บทที่ 2
สรุปสาระสำคัญจากเอกสารที่เกี่ยวข้อง

2.1 วงจรการพัฒนาซอฟต์แวร์ (Software Development Life Cycle, SDLC)

วงจรการพัฒนาระบบซอฟต์แวร์มีวงชีวิตเหมือนกับสิ่งมีชีวิตตั้งแต่เกิดจนตาย วงจรนี้เป็นขั้นตอนที่เป็นลำดับตั้งแต่ต้นจนเสร็จเรียบร้อย เป็นระบบที่ใช้ได้ซึ่งนักวิเคราะห์ระบบต้องทำความเข้าใจว่าในแต่ละขั้นตอนจะต้องทำอะไรและอย่างไร ขั้นตอนในการพัฒนาระบบมีด้วยกัน 7 ขั้นตอนดังนี้

1. เข้าใจปัญหา (Problem Recognition)
2. ศึกษาความเป็นไปได้ (Feasibility Study)
3. วิเคราะห์ (Analysis)
4. ออกแบบ (Design)
5. พัฒนาระบบ (Construction)
6. การปรับเปลี่ยน (Conversion)
7. บำรุงรักษา (Maintenance)

ในโครงการแต่ละโครงการจะมีขั้นตอนดังกล่าวอยู่ในการพัฒนา ในอดีตนักพัฒนาระบบใช้วิธีการที่เรียกว่าแนวทางการพัฒนาแบบน้ำตก (Waterfall approach) ในการดำเนินการตามเทคนิคเอสดีแอลซี (SDLC) คือจะมีการดำเนินงานแต่ละขั้นตอนให้เสร็จเรียบร้อยก่อนที่จะดำเนินการในขั้นตอนต่อไป แต่ในปัจจุบันนักพัฒนาระบบอาจจะดำเนินการย้อนกลับไปกลับมาได้ตามความจำเป็น

ขั้นตอนที่ 1 เข้าใจปัญหา (Problem Recognition)

ก่อนที่จะเริ่มพัฒนาระบบใด ๆ ผู้พัฒนาจำเป็นต้องรู้จักและเข้าใจปัญหาว่าอะไรต้องทำการกำหนดปัญหาให้ได้ก่อน ถือเป็นโจทย์ที่ต้องทำการแก้ไขด้วยขั้นตอนต่อไปของการพัฒนาระบบ

ขั้นตอนที่ 2 ศึกษาความเป็นไปได้ (Feasibility Study)

จุดประสงค์ของการศึกษาความเป็นไปได้ คือ การกำหนดว่าปัญหาคืออะไรและตัดสินใจว่าการพัฒนาสร้างระบบใหม่หรือการแก้ไขระบบเดิมมีความเป็นไปได้หรือไม่โดยเสียค่าใช้จ่ายและเวลาน้อยที่สุดและได้ผลเป็นที่น่าพอใจ นักวิเคราะห์ระบบจะต้องกำหนดให้ได้ว่าการแก้ไขปัญหา

ดังกล่าวมีความเป็นไปได้ทางเทคนิคและบุคลากรรวมถึงปัจจัยด้านอื่น ๆ ที่เกี่ยวข้องกับการพัฒนาสุดท้ายนักวิเคราะห์ระบบต้องวิเคราะห์ให้ได้ว่า ความเป็นไปได้เรื่องค่าใช้จ่ายรวมถึงเวลาที่ใช้ในการพัฒนาระบบและสิ่งที่สำคัญคือผลประโยชน์ที่จะได้รับมีความคุ้มค่าคุ้มทุนหรือไม่ เพื่อจะดำเนินการในขั้นตอนต่อไป

ขั้นตอนที่ 3 วิเคราะห์ (Analysis)

เริ่มเข้าสู่การวิเคราะห์ระบบ การวิเคราะห์ระบบเริ่มต้นตั้งแต่การศึกษาระบบการทำงานของธุรกิจนั้น ในกรณีที่ระบบที่เราศึกษานั้นเป็นระบบที่รู้อยู่แล้วจะต้องศึกษาว่าทำงานอย่างไร เพราะเป็นการยากที่จะออกแบบระบบใหม่โดยที่ไม่ทราบว่าระบบเดิมทำงานอย่างไรหรือธุรกิจดำเนินการอย่างไร หลังจากนั้นกำหนดความต้องการของระบบใหม่เพื่อนำเข้าสู่กระบวนการในขั้นตอนการออกแบบต่อไป

ขั้นตอนที่ 4 การออกแบบ (Design)

ทำการออกแบบระบบใหม่ให้สอดคล้องกับความต้องการของผู้ใช้และฝ่ายบริหาร ซึ่งผลลัพธ์ที่ได้ในขั้นตอนนี้คือ ข้อมูลเฉพาะของการออกแบบ (System Design Specification) โดยอาศัยเครื่องมือต่าง ๆ ในการออกแบบ เช่น แผนภาพการไหลของข้อมูล (Data Flow Diagram), พจนานุกรมข้อมูล (Data Dictionary), รูปแบบข้อมูล (Data Model), ผังงานโครงสร้าง (Structure Charts) เป็นต้น

ขั้นตอนที่ 5 การพัฒนาระบบ (Construction)

ในขั้นตอนนี้โปรแกรมเมอร์จะเริ่มเขียนโปรแกรมและทดสอบโปรแกรมว่าทำงานถูกต้องหรือไม่ ต้องมีการทดสอบกับข้อมูลจริงที่เลือกแล้ว ถ้าทุกอย่างเรียบร้อย จะได้โปรแกรมที่พร้อมที่จะนำไปใช้งานได้จริงต่อไป

ขั้นตอนที่ 6 การปรับเปลี่ยน (Conversion)

เป็นขั้นตอนที่จะนำระบบใหม่มาใช้แทนระบบเก่าภายใต้การดูแลของนักวิเคราะห์ระบบ การป้อนข้อมูลต้องทำให้เรียบร้อยและในที่สุดลูกค้าจะเริ่มต้นใช้งานระบบใหม่ได้ ซึ่งการนำระบบใหม่เข้ามาแทนระบบเก่าควรทำอย่างค่อยเป็นค่อยไปทีละน้อย ที่ดีที่สุดคือ ใช้ระบบใหม่ควบคู่ไปกับระบบเก่าสักระยะหนึ่ง โดยใช้ข้อมูลชุดเดียวกันแล้วเปรียบเทียบผลลัพธ์ว่าตรงกันหรือไม่ ถ้าเรียบร้อยก็พร้อมที่จะเอาระบบเก่าออกได้ แล้วใช้ระบบใหม่ต่อไป

ขั้นตอนที่ 7 บำรุงรักษา (Maintenance)

การบำรุงรักษาได้แก่ การแก้ไขโปรแกรมหลังจากการใช้งานแล้ว สาเหตุที่ต้องแก้ไขโปรแกรมหลังจากใช้งานแล้วส่วนใหญ่ มี 2 ข้อ คือ 1. มีปัญหาในโปรแกรม (Bug) และ 2. การดำเนินงานในองค์กรหรือธุรกิจเปลี่ยนไป

จากสถิติของระบบที่พัฒนาแล้วทั้งหมดประมาณ 40% ของค่าใช้จ่ายใช้ไปในการแก้ไขโปรแกรม เนื่องจากมีบัก (Bug) ดังนั้นนักวิเคราะห์ระบบควรให้ความสำคัญกับการบำรุงรักษาซึ่งปกติจะคิดว่าไม่มีความสำคัญมากนัก

ขั้นตอนและสิ่งที่ต้องทำในแต่ละขั้นตอนของการพัฒนาระบบตามแบบเอสดีแอลซี (SDLC) สามารถสรุปได้ดังตารางที่ 2.1

ขั้นตอน	**สิ่งที่ต้องทำ**
1. เข้าใจปัญหา	1. ตระหนักว่ามีปัญหาในระบบ
2. ศึกษาความเป็นไปได้	1. รวบรวมข้อมูล 2. คาดคะเนค่าใช้จ่าย ผลประโยชน์และอื่นๆ 3. ตัดสินใจว่าจะเปลี่ยนแปลงระบบหรือไม่
3. วิเคราะห์	1. ศึกษาระบบเดิม 2. กำหนดความต้องการของระบบ 3. แผนภาพระบบเก่าและระบบใหม่ 4. สร้างระบบทดลองของระบบใหม่
4. ออกแบบ	1. เลือกซื้อคอมพิวเตอร์ฮาร์ดแวร์และซอฟต์แวร์ 2. เปลี่ยนแผนภาพจากการวิเคราะห์เป็นแผนภาพลำดับขั้น 3. คำนึงถึงความปลอดภัยระบบ 4. ออกแบบ input และ output 5. ออกแบบไฟล์ฐานข้อมูล
5. พัฒนา	1. เตรียมสถานที่ 2. เขียนโปรแกรม 3. ทดสอบโปรแกรม 4. เตรียมคู่มือการใช้และฝึกอบรม

