# บทที่ 2
# วงจรการพัฒนาซอฟต์แวร์
# (Software Development Life Cycle)

*วงจรการพัฒนาซอฟต์แวร์ (Software Development Life Cycle - SDLC)* หรือที่เรียกว่า *กระบวนการทางซอฟต์แวร์ (Software Process)* เป็นกลุ่มของกิจกรรม การกระทำ หรืองานที่ต้องปฏิบัติเมื่อต้องการสร้างผลิตภัณฑ์หนึ่งๆ ไม่ใช่เฉพาะส่วนของการสร้างซอฟต์แวร์เท่านั้น แต่กระบวนการพัฒนาซอฟต์แวร์จะครอบคลุมระยะตั้งแต่การเก็บข้อมูล การวิเคราะห์ความต้องการของผู้ใช้ การทวนสอบข้อกำหนดไปจนถึงการฝึกอบรมผู้ใช้ (Pressman, 2010; Sommerville, 2010) ในบทนี้จะเริ่มด้วยการกล่าวถึงระยะต่างๆ ในกระบวนการพัฒนาซอฟต์แวร์ทั่วไป จากนั้นจะกล่าวถึงแบบจำลองชนิดต่างๆ ที่ใช้ในการพัฒนาซอฟต์แวร์

## 2.1 ระยะในการพัฒนาซอฟต์แวร์ (Software Development Phase)

การพัฒนาซอฟต์แวร์ทั่วไปจะครอบคลุมกิจกรรมที่หลากหลาย ซึ่งสามารถแบ่งเป็นออกเป็นระยะต่างๆ ได้ดังนี้ (ISO, 2008)

### 2.1.1 ระยะวางแผนโครงการ (Project Planning)

ระยะวางแผนโครงการเป็นระยะการวางแผนเพื่อกำหนดขอบเขตของโครงการ การจัดเตรียมเอกสาร และการกำหนดแนวทางสำหรับขั้นตอนอื่นๆ ที่เกี่ยวข้อง

### 2.1.2 ระยะการติดต่อและเก็บรวบรวมความต้องการ (Requirements Gathering)

ระยะการติดต่อและเก็บรวบรวมความต้องการเป็นระยะที่ต้องทำการติดต่อประสานงานกับลูกค้าหรือเจ้าของกิจการเพื่อเก็บรวบรวมข้อมูลที่เกี่ยวข้องกับโครงการ (Ralph & Wand, 2009) ในบางกระบวนการโดยเฉพาะกระบวนการแบบอาไจล์จะมีระยะการติดต่อกระจายตัวอยู่ตลอดกระบวนการ

### 2.1.3 ระยะการวิเคราะห์ระบบ (Analysis)

ระยะการวิเคราะห์ระบบเป็นการนำความต้องการของลูกค้าหรือเจ้าของกิจการมาวิเคราะห์และสร้างแบบจำลองเพื่อนำไปใช้สร้างในระยะอื่นๆ โดยสกัดสิ่งที่จำเป็นในการพัฒนาซอฟต์แวร์ออกมาอยู่ในรูปแบบจำลองหรือต้นแบบ

### 2.1.4 ระยะการออกแบบ (Design)

ระยะการออกแบบเป็นระยะที่นำเอาแบบจำลองในระยะการวิเคราะห์มาออกแบบรายละเอียดเกี่ยวกับวิธีการสร้างเพื่อใช้ในระยะการสร้างต่อไป

### 2.1.5 ระยะการสร้าง (Implementation)

ระยะการสร้างเป็นระยะที่นำแบบจำลองที่วิเคราะห์และออกแบบแล้วมาแปลงเป็นต้นรหัส (Source Code) โปรแกรมที่สามารถทำงานได้

### 2.1.6 ระยะทดสอบ (Testing)

ระยะทดสอบเป็นระยะที่สร้างการทดสอบขึ้นมาเพื่อทดสอบโปรแกรมในระดับต่างๆ เช่น การทดสอบระดับหน่วย (Unit Testing) การทดสอบระดับบูรณาการ (Integration Testing) หรือการทดสอบเพื่อการยอมรับ (Acceptance Testing)

### 2.1.7 ระยะการนำไปใช้ (Deployment)

ระยะการนำไปใช้เป็นระยะที่นำซอฟต์แวร์ไปติดตั้งให้ลูกค้าเพื่อใช้งานหรือเพื่อทดสอบและรวบรวมผลตอบรับ

