรมหลักคือการวางแผน (planning) การจัดการกำหนด (scheduling) การบริหารความเสี่ยง การบริหารทีม (risk management) การบริหารทีม (team management) และ การนำเสนอข้อมูลสารสนเทศของโครงการ

กำหนดการโครงการ (project scheduling)

การกำหนดการเป็นกิจกรรมในการวางแผนการดำเนินการโดยกำหนดกิจกรรมในการพัฒนา กำหนดเวลา ทรัพยากร และงบประมาณที่จะใช้ในแต่ละกิจกรรมโดยมองในภาพรวมในการพัฒนาประกอบกับการพิจารณากระบวนการพัฒนาที่ทีมเลือกใช้เพื่อเลือกวิธีการกำหนดการให้เหมาะสม จากนั้นจึงลงในรายละเอียดเพื่อใช้ในการติดตามและควบคุมการดำเนินการโครงการ

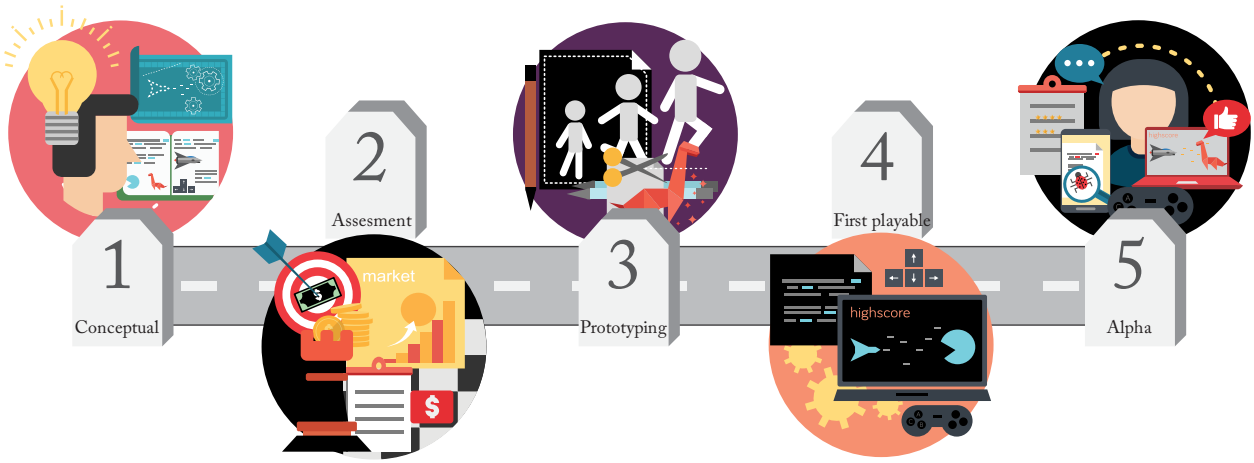
กำหนดการโครงการแตกงานและแจกแจงกิจกรรมย่อยออกมาเพื่อให้ง่ายต่อการบริหารและการดำเนินการ โดยในแต่ละส่วนจะถูกประมาณการทั้งเวลา ทรัพยากร และ ผู้รับผิดชอบ โดยกิจกรรมหลักในการกำหนดคือ การแตกโครงการออกเป็นส่วนย่อยและประเมินทรัพยากรที่ต้องใช้ การจัดการงานให้คุ้มค่าที่สุดตามความเหมาะสมของกำลังคนโดยเน้นการทำงานแบบคู่ขนาน และ การแบ่งงานให้อิสระต่อกันให้มากที่สุด เพื่อให้กำหนดการนำไปสู่แผนดำเนินการโครงการที่ประสิทธิภาพและคุ้มค่า ดังรูปที่ 11.1



รูปที่ 11.1 กระบวนการกำหนดโครงการ ดัดแปลงจาก [11]

การนำเสนอส่วนใหญ่จะใช้รูปกราฟฟิกเพื่อให้เห็นภาพการกำหนดได้อย่างชัดเจน เห็นระยะเวลาและปฏิทินของโครงการได้ชัดเจนโดยมักจะแสดงกิจกรรมต่าง ๆ เทียบกับเวลา นอกจากนั้นยังนิยมใช้ผังรูปเครือข่ายกิจกรรม (activity network) ในการแสดงผลความอิสระต่อกันของกิจกรรมและการแสดงเส้นทางวิกฤติในโครงการ นอกจากนี้การกำหนดควรระบุเส้นตายในแต่ละกิจกรรมให้ชัดเจน ระบุทรัพยากรให้ชัดเจน การประเมินกิจกรรมต่าง ๆ ระบุให้ชัดเจน

การกำหนดการอาจใช้รูปแบบการกำหนดแบบหลักกิโล (milestone) เพื่อกำหนดจุดหมายหรือเป้าหมายย่อยตลอดโครงการ เป็นจุดที่ใช้ในการประเมินความก้าวหน้าของโครงการ ประเมินความเสี่ยงของโครงการ มีคณะกรรมการตรวจและพิจารณาในแต่ละจุดเพื่อพิจารณาและตัดสินว่าโครงการควรจะปรับปรุงในทิศทางไหน ควรดำเนินการต่อหรือยกเลิกโครงการดี ซึ่งกระบวนการนี้พบเห็นได้บ่อยในการพัฒนาซอฟต์แวร์ด้านความบันเทิงโดยเฉพาะการพัฒนาเกม ดังตัวอย่างแสดงในรูปที่ 11.2



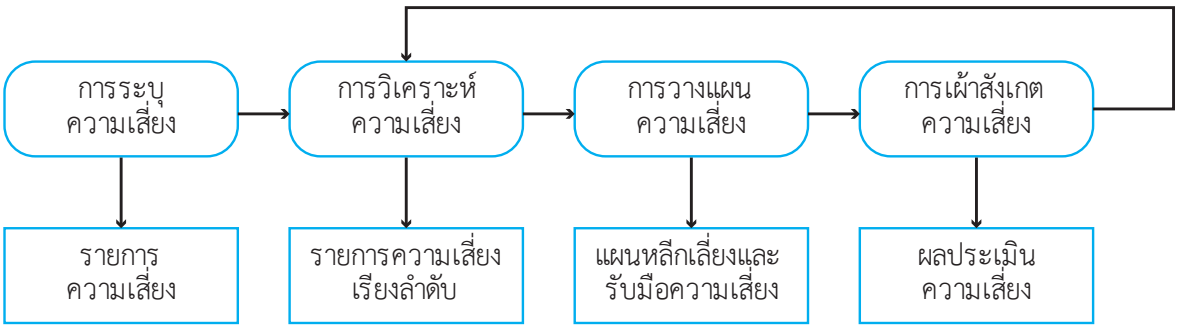
รูปที่ 11.2 กระบวนการหลักกิโล (milestone process) [14]

เครื่องมือที่ใช้ในการกำหนดมีให้เลือกตามความเหมาะสมของโครงการ อาจจะใช้ PERT [15] เป็นเครื่องมือในการกำหนดโครงการ ใช้เครื่องมือในการประมาณการเช่น COCOMO [16] หรืออาจใช้โปรแกรมตารางคำนวณในการบริหารการกำหนดได้ขึ้นอยู่กับความเหมาะสมของโครงการ