ขั้นตอน (ต่อ)	สิ่งที่ต้องทำ (ต่อ)
6. นำมาใช้งานจริง	1. ป้อนข้อมูล 2. เริ่มใช้งานระบบใหม่
7. บำรุงรักษา	1. เข้าใจปัญหา 2. ศึกษาสิ่งที่จะต้องแก้ไข 3. ตัดสินใจว่าจะแก้ไขหรือไม่ 4. แก้ไขเอกสาร คู่มือ 5. แก้ไขโปรแกรม 6. ทดสอบโปรแกรม 7. ใช้งานระบบที่แก้ไขแล้ว

ตารางที่ 2.1 สรุปวงจรการพัฒนาระบบ

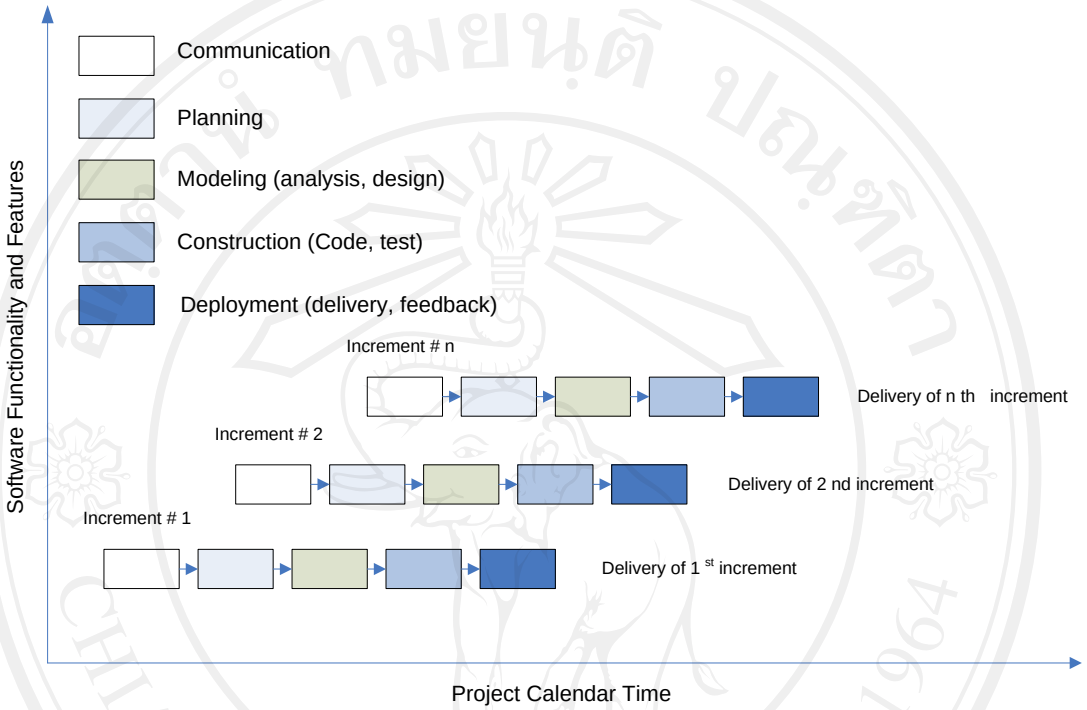
2.2 แบบจำลองกระบวนการค่อยเพิ่มขึ้น (Incremental Process Model)

โรเจอร์ เอส เพรสแมน (Roger S. Pressman) [1] กล่าวถึงแบบจำลองแบบค่อยเพิ่มขึ้นว่า มีบางสถานการณ์ที่ความต้องการเริ่มแรกมีความชัดเจน แต่ขอบเขตงานทั่วไปไม่อาจกำหนดเป็นแบบเชิงเส้นได้หรือมีความจำเป็นต้องสร้างซอฟต์แวร์ที่พอทำงานเริ่มต้นได้ให้ลูกค้าโดยเร็ว จากนั้นจึงขยายหน้าที่การทำงานให้ดีขึ้นในเวอร์ชั่นถัดไป ในกรณีดังกล่าวกระบวนการแบบค่อยเพิ่มขึ้นจะเหมาะสมกับการใช้งาน

2.2.1. แบบจำลองค่อยเพิ่มขึ้น (The Incremental Model)

แบบจำลองค่อยเพิ่มขึ้น ผสมส่วนประกอบของแบบจำลองน้ำตกกับการทำวนซ้ำ รูปที่ 2.1 แสดงให้เห็นว่า แบบจำลองค่อยเพิ่มขึ้นประยุกต์ลำดับเชิงเส้นหลายลำดับ โดยวางให้เหลื่อมกันตามเวลาปฏิบัติแต่ละลำดับเชิงเส้นผลิตซอฟต์แวร์รุ่นใหม่ที่ส่งมอบได้ออกมา ตัวอย่างเช่น เวิร์ดโปรเซสเซอร์ที่พัฒนาโดยใช้แบบจำลองนี้ รุ่นแรกอาจส่งมอบเพียงหน้าที่จัดการไฟล์พื้นฐาน การตัดต่อ และการสร้างเอกสารพื้นฐาน แต่ในรุ่นที่สองอาจเพิ่มหน้าที่การตัดต่อและการสร้างเอกสารให้ดียิ่งขึ้น ในรุ่นที่สามอาจเพิ่มการตรวจสอบสะกดคำ และตรวจสอบไวยากรณ์ ในรุ่นที่สี่อาจเพิ่มความสามารถในการจัดเลย์เอาท์หน้าพิมพ์ โปรดสังเกตว่า กระแสกระบวนการของแต่ละรุ่นที่ผลิตออกมา อาจมีการสร้างต้นแบบรวมอยู่ด้วย

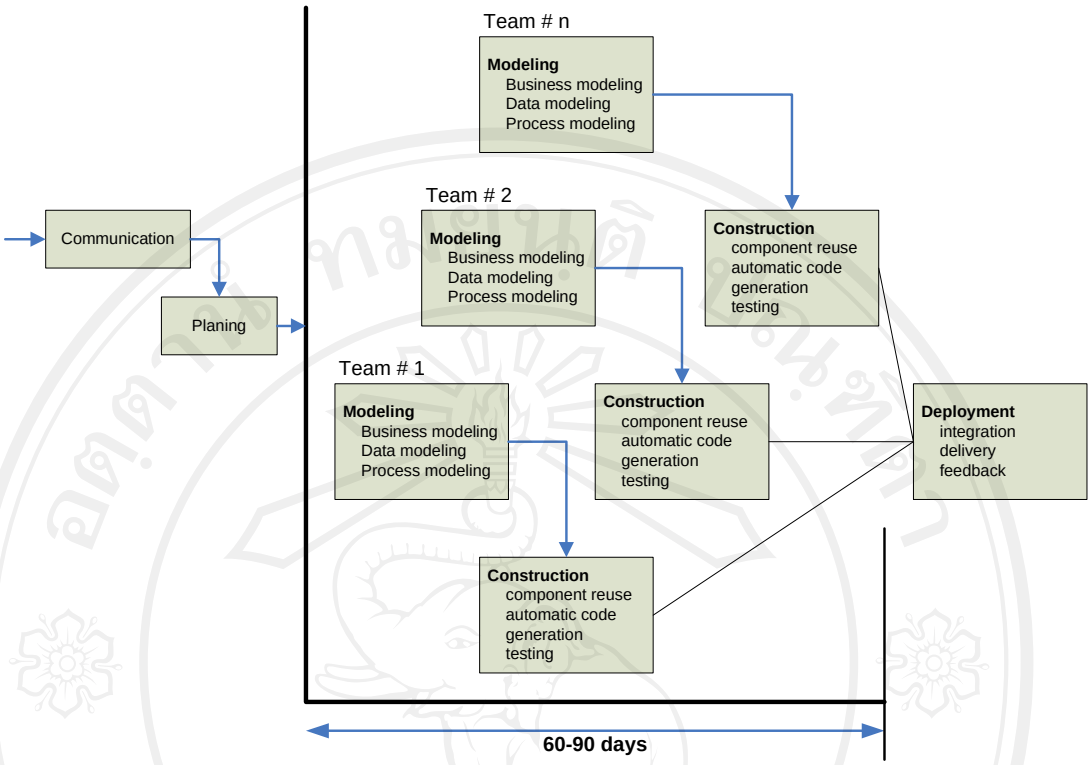
รุ่นแรกของผลิตภัณฑ์ในการทำงานตามแบบจำลองค่อยเพิ่มขึ้นนี้มักจะเป็น ผลิตภัณฑ์แก่น คือตอบสนองเฉพาะความต้องการพื้นฐาน ผลของการใช้งานผลิตภัณฑ์แก่นโดยลูกค้า จะนำมาปรับปรุงรุ่นถัดไปที่ส่งมอบ กระบวนการนี้จะทำซ้ำ ๆ ไปจนกว่าจะส่งมอบผลิตภัณฑ์ที่สมบูรณ์