### 2.1.8 ระยะบำรุงรักษา (Maintenance)

ระยะบำรุงรักษาเป็นระยะหลังการนำไปใช้ โดยผู้ใช้จะรายงานข้อผิดพลาดที่เกิดขึ้นจากตัวระบบเพื่อให้ทีมพัฒนารวบรวมและนำไปปรับปรุงโปรแกรมต่อไป

## 2.2 แบบจำลองกระบวนการทางซอฟต์แวร์ (Software Process Model)

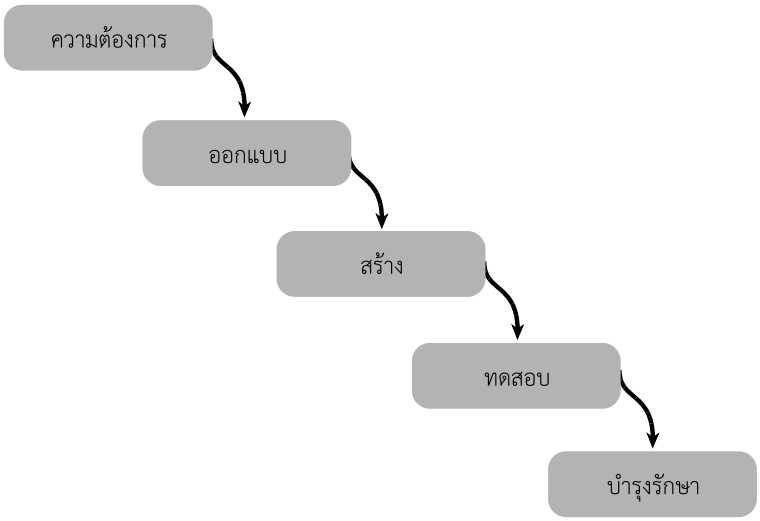
กระบวนการทางซอฟต์แวร์ที่น่าสนในปัจจุบันมีดังต่อไปนี้

1) แบบจำลองน้ำตก (Waterfall Model)
2) แบบจำลองกระบวนการเชิงเพิ่ม (Incremental Process Model)
3) แบบจำลองเชิงวนรอบ (Iterative Model)
4) แบบจำลองเชิงวิวัฒน์ (Evolutionary Model)
5) แบบจำลองสไปรัล (Spiral Model)
6) กระบวนการยูนิฟายด์ (Unified Process)
7) การพัฒนาแบบอาไจล์ (Agile Development)

### 2.2.1 แบบจำลองน้ำตก

แบบจำลองน้ำตก (Waterfall Model) บางครั้งอาจเรียกว่าวงจรพัฒนาแบบดั้งเดิม (Classic Life Cycle) เป็นแนวทางการพัฒนาซอฟต์แวร์ที่มีความเป็นระบบสูงและเป็นลำดับชัดเจน โดยจะเริ่มจากความต้องการซอฟต์แวร์ของลูกค้า ต่อเนื่องไปถึงการวางแผน การออกแบบ การสร้าง และการนำไปใช้ ซึ่งเหมาะกับสถานการณ์ที่มีการเพิ่มความสามารถให้กับระบบที่มีอยู่แล้ว หรือถ้าเป็นการสร้างฟังก์ชันการทำงานใหม่ก็จะเป็นการพัฒนาเพิ่มแบบไม่มากนัก มีความต้องการที่ชัดเจนและไม่เปลี่ยนแปลง (Benington, 1987) ดังรูปที่ 2.1

ต่อมามีแบบจำลองอีกรูปแบบหนึ่งที่ปรับปรุงมาจากแบบจำลองน้ำตก เรียกว่า V-model ที่นำเอาการทดสอบปริมาณเชิงคุณภาพมาเชื่อมโยงไว้กับแต่ละกิจกรรมในแบบจำลองน้ำตก




***รูปที่ 2.1*** *แผนผังแบบจำลองน้ำตก (Benington, 1987)*

แบบจำลองน้ำตกนี้เป็นแบบจำลองกระบวนการที่เก่าแก่ที่สุดในวิศวกรรมซอฟต์แวร์ โดยมีการวิพากษ์วิจารณ์ข้อจำกัดของกระบวนการมาตลอด 40 ปี ซึ่งปัญหาที่พบในแบบจำลองน้ำตกมีดังนี้ (Pressman, 2010; Sommerville, 2010)

1) โครงการในความเป็นจริงจะไม่ค่อยมีกระบวนการที่เป็นลำดับต่อเนื่องกันเป็นเส้นตรงอย่างที่เสนอในแบบจำลอง

