บทที่ 2

เอกสารและงานวิจัยที่เกี่ยวข้อง

ในการทำวิจัยครั้งนี้ผู้ทำการวิจัยได้ศึกษาทฤษฎี แนวคิด หลักการ รวมทั้งเอกสารงานวิจัยที่เกี่ยวข้องในประเทศ และต่างประเทศ เพื่อนำมาประกอบเป็นองค์ความรู้สำหรับการทำวิจัย ซึ่งมีข้อมูลต่างๆ ดังนี้

1. สิ่งแวดล้อมของการพัฒนาซอฟต์แวร์ (Software Development Ecosystems)
2. ภาพรวมของกระบวนการ (A Generic View of Process)
3. กรอบงานของกระบวนการ (A Process Framework)
4. การกำหนดรายละเอียดของกิจกรรมภายในระบบงาน
5. ระเบียบวิธีการพัฒนาซอฟต์แวร์แบบดั้งเดิม (Traditional Methodology)
6. ระเบียบวิธีการพัฒนาซอฟต์แวร์ตามแนวคิด อาไจล์ (Agile Methodology)
7. ประสิทธิภาพของโครงการ (Efficiency of Project)
8. แบบจำลองพื้นฐานสถาปัตยกรรม และวิศวกรรมซอฟต์แวร์ (Model-Based Architecting and Software Engineering)
9. แนวคิดการประเมินผลโครงกาของสตัฟเฟิลบีม (Stufflebeam's CIPP Model)
10. งานวิจัยที่เกี่ยวข้อง

สิ่งแวดล้อมของการพัฒนาซอฟต์แวร์ (Software Development Ecosystems)

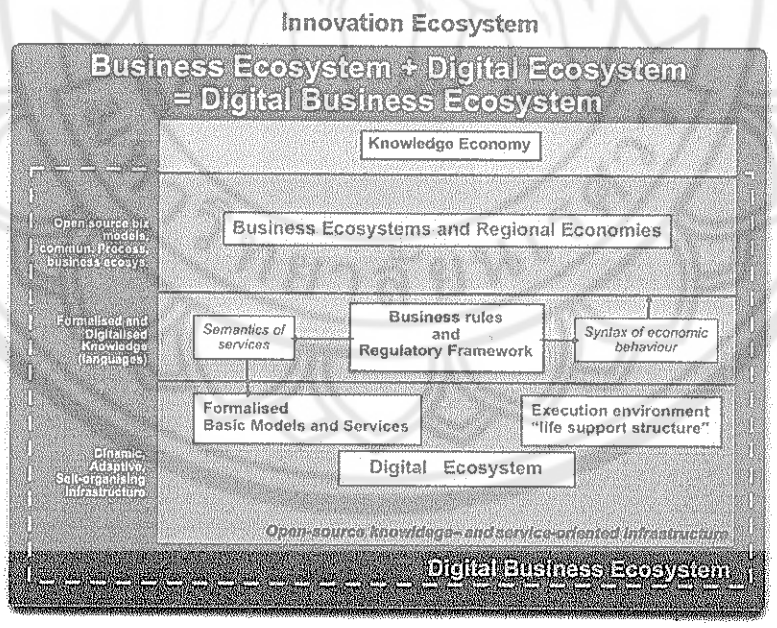
จากการค้นคว้าเอกสาร และงานวิจัยที่เกี่ยวข้อง พบว่า โครงการพัฒนาซอฟต์แวร์ในแต่ละประเภทมีจุดมุ่งหมาย และรูปแบบการดำเนินการวางแผนที่แตกต่างกันออกไป อันเนื่องมาจากบทบาทของทีมพัฒนาที่มีต่อลักษณะของโครงการพัฒนาซอฟต์แวร์ และสิ่งแวดล้อมของการพัฒนาซอฟต์แวร์ถือเป็นปัจจัยที่สำคัญที่เข้ามามีส่วนเกี่ยวข้อง และส่งผลกระทบต่อสิ่งแวดล้อมของการพัฒนาซอฟต์แวร์

European Commission (n.d.) กล่าวถึง สิ่งแวดล้อมของการพัฒนาซอฟต์แวร์ (Software Development Ecosystems) ไว้ดังนี้ ระบบที่เกิดจากความสัมพันธ์ระหว่างสิ่งมีชีวิตกับสิ่งแวดล้อม ซึ่งมีรูปแบบการดำเนินชีวิตที่เป็นธรรมชาติ ถูกให้ความนิยามว่าสังคมทางชีววิทยาเกี่ยวกับการมีผลกระทบต่อกันของระบบและสิ่งแวดล้อมทางกายภาพของมัน ในทำนองเดียวกัน ระบบที่เกิดจากความสัมพันธ์ระหว่างสิ่งแวดล้อมทางธุรกิจ คือ ระบบเครือข่ายของผู้ซื้อ ผู้จัดหาวัตถุดิบ และผู้เข็น

เอกสารทางกฎหมายของผลิตภัณฑ์และบริการที่เกี่ยวข้อง และสภาพแวดล้อมทางเศรษฐกิจ รวมทั้งองค์กรและข้อบังคับของขอบข่ายงาน

ระบบที่เกิดจากความสัมพันธ์ระหว่างสิ่งแวดล้อมทางดิจิตอล (Digital) เป็นโครงสร้างพื้นฐานทางทางดิจิตอล (Digital) ซึ่งมีการจัดระบบของตัวมันเอง มุ่งหมายเพื่อการสร้างสิ่งแวดล้อมทางดิจิตอล (Digital) สำหรับการเชื่อมโยงเครือข่ายภายในองค์กร ซึ่งสนับสนุนการทำงานร่วมกัน การใช้องค์ความรู้ร่วมกัน การพัฒนาในเรื่องของการเผยสู่สาธารณะและปรับตัวเข้ากับเทคโนโลยีและวิวัฒนาการแบบจำลองทางธุรกิจ ระบบที่เกิดจากความสัมพันธ์ระหว่างสิ่งแวดล้อมทางดิจิตอล (Digital) เป็นวิธีที่เปลี่ยนแนวคิดไปยังโลกดิจิตอล (Digital)

การสร้างวิธีการเรียนรู้อย่างแท้จริงเกี่ยวกับระบบที่เกิดจากความสัมพันธ์ระหว่างสิ่งแวดล้อมทางธรรมชาติ ขณะที่ระบบที่เกิดจากความสัมพันธ์ระหว่างสิ่งแวดล้อมที่มีอยู่มากมายในธรรมชาติ ระบบที่เกิดจากความสัมพันธ์ระหว่างสิ่งแวดล้อมดิจิตอล (Digital) มีอยู่อย่างหลากหลาย เนื่องจากการเปลี่ยนแปลงและการพัฒนาเกี่ยวกับผลิตภัณฑ์ และการบริการถูกปรับให้เหมาะสมความต้องการเฉพาะที่ ดังภาพ 1



ภาพ 1 สิ่งแวดล้อมของการพัฒนาซอฟต์แวร์

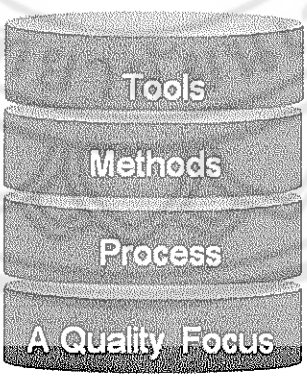
ที่มา: European Commission, n.d.

M. Hadzic, et al. (2007) กล่าวเกี่ยวกับ Digital Ecosystem ไว้ดังนี้ Digital Ecosystem คือ ระบบที่เกิดจากความสัมพันธ์ระหว่างสิ่งแวดล้อมทางดิจิตอล (Digital) เป็นกิจกรรมที่สัมพันธ์กับการปรับเปลี่ยน และการทำงานร่วมกันของชุมชนดิจิตอล (Digital) ประกอบด้วยรูปแบบของการรับส่งข้อมูลทางดิจิตอล (Digital) ซึ่งมีการเชื่อมโยงสัมพันธ์กัน และพึ่งพากันในสิ่งแวดล้อมทางดิจิตอล (Digital) โดยมีผลกระทบต่อหน่วยการทำงานและเชื่อมโยงกันตลอดการไหลของสารสนเทศ และกิจการทางธุรกิจ

ภาพรวมของกระบวนการ (A Generic View of Process)

Fritz Bauer เสนอนิยาม คำว่า วิศวกรรมซอฟต์แวร์ในงานประชุมวิชาการทางด้านวิศวกรรมซอฟต์แวร์ ซึ่งอ้างอิงจาก Roger S. Pressman (1992) ไว้ดังนี้ "วิศวกรรมซอฟต์แวร์ คือ การสร้างและใช้หลักการทางวิศวกรรมที่เหมาะสมเพื่อให้ได้มาซึ่งซอฟต์แวร์ที่มีราคาไม่แพง แต่เชื่อใจได้ และทำงานได้อย่างมีประสิทธิภาพบนเครื่องจักรจริง"

IEEE [IEE93] ให้คำนิยาม คำว่าวิศวกรรมซอฟต์แวร์ ซึ่งอ้างอิงจาก Roger S. Pressman (1992) ดังนี้ วิศวกรรมซอฟต์แวร์ หมายถึง .1) การประยุกต์ใช้แนวทางเชิงปริมาณที่เป็นระบบระเบียบในการพัฒนาดำเนินงาน และบำรุงรักษาซอฟต์แวร์ กล่าวคือ มันเป็นการประยุกต์ใช้วิศวกรรมกับซอฟต์แวร์นั่นเอง 2) การศึกษาแนวทางในข้อ 1



ภาพ 2 ระดับชั้นของวิศวกรรมซอฟต์แวร์

ที่มา: พรฤดี เนติโสภากุล, 2549

พรฤดี เนติโสภากุล (2549) กล่าวถึง วิศวกรรมซอฟต์แวร์: เทคโนโลยีแบบลำดับชั้น ดังนี้ วิศวกรรมซอฟต์แวร์เป็นเทคโนโลยีแบบลำดับชั้น ควรต้องใช้ควบคู่กับนโยบายด้านคุณภาพขององค์กรเพื่อช่วยสนับสนุนการปรับปรุงกระบวนการแบบต่อเนื่อง ซึ่งส่งผลให้เกิดการก้าวไปสู่การพัฒนาแนวคิดด้านวิศวกรรมซอฟต์แวร์ที่มีประสิทธิภาพดี หรือ "ขั้นศูนย์คุณภาพ (A Quality Focus)"

ชั้นกระบวนการ (Process) ถือได้ว่าเป็นรากฐานของวิศวกรรมซอฟต์แวร์ ซึ่งเป็นกระบวนการทางวิศวกรรมซอฟต์แวร์ ทำหน้าที่เสมือนตัวเชื่อมชั้นต่างๆ ที่เกี่ยวกับเทคโนโลยีเข้าด้วยกัน และส่งผลให้เกิดการพัฒนาซอฟต์แวร์อย่างเหมาะสม กระบวนการเป็นตัวกำหนดกรอบ เพื่อให้ได้มาซึ่งเทคโนโลยีที่มีประสิทธิภาพในวิศวกรรมซอฟต์แวร์ กระบวนการซอฟต์แวร์ ถือได้ว่าเป็นหลักสำคัญต่อผู้บริหารจัดการต่อโครงการพัฒนาซอฟต์แวร์ และช่วยให้สามารถประยุกต์ใช้เทคนิคต่างๆ ในผลผลิตของงาน ได้แก่ แบบจำลอง เอกสาร รายงาน เป็นต้น อีกทั้งเมื่อเกิดการเปลี่ยนแปลงความต้องการควรได้รับการบริหารจัดการอย่างเหมาะสม และมีการรับประกันคุณภาพของซอฟต์แวร์