รูปที่ 2.1 แบบจำลองค่อยเพิ่มขึ้น

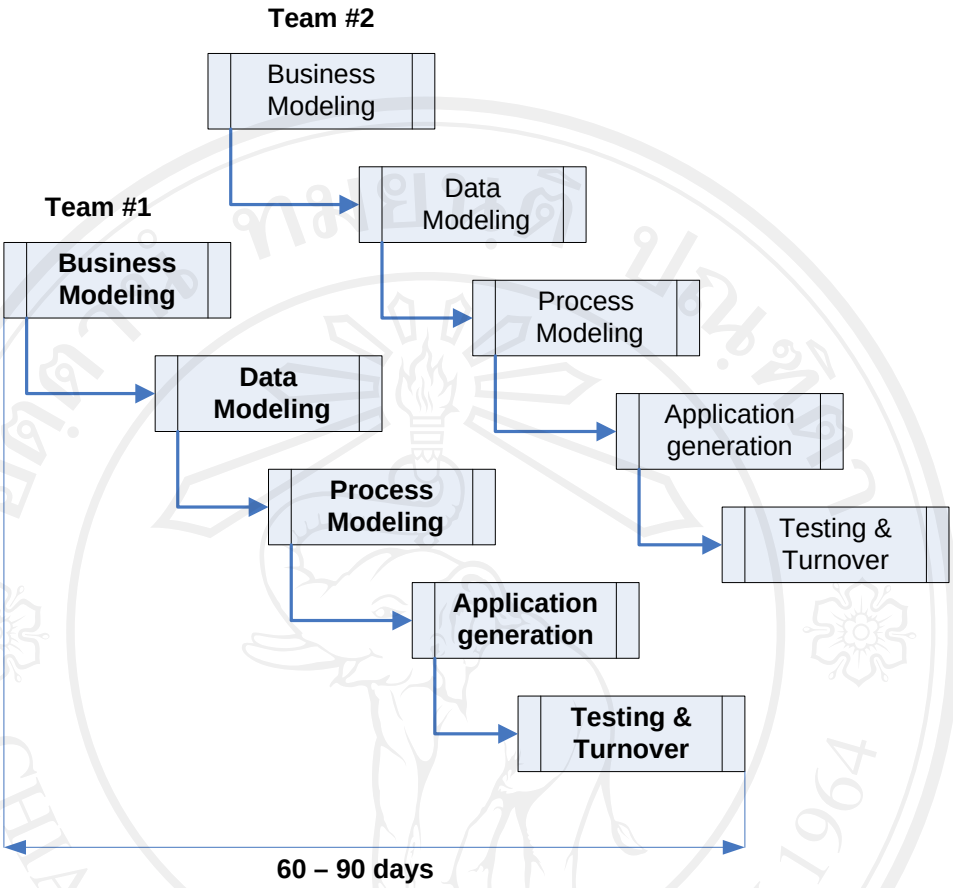
2.2.2. วิธีการพัฒนาซอฟต์แวร์แบบเร็ว (Rapid Application Development-RAD)

การพัฒนาซอฟต์แวร์อย่างรวดเร็ว (Rapid Application Development-RAD) เป็นแบบจำลองกระบวนการซอฟต์แวร์แบบค่อยเพิ่มขึ้นที่เน้น วงจรพัฒนาสั้น ๆ แบบจำลองอาร์เอดีเป็นการดัดแปลงแบบจำลองน้ำตกให้มีความเร็วสูง โดยผลสำเร็จของการพัฒนาอย่างรวดเร็ว เกิดจากการใช้วิธีการประกอบคอมโพเน้นท์ย่อยเข้าด้วยกัน ถ้าเข้าใจความต้องการของระบบได้ดีและขอบเขตโครงการค่อนข้างจำกัด กระบวนการอาร์เอดีจะทำให้ทีมงานพัฒนา สร้างระบบที่ทำงานได้ภายในเวลาอันสั้นมาก คือ 60-90 วัน



รูปที่ 2.2 แบบจำลองการพัฒนาซอฟต์แวร์แบบเร็ว

เงื่อนไขด้านเวลาของโครงการที่จะใช้อาร์เอดีเป็นตัวกำหนดว่า ขอบเขตงานต้องสามารถขยายส่วนได้ในที่นี้คือ แอพพลิเคชั่นต้องสามารถแบ่งเป็นโมดูลที่สามารถทำแต่ละโมดูลให้เสร็จได้ภายในเวลา 3 เดือน ถ้าแอพพลิเคชั่นมีลักษณะนี้ก็สามารถใช้แบบจำลองอาร์เอดีได้ กระบวนการอาร์เอดีมีกิจกรรมกรอบงานดังที่กล่าวมาแล้วคือ งานการสื่อสาร เพื่อให้เข้าใจปัญหาทางธุรกิจ และลักษณะข้อมูลที่ซอฟต์แวร์ต้องรองรับ งานการวางแผนสำคัญมาก เนื่องจากต้องแบ่งทีมงานหลายทีม ให้ทำงานคู่ขนานกันไปในโมดูลต่าง ๆ การสร้างแบบจำลองรวมเอาสามกิจกรรมหลัก คือ การสร้างแบบจำลองทางธุรกิจ การสร้างแบบจำลองข้อมูลและการสร้างแบบจำลองกระบวนการ อันเป็นฐานไปสู่การแทนงานออกแบบ งานก่อสร้างซอฟต์แวร์ขั้นตอนการก่อสร้างเน้นการใช้คอมโพเน้นท์ที่มีอยู่แล้วมาประกอบเข้าด้วยกัน และประยุกต์ใช้โปรแกรมสร้างโค้ดอัตโนมัติ ส่วนขั้นตอนการใช้งานจะรวบรวมข้อมูลมาเพื่อทำการพัฒนาในรอบถัดไปถ้าจำเป็น



รูปที่ 2.3 แสดงส่วนการพัฒนาซอฟต์แวร์ตามรูปแบบการพัฒนาซอฟต์แวร์แบบเร็ว

การสร้างโมเดลทางธุรกิจ (Business modeling)

เป็นการนำข้อมูลสารสนเทศความต้องการทางธุรกิจนำมาสร้างโมเดลทางธุรกิจ ในทางที่ตอบคำถามว่า

- ข้อมูลสารสนเทศใดที่เป็นตัวขับเคลื่อนกิจกรรมในธุรกิจ ?
- ข้อมูลสารสนเทศใดที่ต้องสร้างขึ้น ?
- ใครเป็นคนสร้าง ?
- ข้อมูลสารสนเทศเหล่านั้นจะส่งต่อไปที่ไหน ?
- ใครเป็นคนรับข้อมูลสารสนเทศดังกล่าวไปปฏิบัติ ?