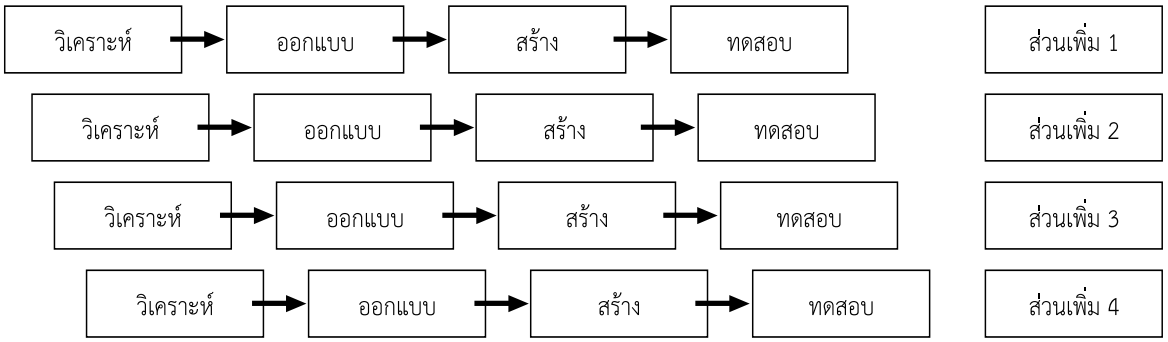
2) เป็นการยากที่ลูกค้าจะสามารถบอกความต้องการทางซอฟต์แวร์ได้อย่างถูกต้องครบถ้วนทั้งหมดตั้งแต่เริ่มโครงการ ซึ่งเป็นสิ่งที่จำเป็นสำหรับกระบวนการแบบน้ำตก

3) ลูกค้าต้องคอยซอฟต์แวร์เป็นเวลานาน เนื่องจากซอฟต์แวร์ที่ทำงานได้จะไม่ปรากฏจนกระทั่งช่วงสุดท้ายของโครงการ ทำให้ความผิดพลาดบางอย่างจะไม่ได้ตรวจพบจนกว่าซอฟต์แวร์จะได้รับการตรวจสอบในช่วงท้าย ซึ่งอาจทำให้เกิดความเสียหายกับโครงการได้ ในการวิเคราะห์กระบวนการนี้จากโครงการจริงพบว่ากระบวนการที่เป็นลำดับต่อเนื่องกันเป็นเส้นตรงอาจทำให้เกิด “การรอ” เพราะสมาชิกในทีมต้องคอยสมาชิกอื่นทำงานบางอย่างให้เสร็จก่อน

### 2.2.2 แบบจำลองกระบวนการเชิงเพิ่ม (Incremental Process Model)

แบบจำลองกระบวนการเชิงเพิ่ม จะรวมเอาลักษณะแบบจำลองที่เป็นลำดับต่อเนื่องกันเป็นเส้นตรงเข้ากับขั้นตอนการทำงานเชิงขนาน โดยที่ปลายกระบวนการของแต่ละรอบจะผลิต “ส่วนเพิ่ม” (Increment) ของซอฟต์แวร์ที่สามารถนำส่งได้ (Larman & Basili, 2003; Pressman, 2010)

ในแบบจำลองนี้จะเรียกรอบแรกว่าการพัฒนาผลิตภัณฑ์หลัก นั่นคือระยะปลายกระบวนการในรอบแรกนั้นจะมีการนำส่งส่วนเพิ่มซึ่งมีคุณสมบัติ (Feature) ที่จำเป็นในขณะที่ความต้องการเพิ่มเติมนั้นจะนำไปพัฒนาในรอบถัดๆ ไป กระบวนการเชิงเพิ่มเหมาะสมกับสถานการณ์ที่จุดเริ่มต้นของโครงการนั้นมีการเตรียมการไว้ค่อนข้างดี แต่มีกระบวนการพัฒนาที่ไม่ต่อเนื่องเป็นเส้นตรงและอาจมีความต้องการให้ผู้ใช้สามารถเข้าถึงฟังก์ชันการทำงานของซอฟต์แวร์เพียงจำนวนหนึ่งก่อนแล้วจึงค่อยขยายความสามารถต่างๆ เพิ่มเติมในภายหลังดังรูปที่ 2.2