การบริหารความเสี่ยง (risk management)

การบริหารความเสี่ยงเป็นกิจกรรมที่เกี่ยวข้องกับการตรวจตราหาความเสี่ยงในโครงการตลอดจนการวางแผนเพื่อรับมือความเสี่ยงต่าง ๆ เพื่อลดผลกระทบที่จะเกิดขึ้นกับโครงการเป็นสำคัญกระบวนการบริหารความเสี่ยงมีความสำคัญในการบริหารโครงการเพราะความเสี่ยงมักเกิดจากการเปลี่ยนแปลงและการเปลี่ยนแปลงมิอาจหลีกเลี่ยงได้ ซ้ำยังเกิดขึ้นได้เสมอในกระบวนการพัฒนา ทำให้หากมีการบริหารความเสี่ยงที่ดีแล้วย่อมจะสามารถช่วยลดผลกระทบที่จะเกิดขึ้นได้จากการคาดการณ์และวางแผนรับมือความเสี่ยงล่วงหน้า

โดยปกติความเสี่ยงมี สอง ประเภทคือความเสี่ยงที่กระทบกับกำหนดการและทรัพยากรของโครงการ กับความเสี่ยงที่กระทบกับคุณภาพของซอฟต์แวร์ กระบวนการบริหารความเสี่ยงประกอบด้วย สี่ กระบวนการพื้นฐานคือ การระบุความเสี่ยง การวิเคราะห์ความเสี่ยง การวางแผนจัดการความเสี่ยง และ การเฝ้าสังเกตความเสี่ยง ดังรูปที่ 11.3



ดังรูปที่ 11.3 กระบวนการบริหารความเสี่ยง ดัดแปลงจาก [11]

การระบุความเสี่ยง การระบุความเสี่ยงในแต่ละโครงการอาจแตกต่างกัน โดยรวมจะขึ้นอยู่กับประสบการณ์ของผู้บริหารโครงการ สำหรับการประเมินจากความเสี่ยงพื้นฐาน เช่น ความเสี่ยงด้านเครื่องมือและเทคโนโลยี ความเสี่ยงด้านความต้องการ ความเสี่ยงด้านการเปลี่ยนแปลง ความเสี่ยงด้านทรัพยากรมนุษย์ ความเสี่ยงด้านองค์กร และ ความเสี่ยงจากการประมาณการคลาดเคลื่อน ความเสี่ยงเหล่านี้จะถูกประเมินและระบุออกมาเป็นรายการความเสี่ยงที่มีความเป็นไปได้ในโครงการ

การวิเคราะห์เสี่ยง กระบวนการที่นำความเสี่ยงต่าง ๆ มาประเมินหาระดับความรุนแรงผลกระทบจากความเสี่ยงว่าอยู่ในระดับหายนะ รุนแรง ทนได้ หรือไม่มีนัยยะ รวมทั้งการวิเคราะห์ความน่าจะเป็นที่จะเกิดความเสี่ยง น้อยมาก น้อย ปานกลาง สูง สูงมาก ซึ่งจะทำให้เห็นว่าความเสี่ยงแต่ละตัวมีผลกระทบอย่างไรบ้างแล้วจะมีโอกาสเกิดมากน้อยเพียงใดเพื่อนำไปจัดเรียงลำดับความเสี่ยงตามความรุนแรงหรือตามความน่าจะเป็นที่ความเสี่ยงจะเกิดขึ้น

การวางแผนความเสี่ยง นำรายการความเสี่ยงมาวางแผนสำหรับ แผนการหลีกเลี่ยงไม่ให้เกิด แผนการลดผลกระทบที่จะเกิดขึ้นจากความเสี่ยงให้ลดน้อยลง แผนการจัดการความเสี่ยงหากความเสี่ยงเกิดขึ้นโดยการจัดทำเป็นแผนกลยุธ์ในการจัดการความเสี่ยงรูปแบบต่าง ๆ ให้ครอบคลุม

การเฝ้าสังเกตความเสี่ยง เป็นกระบวนการคอยเฝ้าสังเกตและตรวจตราความเสี่ยงต่าง ๆ ว่ายังอยู่ในสภาวะปกติหรือเริ่มที่จะก่อปัญหาให้กับโครงการ ผลกระทบจากความเสี่ยงที่เกิดขึ้น เพื่อให้สามารถดำเนินการจัดการความเสี่ยงเหล่านั้นได้ทันท่วงที ทั้งนี้การรายงานผลการเฝ้าสังเกตความเสี่ยงควรดำเนินการอย่างสม่ำเสมอ

ในกระบวนการบริหารความเสี่ยงข้อมูลต่าง ๆ ด้านความเสี่ยงควรได้รับการตรวจตราอย่างสม่ำเสมอให้แน่ใจว่าความเสี่ยงต่าง ๆ ยังอยู่ในการควบคุมเพราะยิ่งตรวจพบเร็วก็สามารถจัดการได้เร็วส่งผลกระทบต่อโครงการลดลง เมื่อความเสี่ยงต่าง ๆ ในโครงการต่ำโอกาสที่โครงการจะประสบความสำเร็จจะมีสูง

การบริหารทีม (team management)

ทีมถือเป็นหัวใจสำคัญในการพัฒนาและเป็นทรัพยากรที่มีค่าจึงจำเป็นต้องได้รับการดูแลเป็นอย่างดีเพื่อสนับสนุนและเสริมสร้างทีมให้แข็งแกร่ง สามัคคี ผ่านการบริหารทีมที่ดี และ มีประสิทธิภาพ ดังนั้นผู้บริหารควรใส่ใจและให้ความสำคัญกับการบริหารทีม สร้างขวัญกำลังใจ สร้างแรงจูงใจให้ทีมร่วมกับทำงานเพื่อให้โครงการดำเนินไปสู่ความสำเร็จ

กระบวนการทางด้านวิศวกรรมซอฟต์แวร์ส่วนใหญ่ดำเนินการเป็นทีม และแน่นอนว่าหากความสัมพันธ์ในทีม การสื่อสารในทีม หรือ การปฏิสัมพันธ์ในทีมเกิดปัญหาหรือไม่สมดุลจะส่งผลกระทบโดยตรงต่อโครงการ ซึ่งปัญหาเหล่านี้สามารถหลีกเลี่ยงได้ด้วยการบริหารทีมที่ดีและมีประสิทธิภาพ สิ่งสำคัญในการบริหารทีมคือการพยายามทำให้ทีมมีความเหนียวแน่นสามัคคี ทำงานร่วมกันเป็นทีม มีเป้าหมายร่วมกัน มีกำลังใจที่ดี และ ทุกคนในทีมมีความสุขกับการทำงานร่วมกันเป็นทีม แต่ในความเป็นจริงการสร้างทีมในอุดมคตินั้นอาจจะยากเนื่องจากแต่ละคนในทีมมีความแตกต่าง ดังนั้นหน้าที่ของผู้บริหารทีมคือการประสานสมาชิกในทีมเข้าด้วยกันอย่างลงตัวและเหมาะสม พยายามสร้างแกนคุณค่า (core value) ในทีมเพื่อให้ทีมแข็งแกร่ง เช่นการเคารพซึ่งกันและกัน ทุกคนรู้หน้าที่ มีน้ำใจนักกีฬา มีความกล้าหาญ ซื่อสัตย์ และ มีความเห็นแก่ทีม