การสร้างโมเดลข้อมูล (Data modeling)

ข้อมูลสารสนเทศที่ได้จากขั้นตอนของบิสเนสโมเดล (Business modeling) เป็นชุดของข้อมูลเชิงวัตถุที่จำเป็นต่อโมเดลทางธุรกิจ ซึ่งลักษณะของข้อมูลเหล่านี้จะเรียกว่าแอตทริบิว จะถูกนำมาสร้างไว้รวมถึงการกำหนดความสัมพันธ์ระหว่างชุดข้อมูลที่ทำการสร้างขึ้น

การสร้างโมเดลกระบวนการ (Process modeling)

ข้อมูลเชิงวัตถุที่ประกาศไว้จะถูกส่งต่อให้กระบวนการสร้างกิจกรรมสำหรับแต่ละชุดข้อมูล ไม่ว่าจะเป็นการตัดสินใจ การสร้าง การแก้ไข การลบหรือการยกเลิกข้อมูล

การสร้างโค้ดคำสั่ง (Application generation)

อาร์เอดีนำเสนอให้ใช้เทคนิคของภาษายุคที่สี่ (Fourth generation techniques) และการใช้เครื่องมือ เช่น VB, VC++, Delphi มากกว่าการใช้ภาษายุคที่สาม (Third generation programming languages) การพัฒนาซอฟต์แวร์แบบเร็วจะใช้โปรแกรมรวมถึงคอมโปเน้นท์ต่าง ๆ ที่เคยถูกสร้างขึ้นแล้ว นำกลับมาใช้ใหม่ทำให้ตอบสนองแนวคิดการโปรแกรมเชิงวัตถุ นอกจากนี้การใช้เครื่องมือในการสร้างโปรแกรมอัตโนมัติ (Automated tools) ก็เป็นอีกทางเลือกหนึ่งสำหรับการสร้างโปรแกรม

การทดสอบระบบและการย้อนกลับ (Testing and turnover)

สำหรับการพัฒนาซอฟต์แวร์แบบเร็ว สิ่งต่าง ๆ ที่นำมาใช้มักจะเป็นสิ่งที่เคยใช้งานได้แล้วและผ่านการทดสอบมาแล้ว ดังนั้นส่วนต่าง ๆ ที่นำมาประกอบกันขึ้นเป็นโปรแกรมจึงมีความพร้อมที่จะนำไปใช้ได้เลยซึ่งถือว่าสำเร็จไปพร้อมกันในขั้นตอนของการพัฒนาโปรแกรมแล้ว

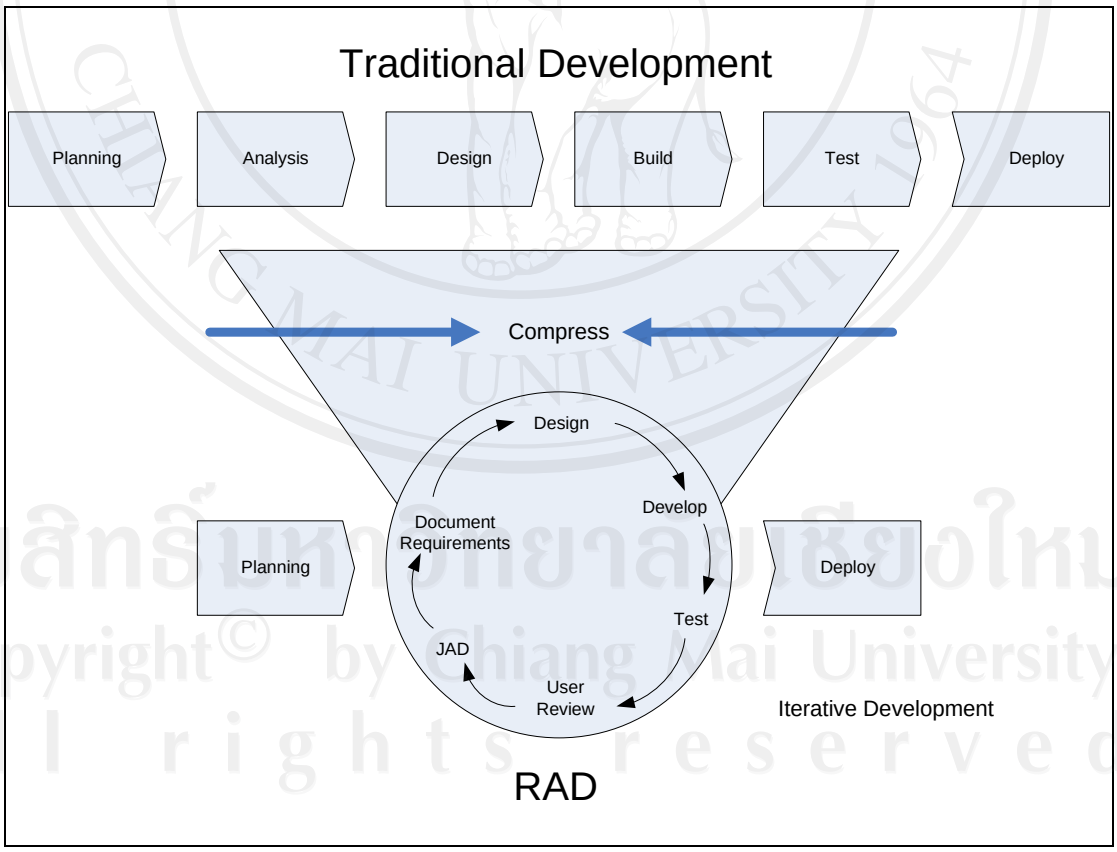
วอล์เตอร์ แมนเนอร์ (Walter Maner) [2] กล่าวถึงหลักการที่ให้พิจารณาในการพัฒนาซอฟต์แวร์ตามแบบการพัฒนาซอฟต์แวร์แบบเร็วไว้ ดังนี้

1. ในสถานการที่แน่นอน 80% ของงานที่ทำจะใช้แค่ 20% ของเวลาทั้งหมดที่ต้องการในการพัฒนา
2. การตอบสนองความต้องการหลัก (Business Requirement) ของระบบได้ถือว่าทำให้เกิดความพึงพอใจสำหรับลูกค้าได้อย่างเต็ม ที่ถึงแม้ว่าความต้องการในระดับปฏิบัติการ (Operational Requirement) จะยังไม่ได้รับการตอบสนองก็ตาม
3. ในด้านการยอมรับระบบที่พัฒนาขึ้น จะประเมินได้จากความเห็นของลูกค้าซึ่งมีทั้งเห็นด้วยและไม่เห็นด้วยแต่อย่างน้อยที่สุดควรคิดเห็นเหล่านั้นสามารถนำไปใช้ประโยชน์ได้มากกว่าความต้องการทั้งหมด

กิตติ ภักดีวัฒนะกุล [3] กล่าวถึงวิธีการพัฒนาซอฟต์แวร์แบบเร็วไว้ว่า วิธีการพัฒนาซอฟต์แวร์แบบเร็ว เป็นวิธีการพัฒนาระบบ (Methodology) วิธีการหนึ่ง ที่รวบรวมเทคนิค (Techniques) เครื่องมือ (Tools) และ เทคโนโลยี เพื่อผสมผสานและประยุกต์ใช้ในการสนับสนุนการพัฒนาระบบให้สามารถลุล่วงโดยใช้เวลาน้อยที่สุด ทั้งนี้ขึ้นอยู่กับความพร้อมของ

องค์กรในขณะนั้น ไม่ว่าจะเป็นเรื่องค่าใช้จ่าย บุคลากร รวมทั้งความต้องการที่แน่นอนของผู้ใช้ระบบ

จากแนวความคิดของวิธีการพัฒนาซอฟต์แวร์แบบเร็ว ทำให้มีการผสมผสานและประยุกต์ใช้เทคนิค เครื่องมือ และเทคโนโลยีเข้าด้วยกันเพื่อพัฒนาระบบ โดยอาจจะมีการแบ่งแยกขั้นตอนในวงจรการพัฒนาระบบให้น้อยลง เลือกใช้เทคนิคการเก็บรวบรวมข้อมูลด้วยวิธีการพัฒนาระบบร่วมกัน (Join Application Development-JAD) และเลือกใช้ตัวต้นแบบ (Prototype) ในการออกแบบ เป็นต้น ทำให้การพัฒนาระบบสามารถดำเนินการได้อย่างรวดเร็ว ดังนั้นวิธีการพัฒนาซอฟต์แวร์แบบเร็ว จึงเหมาะกับองค์กรที่มีความพร้อมในการพัฒนา เช่น ความพร้อมทางด้านการเงิน บุคลากร ประกอบกับผู้ใช้ความต้องการแน่นอนไม่เปลี่ยนแปลง และต้องการระบบใหม่โดยใช้เวลาพัฒนาไม่นาน ลักษณะของวิธีการพัฒนาซอฟต์แวร์แบบเร็วอีกรูปแบบหนึ่งเมื่อเปรียบเทียบกับการพัฒนาซอฟต์แวร์แบบเดิมก็คือการรวบกิจกรรมแต่ละขั้นตอนให้กระชับขึ้นนั่นเองดังแสดงในรูปที่ 2.3



รูปที่ 2.3 แสดงวิธีการพัฒนาซอฟต์แวร์แบบเร็วเปรียบเทียบกับการพัฒนาซอฟต์แวร์แบบเดิม