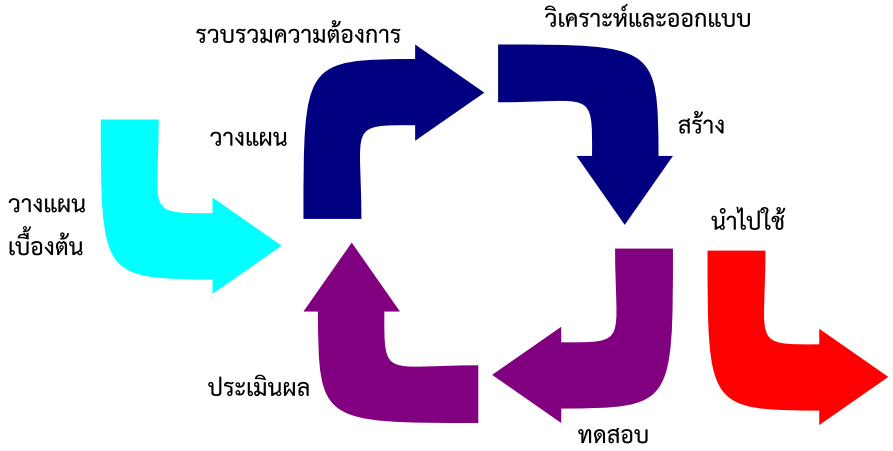



***รูปที่ 2.2*** *แบบจำลองกระบวนการเชิงเพิ่ม (Pressman, 2010)*

**2.2.3 แบบจำลองเชิงวนรอบ (Iterative Model)**

แบบจำลองเชิงวนรอบเป็นแบบจำลองการพัฒนาซอฟต์แวร์ที่อนุญาตให้เกิดการพัฒนาแบบวนรอบได้เพื่อแก้ไขจุดอ่อนสำคัญของกระบวนการแบบน้ำตก แบบจำลองนี้จะเริ่มด้วยการวางแผนและจบด้วยการนำซอฟต์แวร์ไปใช้ โดยมีกิจกรรมของการจัดการความต้องการ การวิเคราะห์และออกแบบระบบ การสร้าง การทดสอบ และการประเมินผลอยู่ในแต่ละรอบ (Larman & Basili, 2003; Pressman, 2010) ตามรูปที่ 2.3

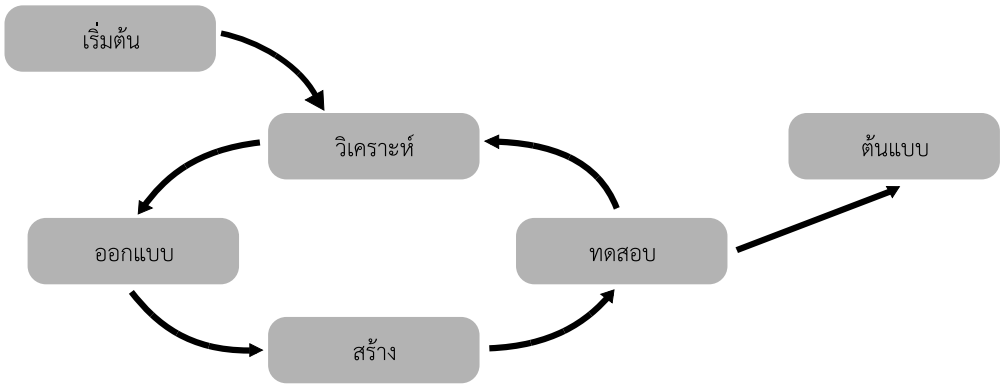
การวนรอบของแบบจำลองมีข้อดีคือทำให้ทีมพัฒนาสามารถเรียนรู้จากการสร้างซอฟต์แวร์ในรอบก่อนหน้านี้เพื่อประเมินและปรับปรุงการทำงานสำหรับรอบต่อๆ ไปได้ ทั้งแบบจำลองเชิงวนรอบและแบบจำลองกระบวนการเชิงเพิ่มเป็นหัวใจสำคัญสำหรับแบบจำลองอื่นๆ เช่น กระบวนการยูนิฟายด์ (Unified Process) และการพัฒนาแบบอาไจล์ (Agile Development)




***รูปที่ 2.3*** *แบบจำลองเชิงวนรอบ (Pressman, 2010)*

### 2.2.4 แบบจำลองเชิงวิวัฒน์ (Evolutionary Model)

แบบจำลองเชิงวิวัฒน์เป็นแบบจำลองที่ใช้เพื่อสร้างให้ซอฟต์แวร์ค่อยๆ สมบูรณ์ขึ้นในแต่ละรอบของการพัฒนาในสถานการณ์ที่ความต้องการซอฟต์แวร์หลักชัดเจนแล้วแต่รายละเอียดของตัวซอฟต์แวร์ที่จะสร้างยังไม่ครบถ้วน กระบวนการในแบบจำลองนี้จะเริ่มจากการสร้างต้นแบบ (Prototyping) ขึ้นมาจากความต้องการที่มีก่อน จากนั้นจึงนำต้นแบบไปให้ผู้ใช้ได้ทดลองใช้งานก่อนแล้วนำผลตอบรับที่ได้มาปรับปรุงต้นแบบให้สมบูรณ์ขึ้นจนกลายเป็นผลิตภัณฑ์จริงที่ทำงานได้ครบถ้วน (Pressman, 2010)