วิธีทางวิศวกรรมซอฟต์แวร์ (Methods) เป็นแนวทางหรือวิธีรวบรวมงานหลายๆ ด้านเข้าด้วยกัน เช่น การติดต่อสื่อสาร การวิเคราะห์ความต้องการ การสร้างแบบจำลองการออกแบบ การพัฒนาโปรแกรม การทดสอบ และการสนับสนุน อีกทั้งยังรวมถึงหลักการพื้นฐานของเทคโนโลยีทุกสาขา การสร้างแบบจำลอง และเทคนิคเชิงพรรณนาแบบอื่นๆ

เครื่องมือทางวิศวกรรมซอฟต์แวร์ (Tools) เป็นเครื่องมือที่ช่วยให้กระบวนการและวิธีทางวิศวกรรมซอฟต์แวร์ ดำเนินไปได้อย่างสะดวกสบายยิ่งขึ้น

กรอบงานของกระบวนการ (A Process Framework)

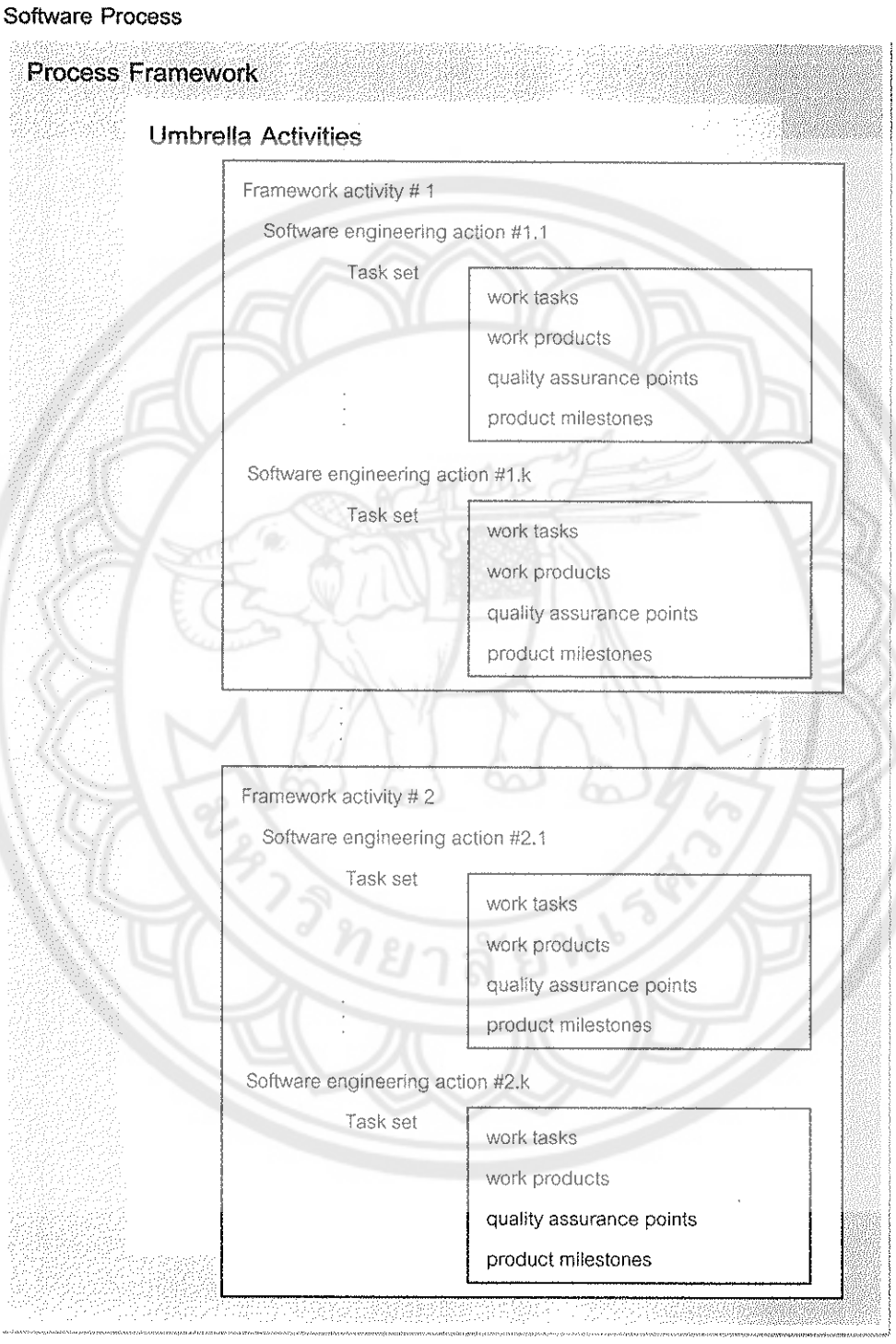
พรฤดี เนติโสภากุล (2549) กล่าวถึง กรอบงานของกระบวนการดังนี้ กรอบงานของกระบวนการถือเป็นพื้นฐานของกระบวนการทางซอฟต์แวร์ กรอบงาน (Framework Activity) สามารถแยกออกเป็นกิจกรรมต่างๆ ซึ่งสามารถประยุกต์ใช้ได้กับทุกโครงการพัฒนาซอฟต์แวร์ โดยกรอบงานครอบคลุมถึงชุดกิจกรรมร่วม (Umbrella Activities) ซึ่งเป็นกิจกรรมที่เกี่ยวข้องกับการพัฒนาซอฟต์แวร์ทั้งกระบวนการ สำหรับชุดของการกระทำด้านวิศวกรรมซอฟต์แวร์ (Software Engineering Actions) โดยแต่ละชุดของกิจกรรมประกอบด้วย กลุ่มงาน (Tasks) ที่เกี่ยวข้องกับการพัฒนาผลิตภัณฑ์ทางวิศวกรรมซอฟต์แวร์ (Work Task) เช่น การออกแบบ (Design) ผลิตภัณฑ์ที่เกี่ยวข้อง (Work Products) การประกันคุณภาพ (Quality Assurance) และหลักไมล์ของโครงการ (Project Milestone)

สำหรับกรอบงานของกระบวนการแบบทั่วไป สามารถนำไปประยุกต์ใช้กับโครงการพัฒนาซอฟต์แวร์มีดังนี้

1. การสื่อสาร (Communication)
2. การวางแผน (Planning)
3. การสร้างแบบจำลอง (Modeling)
4. การสร้าง (Construction)
5. การใช้งาน (Deployment)

กรอบงานของกระบวนการแบบทั่วไปควรได้รับการเสริมด้วยกิจกรรมร่วม (Umbrella Activities) ได้แก่

1. การติดตามและควบคุมโครงการซอฟต์แวร์ (Software Project Tracking and Control)
2. การจัดการความเสี่ยง (Risk Management)
3. การประกันคุณภาพซอฟต์แวร์ (Software Quality Assurance)
4. การพิจารณาทางด้านเทคนิค (Formal Technique Reviews)
5. การวัด (Measurement)
6. การจัดการโครงแบบของซอฟต์แวร์ (Software Configuration Management)
7. การเตรียมและการผลิตชิ้นงาน (Work Product Preparation and Production)



ภาพ 3 กรอบงานของกระบวนการทางซอฟต์แวร์

ที่มา: ดัดแปลงจาก พรฤดี เนติโสภากุล, 2549

การกำหนดรายละเอียดของกิจกรรมภายในระบบงาน

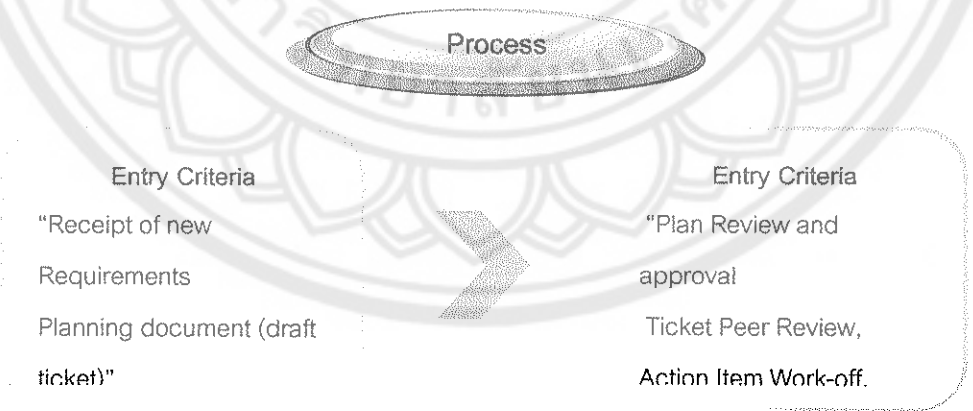
Minoru Nakamura, Yasuhiro Nozue and Mint อ้างอิงจาก สมชาย กิตติชัยกุลกิจ (2549) กล่าวถึงความสำคัญของกระบวนการพัฒนาซอฟต์แวร์ดังนี้ จากการที่ปัจจุบันซอฟต์แวร์มีราคาสูงกว่าฮาร์ดแวร์ อีกทั้งต้นทุนในการพัฒนาไม่สูงนักจึงทำให้มีการเปิดบริษัทผลิตซอฟต์แวร์มากขึ้น แต่เนื่องจากบริษัทเหล่านั้นไม่มีความเชี่ยวชาญมากพอ จึงส่งผลให้ผลิตภัณฑ์ซอฟต์แวร์ที่ผลิตขึ้นไม่มีคุณภาพ และส่งมอบงานล่าช้ากว่ากำหนด และยังส่งผลให้ค่าใช้จ่ายสูงขึ้นจึงก่อให้เกิด "ปัญหาวิกฤตซอฟต์แวร์"

Minoru Nakamura, Yasuhiro Nozue and Mint อ้างอิงจาก สมชาย กิตติชัยกุลกิจ (2549) กล่าวต่อไปว่า จึงมีหลักการเอ็นจิเนียริง (Engineering) เข้ามาช่วยกำหนดขั้นตอนการพัฒนาซอฟต์แวร์ เพื่อให้กระบวนการพัฒนาซอฟต์แวร์มีแนวทางการพัฒนาที่ชัดเจน โดยมีการกำหนดเป้าหมาย 2 ประการ คือ เพื่อเพิ่มประสิทธิภาพในการผลิต และเพื่อลดการพึ่งพาความสามารถของคนใดคนหนึ่ง

การกำหนดรายละเอียดในแต่ละขั้นตอนของการพัฒนากำหนดขึ้นตามหลักการของวงจรการพัฒนาซอฟต์แวร์ (Software Development Life Cycle: SDLC) โดยมีขั้นตอนดังต่อไปนี้

1. **ขั้นตอนการวิเคราะห์** (Analysis)

การดำเนินงานเริ่มตั้งแต่นำความต้องการมาวิเคราะห์ ซึ่งนำไปสู่การวางแผนการพัฒนา

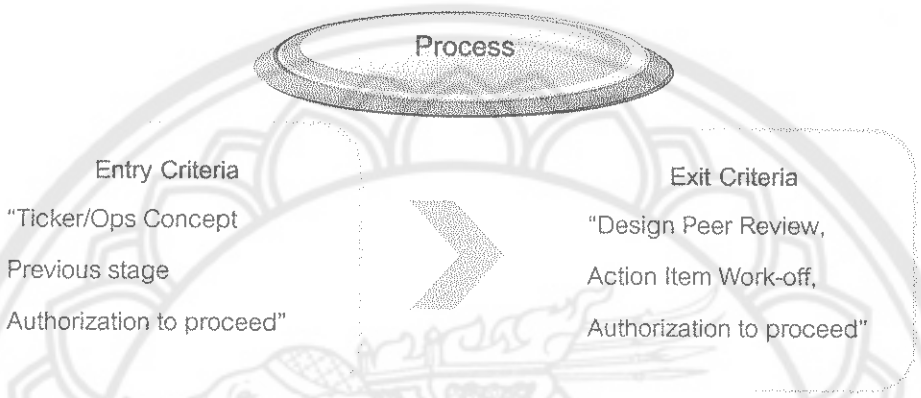


ภาพ 4 ขั้นตอนการวิเคราะห์

ที่มา: ดัดแปลงจาก IEEE Std 730™, 1998

2. **ขั้นตอนการออกแบบ** (Design)