ถ้ามองในมุมมองด้านประสิทธิภาพของทีมการบริหารทีมควรคำนึงถึงการกระจายงานให้สมาชิกในทีมได้ถูกต้องเหมาะสมกับความสามารถและความถนัดของสมาชิกแต่ละคนในทีมเพื่อให้ทีมได้ใช้ความสามารถได้เต็มที่ ในส่วนของการเลือกสมาชิกในทีมเพื่อดำเนินการโครงการพัฒนา ผู้บริหารจำเป็นต้องวิเคราะห์และคัดเลือกสมาชิกให้เหมาะสมกับโครงการทั้งด้านความถนัด และคุณลักษณะส่วนบุคคลเพื่อให้ทีมเกิดความสมดุลและสร้างความเหนียวแน่นในทีม รวมถึงการสร้างการสื่อสารที่ดีในทีมตลอดจนผู้ที่มีส่วนเกี่ยวข้องจะช่วยให้การดำเนินโครงการเป็นไปอย่างราบลื่น

การรวมทีมเป็นอีกหนึ่งกุญแจสำคัญสู่ความสำเร็จของโครงการ ผู้บริหารควรเลือกสร้างทีมที่เหมาะกับโครงการโดยคำนึงถึงความถนัดของสมาชิกในทีมให้ตรงกับการพัฒนาโครงการอย่างสมดุล และต้องพิจารณาคุณลักษณะส่วนบุคคลร่วมด้วยเพื่อช่วยให้ทีมมีความสมดุล โดยคุณลักษณะส่วนบุคคลอาจแบ่งได้ สาม กลุ่มตามแรงจูงใจ ดังนี้

- มุ่งมั่นกับตัวเอง เป็นบุคคลที่มองตัวเองเป็นหลักใช้ความสำเร็จความก้าวหน้าของตัวเองเป็นแรงจูงใจ
- มุ่งมั่นกับงาน เป็นบุคคลประเภทที่ชอบทำงานด้วยตัวเองมีผลงานที่ทำออกมาเป็นแรงจูงใจ
- มุ่งมั่นกับการปฏิสัมพันธ์ เป็นบุคคลที่ชอบการปฏิสัมพันธ์ทำงานเป็นทีม การได้ปฏิสัมพันธ์กับผู้อื่นเป็นแรงจูงใจ

ซึ่งจะเห็นได้ว่าบุคคลแต่ละประเภทมีแรงจูงใจในการสนองความต้องการของตนที่แตกต่างกัน ดังนั้นในการคัดเลือกทีมควรจะมีความสมดุลทั้ง สาม ประเภทไม่เน้นไปประเภทใดประเภทหนึ่งยกตัวอย่าง เช่น ถ้ามีความมุ่งมั่นกับตัวเองเยอะ กลายเป็นทุก ๆ คนอยากเป็นเจ้านาย ถ้ามีมุ่งมั่นกับงานเยอะทุก ๆ คนมุ่งแต่กับงานของตนการสื่อสารอาจจะต่ำ สุดท้ายถ้ามีสมาชิกมุ่งมั่นกับการปฏิสัมพันธ์เยอะจะมีแต่เรื่องคุยกันกิจกรรมปฏิสัมพันธ์มากเกินไปทำให้งานไม่ค่อยเดิน ดังนั้นการปรับสมดุลจึงจำเป็นต้องมีทุกประเภทเข้ามาตามสัดส่วนของโครงการ ฝ่ายบริหารที่มุ่งมั่นกับตัวเอง มีนักพัฒนาที่มุ่งมั่นกับงาน (วิศวกรซอฟต์แวร์ส่วนใหญ่อยู่

ประเภทนี้) และมีประเภทมุ่งมั่นกับการปฏิสัมพันธ์เพื่อสร้างการสื่อสารที่ดีในทีมช่วยประสานและสร้างบรรยากาศที่ดีในทีม นอกจากนั้นการวางแผนกำลังคนต้องพัฒนาจากงบประมาณโครงการเพื่อเลือกสมาชิกตามความถนัดและค่าตอบแทนให้เหมาะสม

การจัดรูปแบบทีม การจัดรูปแบบทีมจะมีการจัดการสองลักษณะหลัก ๆ คือการจัดแบบเป็นทางการและไม่เป็นทางการตามความเหมาะสมของโครงการ พิจารณาได้จาก ขนาดของโครงการ กระบวนการพัฒนาที่เลือกใช้ และ วัฒนธรรมองค์กร โดยการจัดรูปแบบทีมอย่างไม่เป็นทางการนิยมใช้ในทีมที่มีขนาดเล็กไม่ต้องมีลำดับชั้นโครงสร้างการบริหาร เช่นเดียวกับการพัฒนาแบบเอจายล์ที่สนับสนุนการทำงานเป็นทีมแบบไม่เป็นทางการ โดยทีมทำงานในรูปแบบทั้งทีมเป็นหนึ่งเดียว รับรู้ปัญหาร่วมกันตัดสินใจร่วมกัน อาจจะมีหัวหน้าทำหน้าที่ประสานกับภายนอกทีมกับผู้มีส่วนเกี่ยวข้อง ในส่วนของการจัดรูปแบบทีมอย่างเป็นทางการจะใช้การจัดรูปแบบองค์กรแบบมีลำดับชั้นการบริหารเป็นหลัก เพื่อให้ง่ายต่อการบริหารจัดการโดยเฉพาะเมื่อทีมมีขนาดที่ใหญ่

การจัดการสื่อสารในทีม การสื่อสารที่ดีภายในทีมช่วยลดความเสี่ยงและเพิ่มประสิทธิภาพโดยรวมของทีม ข้อมูลข่าวสารต่าง ๆ ควรถูกกระจายให้สมาชิกในทีมทราบอย่างรวดเร็วและถูกต้อง มีการบริหารการสื่อสารที่ดี สถานะต่าง ๆ ของโครงการสมาชิกในทีมต้องเข้าถึงได้ เช่น ถ้ามีการเปลี่ยนแปลงเกิดขึ้นในโครงการสมาชิกในทีมควรรับทราบข้อมูลการเปลี่ยนแปลงนี้ และสามารถตอบสนองได้อย่างมีประสิทธิภาพ การสื่อสารที่ดีช่วยทำให้ทีมเหนียวแน่นและแข็งแกร่ง รูปแบบการสื่อสารแต่ละทีมก็อาจมีความแตกต่างกันไปขึ้นอยู่กับการจัดรูปแบบทีม ขนาดของทีม กระบวนการพัฒนาที่เลือกใช้ คุณลักษณะส่วนบุคคลภายในที่และสภาพแวดล้อมในสถานที่ทำงาน โดยผู้บริหารควรจัดการสร้างบรรยากาศและสภาพแวดล้อมให้เหมาะสมเพื่อสนับสนุนการสื่อสาร