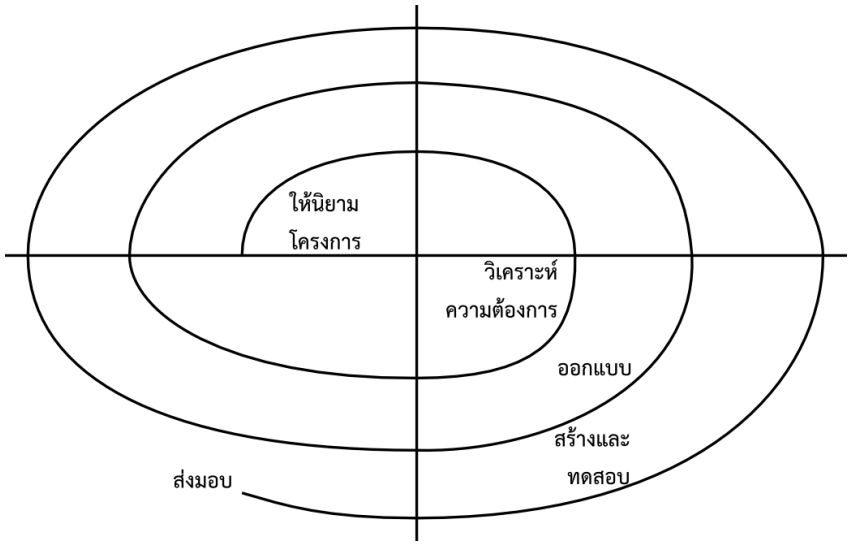



***รูปที่ 2.4*** *แบบจำลองเชิงวิวัฒน์ (Pressman, 2010)*

แนวทางการสร้างต้นแบบมี 2 แนวคิดใหญ่ๆ คือ 1) การสร้างต้นแบบแล้วทิ้ง และ 2) การสร้างต้นแบบแล้วค่อยๆ ปรับเป็นระบบจริง ในแบบจำลองนี้จะสนับสนุนแนวทางการสร้างต้นแบบเพื่อให้สามารถปรับปรุงเป็นระบบจริงที่ใช้งานได้ต่อไป

### 2.2.5 แบบจำลองสไปรัล (Spiral Model)

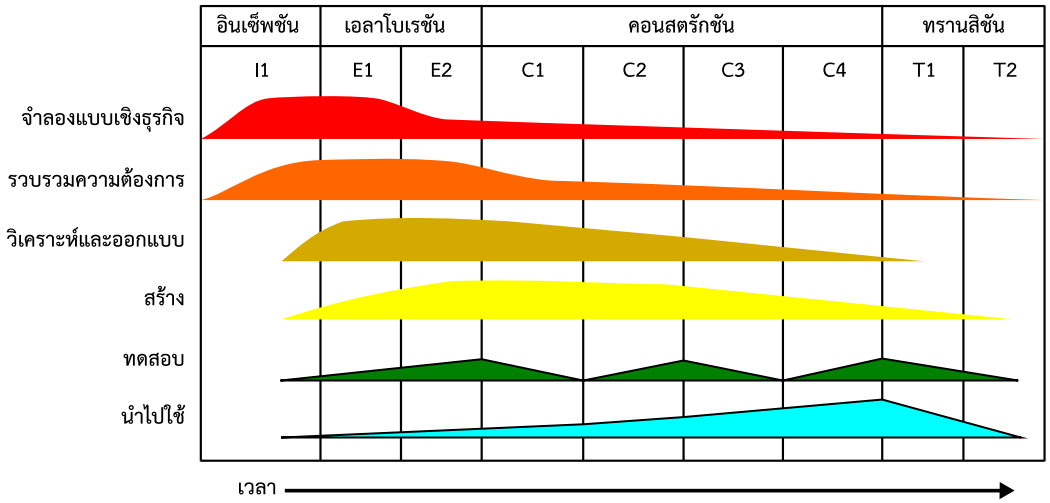
แบบจำลองสไปรัล เป็นการรวมเอากระบวนการแบบจำลองเชิงวนรอบเข้ากับความเป็นระบบของกระบวนการแบบน้ำตก (Boehm, 1986) โดยแต่ละรอบของการพัฒนาเรียกว่า “สไปรัล” จะมีช่วงกว้างประมาณ 6 เดือนถึง 2 ปี ซึ่งผลลัพธ์จากแต่ละรอบจะเป็นซอฟต์แวร์ที่สมบูรณ์ขึ้นเรื่อยๆ ในลักษณะเดียวกับผลที่ได้ตามแบบจำลองเชิงวิวัฒน์ หรืออาจเป็นสิ่งที่สามารถส่งมอบได้ เช่น ข้อกำหนดของซอฟต์แวร์ (Software Specification) หรือแบบจำลองจากการออกแบบ เป็นต้น ในรอบแรกของสไปรัลมักจะเป็นการกำหนดข้อกำหนดของซอฟต์แวร์ จากนั้นในสไปรัลที่สองและถัดมาจะเป็นซอฟต์แวร์ที่สมบูรณ์ขึ้นหรือซับซ้อนขึ้น ในแต่ละรอบจะมีการผ่านระยะการวางแผน ซึ่งในจุดนี้จะเป็นการวิเคราะห์ต้นทุน ตารางเวลา และปรับให้เหมาะสมตามการตอบรับที่ได้จากลูกค้า ซึ่งทดลองใช้ระบบที่ส่งมอบจากสไปรัลก่อนหน้านี้