การดำเนินการเริ่มตั้งแต่การนำความต้องการที่ออกแบบไว้ในขั้นตอนการวิเคราะห์มาเข้าสู่กระบวนการออกแบบ ดังภาพ 5

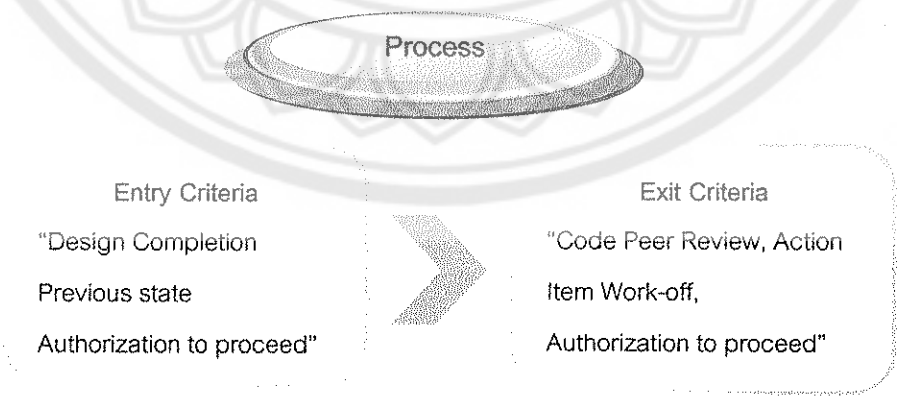


ภาพ 5 ขั้นตอนการออกแบบ

ที่มา: ดัดแปลงจาก IEEE Std 730™, 1998

3. **ขั้นตอนการพัฒนาซอฟต์แวร์** (Programming)

การดำเนินงานเริ่มตั้งแต่นำผลลัพธ์ที่ได้จากการออกแบบนำมาพัฒนา



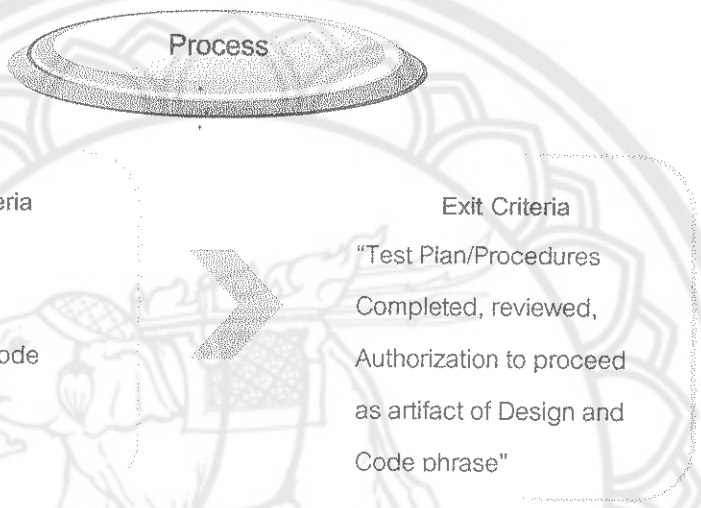
ภาพ 6 ขั้นตอนการพัฒนาซอฟต์แวร์

ที่มา: ดัดแปลงจาก IEEE Std 730™ , 1998

4. **ขั้นตอนการทดสอบและการเชื่อมต่อ** (Testing and Integration)

การดำเนินงานเริ่มตั้งแต่นำผลลัพธ์ที่ได้จากขั้นตอนพัฒนามาเชื่อมต่อเพื่อทดสอบการทำงานของระบบ โดยมีการวางแผนการทดสอบความสมบูรณ์ของระบบ

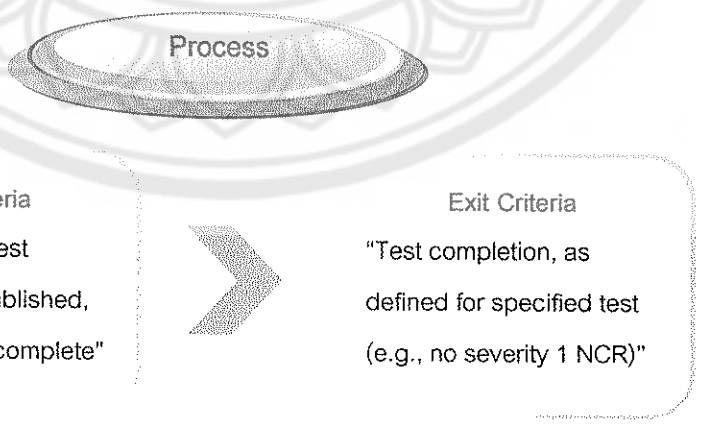
4.1 การวางแผนการทดสอบ



ภาพ 7 ขั้นตอนการทดสอบ

ที่มา: ดัดแปลงจาก IEEE Std 730™, 1998

4.2 การทดสอบ



ภาพ 8 ขั้นตอนการทดสอบการเชื่อมต่อ

ที่มา: ดัดแปลงจาก IEEE Std 730™, 1998

จากแนวคิด และองค์ประกอบต่างๆ ที่ได้กล่าวไว้ข้างต้นถือเป็นส่วนพื้นฐานสำหรับการกำหนดรายละเอียดกิจกรรมการดำเนินงานสำหรับการพัฒนาซอฟต์แวร์ เพื่อให้สามารถพัฒนาซอฟต์แวร์ได้อย่างเป็นระบบด้วยเหตุนี้จึงมีผู้นำเสนอทฤษฎีและระเบียบวิธีการพัฒนาซอฟต์แวร์แบบต่างๆ เพื่อให้การพัฒนาซอฟต์แวร์มีประสิทธิภาพมากขึ้น โดยระเบียบวิธีการพัฒนาซอฟต์แวร์มีวิวัฒนาการดังนี้

ระเบียบวิธีการพัฒนาซอฟต์แวร์แบบดั้งเดิม (Traditional Methodology)

ระเบียบวิธีการพัฒนาซอฟต์แวร์ (Methodology) หมายถึง การนำระเบียบวิธีการทางแนวคิดของวงจรการพัฒนาซอฟต์แวร์มาสู่ขั้นตอนการปฏิบัติจริงเพื่อนำมาใช้ในการพัฒนาซอฟต์แวร์ (กิตติ ภักดีวัฒนะกุล และพนิดา พานิชกุล, 2548)

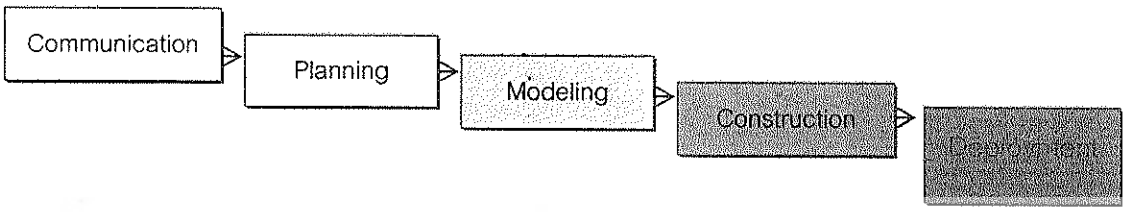
ปัจจุบันมีการพัฒนาระเบียบวิธีการพัฒนาซอฟต์แวร์ (Methodology) ไว้หลากหลายแนวคิด ซึ่งแต่ละแนวคิดจะมีการกำหนดขั้นตอนการดำเนินการที่แตกต่างกัน ดังนี้

แบบจำลองที่มีระเบียบแบบแผน (Disciplined Model) ตัวอย่างโมเดล: โมเดลที่มีระเบียบแบบแผน (Model Example: Disciplined Model) โดยแต่ละโมเดลมีการกำหนดกระบวนการและขั้นตอนในการดำเนินการไว้อย่างชัดเจนหรือมีกระบวนการที่เป็นขั้นตอนแน่นอนหรือมีลักษณะที่เป็น Disciplined

1. Waterfall Model Series

1.1 **แบบจำลองน้ำตก** (Waterfall Model) วงจรการพัฒนาซอฟต์แวร์ (Software Development Life Cycle: SDLC) แบบจำลองน้ำตก (Waterfall Model) ได้รับความนิยมมากในช่วงปี ค.ศ. 1980 สำหรับการพัฒนาซอฟต์แวร์ในแบบ Waterfall จะอธิบายขั้นตอนการปฏิบัติงานเป็นลำดับขั้นตอนที่ชัดเจนในรูปแบบจำลอง ซึ่งช่วยให้ผู้พัฒนาสามารถเข้าใจภาพรวมของระบบได้ดียิ่งขึ้น อีกทั้งแบบจำลองยังสนับสนุนกระบวนการสื่อสารระหว่างผู้ที่เกี่ยวข้องกับระบบอีกด้วย

ในยุคแรกๆ ไม่อนุญาตให้มีการย้อนกลับไปแก้ไขในขั้นตอนก่อนหน้า ดังนั้นนักพัฒนาระบบจึงจำเป็นทำขั้นตอนปัจจุบันให้ดีที่สุดก่อนทำในขั้นตอนถัดไป แต่ในปัจจุบันมีการพัฒนาโมเดลโดยอนุญาตให้มีการย้อนกลับไปแก้ไขในขั้นตอนก่อนหน้าได้ เพื่อให้สอดคล้องกับสภาวการณ์ที่เปลี่ยนแปลงไปอย่างรวดเร็วของความต้องการของลูกค้า เทคโนโลยี การรับรู้ รวมทั้งนวัตกรรมใหม่ๆ (กิตติ ภักดีวัฒนะกุล และพนิดา พานิชกุล, 2548)



ภาพ 9 ระเบียบวิธีการพัฒนาซอฟต์แวร์แบบน้ำตก (Waterfall Model)

ที่มา: ดัดแปลงจาก Roger S. Pressman,1992

1.2 **Rapid Prototype Development Model** แนวคิดของการเพิ่มส่วนของต้นแบบ (Prototype) เข้าไปในส่วนของ แบบจำลองน้ำตก (Waterfall Model) นั้นเกิดจากแนวคิดที่ว่าการสร้างต้นแบบ (Prototype) จะช่วยในการสื่อสารระหว่างผู้ที่เกี่ยวข้องสามารถเข้าใจและมองเห็นภาพรวม ของระบบก่อนการทำงานจริงตามแนวคิดของ What You See Is What You Get (WYSIWYG) หรือสิ่งที่มองเห็น คือ สิ่งที่ได้รับ โดยพัฒนาต้นแบบ (Prototype) ขึ้นเพื่อจำลองภาพระบบจริงที่จะเกิดขึ้น โดยแนวคิดนี้ค่อนข้างได้รับความนิยม เนื่องจากสามารถลดปัญหาการพัฒนาสิ่งที่ผู้ใช้ไม่ต้องการ (Fail at first launch) โดยแนวคิดของการเพิ่มต้นแบบ (Prototype) เข้าไปนั้นจะสามารถเพิ่มในขั้นตอนการวิเคราะห์และสามารถปรับปรุงพัฒนาต้นแบบ (Prototype) และทำการจำลองการทำต้นแบบ (Prototype) ก่อนการพัฒนาจริง