การวัดและการประเมินซอฟต์แวร์ (software measurement and estimation techniques)

ซอฟต์แวร์เป็นผลิตภัณฑ์ที่เป็นนามธรรม ดังนั้นการวัดและประเมินซอฟต์แวร์เป็นกระบวนการที่เกิดจากการประมาณการจากค่าชี้วัดต่าง ๆ ที่สกัด ออกมาจากซอฟต์แวร์และกระบวนการ เพื่อชี้วัดปริมาณคุณภาพและเทียบมาตรฐานของซอฟต์แวร์ ซึ่งกระบวนการวัดและประเมินซอฟต์แวร์นั้นมีความยากและคลาดเคลื่อน มาตรฐานที่ออกมาก็ยังน้อย ทำให้องค์กรส่วนมากก็ไม่ได้ดำเนินการวัดและประเมินซอฟต์แวร์อย่างเป็นระบบ การประเมินส่วนใหญ่อยู่ในรูปการณ์ประเมินเอกสารจากข้อมูลตัวชี้วัดต่าง ๆ ที่ทำการรวบรวมมาวิเคราะห์และประมาณการ

ในการบริหารโครงการมีการวัดและประเมินในส่วนของกระบวนการพัฒนาเพื่อตรวจดูความก้าวหน้าและคุณภาพของซอฟต์แวร์ที่กำลังพัฒนาว่าเป็นไปตามมาตรฐานและตรงตามความต้องการหรือไม่ ตัวชี้วัดที่นำมาพิจารณาจากกระบวนการพัฒนาได้แก่ เวลาที่ใช้ในการพัฒนาในแต่ละส่วนคิดเป็นวันเวลารวมที่ต้องใช้ ทรัพยากรที่ใช้ในการพัฒนาแต่ละส่วนในรูปแบบ แรงงานคน ค่าใช้จ่าย หรือทรัพยากรที่ลงทุน จำนวนเหตุการณ์ที่เกิดในแต่ละส่วน เช่น จำนวนข้อผิดพลาดที่ตรวจพบ จำนวนการเปลี่ยนแปลงที่เกิดขึ้น จำนวนความต้องการที่เปลี่ยนแปลง จำนวนบรรทัดที่พัฒนาที่แก้ไข เป็นต้น นอกจากนั้นยังสามารถนำตัวชี้วัดความซับซ้อนของโปรแกรมมาร่วมพิจารณา โดยเป้าหมายในการวัดคือการประเมินคุณภาพของซอฟต์แวร์ว่าอยู่ในมาตรฐานและระดับที่ยอมรับได้หรือไม่

ปัญหาที่เกิดขึ้นในการวัดและประเมินซอฟต์แวร์มาจากการที่ไม่สามารถนำตัวชี้วัดมาประเมินได้อย่างถูกต้องตรงไปตรงมาได้ ไม่มีมาตรฐานในการกำหนดตัวชี้วัด มาตรฐานในการวัดและการประเมิน หรือมาตรฐานในการวิเคราะห์แต่อย่างใด การเพิ่มกระบวนการวัดและประเมินซอฟต์แวร์ เพิ่มค่าใช้จ่ายอื่นให้กับการพัฒนา หลาย ๆ องค์กรจึงไม่ได้ทำกระบวนการนี้อย่างจริงจัง

อย่างไรก็ตามในเมื่อการวัดที่ซอฟต์แวร์ตรง ๆ นั้นทำได้ยากและอาจจะไม่สะท้อนถึงคุณภาพของซอฟต์แวร์ได้อย่างแม่นยำ ดังนั้น การวัดและประเมินซอฟต์แวร์อาจเปลี่ยนมุมมองมาวันที่กระบวนการแทน โดยมีสมมุติฐานว่าถ้าหากกระบวนการพัฒนาดีภายใต้องค์กรที่มีคุณภาพผ่านการประเมินแล้วนั้น คาดได้ว่าซอฟต์แวร์ที่ผ่านการพัฒนานั้นจะดีและมีคุณภาพตามมาตรฐานด้วย เช่น ลูกค้ามีความมั่นใจในซอฟต์แวร์ที่ผลิตจากบริษัทที่ผ่านการประเมินจนได้รับรองระดับตามมาตฐาน CMMI เป็นต้น

ซีเอ็มเอ็มไอ (capability maturity model integration: CMMI) [17] คือกระบวนการประเมินคุณภาพและความสามารถของทีมพัฒนาซอฟต์แวร์ด้วยการใช้ แบบจำลอง CMMI ในการเรียนรู้และประเมินระดับความสามารถและประสิทธิภาพขององค์กร และนำเสนอแนวทางการเพิ่มประสิทธิภาพทางธุรกิจ เป็นอีกหนึ่งมาตรฐานในการปรุบปรุงคุณภาพซอฟต์แวร์ที่ได้รับการยอมรับทั่วโลก องค์กรไหนได้รับการรับรองระดับ CMMI ถือว่ามีกระบวนการที่มีคุณภาพและได้มาตรฐานในการที่จะผลิตซอฟต์แวร์ที่มีประสิทธิภาพ มีความน่าเชื่อถือ

แบบจำลอง CMMI เป็นชุดแนวทางสำหรับการดำเนินงานและแนวทางปฏิบัติที่ดีได้รับการยอมรับ เพื่อช่วยปรับปรุงให้องค์กรมีศักยภาพที่สูงขึ้น โดยแบบจำลองถูกออกแบบมาให้เข้าใจได้ เข้าถึงได้ ยืดหยุ่น และ สามารถปรับรวมเข้ากับกระบวนการพัฒนาต่าง ๆ ได้เช่นเอจายล์ โดย CMMI มีทั้งหมด 5 ระดับเพื่อใช้ในการระบุระดับวุฒิภาวะ (maturity level) ขององค์กร โดยในแต่ละระดับจะมีการกำหนด โปรเซสแอเรีย (process area) ที่จำเป็นต้องมีในแต่ละระดับ ดังต่อไปนี้

- *Maturity level 1 initial* ไม่มีการกำหนดโปรเซสแอเรียเป็นระดับเริ่มต้น
- *Maturity level 2 Managed* มีการกำหนดกระบวนการบริหารขั้นพื้นฐานใน 7 โปรเซสแอเรีย
    - CM - Configuration Management
    - MA - Measurement and Analysis
    - PMC - Project Monitoring and Control
    - PP - Project Planning
    - PPQA - Process and Product Quality Assurance
    - REQM - Requirements Management
    - SAM - Supplier Agreement Management
- *Maturity level 3 Defined* มีการกำหนดกระบวน 11 โปรเซสแอเรีย และต้องทำทั้ง 7 โปรเซสแอเรียในระดับ 2 ด้วย
    - DAR - Decision Analysis and Resolution
    - IPM - Integrated Project Management +IPPD
    - OPD - Organizational Process Definition +IPPD
    - OPF - Organizational Process Focus
    - OT - Organizational Training
    - PI - Product Integration
    - RD - Requirements Development
    - RSKM - Risk Management
    - TS - Technical Solution