***รูปที่ 2.5*** *แบบจำลองสไปรัล (Boehm, 1986)*

### 2.2.6 กระบวนการยูนิฟายด์ (Unified Process)

กระบวนการยูนิฟายด์ เป็นกระบวนการพัฒนาซอฟต์แวร์ที่พยายามดึงเอาคุณสมบัติและลักษณะเด่นของหลายๆ แบบจำลองมาไว้ด้วยกัน กระบวนการยูนิฟายด์ให้ความสำคัญกับการสื่อสารกับลูกค้าและอธิบายความต้องการของซอฟต์แวร์ให้อยู่ในรูปแบบยูสเคส (Use Case) จากมุมมองของลูกค้า อีกทั้งกระบวนการยูนิฟายด์ยังเน้นความสำคัญของสถาปัตยกรรมซอฟต์แวร์และใช้กระบวนการทำงานเป็นแบบวนรอบ ซึ่งแต่ละรอบจะได้ผลลัพธ์เป็นส่วนเพิ่มของซอฟต์แวร์ (Jacobson et al., 1999; Kruchten, 2003) รอบหนึ่งรอบในระยะต่างๆ กันของกระบวนการยูนิฟายด์จะทำกิจกรรมในกระบวนการไม่เท่ากัน จากรูปที่ 2.6 เป็นการพัฒนาแบบวนรอบของกระบวนการยูนิฟายด์ที่มีการส่งมอบงานที่เกิดขึ้นอย่างต่อเนื่องตามรอบที่อยู่ในกรอบเวลาและมีการทำงานหลายกิจกรรม กระบวนการยูนิฟายด์แบ่งออกเป็น 4 ระยะดังต่อไปนี้




***รูปที่ 2.6*** *ระยะการพัฒนาต่างๆ ในกระบวนการยูนิฟายด์ (Kruchten, 2003)*

**1) ระยะอินเซ็พชัน (Interception)** ในระยะอินเซ็พชันจะเน้นการสื่อสารกับลูกค้าและการวางแผน มีการเริ่มวางสถาปัตยกรรม และมีการออกแบบ การสร้าง และการทดสอบบ้างเล็กน้อย นอกจากนี้ยังมีการเก็บความต้องการให้อยู่ในรูปแบบของยูสเคสเบื้องต้น

**2) ระยะเอลาโบเรชัน (Elaboration)** ในระยะเอลาโบเรชันนี้จะเน้นการสื่อสารมากที่สุด โดยทำการลงรายละเอียดยูสเคสที่ได้จากระยะอินเซ็พชันให้มากขึ้น มีการวางสถาปัตยกรรมที่ครบถ้วน และสร้างแบบจำลองยูสเคส แบบจำลองการวิเคราะห์ แบบจำลองการออกแบบ แบบจำลองการสร้าง และแบบจำลองการนำไปใช้ ในช่วงปลายของระยะนี้อาจจะมีการสร้างซอฟต์แวร์ที่ทำงานได้เป็นตัวแรกออกมาให้กับผู้ใช้ รวมทั้งแผนงานจะถูกตรวจทานเพื่อให้มั่นใจว่าระยะเวลาสำหรับการส่งงานยังสมเหตุสมผลอยู่หรือไม่ ซึ่งการเปลี่ยนแปลงแผนมักจะกระทำในระยะนี้