1.3 **แบบจำลองแบบวี** (V Model) แนวคิดของ V Model มาจากการที่พัฒนาซอฟต์แวร์ตามแบบจำลองน้ำตก (Waterfall Model) โดยเพิ่มแนวคิดการตรวจสอบ (Verification) แต่ละขั้นตอนเพื่อเพิ่มประสิทธิภาพในการทำงานของแบบจำลอง และตรวจสอบกระบวนการพัฒนาให้สามารถดำเนินไปได้อย่างดี ดังนั้น จึงกำหนดคุณลักษณะของการตรวจสอบไว้ เมื่อแบ่งการทำงานเป็น 8 ขั้นตอน โดยแบ่งเป็นสองส่วนหลัก ส่วนแรกเปรียบเสมือนขั้นตอนในการจัดสร้างเป็นขาด้านแรกของแบบจำลองแบบวี (V Model) ส่วนขาด้านที่สองที่เป็นด้านขาขึ้นของแบบจำลองแบบวี (V Model) เสมือนเป็นด้านที่คอยตรวจสอบการทำงานสอดคล้องกัน ดังนั้น หากต้องการจะทราบว่าความต้องการที่ได้กำหนดไว้ ได้ถูกนำไปจัดสร้างในระบบงานและรองรับความต้องการได้จริงหรือไม่ จะตรวจสอบในขั้นตอนของการทดสอบการยอมรับ (Acceptance Testing) ในขณะที่การทดสอบระบบ (System Testing) จะตรวจสอบขั้นตอนการวิเคราะห์ ดังนั้น จะเห็นได้ว่า (Analysis) แนวคิดดังกล่าวพยายามสร้างความมั่นใจว่าระบบที่พัฒนาขึ้นนั้นถูกต้องตามความต้องการจริง โดยมีการเตรียมพร้อมในขั้นตอนต้น เพื่อนำไปใช้ในการตรวจสอบย้อนกลับ (เมสินี นาคมณี, 2547, หน้า 29)

1.4 แบบจำลองแบบฟันปลา (Saw tooth Model) แนวคิดของแบบจำลองแบบฟันปลา (Saw tooth Model) พัฒนาเพิ่มเติมจากแนวคิดการตรวจสอบของแบบจำลองแบบวี (V Model) เนื่องจากในส่วนของแบบจำลองแบบวี (V Model) ให้ความสำคัญเฉพาะการตรวจสอบย้อนกลับแต่ละขั้นตอน โดยไม่ได้ให้ความสำคัญส่วนของปัจจัยหลักที่ส่งผลในการตัดสินคุณภาพของนั่นคือส่วนของลูกค้า หรือผู้ใช้งานจริง ดังนั้นแบบจำลองแบบฟันปลา (Saw tooth Model) จึงพยายามที่จะดึงความพึงพอใจของลูกค้าเข้ามาใช้เป็นส่วนหนึ่งในการตรวจสอบการทำงานในแต่ละขั้นตอนด้วย โดยนำการสร้างต้นแบบ (Prototype) เข้ามาช่วยในการตรวจสอบความถูกต้องระหว่างกลุ่มผู้พัฒนากับกลุ่มลูกค้า แนวคิดแบบจำลองแบบฟันปลา (Saw tooth Model) เริ่มมีผู้ให้ความสนใจอย่างมาก เนื่องจากไม่เพียงจัดกระบวนการตามแนวคิดของ วิศวกรรม (Engineering) เท่านั้น หากมุ่งเน้นความสนใจส่วนของการบริหารจัดการ (Management) โดยแนวคิดดังกล่าวให้ความสำคัญกับการสร้างความสัมพันธ์ และความพึงพอใจกับลูกค้า ซึ่งเป็นผู้ตัดสินว่าซอฟต์แวร์มีประสิทธิภาพหรือมีคุณภาพตามที่ต้องการหรือไม่ (เมสินี นาคมณี, 2547, หน้า 30)

1.5 แบบจำลองฟันฉลาม (Shark tooth Model) แบบจำลองฟันฉลาม (Shark tooth) นั้นกล่าวได้ว่ามีความคล้ายคลึงกับ แบบจำลองแบบฟันปลา (Saw tooth Model) เพียงแต่เพิ่มส่วนของการบริหารจัดการ (Management) เข้ามาในขั้นตอนพัฒนา เพื่อสร้างกระบวนการการตรวจสอบคุณภาพ และประสานระหว่างกลุ่มลูกค้าและกลุ่มผู้พัฒนา ซึ่งช่วยเพิ่มประสิทธิภาพของกระบวนการในการพัฒนาซอฟต์แวร์ดียิ่งขึ้น

2. Modern Approach Series

Lan Sommerville อ้างอิงใน เมสินี นาคมณี (2547) กล่าวว่า ความต้องการของระบบเปลี่ยนแปลงตลอดเวลาในการทำโครงการ กระบวนการวงรอบ (Iteration) ในขั้นตอนแรกๆ มีการทำซ้ำสมอเป็นส่วนหนึ่งของกระบวนการพัฒนาระบบขนาดใหญ่

แนวคิดดังกล่าวเกิดขึ้นจากแนวคิดที่ว่าระหว่างการพัฒนาระบบความต้องการของซอฟต์แวร์มักจะเปลี่ยนแปลงอยู่เสมอ ดังนั้นกระบวนการแบบจำลองน้ำตก (Waterfall Model) ซึ่งพยายามกำหนดความต้องการระบบเป็นจุดเริ่มต้นสำหรับการพัฒนาซอฟต์แวร์อาจจะไม่สามารถทำได้ เนื่องจากการปรับเปลี่ยนความต้องการซอฟต์แวร์ที่เกิดขึ้นระหว่างการพัฒนา ดังนั้น วงจรการซอฟต์แวร์ที่ดีควรจะนำแนวคิดของการเปลี่ยนแปลงของความต้องการซอฟต์แวร์เข้ามาส่วนหนึ่งในการกำหนดขั้นตอนกระบวนการพัฒนาซอฟต์แวร์ นั่นคือกระบวนการวงรอบ (Iteration) เพื่อรองรับการปรับเปลี่ยนของความต้องการซอฟต์แวร์อยู่ตลอดเวลาและได้มีการนำแนวคิดแบบการวงรอบ (Iteration) นี้เองไปปรับเปลี่ยนเป็นแนวคิดของโมเดลต่างๆ นั่นเอง

โดยในลักษณะของแบบจำลองการวงรอบ (Iteration Model) นั้นแบ่งออกเป็นสองประเภทหลักนั้นคือ ลักษณะของวงรอบแบบพัฒนา (Evolutionary Model) และวงรอบเพิ่มเติม (Incremental Model) โดยในส่วนของวงรอบแบบพัฒนา (Evolutionary Model) นั้นมุ่งเน้นการพัฒนาในแต่ละวงรอบที่มีการทำงานที่แตกต่างกัน ในขณะที่ลักษณะของวงรอบเพิ่มเติม (Incremental Model) จะเป็นการวนรอบที่มีลักษณะของการทำงานซ้ำแต่เป็นการเพิ่มเติมส่วนงาน เช่น Rational Unified Process นั่นเอง

2.1 The Spiral Model การพัฒนาซอฟต์แวร์แบบ The Spiral Model เป็นโมเดลที่มีการนำความเสี่ยง (Risk) เข้ามาเป็นตัวขับเคลื่อนกระบวนการพัฒนา ซึ่งเป็นเครื่องมือช่วยในการตัดสินใจของผู้ที่เกี่ยวข้อง วงจรการพัฒนาจะวนเป็นวงกลมและวงจะกว้างขึ้นเรื่อยๆ ตามระดับการพัฒนา ในขณะที่ความเสี่ยงจะลดลงตามลำดับ และจุดตรวจสอบ (Milestone) ซึ่งจะมีการตรวจสอบความต้องการและวิเคราะห์ความเสี่ยง ช่วยทำให้ผู้ที่เกี่ยวข้องกับการพัฒนาซอฟต์แวร์ร่วมกันพิจารณาความเป็นไปได้ในการพัฒนาซอฟต์แวร์ (Roger S. Pressman, 1992)

เมสินี นาคมณี (2547) กล่าวถึงลักษณะเฉพาะของแบบจำลองดังนี้

ระยะที่ 1 ด้านวิศวกรรม (Engineering Stage) กิจกรรมหลักของด้านวิศวกรรม คือ การกำหนดกรอบงานอย่างคราวๆ เพื่อกำหนดทิศทางสำหรับการพัฒนาซอฟต์แวร์

ขั้นตอน 1 การศึกษาความเป็นไปได้ (Feasibility Iteration) ขั้นตอนนี้เป็นการวางแผนการดำเนินงานโดยรวม (Inception) เพื่อแสดงภาพรวมของระบบจากการศึกษาความเป็นไปได้ (Feasibility) และศึกษาความต้องการของระบบ

ขั้นตอน 2 การวางโครงสร้างรูปแบบ (Architecture Iteration) ขั้นตอนนี้เป็นการวางแผนการดำเนินงานอย่างละเอียด (Elaboration) และออกแบบโครงสร้างของระบบ

ระยะที่ 2 ด้านการผลิต (Manufacturing Stage)

ขั้นตอน 3 การจัดสร้างเพื่อการใช้งาน (Usable Iteration) ขั้นตอนี้เป็นการพัฒนาซอฟต์แวร์เพื่อใช้งาน หากระบบมีความซับซ้อนมากจะมีการแบ่งขั้นตอนนี้เป็นส่วนย่อยๆ

ขั้นตอน 4 การนำไปใช้งาน (Product Release) ขั้นตอนนี้เป็นขั้นตอนสุดท้ายของการวงรอบ (Iteration) เพื่อดำเนินการส่งมอบซอฟต์แวร์ให้ผู้ที่เกี่ยวข้องต่อไป

2.2 **การพัฒนาซอฟต์แวร์แบบ** Relational Unified Process: RUP เป็นแบบจำลองที่ได้รับความนิยมอย่างมาก แบบจำลอง RUP นำแนวคิดเดียวกับแบบจำลอง The Spiral มาผสมผสานกับแบบจำลองที่เกี่ยวข้องกับ UML โดยมีลักษณะเฉพาะของแบบจำลองดังนี้

2.2.1 ลักษณะการพัฒนาแบบวงรอบ และมีการพัฒนาเพิ่ม (Iteration and Incremental)

2.2.2 นำข้อกำหนดของความต้องการซอฟต์แวร์เป็นตัวผลักดันการพัฒนา โดยใช้เครื่องมือ เช่น ยูสเคส (Use case)

2.2.3 ให้ความสำคัญที่สถาปัตยกรรม (Architecture-Centric)

2.2.4 รูปแบบภาษาเชิงโมเดลเป็นเครื่องมือในการพัฒนา (Using Unified Modeling Language: UML)

ระเบียบวิธีการพัฒนาซอฟต์แวร์ตามแนวคิดอาไจล์ (Agile Methodology)

ปี ค.ศ. 2001 ผู้เชี่ยวชาญด้านการพัฒนาซอฟต์แวร์ 17 ท่านนำโดย Kent Beck ได้ประกาศคำแถลงการณ์เรื่องวิธีการพัฒนาซอฟต์แวร์แบบ อาไจล์ (Agile) โดยคำแถลงการณ์มีดังนี้

"Agile Manifesto"

We are uncovering better way of Developing software by doing it and helping others do it.

Though this work we have come to value:

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

ภาพ 10 คำแถลงการณ์เรื่องวิธีการพัฒนาซอฟต์แวร์แบบ อาไจล์ (Agile Manifesto)

ที่มา: ดัดแปลงจาก Kent Bect., et al., n.d.