- VAL - Validation
- VER - Verification

- *Maturity level 4 Quantitatively Managed* มีการใช้การวัดเชิงปริมาณในการจัดการกระบวนการมี 2 โปรเซสแอเรีย
  - QPM - Quantitative Project Management
  - OPP - Organizational Process Performance
- *Maturity level 5 Optimizing* มี2 โปรเซสแอเรียเป็นระดับที่สามารถวิเคราะแก้ไขปัญหาเพื่อปรับปรุงกระบวนการ
  - CAR - Causal Analysis and Resolution
  - OID - Organizational Innovation and Deployment

การประเมิน (appraisal) CMMI ช่วยในการระบุจุดแข็งและจุดอ่อนของกระบวนการต่าง ๆ ภายในองค์กร กระบวนการใดบ้างใกล้เคียงกับแนวทางปฏิบัติของ CMMI เพื่อนำไปดำเนินการปรับปรุงประสิทธิภาพปรับปรุงความสามารถของกระบวนการนั้น ๆ โดยมุ่งเน้นกระบวนการที่มีผลกระทบต่อธุรกิจและกระบวนการพัฒนาซอฟต์แวร์ การประเมินจะช่วยให้องค์กรสามารถระบุและจัดลำดับความสำคัญของการกระบวนการต่าง ๆ การได้รับระดับวุฒิภาวะตามมาตรฐาน CMMI ถือเป็นเครื่องชี้วัดระดับความสำเร็จ ระดับความสามารถ ระดับความเป็นมืออาชีพ เพื่อแสดงให้ลูกค้ามั่นใจได้ว่าซอฟต์แวร์ที่ผ่านการผลิตจากองค์กรจะเป็นซอฟต์แวร์ที่มีคุณภาพและได้มาตรฐาน นอกจากนั้นในการรับงานหรือประมูลบางโครงการจะมีการระบุระดับ CMMI ที่สามารถเข้ามารับหรือประมูลงานได้ หลาย ๆ องค์กรจึงเข้ารับการประเมินวัดระดับ โดยในประเทศไทยมีหน่วยงานเขตอุตสาหกรรมซอฟต์แวร์ประเทศไทย [18] คอยสนับสนุนให้องค์กรต่าง ๆ ประเมินวัดระดับ เพื่อยกระดับอุตสาหกรรมซอฟต์แวร์ไทยได้มาตรฐานสากล โดยมีปกติระยะเวลาในการให้คำปรึกษาและประเมินในแต่ละระดับ ดังนี้

- *Maturity level 2* ใช้เวลาโดยประมาณ 12 เดือน
- *Maturity level 3* ใช้เวลาโดยประมาณ 16 เดือน
- *Maturity level 4 และ 5* ใช้เวลาโดยประมาณ 24 เดือน

การดำเนินการให้คำปรึกษาและประเมินประกอบด้วย 5 ขั้นตอนพื้นฐานคือ 1 การวินัจฉัยเบื้องต้นเพื่อเข้าใจองค์กร 2 การจัดทำกระบวนการและเอกสารที่เกี่ยวข้องสำหรับกระบวนการต่างๆ 3 การกระบวนการไปใช้จริงตามกระบวนการทำกำหนดและมีการประเมินย่อยเทียบเคียงกับโปรเซสแอเรีย 4 ทำการทดลองประเมินเพื่อตรวจสอบความพร้อมก่อนรับการประเมินจริง 5 เข้ารับการประเมินระดับวุฒิภาวะอย่างเป็นทางการ

บทที่ 12 เครื่องมือในงานวิศวกรรมซอฟต์แวร์ Software tools and environments

วัตถุประสงค์

- *เข้าใจเครื่องมือในงานวิศวกรรมซอฟต์แวร์*
- *สามารถเลือกเครื่องมือในงานวิศวกรรมซอฟต์แวร์ให้เหมาะสมกับการพัฒนา*

เครื่องมือที่ใช้ในการพัฒนาซอฟต์แวร์ประกอบด้วยเครื่องมือที่หลากหลาย เพื่อสนองการทำงานในแต่ละด้านตั้งแต่การเก็บข้อมูล วิเคราะห์ ออกแบบ พัฒนา ทดสอบ บำรุงรักษาซอฟต์แวร์ และการบริหารจัดการโครงการ แต่ละทีมพัฒนาก็เลือกใช้ เครื่องมือพัฒนาที่แตกต่างกันตามความเหมาะสมโดยคำนึงถึงศักยภาพของทีมและทรัพยากรที่มีอยู่ โดยในบทนี้ ได้อธิบายถึงเครื่องมือหลักที่ใช้ในการพัฒนา ซอฟต์แวร์

สภาพแวดล้อมการพัฒนาโปรแกรม (programming environments)

เป็นเครื่องมือที่ใช้ในการพัฒนาโปรแกรมโดยส่วนมาก จะอยู่ในรูปแบบของสภาพแวดล้อมการพัฒนาโปรแกรม ที่ประกอบด้วยเครื่องมือย่อย สำหรับการพัฒนาโปรแกรมโดยเน้นไปที่การเขียนรหัสต้นฉบับ การสร้างโปรแกรมการรวมโปรแกรมและการทดสอบ โดยเครื่องมือต่าง ๆ ที่ใช้ในการพัฒนาจะถูกนำมาประยุกต์ใช้งานเพื่อสร้างเป็น สภาพแวดล้อมการพัฒนาโปรแกรม ที่มีความเหมาะสมกับทีมพัฒนา อาจมีการรวมเอาระบบบริหารจัดการ รหัสต้นฉบับ และการควบคุมรุ่น เข้ามาใช้งาน เพื่อให้ทีมพัฒนาสามารถทำงานร่วมกันได้อย่างมีประสิทธิภาพ

เครื่องมือในการพัฒนา (development tools)