**3) ระยะคอนสตรักชัน (Construction)** ระยะคอนสตรักชันเป็นระยะที่เน้นไปยังการพัฒนาโดยใช้แบบจำลองเชิงสถาปัตยกรรมเป็นตัวป้อนเข้า ในระยะนี้จะเป็นการสร้างหรือหาชิ้นส่วนซอฟต์แวร์ (Software Component) มาใช้เพื่อทำให้ยูสเคสสามารถทำงานได้จริง และเพื่อให้การพัฒนาเป็นไปอย่างลุล่วง แบบจำลองต่างๆ ควรมีความสมบูรณ์มาจากระยะเอลาโบเรชัน ทุกๆ คุณสมบัติที่จำเป็นสำหรับส่วนเพิ่มนี้จะถูกสร้างออกมาเป็นซอฟต์แวร์ในรูปแบบของต้นรหัส และในทุกๆ ชิ้นส่วนที่พัฒนาก็จะมีการทดสอบระดับหน่วยประกอบไปด้วย ในส่วนปลายระยะอาจจะมีการประกอบชิ้นส่วนซอฟต์แวร์เข้าด้วยกันและมีการทดสอบเพื่อการยอมรับสำหรับแต่ละยูสเคสก่อนจะเข้าสู่ระยะต่อไป

**4) ระยะทรานสิชัน (Transition)** ระยะทรานสิชันเป็นระยะที่รวมเอาช่วงสุดท้ายของกระบวนการสร้าง และช่วงแรกของกระบวนการนำไปใช้เข้าด้วยกัน โดยจะนำซอฟต์แวร์ไปให้ผู้ใช้ทดสอบเพื่อรวบรวมผลตอบรับ จากนั้นก็จะเป็นขั้นตอนการเขียนคู่มือ เอกสารการแก้ปัญหา และวิธีการติดตั้ง เป็นต้น ในปลายระยะนี้ ส่วนเพิ่มของซอฟต์แวร์จะกลายเป็นการปล่อย (Release) ซอฟต์แวร์ที่ใช้งานได้

### 2.2.7 การพัฒนาแบบอาไจล์ (Agile Development)

กระบวนการพัฒนาแบบอาไจล์ เป็นการรวมเอาปรัชญาและแนวทางแบบอาไจล์เข้าไว้ด้วยกัน โดยปรัชญาของการพัฒนาแบบอาไจล์จะเน้นการทำให้ลูกค้าพึงพอใจ การส่งมอบซอฟต์แวร์ในระยะสั้นๆ อย่างต่อเนื่อง การมีทีมขนาดเล็กที่พร้อมและคล่องตัว การใช้หลักวิศวกรรมซอฟต์แวร์อย่างพอดี และการพัฒนาที่มีกระบวนการเรียบง่ายและมีประสิทธิภาพ สำหรับแนวทางการพัฒนาแบบอาไจล์จะเน้นการวิเคราะห์และออกแบบให้พอเหมาะเพื่อให้ซอฟต์แวร์สามารถส่งมอบได้ทันเวลาและมีการสื่อสารกันอย่างต่อเนื่องระหว่างลูกค้าและนักพัฒนา (Highsmith, 2002)

การพัฒนาแบบอาไจล์สร้างขึ้นเพื่อตอบสนองการเปลี่ยนแปลงที่เกิดขึ้นอย่างรวดเร็วตามสถานการณ์การพัฒนาซอฟต์แวร์ในปัจจุบัน ซึ่งมีผลทำให้ต้นทุนในการพัฒนาซอฟต์แวร์เพิ่มมากขึ้นตามไปด้วย การเปลี่ยนแปลงจึงเป็นสิ่งที่อันตรายหากไม่มีการจัดการที่เหมาะสม การพัฒนาแบบอาไจล์มีข้อดีที่จะสามารถช่วยลดต้นทุนของการเปลี่ยนแปลงในระหว่างการพัฒนา โดยต้นทุนของการเปลี่ยนแปลงจะเพิ่มมากขึ้นอย่างไม่เป็นเส้นตรงระหว่างเวลาที่โครงการกำลังดำเนินไป การเปลี่ยนแปลงนั้นสามารถทำได้ง่ายในช่วงต้นของโครงการ แต่จะทำได้ยากมากเมื่อโครงการดำเนินไปแล้วหลายเดือน และถ้าการเปลี่ยนแปลงมีผลกระทบกับตัวสถาปัตยกรรมจะทำให้ต้นทุนของการเปลี่ยนแปลงดังกล่าวเพิ่มมากขึ้นอีกหลายเท่าตัว