วิธีการพัฒนาระบบแบบรวดเร็วสามารถลดขั้นตอนของวงจรพัฒนาระบบ (SDLC) จาก 7 ขั้นตอน เหลือเพียง 4 ขั้นตอน ดังแสดงในตารางที่ 2.2

SDLC	RAD
1. การวิเคราะห์ปัญหา 2. การศึกษาความเป็นไปได้ 3. การวิเคราะห์ความต้องการของระบบ 4. การวิเคราะห์เพื่อตัดสินใจ 5. การออกแบบ 6. การสร้างระบบ 7. การใช้ระบบ	1. การวางแผนกำหนดความต้องการ (requirement planning) 2. การออกแบบโดยผู้ใช้ (user design) 3. การสร้างระบบ (construction) 4. การเปลี่ยนระบบ (cutover)

ตารางที่ 2.2 แสดงการเปรียบเทียบขั้นตอนการพัฒนาระหว่าง SDLC กับ RAD

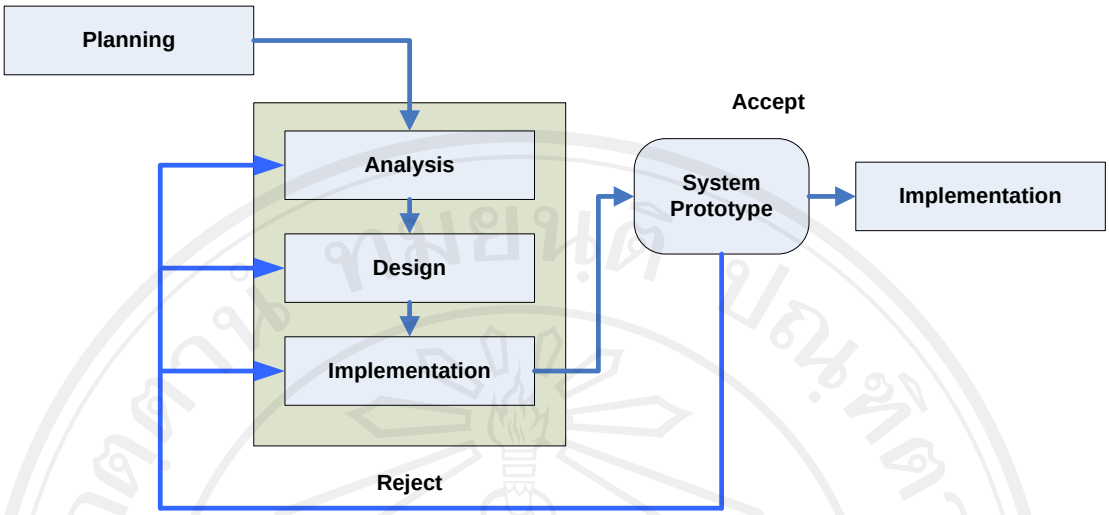
2.3 การพัฒนาระบบร่วมกัน (Joint Application Development – JAD)

การพัฒนาระบบร่วมกันหรือ เจเอดี เป็นอีกวิธีหนึ่งของการพัฒนาระบบอย่างรวดเร็ว ระหว่างผู้ใช้ระบบและผู้พัฒนา โดยผู้ใช้และผู้พัฒนาระบบมีส่วนร่วมในการกำหนดความต้องการของระบบร่วมกัน (Joint Requirement Planning – JRP) และออกแบบระบบร่วมกัน (Joint Application Design – JAD) เพื่อลดเวลา ค่าใช้จ่าย และขั้นตอนการรวบรวมข้อมูล เป็นวิธีที่บริษัทไอบีเอ็มพัฒนาขึ้นในช่วงปลายทศวรรษ 1970 และเป็นที่นิยมใช้เป็นเทคนิคในการรวบรวมข้อมูลขององค์การด้านธุรกิจ ในการรวบรวมข้อมูลร่วมกันผู้ที่เกี่ยวข้องจะต้องมีการวางแผนดำเนินการที่ดีเพราะเป็นกระบวนการที่ต้องใช้เวลา และค่าใช้จ่าย แต่ผลที่ได้ก็จะคุ้มค่า

2.4 การจัดทำต้นแบบ (Prototyping)

อภิรักษ์ จิรายุสกุล [4] กล่าวถึงการทำต้นแบบไว้ว่า กรรมวิธีการสร้างต้นแบบของระบบ (Prototype) มีวัตถุประสงค์ เพื่อให้ผู้ใช้สามารถยืนยันความต้องการบางส่วน และให้ผู้ใช้ได้เห็นถึงรายละเอียดบางอย่างของระบบที่อาจขาดตกบกพร่อง รวมทั้งมีส่วนช่วยให้ผู้ใช้ยอมรับระบบที่พัฒนาใหม่ได้ง่ายขึ้น

ขั้นตอนการจัดทำต้นแบบ สามารถแสดงได้ดังรูปที่ 2.4



รูปที่ 2.4 แสดงขั้นตอนของการทำต้นแบบ

ข้อดี ของการใช้ตัวต้นแบบ คือ เหมาะสมและมีประโยชน์มากที่สุดกับการที่ไม่สามารถกำหนดความต้องการของระบบที่ชัดเจนได้ การที่ผู้ใช้มีส่วนร่วมทำให้ได้ระบบที่ตรงกับความต้องการของผู้ใช้ได้มากที่สุด

ข้อด้อย คือ การจัดทำระบบอย่างรวดเร็ว อาจจะข้ามขั้นตอนที่สำคัญของการออกแบบระบบ โดยไม่ได้คำนึงถึงการทำงานกับข้อมูลจำนวนมาก เมื่อนำไปใช้งานจริงอาจประสบปัญหาประสิทธิภาพการทำงานของระบบได้

2.5 การวิเคราะห์และออกแบบเชิงวัตถุด้วยยูเอ็มแอล

โอภาส เอี่ยมสิริวงศ์ [5] กล่าวถึงเรื่องการวิเคราะห์และออกแบบเชิงวัตถุไว้ว่า หลักการพัฒนาระบบเชิงวัตถุ จะประกอบด้วยกลุ่มของวัตถุ (Class of Object) ต่าง ๆ ที่ทำงานร่วมกัน โดยแบ่งบทบาทหน้าที่ความรับผิดชอบ ซึ่งใช้หลักการจัดแบ่งประเภทของวัตถุในลักษณะทางนามธรรม (Abstract) ออกเป็นกลุ่ม ๆ ที่เรียกว่าคลาส (Class) แต่ละคลาสจะมีสถานะ (States) รวมทั้งพฤติกรรม (Behavior) ตามบทบาทของตน โดยมีข้อมูลรายละเอียดหรือคุณสมบัติ (Characteristic) ที่เก็บซ่อน (Encapsulate) ในคลาสของตนโดยไม่มีการปะปนกับคลาสอื่น ๆ แต่ในการติดต่อสื่อสารหรือการร้องของใช้บริการ สามารถติดต่อสื่อสารกันได้ด้วยเมจเสจ (Message)

แนวคิดเชิงโครงสร้างนั้น เป็นโครงสร้างที่โปรแกรมกับข้อมูลนั้นแยกออกจากกัน แต่แนวคิดเชิงวัตถุนั้น จะมองเป็นออบเจ็กต์หนึ่งที่เป็นแหล่งรวมของข้อมูล (Data) วิธีการ (Method) โดยมีคลาสเป็นตัวกำหนดคุณสมบัติของออบเจ็กต์นั้น ซึ่งคุณสมบัติยังสามารถทำการสืบทอด

(Inheritance) ในลักษณะ Subclass ต่าง ๆ ดังนั้นหากมีคลาสที่เป็นต้นแบบที่ดีอยู่แล้ว ผู้พัฒนาสามารถนำคุณสมบัติของคลาสต้นแบบนั้นมาใช้งานได้ทันที ซึ่งเป็นการนำกลับมาใช้ใหม่ (Reusable) ทำให้ช่วยลดเวลาในการพัฒนาและลดค่าใช้จ่าย ประกอบกับความมั่นใจในคลาสต้นแบบที่ใช้งานมานาน จะบ่งบอกถึงความถูกต้องซึ่งก่อให้เกิดข้อผิดพลาดได้น้อย