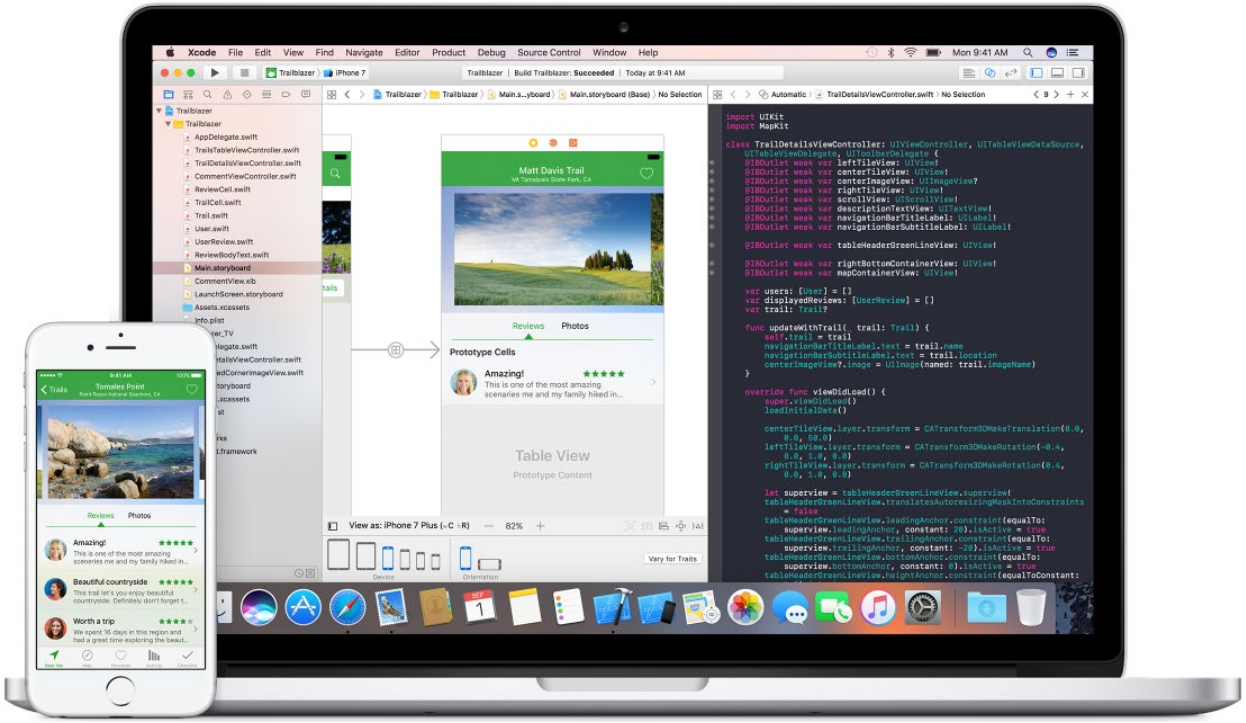
เครื่องมือในการพัฒนาซอฟต์แวร์ที่นักพัฒนาจะคิดถึงเป็นอันดับแรกคือ สภาพแวดล้อมการพัฒนาแบบบูรณาการ หรือ ไอดีอี (integrated development environment: IDE) นักพัฒนาใช้เขียนรหัสต้นฉบับ ครอบคลุมถึงสิ่งอำนวยความสะดวกที่จำเป็นสำหรับการพัฒนาซอฟต์แวร์ แก้ไขรหัสต้นฉบับ คอมไพเลอร์ ตัวแปร ตัวสร้าง เครื่องมือทดสอบ และดีบักเกอร์ ดังรูปที่ 12.1 โดยสามารถใช้ซอฟต์แวร์หรือสร้างเครื่องมือช่วยพัฒนาซอฟต์แวร์ก็สามารถทำได้ ส่วนใหญ่รองรับ GUI และกระบวนทัศน์การเขียนโปรแกรมและภาษาที่หลากหลาย บางเครื่องมือสามารถติดตั้งคุณสมบัติปลั๊กอินเพิ่มเติมช่วยให้สามารถขยายความสามารถในการพัฒนาได้

นักพัฒนาควรพิจารณาภาษาและเทคโนโลยีที่รองรับสำหรับแพลตฟอร์มเป้าหมาย แล้วจึงเลือกไอดีอีที่เหมาะสมสำหรับการพัฒนา ได้แก่ C/C++, C#, Java, Swift, Flash, PHP, และ HTML5 ตัวอย่างไอดีอี Xcode เน้นไปที่ภาษา Swift และ Objective C ในขณะที่ NetBeans เน้นสำหรับภาษา Java นอกจากนั้นการเลือกไอดีอี ที่เหมาะสมสามารถพิจารณาจากเทคโนโลยีต่าง ๆ เช่น: เทคโนโลยีข้ามแพลตฟอร์ม เทคโนโลยีหลังบ้าน การออกแบบส่วนประสานผู้ใช้บูรณาการบริการ เทคโนโลยีข้ามแพลตฟอร์ม ดังนั้นนักพัฒนาต้องระวังในการเลือกเครื่องมือไอดีอี ที่เหมาะสมในการทำงานของทีม

IDE's components

Code editor
Compiler
Interpreter
Builder
Debugger
Object browser
Code completion
Refactoring
Version control
Automatic test
Visual programming

รูปที่ 12.1 ส่วนประกอบหลักของไอดีอี



รูปที่ 12.2 ไอดีอี Xcode [19]

โดยสรุปแล้วไอดีอีมีเป็นเครื่องมือสำคัญในการพัฒนาซอฟต์แวร์ โดยสามารถสนับสนันการพัฒนาซอฟต์แวร์ได้แทบทุกด้านอาจกล่าวได้ว่า นักพัฒนามีเพียงไอดีอีก็สามารถสร้างซอฟต์แวร์ขึ้นมาได้ทั้งระบบ สำหรับค่าใช้จ่ายของไอดีอีมีทั้งแบบเสียและไม่เสียค่าใช้จ่าย มีทั้งระบบใช้งานโดดเดี่ยว ใช้งานเป็นเครือข่าย ใช้งานบนคลาวน์ ให้เลือกตามความต้องการของนักพัฒนา ในส่วนของข้อดีและประโยชน์ในการใช้ไอดีอีพัฒนาซอฟต์แวร์มี ดังนี้

- มีคุณสมบัติและเครื่องมือทั้งหมดที่จำเป็นสำหรับการสร้างซอฟต์แวร์ในชุดเดียวไม่ว่าจะเป็น ฟังก์ชันการพัฒนาโปรแกรมที่จำเป็นทั้งหมด เช่น การเข้ารหัส การทดสอบ การดีบัก และ การคอมไพล์ โดยที่ไม่จำเป็นต้องสลับไปมาระหว่างเครื่องมือต่าง ๆ ทำให้สามารถช่วยประหยัดเวลา
- ตรวจสอบข้อผิดพลาดให้โดยอัตโนมัติ ซึ่งจะช่วยให้สามารถส่งมอบรหัสต้นฉบับคุณภาพสูงได้
- สามารถเพิ่มความสามารถไอดีอี ได้ด้วยระบบปลั๊กอินต่าง ๆ
- อำนวยความสะดวกในการทำงานร่วมกันเป็นทีม

เครื่องมือในการทดสอบ (testing tools)

ในกระบวนการทดสอบซอฟต์แวร์สามารถใช้เครื่องมือในการช่วยในการทดสอบโดยเฉพาะการทดสอบในกระบวนการพัฒนาแบบเอจายล์จะใช้เครื่องมือทดสอบแบบอัตโนมัตเพื่อความสะดวกและรวดเร็วในการทดสอบ นอกจากนั้นเครื่องมือหลักอีกชิ้นคือดีบักเกอร์ที่ใช้สำหรับการตรวจสอบโปรแกรมในระดับลึกและละเอียด เพื่อช่วยให้นักพัฒนาและนักทดสอบสามารถเข้าถึงข้อมูลต่าง ๆ ในขณะที่ซอฟต์แวร์ดำเนินการทดสอบ เครื่องมือในการทดสอบนั้นอาจรวมถึงกรอบการทำงานเพื่อทดสอบ ที่นำมาใช้ร่วมกับการพัฒนาอีกด้วย เช่น การใช้กรอบการทำงานแบบหน่วยย่อยทดสอบ เป็นต้น