จากการศึกษาพบว่าการพัฒนาแบบอาไจล์ช่วยให้กราฟต้นทุนของการเปลี่ยนแปลงดังกล่าวแบนลง นั่นคือยอมให้การเปลี่ยนแปลงเกิดขึ้นในช่วงปลายของการพัฒนาได้โดยไม่เกิดผลกระทบมากนัก และหนึ่งในหลักการของอาไจล์คือ การยินดีที่จะเปลี่ยนแปลงความต้องการของซอฟต์แวร์แม้จะอยู่ในช่วงปลายของการพัฒนาก็ตาม (Pressman, 2010)

การพัฒนาแบบอาไจล์ที่น่าสนใจในปัจจุบันมีดังต่อไปนี้

1) การพัฒนาแบบสกรัม (Scrum Development)
2) การจำลองแบบเชิงอาไจล์ (Agile Modeling)
3) กระบวนการยูนิฟายด์เชิงอาไจล์ (Agile Unified Process)
4) เอ็กซ์ตรีมโปรแกรมมิง (Extreme Programming)
5) กระบวนการยูนิฟายด์แบบเปิด (OpenUP)

อย่างไรก็ดีการพัฒนาแบบอาไจล์เป็นแนวคิดที่แฝงอยู่ในกระบวนการพัฒนาซอฟต์แวร์มาตั้งแต่อดีตแต่ก็เพิ่งมีแนวทางชัดเจนเมื่อไม่นานมานี้ โดยสรุปแล้วการพัฒนาแบบอาไจล์สร้างขึ้นมาเพื่อแก้ไขจุดอ่อนของวิศวกรรมซอฟต์แวร์แบบดั้งเดิม ซึ่งแม้ว่าวิธีการแบบอาไจล์นี้จะมีประโยชน์ที่เห็นได้อย่างชัดเจน แต่ก็คล้ายกับกระบวนการพัฒนาประเภทอื่นที่ไม่สามารถใช้ได้กับทุกโครงการ ทุกองค์กร หรือทุกบุคคลเสมอไป โดยต้องมีการปรับใช้อย่างเหมาะสมเพื่อให้เกิดประโยชน์สูงสุด

## 2.3 บทสรุป

ในบทนี้ได้กล่าวถึงวงจรการพัฒนาซอฟต์แวร์หรือที่เรียกกันว่ากระบวนการพัฒนาซอฟต์แวร์ ซึ่งโครงสร้างทั่วไปจะมีวงจรการพัฒนาซอฟต์แวร์ที่ประกอบไปด้วยระยะในการพัฒนาจำนวนหนึ่งที่ครอบคลุมตั้งแต่การวางแผน การรวบรวมความต้องการ ไปจนถึงการนำไปใช้และการบำรุงรักษา ระยะเหล่านี้อาจวางอยู่ในลำดับเวลาที่แตกต่างกันในแต่ละแบบจำลองของกระบวนการพัฒนา ซึ่งในบทนี้ได้กล่าวถึงแบบจำลองประเภทต่างๆ ที่พบตั้งแต่ยุคเริ่มต้นของวิศวกรรมซอฟต์แวร์ เช่น แบบจำลองน้ำตก จนถึงแบบจำลองที่คิดค้นขึ้นในช่วงปัจจุบัน เช่นกระบวนการยูนิฟายด์ และการพัฒนาแบบอาไจล์ ซึ่งการพัฒนาแบบอาไจล์นั้นจะกล่าวถึงรายละเอียดต่อไปในบทที่ 3

## 2.4 คำถามท้ายบทที่ 2

1) SDLC คืออะไร มีระยะอะไรบ้าง

2) แบบจำลองน้ำตก คืออะไร มีข้อดีและข้อเสียอย่างไร

3) แบบจำลองเชิงเพิ่ม คืออะไร มีข้อดีและข้อเสียอย่างไร

4) แบบจำลองเชิงวนรอบ คืออะไร มีข้อดีและข้อเสียอย่างไร

5) แบบจำลองเชิงวิวัฒน์ คืออะไร มีข้อดีและข้อเสียอย่างไร

6) อธิบายความแตกต่างระหว่างแบบจำลองเชิงเพิ่ม และแบบจำลองเชิงวนรอบ

7) อธิบายความแตกต่างระหว่างแบบจำลองเชิงเพิ่ม และแบบจำลองเชิงวิวัฒน์

8) อธิบายความแตกต่างระหว่างแบบจำลองเชิงวนรอบ และแบบจำลองเชิงวิวัฒน์

9) ระยะการพัฒนาในกระบวนการยูนิฟายด์มีอะไรบ้าง

10) ค้นคว้ากระบวนการการพัฒนาซอฟต์แวร์ที่พบในปัจจุบันเพิ่มเติมจากอินเตอร์เน็ต