จึงสามารถกล่าวโดยสรุปได้ว่า การวิเคราะห์และออกแบบเชิงวัตถุนี้ เป็นแนวคิดที่พยายามจัดระบบกระบวนการพัฒนาระบบงานให้มีระเบียบ และสามารถนำโปรแกรมที่เคยเขียนมาก่อนให้สามารถกลับมาใช้งานใหม่ ซึ่งถ้าเปรียบเทียบกับการเขียนโปรแกรมเชิงโครงสร้าง ถึงแม้ระบบงานที่มีความใกล้เคียงกัน แต่โมดูลที่จะนำมาใช้งานจะต้องมีการปรับเปลี่ยนมากมายแทบจะเริ่มต้นเขียนขึ้นใหม่ทั้งหมด เป็นเพราะแนวทางการพัฒนาซอฟต์แวร์เชิงโครงสร้าง (Structured) นั้นมีลักษณะเป็นนามธรรมซึ่งเกิดจากการจิตนาการ ดังนั้นระบบงานที่พัฒนาตามแนวคิดเชิงโครงสร้างในแต่ละระบบ จะเกิดจากการจิตนาการของแต่ละบุคคล จินตนาการของแต่ละบุคคลต่างก็มีแนวคิดที่ต่างกัน ดังนั้นจึงเห็นซอฟต์แวร์จำนวนมากมายทั้งที่เป็นะบบเดียวกันและใช่ว่าจะสามารถนำกลับใช้ใหม่ได้ทั้งหมด

2.6 ยูเอ็มแอล (UML)

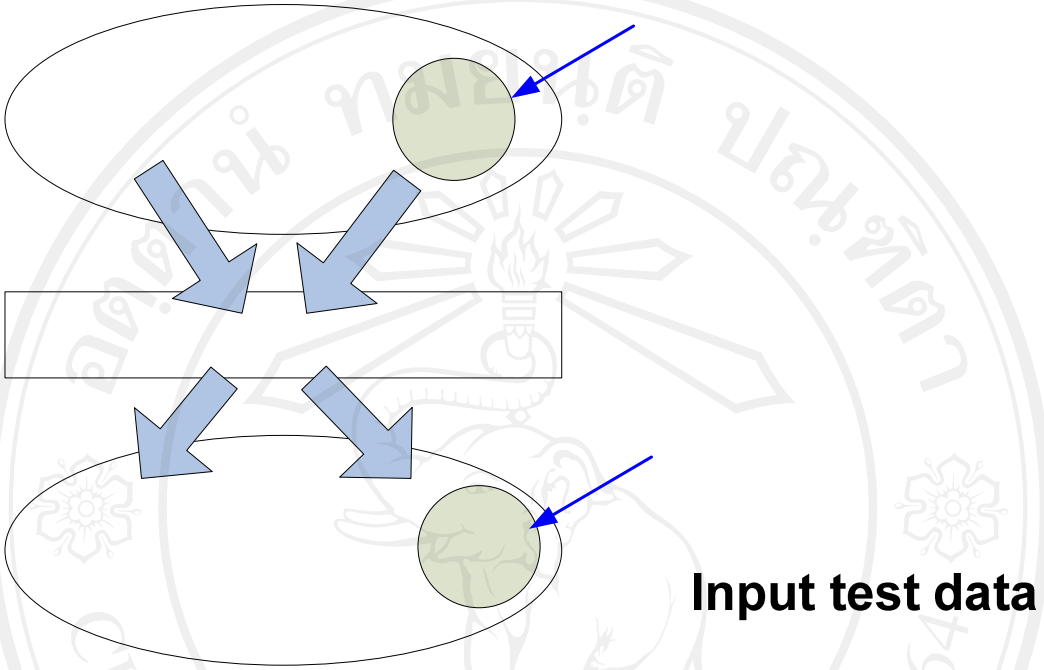
นิยาม UML (Unified Modeling Language) ที่ Grady Booch, Ivar Jacobson และ Jim Rumbaugh ทั้งสามร่วมมือกันพัฒนาขึ้นมาได้นิยามไว้ว่า ยูเอ็มแอล เป็นสัญญลักษณ์ (Notation) ที่ใช้อธิบาย แสดงรายละเอียดจำองการสร้าง และจัดการกับเอกสารต่าง ๆ ในระบบ เพื่อให้การออกแบบซอฟตแวร์สามารถทำได้โดยง่าย และปรับปรุงวิธีการทำงานให้ดีขึ้น

แต่เดิมนั้นทั้งสามต่างมีโมเดลในการพัฒนาระบบเชิงวัตถุ ซึ่งต่อมาบริษัทเรชั่นแนล (Rational) ได้ร่วมมือให้บุคคลทั้งสามทำการพัฒนาโมเดลร่วมกัน จึงเป็นที่มาของยูเอ็มแอล ซึ่งเป็นโมเดลที่สื่อสารด้วยภาพ โดยแต่ละโมเดลจะแสดงมุมมองที่มีต่อระบบแตกต่างกัน

ดังนั้นยูเอ็มแอล จึงเป็นระเบียบวิธี (Methodology) หนึ่งเช่นเดียวกับการวิเคราะห์ระบบเชิงโครงสร้างที่ใช้ Data Flow Diagram (DFD) และ Entity Relationship Diagram (ERD)

2.7 การทดสอบแบบกล่องดำ (Black-box Testing)

กิตติภณ เพ็ชรสว่าง [6] กล่าวถึงการทดสอบแบบกล่องดำ (Black-box Testing) ไว้ว่าเป็นการตรวจสอบที่วิธีที่ใช้ในการตรวจสอบถูกสร้างขึ้นจากเกณฑ์การทำงานของระบบ แนวคิดของการตรวจสอบวิธีนี้มองว่า ระบบคือ “กล่องดำ” ที่ไม่สามารถมองเห็นสิ่งที่อยู่ภายในได้ หากต้องการตัดสินว่าการทำงานถูกต้องหรือไม่ ทำได้โดยศึกษาจากข้อมูลนำเข้าที่ป้อนเข้าสู่ระบบและผลลัพธ์ที่สัมพันธ์กับข้อมูลนำเข้านั้นที่ได้จากระบบ แสดงดังรูปที่ 2.5



รูปที่ 2.5 แสดงแนวทางของ Black-box testing

เมื่อป้อนข้อมูลนำเข้าที่ก่อให้เกิดการทำงานที่บกพร่อง ระบบจะนำข้อมูลเหล่านี้ไปประมวลผลและจะได้ผลลัพธ์ที่ไม่ถูกต้องออกมา ในที่สุดจะตรวจพบข้อบกพร่องในระบบได้ ปัญหาสำคัญสำหรับการตรวจสอบด้วยวิธีนีคือ การเลือกข้อมูลนำเข้าที่จะทำให้ระบบทำงานผิดพลาด (Ie) หากทีมตรวจสอบเลือกข้อมูลนำเข้าที่ไม่อยู่ในกลุ่มนี้ ระบบจะไม่แสดงข้อผิดพลาดให้เห็น ทำให้ทีมตรวจสอบเข้าใจว่าระบบทำงานได้สมบูรณ์แล้วก็เป็นได้

Equivalence partitioning

ข้อมูลนำเข้าที่จะป้อนสู่ระบบมีอยู่หลายประเภท ซึ่งมีคุณสมบัติเฉพาะตัวแตกต่างกันออกไป เช่น จำนวนเต็มบวก, จำนวนเต็มลบ, เลขทศนิยม, ตัวอักษณล้วน, ตัวอักษรที่มีช่องว่างปะปน เป็นต้น โดยปกติแล้วโปรแกรมจะตอบสนองต่อข้อมูลนำเข้าประเภทต่าง ๆ ไม่เหมือนกัน แต่จะตอบสนองแบบเดียวกันสำหรับข้อมูลประเภทเดียวกัน เช่น การป้อน 100 กับการป้อน 100.75 เข้าสู่ระบบ วิธีการตอบสนองของระบบต่อข้อมูลเหล่านี้จะไม่เหมือนกัน เพราะเป็นข้อมูลคนละประเภทกัน แต่การป้อน 100 กับการป้อน 200 เข้าสู่ระบบ วิธีการตอบสนองของระบบต่อข้อมูลเหล่านี้จะเหมือนกัน เพราะเป็นข้อมูลประเภทเดียวกัน ด้วยเหตุนี้หากทีมตรวจสอบสามารถ

แบ่งขอบเขตของค่าที่จะป้อนเข้าสู่ระบบออกเป็นกลุ่ม ๆ โดยมั่นใจว่าระบบจะตอบสนองต่อค่าแต่ละค่าในกลุ่มเดียวกันเหมือนกันได้ การทดสอบแบบกล่องดำ (Black-box Testing) จะสามารถใช้การเลือกค่าเพียงบางค่าของแต่ละกลุ่มมาเป็นข้อมูลที่จะป้อนเข้าสู่ระบบ แทนที่จะต้องใช้ทุกค่าได้