เครื่องมือในการออกแบบและสร้างชุดทดสอบ เป็นเครื่องมือที่ช่วยให้สามารถทำการออกแบบและสร้างชุดทดสอบให้ตรงกับคุณลักษณะและความต้องการของซอฟต์แวร์ โดยจะเป็นเครื่องมือที่ช่วยให้สร้างชุดทดสอบให้ตรงและครอบคลุมการทดสอบในระดับที่ยอมรับได้

เครื่องมือช่วยทดสอบ ในกระบวนการทดสอบมีเครื่องมือช่วยในการทดสอบมากมาย ไอดีอีส่วนใหญ่ก็มีเครื่องมือทดสอบเหล่านี้ฝังอยู่ในตัวเพื่อให้นักพัฒนาและนักทดสอบสามารถทำการทดสอบได้อย่างสะดวกและรวดเร็ว มีกรอบการทำงานเพื่อทดสอบให้เลือกใช้งาน เพื่อช่วยสนับสนุนการทดสอบทั้งแบบกึ่งอัตโนมัติและการทดสอบอัตโนมัติ เครื่องมือทดสอบความถูกต้อง เครื่องมือทดสอบประสิทธิภาพ เครื่องมือทดสอบความน่าเชื่อถือ เครื่องมือทดสอบความผิดพร่อง ล็อก ดีบักเกอร์ และนอกจากนั้นยังมีเครื่องมือที่ช่วยติดตามการทดสอบ ปรับปรุงข้อมูลและสถานะต่าง ๆ ในกระบวนการทดสอบ เช่นเครื่องมือติดตามข้อผิดพลาด เครื่องมือจัดการรุ่นของซอฟต์แวร์เพื่อติดตามการทดสอบและการเปลี่ยนแปลงของซอฟต์แวร์ เครื่องมือในการทดสอบสำหรับวิศวกรรมซอฟต์แวร์มีให้เลือกมากทั้งมีและไม่มีค่าใช้จ่าย การเลือกมาใช้งานขึ้นอยู่กับรูปแบบกระบวนการพัฒนา เครื่องมือและภาษาที่ใช้สำหรับการพัฒนา มาตรฐานที่กำหนด และความเหมาะสมกับทีมพัฒนาและองค์กร

เครื่องมือในการบริหาร (management tools)

เครื่องมือในการบริหารอาจกล่าวถึงเครื่องมือในการบริหารโครงการพัฒนาซอฟต์แวร์เป็นหลัก (project management software: PMS) ซึ่งเป็นเครื่องมือที่ใช้ในการบริหารจัดการโครงการให้ดำเนินไปได้อย่างราบลื่นมีเป้าหมายเพื่อให้โครงการดำเนินไปจนประสบความสำเร็จภายใต้ทรัพยากรที่กำหนดและภายในระยะเวลาที่วางแผนไว้ โดยตัวเครื่องมือนั้นอาจสามาถที่จะทำการประมาณการโครงการ วางแผน การจัดกำหนดการ การควบคุมและบริหารงบประมาณ การบริหารทรัพยากร การสื่อสาร การตัดสินใจ การควบคุมคุณภาพ การจัดการเอกสารและรายงาน และการบริการอื่น ๆ ที่เกี่ยวข้องกับการบริหารโครงการ

เป้าหมายของการบริหารโครงการพัฒนาซอฟต์แวร์คือโครงการประสบความสำเร็จสามารถส่งมอบผลิตภัณฑ์ซอฟต์แวร์ที่มีคุณภาพสูงตามมาตรฐานให้กับลูกค้าในเวลาที่กำหนด ในกรอบค่าใช้จ่ายที่คุ้มค่า โดยเครื่องมือที่ใช้อาจอยู่ในรูปแบบ ซอฟต์แวร์ตั้งโต๊ะ ซอฟต์แวร์บนเว็บ ซอฟต์แวร์โอเพนซอร์ส ซอฟต์แวร์บนอุปกรณ์เคลื่อนที่ ซอฟต์แวร์ส่วนตัว ซอฟต์แวร์ผู้ใช้เดียว ซอฟต์แวร์ทำงานร่วมหลายผู้ใช้ โดยจะขึ้นอยู่กับรูปแบบการใช้งาน ขนาดและความซับซ้อนของโครงการ รูปแบบและขนาดของทีมพัฒนาเพื่อเลือกเครื่องมือที่เหมาะสมในการใช้งานในแต่ละโครงการ

กิจกรรมหลักของการบริหารโครงการคือการวางแผนและควบคุมให้โครงการดำเนินไปตามแผนดังนั้นเครื่องมือบริหารโครงการจึงเน้นที่การจัดการกำหนด (scheduling) และการนำเสนอข้อมูลของโครงการเพื่อประกอบการบริหารและตัดสินใจ โดยในส่วนของการจัดการกำหนดจะเน้นการวางแผนกำหนดกิจกรรมต่าง ๆ ในการพัฒนา กำหนดวันและทรัพยากรที่ต้องใช้ สำหรับลายละเอียดในแต่ละกิจกรรเครื่องมือสามารถช่วยสร้างออกมาให้ ขึ้นอยู่กับเครื่องมือที่ใช้และรูปแบบกระบวนการพัฒนา เครื่องมือการจัดการกำหนดอาจมีฟังก์ชัน การกำหนดตารางเวลา การกำหนดทรัพยากร การจัดการเส้นทางวิกฤต การประมาณระยะกิจกรรม การกำหนดค่าใช้จ่าย เป็นต้น

การนำเสนอข้อมูลสารสนเทศโครงการเป็นอีกหนึ่งฟังก์ชันหลักเพื่อนำเสนอข้อมูลโครงการประกอบการบริหารและตัดสินใจ ในรูปแบบต่าง ๆ ขึ้นอยู่กับผู้มีส่วนเกี่ยวข้องในแต่ละบทบาทที่ต่างกัน ซึ่งข้อมูลสารสนเทศเหล่านี้อาจใช้เป็นตัวชี้วัดในการประเมิน การประมาณ หรือการวัดความก้าวหน้าของโครงการโดยมักจะมี

- ข้อมูลสารสนเทศเกี่ยวกับระยะเวลาในแต่ละกิจกรรม
- ภาระงาน
- ข้อมูลสารสนเทศเกี่ยวกับความเสี่ยง
- ข้อมูลสารสนเทศความก้าวหน้าของโครงการ
- ข้อมูลสารสนเทศด้านค่าใช้จ่าย
- หลักฐานเชิงประจักษ์
- การสื่อสารภายในทีมและนอกทีม

เครื่องมือในการรวม (tool integration mechanism)