แนวคิดดังกล่าวมุ่งเน้นการให้ความสำคัญกับบุคคลและการสื่อสารมากกว่ากระบวนการและเครื่องมือ และให้ความสำคัญแก่การทำงานของซอฟต์แวร์ที่ทำงานได้จริงมากกว่าการจัดทำเอกสาร รวมทั้งการให้ความสำคัญแก่การทำงานร่วมกับลูกค้า โดยยอมรับลูกค้ามาเป็นส่วนหนึ่งของทีมมากกว่าข้อตกลงตามสัญญา อีกทั้งยังมุ่งเน้นให้ความสำคัญแก่การยอมรับและตอบสนองต่อการเปลี่ยนแปลงมากกว่าการปฏิบัติตามขั้นตอนหรือแผนงานที่ได้กำหนดไว้ (เมสินี นาคมณี, 2547) และยังมีผู้แสดงความคิดเห็นเกี่ยวกับ วิธีการพัฒนาซอฟต์แวร์ แบบอาไจล์ (Agile) ดังนี้

Ivar Jacobsm อ้างอิงจาก พรฤดี เนติโสภากุล (2549) ได้แสดงความคิดเห็นเกี่ยวกับวิธีการ อาไจล์ (Agile) ดังนี้ "วิธีการ อาไจล์ (Agile) ได้กลายเป็นคำพูดที่นิยมพูดกันในการอธิบายกระบวนการซอฟต์แวร์สมัยใหม่ วิธีการทุกอย่างกลายเป็นแบบ อาไจล์ (Agile) ทีมอาไจล์ (Agile) เป็นทีมเล็กๆ ที่สามารถตอบสนองต่อการเปลี่ยนแปลง การเปลี่ยนแปลงเป็นทุกๆ สิ่งของการพัฒนาซอฟต์แวร์ การเปลี่ยนแปลงของตัวซอฟต์แวร์ที่ถูกสร้างขึ้น การเปลี่ยนแปลงตัวทีมงาน การเปลี่ยนแปลงไปสู่เทคโนโลยีใหม่ๆ การเปลี่ยนแปลงทุกๆ ชนิดที่อาจมีผลต่อผลิตภัณฑ์ หรือต่อโครงการที่สร้างผลิตภัณฑ์ ควรจะต้องมีกลไกรองรับการเปลี่ยนแปลงแบบฝังตัวในทุกสิ่งทุกอย่างที่เราทำกับซอฟต์แวร์กลไกบางอย่างที่เป็นจิตวิญญาณของซอฟต์แวร์ ทีมงาน อาไจล์ (Agile) ความระลึกไว้ว่า ซอฟต์แวร์ถูกพัฒนาโดยบุคคลอิสระที่ทำงานในทีม ความเชี่ยวชาญของบุคคลเหล่านี้ รวมทั้งความร่วมมือ เป็นหัวใจหลักของความสำเร็จของโครงการ"

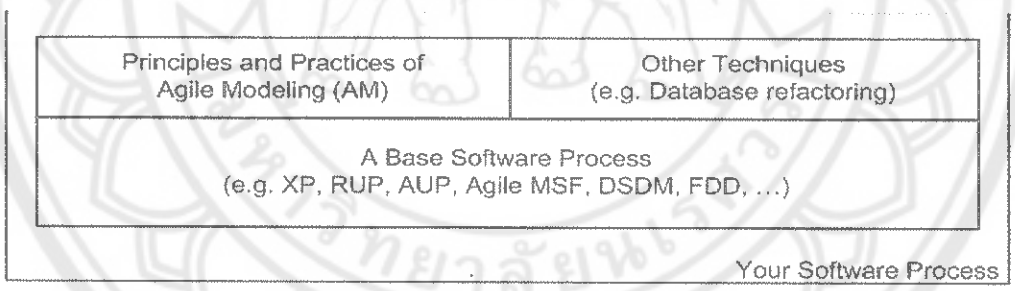
Roger S. Pressman (2006) ได้แสดงความคิดเห็นเพิ่มเติมเกี่ยวกับวิธีการ อาไจล์ (Agile) ดังนี้ "ความคิดเห็นข้างต้นแสดงว่า ความเปลี่ยนแปลงเป็นแรงผลักดันของวิธีการ อาไจล์ (Agile) วิศวกรรมซอฟต์แวร์ต้องไวต่อการเปลี่ยนแปลงและรองรับความเปลี่ยนแปลงอย่างรวดเร็วได้ แต่วิธีการ อาไจล์ (Agile) มีอะไรมากกว่าการรองรับการเปลี่ยนแปลง มันยังรวมถึงปรัชญาที่กล่าวถึงในแถลงการณ์ข้างต้นบทนี้ ซึ่งรวมถึงโครงสร้างของทีมงาน ทัศนคติที่ต้องสื่อสารระหว่างกัน ระหว่างนักเทคนิคและนักธุรกิจ ระหว่างวิศวกรกับผู้จัดการ เน้นการส่งมอบซอฟต์แวร์ที่ทำงานได้อย่างรวดเร็ว และลดความสำคัญของผลิตผลตัวกลาง วิธีการนี้รับลูกค้ามาเป็นส่วนหนึ่งของทีมงาน ไม่มีการแบ่งแยกเขาและเราและยอมรับว่า การวางแผนที่สมบูรณ์แบบอาจทำไม่ได้ในโลกจริง และเน้นการวางแผนที่ยืดหยุ่นแทน"

เมสินี นาคมณี (2547) กล่าวถึง แนวความคิดเกี่ยวกับ อาไจล์ (Agile) ดังนี้ "อาไจล์ (Agile) คือ การที่จะทำอย่างไรให้กระบวนการที่เกิดขึ้นนั้นเป็นกระบวนการที่สามารถทำได้อย่างยืดหยุ่น และคล่องตัวสามารถแก้ปัญหาเฉพาะหน้าได้เป็นอย่างดี โดยมีความรวดเร็วและยืดหยุ่นในการทำงานสูงขึ้นหรือทำให้การทำงานนั้นง่ายขึ้นนั่นเอง"

จากแนวคิดของผู้เชี่ยวชาญหลายท่านที่กล่าวในข้างต้นจึงสามารถนำมาสรุปแนวคิดของเทคนิคการพัฒนาซอฟต์แวร์แบบ อาไจล์ (Agile) ได้ดังต่อไปนี้ วิธีการ อาไจล์ (Agile) เป็นวิธีการพัฒนาซอฟต์แวร์ที่สามารถรองรับการเปลี่ยนแปลงที่อาจเกิดขึ้นจากสภาวการณ์ภายนอกได้อย่างรวดเร็วและยืดหยุ่นโดยไม่ยึดติดกับขั้นตอนการทำงานแบบตายตัว

การประยุกต์ใช้ อาไจล์ (Agile Modeling)

อาไจล์ (Agile Modeling: AM) เป็นหลักการพื้นฐานของวิธีการปฏิบัติสำหรับกระบวนการจัดเตรียมเอกสารและแบบจำลองที่มีประสิทธิภาพของระบบซึ่งเป็นพื้นฐานของซอฟต์แวร์ อาไจล์ (Agile Modeling: AM) เป็นการรวมกลุ่มเกี่ยวกับการประเมินค่าหลักทฤษฎี วิธีปฏิบัติสำหลับการสร้างแบบจำลองซอฟต์แวร์ ซึ่งสามารถนำไปประยุกต์ใช้กับโครงการพัฒนาซอฟต์แวร์อย่างมีประสิทธิผล และการจัดการที่ภาระงานบางเบาลง (Light-weight) ดังภาพ 11 อาไจล์ (Agile Modeling: AM) เป็นตัวกลางเพื่อสร้างหรือทำให้เหมาะสมกับความต้องการที่เฉพาะเจาะจงของแบบจำลองอื่นๆ วิธีการพัฒนาแบบสมบูรณ์ อย่างเช่น XP, RUP โดยสามารถที่จะพัฒนากระบวนการซอฟต์แวร์กับความต้องการที่เหมาะสมของโครงการ (Scott W. Ambler, ไม่ระบุปี)



ภาพ 11 การประยุกต์ใช้แบบจำลองอาไจล์ (Agile) กับกระบวนการพัฒนาซอฟต์แวร์แบบอื่น

ที่มา: Scott W. Ambler., n.d.

หลักการของ อาไจล์ (Agile Principles)

หลักการของ อาไจล์ (Agile) มีที่มามาจากแนวคิดหลักของ อาไจล์ (Agile) ที่มุ่งเน้นการทำงานที่มีความยืดหยุ่นสามารถปรับเปลี่ยนได้ตามสถานการณ์มากกว่าการทำงานตามขั้นตอนที่ได้กำหนดไว้ตามลำดับที่แน่นอน จึงได้มีการกำหนดหลักการพัฒนาซอฟต์แวร์แบบ อาไจล์ (Agile) ขึ้นดังต่อไปนี้

1. มุ่งเน้นความสำคัญกับความพึงพอใจของลูกค้า โดยการที่สามารถจัดส่งซอฟต์แวร์ได้อย่างรวดเร็ว และต่อเนื่อง

2. ส่งมอบซอฟต์แวร์ให้ลูกค้าได้เห็นความคืบหน้าของผลิตภัณฑ์อย่างสม่ำเสมอเป็นระยะเพื่อให้ลูกค้ามีข้อเสนอแนะเกี่ยวกับซอฟต์แวร์

3. ความก้าวหน้าของโครงการวัดจากซอฟต์แวร์ที่สามารถทำงานได้จริง

4. ยอมรับความเปลี่ยนแปลงความต้องการของลูกค้าที่อาจเกิดขึ้นได้ตลอดเวลา

5. สร้างแนวคิดการทำให้ทุกขั้นตอนเป็นเรื่องง่ายเป็นหัวใจหลักของการพัฒนา เพื่อรองรับการเปลี่ยนแปลง

6. มอบหมายสิทธิในการตัดสินใจ และให้ความไว้วางใจในการทำงานเพื่อให้บรรลุวัตถุประสงค์ของการพัฒนา

7. สร้างกระบวนการบริหารจัดการภายในทีมที่ดีจะส่งผลให้ผลการดำเนินการเป็นไปได้ด้วยดี

8. สร้างกระบวนการทำงานที่ทีมพัฒนาแสดงความคิดเห็น แลกเปลี่ยนความคิดภายในทีมเพื่อลดข้อผิดพลาด และเพิ่มเข้าใจที่ตรงกันในการทำงานร่วมกัน

9. สร้างความร่วมมือและแลกเปลี่ยนเครื่องมือ รวมทั้งกำหนดวิธีการต่างๆ โดยคำนึงถึงประสิทธิภาพของการพัฒนาซอฟต์แวร์ เพื่อปรับปรุงการทำงานภายในทีมดีขึ้น

10. พยายามควบคุมการทำงานให้มีความสมดุลและมีความต่อเนื่องในทุกระยะ

11. ยินยอมให้ผู้ที่เกี่ยวข้องทั้งทางด้านผู้ใช้งานผู้พัฒนา และผู้ที่เกี่ยวข้องเข้ามามีส่วนร่วมในการทำงาน โดยมีการพูดคุยหรือประชุมร่วมกันเป็นประจำทุกวัน

12. มุ่งเน้นความสนใจเกี่ยวกับเทคนิคต่างๆ เมื่อมีการค้นพบเทคนิคที่จะทำให้การทำงานง่ายขึ้นควรนำมาแลกเปลี่ยน (Kent Bect., et al., n.d.)