เมื่อทีมตรวจสอบสามารถจำแนกข้อมูลออกได้เป็นกลุ่ม ๆ ควรเลือกค่าซึ่งเป็นตัวแทนที่ดีของแต่ละกลุ่มเพื่อใช้เป็นข้อมูลป้อนเข้าสู่ระบบ แนวทางที่ดีแนวทางหนึ่งในการเลือกค่าจากแต่ละกลุ่มคือ เลือกค่าซึ่งเป็นขอบเขตของแต่ละกลุ่มกับค่ากลางของกลุ่มนั้น ๆ เนื่องจากผู้ออกแบบและโปรแกรมเมอร์มักมีแนวโน้มจะใส่ใจกับค่าปกติทั่วไปมากกว่า จึงเลือกค่ากลางของกลุ่มเพื่อตรวจสอบ ในขณะที่ค่าไม่ค่อยปกตินัก เช่น ค่าต่ำสุด หรือ ค่าสูงสุด มักถูกโปรแกรมเมอร์มองข้ามทำให้ระบบทำงานผิดพลาดบ่อย ๆ เมื่อต้องประมวลผลกับข้อมูลเหล่านี้

การตรวจสอบซอฟต์แวร์ตามหลักการทดสอบแบบกล่องดำ (Black-box Testing) นั้น ผู้ตรวจสอบไม่จำเป็นจะต้องมีความรู้ ความเข้าใจในโครงสร้างของการเขียนโปรแกรมภายในซอฟต์แวร์ เพียงแต่เข้าใจถึงขอบเขตการทำงานของระบบ ลักษณะของข้อมูลเข้าและผลลัพธ์ที่คาดว่าจะได้รับจากระบบ ประกอบกับการเลือกข้อมูลนำเข้าที่จะใช้ตรวจสอบระบบอย่างเหมาะสมตามหลักการ Equivalence partitioning ผู้ตรวจสอบก็สามารถออกแบบการตรวจสอบซอฟต์แวร์อย่างมีประสิทธิภาพได้

2.8 ทฤษฎีเกี่ยวกับความพึงพอใจ

2.8.1 ความหมายของความพึงพอใจ

วิมลสิทธิ์ หรยางกูร [7] ให้ความหมายว่า ความพึงพอใจ เป็นการให้ค่าความรู้สึกของคนเราที่สัมพันธ์กับโลกทัศน์ ที่เกี่ยวกับความหมายของสภาพแวดล้อม ค่าความรู้สึกของบุคคลที่มีต่อสภาพแวดล้อมจะแตกต่างกัน เช่น ความรู้สึก ดี - เลว พอใจ – ไม่พอใจ สนใจ – ไม่สนใจ เป็นต้น

กิติมา ปรีดีดิลก [8] สรุปไว้ว่า ความพึงพอใจ หมายถึง ความรู้สึกชอบ หรือพอใจที่มีต่อองค์ประกอบ และสิ่งจูงใจในด้านต่าง ๆ ของงาน และผู้ปฏิบัติงานได้รับการตอบสนองตามความต้องการของเขาได้

จากความหมายของความพึงพอใจที่กล่าวมา สรุปได้ว่า ความพึงพอใจหมายถึงความรู้สึกที่ดีของบุคคลซึ่งมักเกิดจากการได้รับการตอบสนองตามที่ตนต้องการจะเกิดความรู้สึกที่ดีในสิ่งนั้น และในทางตรงกันข้ามหากความต้องการไม่ได้รับการตอบสนอง จะทำให้เกิดความไม่พึงพอใจ ซึ่งการวัดความพึงพอใจสามารถกระทำได้หลายวิธี ดังต่อไปนี้

1) การใช้แบบสอบถาม ซึ่งเป็นวิธีการที่นิยมใช้กันแพร่หลายวิธีหนึ่ง โดยการร้องขอหรือขอความร่วมมือ จากกลุ่มบุคคลที่ต้องการวัด แสดงความคิดเห็นลงในแบบฟอร์มที่กำหนดคำตอบไว้ให้เลือกตอบหรือเป็นคำตอบอิสระ โดยคำถามที่ถามอาจจะถามถึงความพึงพอใจในด้านต่าง ๆ ที่มีต่อซอฟต์แวร์ที่ใช้

2) การสัมภาษณ์ เป็นอีกวิธีที่จะทำให้ทราบถึงระดับความพึงพอใจของเจ้าของงาน ซึ่งเป็นวิธีการที่ต้องอาศัยเทคนิคและความชำนาญพิเศษของผู้สัมภาษณ์ที่จะจูงใจให้ผู้ถูกสัมภาษณ์ตอบคำถามให้ตรงกับข้อเท็จจริง การวัดความพึงพอใจโดยวิธีสัมภาษณ์นับว่าเป็นวิธีที่ประหยัดและมีประสิทธิภาพอีกวิธีหนึ่ง

3) การสังเกต โดยใช้วิธีการสังเกตพฤติกรรมของเจ้าของงาน โดยพิจารณาจากกิริยาท่าทาง การพูด สีหน้า และความถี่ของการใช้งานระบบ การวัดความพึงพอใจโดยวิธีนี้ ผู้วัดจะต้องกระทำอย่างจริงจังและมีแบบแผนที่แน่นอน จึงจะสามารถประเมินถึงระดับความพึงพอใจได้อย่างถูกต้อง

2.8.2 เจตคติและการวัดเจตคติ

บุญธรรม กิจปรีดาบริสุทธิ์ [9] กล่าวถึงเจตคติไว้ว่า เป็นกริยาท่าทีรวม ๆ ของบุคคลที่เกิดจากความพร้อมหรือความโน้มเอียงของจิตใจ ซึ่งแสดงออกต่อสิ่งเร้าหนึ่ง ๆ เช่น วัตถุ สิ่งของและสถานการณ์ต่าง ๆ ในสังคม โดยแสดงออกมาในทางสนับสนุน ซึ่งมีความรู้สึกเห็นดี เห็นชอบต่อสิ่งเร้านั้น ๆ หรือในทางต่อต้าน ซึ่งมีความรู้สึกที่ไม่เห็นดี ไม่เห็นชอบต่อสิ่งเร้านั้น ฉะนั้น การวัดเจตคติจึงต้องพิจารณากริยาท่าทีหรือการตอบสนองต่อสิ่งเร้าในหลายด้านหลายประการรวมกัน

2.8.2.1 หลักการวัดเจตคติ

การวัดเจตคติมีหลักการเบื้องต้นที่ต้องทำความเข้าใจ 3 ประการ ดังนี้

1) เนื้อหา (Content) การวัดเจตคติต้องมีสิ่งเร้าไปกระตุ้นให้แสดงกิริยาท่าทีออกมา สิ่งเร้าโดยทั่วไปได้แก่เนื้อหาที่ต้องการวัด เช่น ต้องการวัดเจตคติต่อการตัดสินใจเกี่ยวกับชีวิตครอบครัวของบุคคล เนื้อหาที่เป็นสิ่งเร้าในที่นี้คือ สถานการณ์การตัดสินใจเกี่ยวกับชีวิตครอบครัว และความสัมพันธ์ภายในครอบครัว เป็นต้น
2) ทิศทาง (Direction) การวัดเจตคติโดยทั่วไปกำหนดให้เจตคติมีทิศทางเป็นเส้นตรงและต่อเนื่องกันในลักษณะเป็นซ้าย-ขวา หรือ บวกกับลบ กล่าวคือเริ่มจากเห็นด้วยอย่างยิ่งและความเห็นด้วยลงเรื่อย ๆ จนถึงมีความรู้สึกเฉย ๆ และลดต่อไปเป็นไม่เห็นด้วยจนไม่เห็นด้วยอย่างยิ่ง ลักษณะของการเห็นด้วยและไม่เห็นด้วยอยู่เป็นเส้นตรงเดียวกันและต่อเนื่องกัน

3) ความเข้มข้น (Intensity) กิริยาท่าทีหรือความรู้สึก ที่แสดงออกต่อสิ่งเร้านั้นมีปริมาณมากน้อยแตกต่างกัน ถ้ามีความเข้มข้นสูงไม่ว่าจะเป็นไปในทิศทางใดก็ตามจะมีความรู้สึกหรือกิริยาท่าที่รุนแรงกว่า