ในกระบวนการพัฒนาซอฟต์แวร์ที่มีขนาดใหญ่และซับซ้อนการรวมส่วนของซอฟต์แวร์เข้าด้วยกันให้สามารถทำงานร่วมกันได้เป็นงานที่ยุ่งและซับซ้อน ดังนั้นการใช้เครื่องมือในการรวม จึงมีความสำคัญต่อการพัฒนาซอฟต์แวร์โดยมีวัตถุประสงค์เพื่อรวมและประกอบซอฟต์แวร์ ส่วนของซอฟต์แวร์ ระบบย่อย เข้าด้วยกันให้สามารถใช้งานร่วมกันได้ทำงานประสานกันได้โดยไม่มีข้อผิดพลาด โดยเฉพาะการรวมส่วนของซอฟต์แวร์ที่มาจากภายนอก หรือผ่านทางไกล จากหลายแหล่งผู้ผลิต เครื่องมือในการรวมจะช่วยให้การพัฒนาและการดำเนินการสะดวก เข้ากันได้ เพิ่มความน่าเชื่อถือได้ โดยเครื่องมือการรวม สามารถแบ่งเป็นหลายประเภทขึ้นอยู่กับการใช้งานเช่น เครื่องมือการรวมและปรับข้อมูล เครื่องมือการรวมส่วนประกอบซอฟต์แวร์ เครื่องมือการรวมระบบ เครื่องมือการรวมระบบดั้งเดิม และ เครื่องมือการรวมแพลตฟอร์ม

หลักการพื้นฐานเครื่องมือการรวมคือการปรับส่วนต่าง ๆ ให้สามารถทำงานร่วมกันได้ มีคุณภาพ มีความน่าเชื่อถือ มีประสิทธิภาพ และ มีความคุ้มค่า ทั้งการปรับรูปแบบการประสาน รูปแบบข้อมูล ลำดับการทำงาน การประสานการเข้าจังหวะเพื่อให้ส่วนต่าง ๆ ที่นำมาประกอบรวมกันสามารถทำงานเข้ากันได้ เกณฑ์อันดับแรกในการเลือกเครื่องมือมาใช้ในการรวมคือ ความเข้ากันได้และการรองรับ วิศวกรซอฟต์แวร์ต้องตรวจสอบดูว่าเครื่องมือสามารถรองรับการรวมส่วนต่าง ๆ ที่มีอยู่ในระบบหรือไม่ ส่วนประกอบจากผู้ผลิตที่เลือกมาใช้ในระบบ เครื่องมือรองรับหรือไม่ เทคโนโลยีเข้ากันได้หรือไม่ สุดท้ายคือคุ้มค่าหรือไม่

เครื่องมือทางวิศวกรรมซอฟต์แวร์มีให้เลือกใช้งานมากมาย มีทั้งแบบที่มีค่าใช้จ่ายและไม่มีค่าใช้จ่ายในการใช้งาน เทคโนโลยีด้านเครื่องมือพัฒนามีการพัฒนาอย่างต่อเนื่อง มีผลิตภัณฑ์ใหม่ ๆ ออกสู่ท้องตลาดให้เลือกใช้ตลอด เทคโนโลยีและฟังก์ชันการรองรับต่าง ๆ ทางด้านวิศวกรรมซอฟต์แวร์ถูกพัฒนาออกมาเป็นเครื่องมือเพื่อช่วยอำนวยความสะดวก เพิ่มคุณภาพ เพิ่มประสิทธิภาพ และ เพิ่มความคุ้มค่าในการพัฒนาซอฟต์แวร์ โดยเกิดจากการผลักดันจากทางภาคอุตสาหกรรมและทางฝั่งวิชาการจากมหาวิทยาลัยต่าง ๆ เพื่อมุ่งเน้นการสร้างเครื่องมือให้รองรับงานด้านวิศวกรรมซอฟต์แวร์ นอกจากนั้นยังเริ่มมีการใช้ปัญญาประดิษฐ์มาช่วยทำให้เครื่องมือมีประสิทธิภาพและเพิ่มฟังก์ชันการทำงานให้สะดวกมากขึ้น เช่น เครื่องมือระบบตรวจหาข้อผิดพลาดด้วยปัญญาประดิษฐ์หรือการพยายามให้ปัญญาประดิษฐ์เขียนรหัสต้นฉบับหรือแก้ไขรหัสต้นฉบับได้เอง

Bibliography

[1] I. Sommerville, "Software engineering history," 2008. [Online]. Available: https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/History/. [Accessed 11 11 2021].

[2] Software Engineering: A Practitioner's Approach, McGraw-Hill Education, 2019.

[3] “พจนานุกรม ฉบับราชบัณฑิตยสถาน พ.ศ.๒๕๕๔,” [ออนไลน์]. Available: https://dictionary.orst.go.th/. [เข้าถึง 1 1 2021].

[4] "Software Engineering Code of Ethics and Professional Practice," , . [Online]. Available: http://www.acm.org/about/se-code. [Accessed 10 2 2022].

[5] "Guide to Agile Practices," , . [Online]. Available: http://guide.agilealliance.org/. [Accessed 10 2 2022].

[6] K. . Beck, J. . Grenning, R. C. Martin, M. . Beedle, J. . Highsmith, S. . Mellor, A. v. Bennekum, A. . Hunt, K. . Schwaber, A. . Cockburn, R. . Jeffries, J. . Sutherland, W. . Cunningham, J. . Kern, D. . Thomas, M. . Fowler and B. . Marick, "Principles behind the Agile Manifesto," , 2001. [Online]. Available: http://agilemanifesto.org/principles.html. [Accessed 10 2 2022].

[7] "12 Principles Behind the Agile Manifesto - Agile Alliance," , . [Online]. Available: https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/. [Accessed 10 2 2022].

[8] "Agile Manifesto for Software Development | Agile Alliance," [Online]. Available: https://www.agilealliance.org/wp-content/uploads/2019/09/agile-manifesto-download-2019.pdf. [Accessed 11 11 2021].

[9] "หลักการเบื้องหลังคำแถลงอุดมการณ์แห่งอไจล์," [Online]. Available: https://agilemanifesto.org/iso/th/principles.html. [Accessed 10 10 2021].

[10] K. Schwaber and J. Sutherland, "2017-Scrum-Guide-Thai," 2017. [Online]. Available: https://scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Thai.pdf. [Accessed 12 12 2021].

[11] I. Sommerville, Software Engineering, 10th Edition, Pearson, 2016.

[12] G. Erich, R. Helm, R. Johnson, J. Vlissides and G. Booch, Design Patterns: Elements of Reusable Object-Oriented Software, 1st edition, Addison-Wesley Professional, 1994.

[13] S. Bechtold, S. Brannen, J. Link, M. Merdes, M. Philipp, J. d. Rancourt and C. Stein, "JUnit 5 User Guide," junit.org, 12 4 2021. [Online]. Available: https://junit.org/junit5/docs/current/user-guide/. [Accessed 10 4 2022].

[14] W. Feungchan, VIDEO GAME DESIGN, Khon Kaen, 2021.

[15] U. D. o. t. Navy, "Program evaluation research task. Summary report Phase 1 and 2," U.S. Government Printing Office, 1958.

[16] B. Boehm, Software Engineering Economics, Prentice-Hall, 1981.

[17] "CMMI Institute - Home," ISACA, 29 04 2022. [Online]. Available: https://cmmiinstitute.com/. [Accessed 29 04 2022].

[18] "CMMI - เกี่ยวกับ CMMI," SOFTWARE PARK THAILAND, 29 04 2022. [Online]. Available: http://www.swpark.or.th/cmmi/implementguide.php. [Accessed 29 04 2022].

[19] Apple Inc, "Xcode - IDE," Apple, 25 03 2022. [Online]. Available: https://developer.apple.com/xcode/ide/. [Accessed 25 3 2022].