แบบจำลองกระบวนการอาไจล์ (Agile Process Models)

ประวัติศาสตร์วิศวกรรมซอฟต์แวร์ เต็มไปด้วยกรรมวิธีและกระบวนการที่ล้าสมัย โดยมีกระบวนการใหม่ๆ เกิดขึ้นมาตลอด ต่างมีสัญลักษณ์และสิ่งใหม่ ๆ ที่ดีกว่าวิธีการเดิมๆ เส้นทางของอาไจล์ (Agile) ก็เดินตามทางเหล่านี้ ในส่วนนี้จะนำเสนอภาพรวมของแบบจำลองกระบวนการอาไจล์ (Agile) ในแบบต่างๆ

1. **เอ็กซ์ทรีม โปรแกรมมิ่ง-เอ็กซ์พี** (Extreme Programming – XP) แนวคิดและวิธีการเอ็กซ์ทรีมโปรแกรมมิ่ง (เอ็กซ์พี) นิยมใช้แนวทางเชิงวัตถุในการพัฒนาระบบ โดยมีชุดของกฎและข้อปฏิบัติที่ต้องทำระหว่างกิจกรรมกรอบงาน และงานย่อยที่ทำระหว่างแต่ละกิจกรรมดังนี้

1.1 **การวางแผนเพื่อการเล่นเกม** (Planning game) การกำหนดรายละเอียดกิจกรรมที่ต้องทำ และจัดส่งแต่ละการวงรอบ เมื่อกิจกรรมแต่ละวงรอบเสร็จสิ้นแล้วจะวางแผนเพื่อเข้าสู่การดำเนินกิจกรมในวงรอบถัดไป เอ็กซ์ทรีมโปรแกรมมิ่ง (เอ็กซ์พี) พยายามใช้เทคนิคต่างๆ เข้ามาช่วยสำหรับการเข้าถึงความต้องการของผู้ใช้ เช่น การเข้าถึงความต้องการด้วยการสร้างเรื่องราวจากลูกค้า (User stories) เป็นเครื่องมือที่มองภาพรวมของระบบงานจากมุมมองผู้ใช้ ดังนั้นการวางแผนแต่ละครั้งผู้ที่เกี่ยวข้องจะร่วมกันวางแผนเพื่อให้ซอฟต์แวร์ที่ได้มีความเหมาะสม

1.2 **การส่งมอบงานบ่อย** (Small, Frequent Releases) การพัฒนาระบบ และการส่งมอบจะนำเสนอแก่ลูกค้าบ่อย (Small Frequent Releases) โดยค่อยๆ พัฒนาขึ้นเป็นลำดับอย่างต่อเนื่อง เพื่อให้ลูกค้าหรือผู้ใช้เห็นความคืบหน้าของโครงการอย่างต่อเนื่อง และเป็นตัวกระตุ้นให้ลูกค้าบอกความต้องการที่แท้จริงโดยการยินยอมให้ลูกค้าแสดงข้อเสนอแนะเกี่ยวกับซอฟต์แวร์ที่ส่งมอบ ซึ่งแนวคิดนี้มุ่งเน้นคุณค่าของการลงทุนของลูกค้าส่งผลให้ลูกค้าเกิดความพึงพอใจในซอฟต์แวร์ยิ่งขึ้น

1.3 **การสร้างวิสัยทัศน์ในการออกแบบร่วมกัน** (System Metaphors) แนวคิดของการสร้างวิสัยทัศน์ในการออกแบบร่วมกัน (System Metaphors) เสนอให้มีการใช้ตัวกลางมาช่วยในการออกแบบ และพัฒนาระบบ เช่น พยายามซ่อนข้อมูลเชิงเทคนิค โดยนำเสนอข้อมูลที่ง่ายทุกฝ่ายสามารถเข้าใจการนำเสนอเพื่อให้ลูกค้าสามารถเข้าใจการทำงานของระบบ (เมสินี นาคมณี, 2547)

1.4 **การออกแบบง่าย ๆ (Simple Design)** วิธีการ Simple Design จากแนวคิดของการพัฒนาแบบการพัฒนาเพิ่ม (Incremental) และพัฒนาแบบวงรอบ (Iteration) ที่ว่าการพัฒนาซอฟต์แวร์ประกอบด้วยการทำซ้ำหลายๆ รอบโดยแต่ละรอบเป็นแบบการส่งมอบงานทีละน้อย (Small Release) โดยค่อยๆ พัฒนาขึ้นเป็นลำดับอย่างต่อเนื่อง ดังนั้น XP จึงเสนอแนวคิดให้การ

ออกแบบโครงสร้างแบบง่ายๆ เพื่อรองรับการเปลี่ยนแปลงในอนาคต ซึ่งนอกจากจะส่งผลให้การทำงานในแต่ละรอบง่ายแล้วยังช่วยให้ประหยัดเวลาในการพัฒนามากขึ้น เมื่อเทียบกับการออกแบบที่สมบูรณ์และซับซ้อนแต่อาจไม่ได้ใช้ในอนาคต (เมสินี นาคมณี, 2547)

1.5 **การนำผลการทดสอบขับเคลื่อนการพัฒนาซอฟต์แวร์** (Test Driven Development: TDD) การทดสอบคุณภาพของการพัฒนาซอฟต์แวร์ โดยมีแนวคิดที่ว่านำการทดสอบและผลจากทดสอบมาช่วยขับเคลื่อนให้เกิดการพัฒนาซอฟต์แวร์ กล่าวคือ เมื่อมีการแบ่งงานออกเป็นการส่งมอบงานทีละน้อย (Small Release) หลังจากที่งานเสร็จสิ้นอนุญาตให้ได้ทดสอบการใช้งานซอฟต์แวร์และนำผลจากการทดสอบมาใช้ในการวางแผนการทำงานในวงรอบถัดไป (เมสินี นาคมณี, 2547)

1.6 **การปรับปรุงโครงสร้างการโปรแกรม** (Code Refactoring) กระบวนการ Code Refactoring เป็นการพยายามปรับปรุงโครงสร้างการพัฒนาโปรแกรมโดยแยกส่วนของการออกแบบออกจากส่วนงานของการทำงานของซอฟต์แวร์ ทำให้การทำงานมีอิสระมากขึ้น ซึ่งช่วยให้สามารถรองรับการเปลี่ยนแปลงได้ตลอดเวลาและเกิดความยืดหยุ่นในการเปลี่ยนแปลง นอกจากนี้การปรับปรุงโครงสร้างการโปรแกรม (Refactoring) ยังเป็นการจัดโครงสร้างของโค้ด (Code) ใหม่โดยที่ความหมายและพฤติกรรมของโค้ดไม่เปลี่ยนแปลง

1.7 **การพัฒนาโปรแกรมเป็นคู่** (Pair Programming) การพัฒนาโปรแกรมเป็นคู่โดยมีแนวคิดที่ว่านักพัฒนาสลับหน้าที่กันระหว่างพัฒนาโปรแกรม และการตรวจสอบคุณภาพ หรือความถูกต้องของการพัฒนาโปรแกรม เสมือนว่าใช้คอมพิวเตอร์เครื่องเดียวกัน ซึ่งวิธีนี้ช่วยให้ซอฟต์แวร์ที่ได้มีคุณภาพที่ดีและยังถือเป็นการแลกเปลี่ยนความรู้ระหว่างทีมพัฒนา ซึ่งช่วยพัฒนาทักษะภายในทีม และช่วยปรับเปลี่ยนความเสี่ยงในกรณีที่บุคคลากรที่ขาดความเชี่ยวชาญด้านการพัฒนาซอฟต์แวร์ ซึ่งปัญหาดังกล่าวถือได้ว่ามีความสำคัญอย่างมากเนื่องจากอาจทำให้โครงการล้มเหลว รวมทั้งลดความเสี่ยงหากสมาชิกในทีมออกจากการพัฒนาซอฟต์แวร์ ทั้งนี้ในการพัฒนาซอฟต์แวร์แบบการพัฒนาโปรแกรมเป็นคู่ (Pair Programming) นี้ อาจมีความเสี่ยงหากนักพัฒนามีความแตกต่างกันมาก หรือมีความขัดแย้งกัน (เมสินี นาคมณี, 2547)

1.8 **การสร้างการมีส่วนร่วมในการพัฒนา** (Collective Code Ownership) แนวคิดของการสร้างการมีส่วนร่วมในการพัฒนา (Collective Code Ownership) คือ การเห็นความสำคัญของการทำงานร่วมกัน โดยพยายามทำให้สมาชิกในทีมพัฒนาทุกคนมีความรับผิดชอบร่วมกัน ผู้พัฒนาทุกคนสามารถเข้าถึงโค้ด (Code) และช่วยในการแลกเปลี่ยนความรู้ภายในทีมอีกด้วย ทั้งนี้เพื่อให้โค้ดที่พัฒนาขึ้นเป็นไปในทิศทางเดียวกัน (เมสินี นาคมณี, 2547)

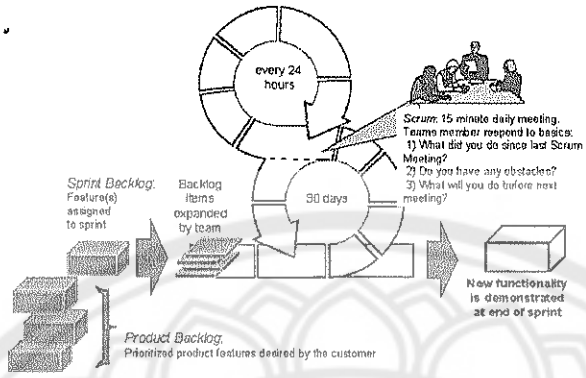
1.9 **การเชื่อมต่อระบบอย่างต่อเนื่อง** (Continuous Integration) การเชื่อมต่อระบบอย่างต่อเนื่องเป็นผลมาจากการแบ่งงานออกเป็นการส่งมอบงานทีละน้อย (Small Release) หลังจากที่นักพัฒนาทำงานในส่วนที่ได้รับมอบหมายเสร็จและผ่านการทดสอบย่อยแล้วควรมีการเชื่อมต่ออย่างสม่ำเสมอเพื่อลดปัญหาจากการที่แต่ละส่วนงานยังไม่สมบูรณ์ ซึ่งอาจส่งผลให้เสียเวลาในการปรับแก้สำหรับกรณีที่ทำการเชื่อมต่อกับส่วนอื่นๆ หลังจากทุกส่วนเสร็จสิ้นในครั้งเดียว (เมสินี นาคมณี, 2547)

1.10 **การสร้างมาตรฐานในการพัฒนาโปรแกรม** (Coding Standards) การใช้มาตรฐานในการพัฒนาโปรแกรมแบบเดียวกัน ซึ่งช่วยให้การสื่อสารระหว่างทีมพัฒนาง่ายขึ้น และช่วยในการนำโค้ด (Code) เดิมมาปรับปรุงหรือนำกลับมาใช้ เช่น การจัดกลุ่มคลาส (Class) หรือการตั้งชื่อ (Name Space) (เมสินี นาคมณี, 2547)

2. **สครัม** (Scrum) ถูกพัฒนาขึ้นเมื่อปี 1995 โดย Ken Schwaber ได้แนวคิดจากกีฬารักบี้ โดยมองกระบวนการพัฒนาซอฟต์แวร์เสมือนการเล่นเกมรักบี้ ซึ่งมีการรับ และส่งลูกตลอดเวลา เสมือนการทำงานร่วมกันกับคนหลายๆ ฝ่าย ที่ต้องทำงานสอดคล้องกัน ซึ่งหลักการสครัม (Scrum) มีลักษณะเป็นกระบวนการที่เรียกว่า light-Weight Management Process ที่เน้นการบริหารจัดการมากกว่าขั้นตอนตายตัวตามลักษณะวิศวกร โดยลักษณะการทำงานแบ่งออกเป็น 2 ส่วนหลักดังนี้

2.1 **สครัม** (Scrum) การทำงานร่วมกันของทีมพัฒนา โดยมีการวิเคราะห์ปัญหาที่เกิดขึ้นระหว่างพัฒนาร่วมกันระหว่างพัฒนา