2.8.2.2 มาตรวัดเจตคติ (Attitude Scale) เครื่องมือที่ใช้วัดเจตคติ เรียกว่ามาตราส่วนประมาณค่า (Rating Scale) เครื่องมือวัดเจตคติที่นิยมใช้และรู้จักกันแพร่หลายมี 4 ชนิด ได้แก่

1) มาตรวัดแบบเธอร์สโตน (Thurstone Type Scale) ใช้วิธีพยายามสร้างข้อความเจตคติตามโครงสร้างของเจตคติที่กำหนดให้มากที่สุดเท่าที่จะมากได้ แล้วนำข้อความที่สร้างทั้งหมดนั้นไปให้คณะบุคคล (50 -100 คน) ตัดสินว่าข้อความนั้นควรอยู่ในมาตราใดระหว่าง A ถึง K หรือ 1 ถึง 11 หมายความว่าเห็นด้วยมากที่สุดและลดน้อยลงตามลำดับของความรู้สึก

2) มาตรวัดแบบลิเคิต (Likert Scale) การสร้างมาตรวัดแบบนี้มีข้อตกลงเบื้องต้นว่าลักษณะการกระจายของเจตคติมีการแจกแจงเป็นปกติ (Normal Curve) จากข้อตกลงนี้ลิเคิตได้ใช้เป็นหลักหาค่า s คะแนนของมาตรวัดโดยนำไปทดลองกับกลุ่มที่ต้องการวัด ไม่ต้องให้คณะบุคคลตัดสินเหมือนของแบบเธอร์สโตน และจากการศึกษาต่อไปลิเคิดพบอีกว่าค่า s ที่หามาจากการใช้ Normal Deviate กับการกำหนดแบบให้ค่าคะแนนมาตรวัดโดยแต่ละมาตรวัดห่างกันเท่ากันเป็น 0, 1, 2, 3, 4 สำหรับข้อความวัดเจตคติเป็นลบ (Negative or unfavorable) และเป็น 4, 3, 2, 1, 0 สำหรับข้อความวัตเจตคติเป็นบวก (Positive of favorable) นั้นได้ผลไม่แตกต่างกัน ซึ่งพบว่ามีค่าสหสัมพันธ์กันสูงถึง 0.99

มาตรวัดเจตคติแบบลิเคิตสามารถใช้วัดเจตคติได้อย่างกว้างขวางกว่าแบบอื่น ๆ และสามารถวัดเจตคติได้เกือบทุกเรื่อง ยิ่งกว่านั้นมักจะมีค่าความเที่ยงสูงกว่าแบบอื่นอีกด้วย

3) มาตรวัดแบบกัตต์แมน (Guttman Scale) เป็นวิธีการประเมินชุดของข้อความวัดเจตคติที่สร้างขึ้น ซึ่งกัตต์แมนเรียกวิธีของเขาว่าเป็นการวิเคราะห์มาตราส่วน (Scalogram Analysis) กล่าวคือ ในชุดของข้อความวัดเจตคติหนึ่ง ๆ นั้น ถ้าหากผู้ตอบเห็นด้วยกับข้อความ 2 แล้ว เขาจะต้องเห็นด้วยกับข้อความ 1 ก่อน และถ้าเห็นด้วยกับข้อความ 3 ก็ต้องเห็นด้วยกับข้อความ 2 และข้อความ 1 มาก่อน ในลักษณะนี้เรื่อย ๆ ไป ฉะนั้นการวัดเจตคติในลักษณะนี้จึงสามารเห็นแบบแผน (Pattern) ของเจตคติที่มีต่อเรื่องนั้นของกลุ่มบุคคลที่วัดได้อีกด้วย

4) มาตรวัดแบบออสกูด (Osgood Scale) วิธีการวัดแบบนี้มีชื่อเรียกอีกอย่างหนึ่งว่า วิธีการแห่งความแตกต่างของความหมาย (Semantic Differential Method) ซึ่งมีลักษณะคล้ายกับการหาความหมายของมโนทัศน์ (Concept) ซึ่งมาตรวัดเจตคติแบบนี้นอกจากจะใช้วัดเจตคติแล้ว ยังสามารนำไปใช้วัด บุคลิกภาพ ความคิดเห็น ความเชื่อและความรู้สึกที่มีต่อสิ่งต่าง ๆ ได้อีกด้วย รวมทั้งสามารถแยกความแตกต่างของมโนทัศน์ต่าง ๆ ได้เป็นอย่างดี แต่ก็มีความจำกัดในการเลือกใช้คำคุณศัพท์มาใช้ มาตรวัดแบบนี้จะสร้างข้อความในลักษณะมโนทัศน์ขึ้น ซึ่งอาจจะเป็นคำ ข้อความ วลี หรือประโยคสั้น ๆ ที่ได้ใจความสมบูรณ์ก็ได้ โดยการเลือกคำคุณศัพท์ที่มีความหมายตรงกันข้ามซึ่งมักมี 3 องค์ประกอบ

1) คำคุณศัพท์ที่แสดงการประเมินผล (Evaluative Adjective) เช่น ดี – เลว, สวย – ขี้เหร่
2) คำคุณศัพท์ที่แสดงถึงศักยภาพ (Potency Adjective) เช่น อ้วน – ผอม, หนัก – เบา
3) คำคุณศัพท์ที่แสดงการเคลื่อนไหวของมโนทัศน์ เช่น ร่าเริง – เศร้า, เร็ว – ช้า

ในทางปฏิบัติผู้วิจัยอาจจะสนใจคำคุณศัพท์เพียงลักษณะใดลักษณะหนึ่งก็ได้ ไม่จำเป็นต้องเลือกให้ครบทั้ง 3 องค์ประกอบเสมอไป

2.9 มาตรฐานคุณภาพซอฟต์แวร์ไทย (Thai Quality Software)

สมาคมอุตสาหกรรมซอฟต์แวร์ไทย [10] กล่าวถึงมาตรฐานคุณภาพซอฟต์แวร์ไทย Thai Quality Software (TQS) ไว้ว่า ถูกกำหนดขึ้นในปี พ.ศ. 2544 โดยสมาคมอุตสาหกรรมซอฟต์แวร์ไทย (ATSI) เพราะเห็นว่าประเทศไทยควรสร้างมาตรฐานในด้านการพัฒนาซอฟต์แวร์ โดยมีจุดมุ่งหมายเพื่อการยกระดับคุณภาพและปรับปรุงความสามารถในการพัฒนาซอฟต์แวร์ของวิสาหกิจไทย เพื่อเพิ่มโอกาสในการรับพัฒนาโครงการซอฟต์แวร์ภาครัฐและเพิ่มขีดความสามารถทางการแข่งขันในตลาดต่างประเทศ ซึ่งทางสมาคมอุตสาหกรรมซอฟต์แวร์ไทยได้ทำการกำหนดมาตรฐานคุณภาพซอฟต์แวร์นี้ให้เหมาะสมกับวิสาหกิจขนาดกลางและขนาดย่อม (SME) ของไทยโดยเฉพาะ ซึ่งทำการอ้างอิงจากมาตรฐาน ISO/IEC 12207 ซึ่งเป็นมาตรฐานนานาชาติในส่วนที่เกี่ยวกับการควบคุมคุณภาพในกระบวนการผลิตซอฟต์แวร์

มาตรฐานคุณภาพซอฟต์แวร์ไทยมี 5 ระดับ ดังนี้

1. Basic Software Engineering เป็นระดับการกำหนดกระบวนการต่าง ๆ ที่ต้องทำ
2. Define เป็นการวางแผนสิ่งที่ต้องทำเพื่อมุ่งไปสู่การรับการตรวจสอบ

3. Perform เป็นการจัดทำโครงงานทดสอบตามแผนที่วางไว้
4. Measure เป็นการวัดผลและประเมินผล
5. Continuous Improvement รักษาคุณภาพและพัฒนาให้ดีขึ้นไป

ขั้นตอนต่าง ๆ ในมาตรฐานคุณภาพซอฟต์แวร์ไทย ไม่ได้มีการดัดแปลงหรือเปลี่ยนแปลงไปจาก ISO/IEC 12207 เพียงแต่คัดเลือกขั้นตอนต่าง ๆ ที่เหมาะสมกับระดับความสามารถในแต่ละระดับเพื่อมุ่งไปสู่มาตรฐานสากลได้อย่างเป็นระบบ มิใช่การสร้างมาตรฐานใหม่ให้เข้ากับรูปแบบการพัฒนาซอฟต์แวร์แบบคนไทย เพราะมาตรฐานที่อ้างอิงนั้นเป็นมาตรฐานที่เป็นที่ยอมรับกันในระดับนานาชาติ