2.2 **สปริ้น** (Sprint) ส่วนของการทำงานแต่ละวงรอบ โดยยินยอมให้ผู้ที่เกี่ยวข้องกับโครงการ เข้ามามีส่วนร่วมในการพิจารณาความก้าวหน้า และการส่งมอบงานแต่ละวงรอบ เพื่อผลักดันการพัฒนาวงรอบถัดไป (เมสินี นาคมณี, 2547)



ภาพ 12 การพัฒนาซอฟต์แวร์แบบ Scrum

ที่มา: Roger S. Pressman, 2005

3. **การพัฒนาซอฟต์แวร์แบบปรับตัว-เอเอสดี** (Adaptive Software Development -ASD) การพัฒนาซอฟต์แวร์แบบปรับตัว (เอเอสดี) ถูกนำเสนอโดย Jim Highsmith สร้างขึ้นเพื่อสนับสนุนการพัฒนาระบบที่ซับซ้อน ให้ความสำคัญกับการทำงานร่วมกันระหว่างบุคคล และการจัดระเบียบทีมงาน โดยลักษณะการทำงานแบ่งออกเป็น 3 ส่วนหลักดังนี้

3.1 **การคาดเดา** (Speculation) เริ่มต้นการวางแผนโดยใช้ข้อมูลเบื้องต้น ได้แก่ ข้อมูลเกี่ยวกับภารกิจของลูกค้า เงื่อนไขต่างๆ ของโครงการและความต้องการพื้นฐาน เพื่อกำหนดชุดของวงจรการวงรอบสำหรับซอฟต์แวร์ที่โครงการจะต้องผลิต

3.2 **การร่วมมือ** (Collaboration) การร่วมมือกันระหว่างผู้ที่เกี่ยวข้องเป็นวิธีที่ใช้ในทุกๆ วิธีการแบบอาไจล์ สำหรับการพัฒนาซอฟต์แวร์แบบปรับตัว (เอเอสดี) ให้ความสำคัญกับการทำงานร่วมกัน ดังนี้ 1) การวิพากษ์วิจารณ์ชิ้นงาน 2) การช่วยเหลือกัน 3) การกระจายงานอย่างเท่าเทียมกัน 4) สร้างความชำนาญเฉพาะที่จะเป็นประโยชน์ต่องาน 5) มีการพูดคุยถึงปัญหาหรือข้อกังวลสงสัยเพื่อนำไปสู่การทำงานที่ได้ประสิทธิผล

3.3 **การเรียนรู้** (Learning) High smith อ้างว่าทีมพัฒนาซอฟต์แวร์มักจะประมาณความเข้าใจของตนเองด้านเทคโนโลยีและด้านอื่นๆ มากเกินจริงดังนั้นสมาชิกของทีมเอเอสดีเริ่มพัฒนาขึ้นส่วนที่เป็นส่วนหนึ่งของวงจรปรับตัว ทีมพัฒนาซอฟต์แวร์ควรเรียนรู้ไปพร้อมๆ กับความก้าวหน้าของกระบวนการพัฒนา โดยกระบวนการเรียนรู้มี 3 วิธีคือ

3.3.1 กลุ่มเฉพาะทาง (Focus Groups) เมื่อลูกค้า/ผู้ใช้งานให้ผลตอบกลับในการใช้งาน ในแต่ละรุ่นของซอฟต์แวร์ที่ส่งมอบไป สิ่งนี้จะเป็นตัวบ่งชี้ว่าผลิตภัณฑ์ได้ตอบสนองความจำเป็นทางธุรกิจหรือไม่

3.3.2 การทบทวนทางเทคนิคอย่างเป็นทางการ (Formal Technical Review) ทีมเอเอสดีมีการทบทวนชิ้นส่วนซอฟต์แวร์ที่กำลังพัฒนาอยู่ ขณะเดียวกันมีการปรับปรุงคุณภาพและเรียนรู้ไปพร้อมๆ กัน

3.3.3 การตรวจสอบภายหลัง (Postmortems) เมื่อเสร็จสิ้นงานแล้ว ทีมงานเอเอสดีกลายเป็นผู้วิจารณ์งานและคุณภาพของตนเอง ซึ่งเป็นเป้าหมายของการเรียนรู้เพื่อปรับปรุงวิธีการทำงาน (พรฤดี เนติโสภากุล, 2549)

4. **วิธีการพัฒนาระบบไม่หยุดนิ่ง-ดีเอสดีเอ็ม** (Dynamic System Development Method-DSDM) วิธีการพัฒนาระบบแบบไม่หยุดนิ่ง (ดีเอสดีเอ็ม) เป็นการพัฒนาซอฟต์แวร์อาไจล์ที่มีการกำหนดกรอบงานสำหรับการพัฒนาโดยมีข้อจำกัดด้านเวลา ซึ่งใช้การพัฒนาต้นแบบและพัฒนาเพิ่มขึ้น ในสิ่งแวดล้อมโครงการที่มีการควบคุม โดยพยายามพัฒนาชิ้นงาน 80 เปอร์เซ็นต์ของแอพพลิเคชั่นให้เสร็จภายในเวลา 20 เปอร์เซ็นต์ ของเวลาทั้งหมดที่ใช้ในการพัฒนา

ดีเอสดีเอ็มเป็นกระบวนการวงรอบ (Iteration) เช่นเดียวกับเอ็กซ์พีและเอเอสดี มีความแตกต่างตรงที่แต่ละรอบของวงจรดีเอสดีเอ็ม จะเป็นไปตามกฎ 80 เปอร์เซ็นต์ คือ ทำงานให้เท่าที่จำเป็นในแต่ละวงรอบ เพื่อให้เคลื่อนไปสู่วงรอบถัดไป ส่วนรายละเอียดที่เหลือสามารถทำให้เสร็จได้ภายหลัง เมื่อทราบความต้องการทางธุรกิจเพิ่มเติม หรือเมื่อได้รับการร้องขอให้เปลี่ยนแปลง โดยลักษณะการทำงานแบ่งออกเป็น 5 ส่วนหลักดังนี้

4.1 **การศึกษาความเป็นไปได้** (Feasibility Study) สร้างรายการความต้องการและข้อกำหนดทางธุรกิจพื้นฐานของแอพพลิเคชั่นที่จะสร้าง ซึ่งมีการประเมินว่าแอพพลิเคชั่นมีความเป็นไปได้ สำหรับกระบวนการดีเอสดีเอ็มหรือไม่

4.2 **การศึกษาด้านธุรกิจ** (Business Study) จัดสร้างความต้องการเชิงข่าวสาร และเชิงหน้าที่ของแอพพลิเคชั่น นิยามสถาปัตยกรรมและระบุความต้องการในการบำรุงรักษาแอพพลิเคชั่น

4.3 **การทำวนซ้ำแบบจำลองเชิงหน้าที่** (Function Model Iteration) พัฒนาต้นแบบและพัฒนาเพิ่มขึ้น และวิวัฒนาการไปเป็นแอพพลิเคชั่นที่ส่งมอบได้ การทำวนซ้ำเป็นการรวบรวมความต้องการเพิ่มเติมจากการตอบสนองกลับมาของผู้ใช้

4.4 **การทำวนซ้ำการออกแบบและการสร้าง** (Design and Build Iteration) ต้นแบบที่พัฒนาขึ้นระหว่างกระบวนการวนซ้ำการสร้างแบบจำลองเชิงหน้าที่ เพื่อให้มั่นใจว่าซอฟต์แวร์

สามารถทำงานได้จริง บางกรณี การทำวนซ้ำการสร้างแบบจำลองเชิงหน้าที่และการทำวนซ้ำการออกแบบและการสร้างเกิดขึ้นพร้อมๆ กัน

4.5 **การอิมพลีเมนต์** (Implementation) การใช้งานซอฟต์แวร์ภายใต้สิ่งแวดล้อมการทำงานจริง 1) ซอฟต์แวร์อาจไม่ต้องสมบูรณ์ร้อยเปอร์เซ็นต์ หรือ 2) การร้องขอการเปลี่ยนแปลงอาจเกิดขึ้นได้ขณะใช้งานอยู่ ในกรณีใดกรณีหนึ่งการพัฒนาดีเอสดีเอ็มจะดำเนินต่อเนื่องไป โดยย้อนกลับมาทำกิจกรรมการวนซ้ำ (พรฤดี เนติโสภากุล, 2549)

5. **คริสตัล** (Crystal) Alistair Cockburn และ Jim Highsmith ได้สร้างแบบจำลอง คลิสตัล (Crytal) ของวิธีการอาไจล์ (Agile) เพื่อให้ได้มาซึ่งวิธีการพัฒนาซอฟต์แวร์ที่เน้นความสามารถในการพัฒนาแบบนำร่อง สำหรับโครงการที่มีทรัพยากรจำกัด และต้องร่วมมือกัน ในการการสื่อสารโดยมีเป้าหมายหลัก คือ ส่งมอบซอฟต์แวร์ที่มีประโยชน์ทำงานได้ และเป้าหมายรองคือ จัดเตรียมความพร้อมสำหรับการวงรอบถัดไป (พรฤดี เนติโสภากุล, 2549)

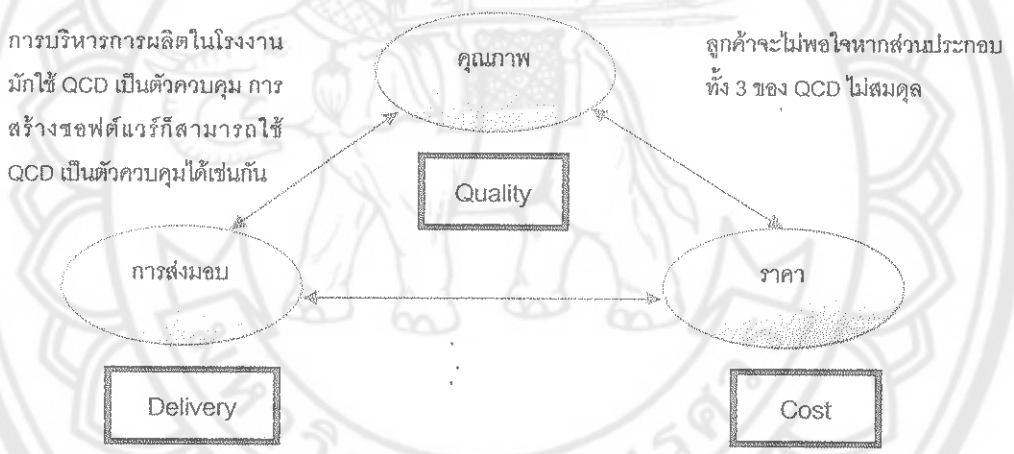
6. **การพัฒนาที่ขับเคลื่อนด้วยคุณลักษณะของซอฟต์แวร์-เอฟดีดี** (Feature Driven Development-FDD) การพัฒนาที่ขับเคลื่อนด้วยคุณลักษณะของซอฟต์แวร์ (เอฟดีดี) เริ่มขึ้นจากแนวคิดของ Peter Coad และคณะ เพื่อสร้างแบบจำลองกระบวนการเชิงปฏิบัติการของวิศวกรรมซอฟต์แวร์เชิงวัตถุ จากนั้น Stephen Palmer และ John Felsing ได้พัฒนาเพิ่มเติมงานของ Coad เพื่อประยุกต์กับโครงการขนาดขนาดใหญ่ขึ้น

เนื่องจากคุณลักษณะเป็นส่วนเล็กๆ ของซอฟต์แวร์ที่ทำงานได้ จึงสามารถอธิบายโครงสร้างและความสัมพันธ์ระหว่างกันได้ง่ายกว่า อีกทั้งยังสามารถทบทวนได้ดีกว่าหากเกิดข้อผิดพลาดจึงง่ายต่อการตรวจทานอย่างละเอียด

เอฟดีดีให้รายละเอียดและเทคนิคของการจัดการโครงการมากกว่าวิธีอาไจล์อื่นๆ เมื่อโครงการมีความซับซ้อนมากขึ้น การจัดการโครงการเฉพาะกิจอาจไม่เพียงพอ จึงจำเป็นที่ผู้ที่เกี่ยวข้องต้องเข้าใจสถานะของโครงการ ว่าได้บรรลุเป้าหมายอะไรไปแล้วบ้าง และมีปัญหาอะไรบ้างที่พบ กิจกรรมการพัฒนาระบบแบ่งเป็น "การออกแบบคร่าวๆ การออกแบบ การตรวจทานการออกแบบ การโค้ด การตรวจทานการโค้ด และการส่งเสริมการสร้าง" (พรฤดี เนติโสภากุล, 2549) การเปรียบเทียบแนวปฏิบัติหลักกับแถลงการณ์ของอาไจล์ (Agile)

ประสิทธิภาพของโครงการ (Efficiency of Project)

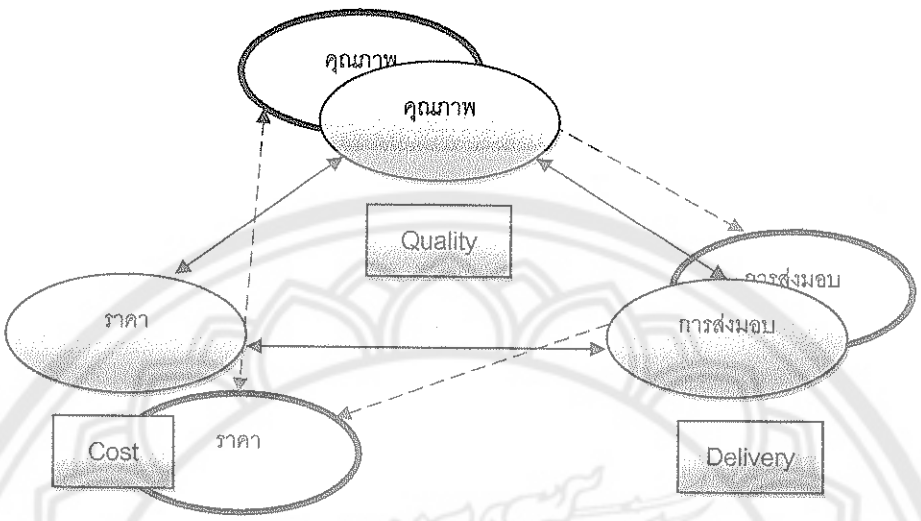
ประสิทธิภาพของโครงการ (Efficiency of Program) หมายถึง การดำเนินกิจกรรมของโครงการนั้นๆ บรรลุตามวัตถุประสงค์ที่กำหนดไว้ โดยได้รับผลตอบแทนที่คุ้มค่า ภายใต้การใช้ทรัพยากรในการดำเนินงานน้อยที่สุด พิจารณาถึงประสิทธิภาพของการบริหารการพัฒนาซอฟต์แวร์ประกอบการด้วย การควบคุมความคืบหน้า การควบคุมคุณภาพ และการควบคุมค่าใช้จ่าย เรียกการพัฒนาซอฟต์แวร์นี้ว่า QCD (Quality, Cost, Delivery) ซึ่งหลักการบริหารซอฟต์แวร์ผู้พัฒนาต้องการให้ซอฟต์แวร์ที่พัฒนาขึ้นสามารถรองรับความต้องการทั้ง 3 ด้าน ที่กล่าวมาข้างต้น จึงจำเป็นต้องทำ QCD ให้สมดุล ดังภาพ 13



ภาพ 13 หลักการบริหารซอฟต์แวร์

ที่มา: ดัดแปลงจาก Roger S. Pressman, 2005

เป้าหมายของการบริหารซอฟต์แวร์แบบ QCD ได้จากการทำความเข้าใจระหว่างผู้ที่เกี่ยวข้องทุกฝ่ายจากนั้นทำความเข้าใจเกี่ยวกับผลิตภัณฑ์ที่พัฒนาขึ้นและกำหนดข้อตกลงรวมกันเพื่อลดปัญหาที่อาจเกิดขึ้นตามมา ถึงแม้ว่าแต่ละฝ่ายจะพยายามเข้าใจทำความเข้าใจเกี่ยวกับผลิตภัณฑ์ที่พัฒนาขึ้น แต่อาจเกิดความขัดแย้งกัน เนื่องจากแต่ละฝ่ายมีเป้าหมายแตกต่างกันตามหน้าที่ของตน ดังนั้นจึงต้องมีการประชุมแต่ละฝ่ายเพื่อกำหนดความสำคัญของงานที่จะทำเพื่อให้ปัญหาที่อาจเกิดขึ้นลดน้อยลง ดังภาพ 14 (Roger S. Pressman, 2005)



ภาพ 14 ความคาดเคลื่อนหลักการบริหารซอฟต์แวร์

ที่มา: ดัดแปลงจาก Roger S. Pressman, 2005

แบบจำลองพื้นฐานของสถาปัตยกรรม และวิศวกรรมซอฟต์แวร์ (Model-Based (System) Architecting and Software Engineering)

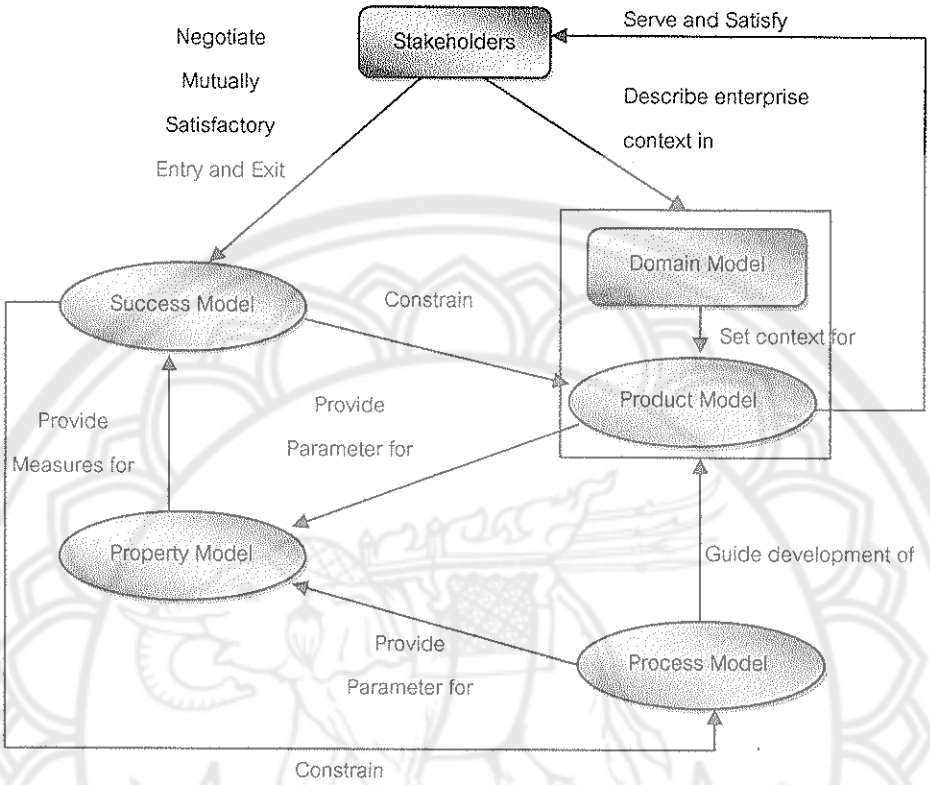
แนวคิด Model-Based (System) Architecting and Software Engineering กล่าวถึงความประสบความสำเร็จ หรือล้มเหลวของโครงการหนึ่งๆ เกิดจากความต้องการของผู้ที่เกี่ยวข้องในระบบมีความสอดคล้อง หรือขัดแย้งกัน ประกอบด้วยหลัก 4 ประการดังนี้

1. โมเดลความสำเร็จ (Success Model) ผู้ที่เกี่ยวข้องตั้งเป้าหมายความสำเร็จไว้แตกต่างกัน

2. โมเดลผลิตภัณฑ์ (Product Model) ผู้ที่เกี่ยวข้องตั้งเป้าหมายเกี่ยวกับลักษณะของผลิตภัณฑ์ไว้แตกต่างกัน

3. โมเดลกระบวนการ (Process Model) ลักษณะกิจกรรมภายในกระบวนการพัฒนาซอฟต์แวร์ สำหรับผู้ที่เกี่ยวข้องกับโครงการอาจมีความแตกต่างกัน

4. โมเดลคุณลักษณะซอฟต์แวร์ (Property Model) ผู้ที่เกี่ยวข้องตั้งเป้าหมายเกี่ยวกับคุณสมบัติของระบบงาน หรือซอฟต์แวร์ต่างกัน (เมสินี นาคมณี, 2547)



ภาพ 15 Model-Based (System) Architecting and Software Engineering

ที่มา: ดัดแปลงจาก เมสินี นาคมณี, 2547

แนวคิดการประเมินผลโครงกาของสตัฟเฟิลบีม (Stufflebeam's CIPP Model)

แบบจำลอง (Model) หมายถึง วิธีการสื่อสารทางความคิด ความเข้าใจ ตลอดจนจินตนาการที่มีต่อเรื่องราวใดๆ ให้ปรากฏโดยใช้การสื่อในลักษณะต่างๆ เช่น แผนผังระบบสมการ แผนภูมิ เป็นต้น เพื่อให้สามารถนำเสนอปรากฏการณ์ต่างๆ ได้อย่างมีระบบ การประเมินผลโครงการนั้น มีแนวคิดและแบบจำลองหลายอย่าง ณ ที่นี้ ขอเสนอแนวคิดและแบบจำลองการประเมินแบบซิป์ป์ หรือ CIPP Model ของสตัฟเฟิลบีม (Danial. L. Stufflebeam) เนื่องจากเป็นแบบจำลองที่ได้รับการยอมรับกันทั่วไปในปัจจุบัน

แนวคิดการประเมินของสตัฟเฟิลบีม (Stufflebeam's CIPP Model) เกิดขึ้นเมื่อปี ค.ศ. 1971 สตัฟเฟิลบีม และคณะ ได้เผยแพร่หนังสือทางการประเมิน ชื่อ "Educational Evaluation and decision Making" โดยเป็นที่ยอมรับกันอย่างกว้างขวาง เนื่องจากแนวคิดและวิธีการทางการวัด

และประเมินผลมีความน่าสนใจอย่างมาก จึงกล่าวได้ว่า สตัฟเฟิลบีมเป็นผู้มีบทบาทสำคัญในการพัฒนาทฤษฎีการประเมินเรียกว่า CIPP Model ซึ่งเป็นวิธีการประเมินกระบวนการอย่างต่อเนื่อง สามารถใช้ควบคู่กับการบริหารโครงการ เพื่อหาข้อมูลประกอบการตัดสินใจ ซึ่งแบ่งประเด็นการประเมินผลออกเป็น 4 ประเภท (สมคิด พรมจุ้ย, 2544)

1. **การประเมินด้านบริบทหรือสภาวะแวดล้อม** (Context Evaluation: C) การประเมินสภาวะแวดล้อมช่วยสนับสนุนการตัดสินเกี่ยวกับปัจจัยทางสภาพแวดล้อมของโครงการที่จะช่วยให้โครงการบรรลุเป้าหมายและตรวจสอบความเหมาะสมของโครงการว่าสอดคล้องกับความต้องการที่แท้จริงหรือไม่ รวมทั้งตรวจสอบความชัดเจนของวัตถุประสงค์ของโครงการว่ามีความสอดคล้องกับนโยบายขององค์กรหรือไม่ (สุพักตร์ พิบูลย์, 2544)

2. **การประเมินปัจจัยเบื้องต้นหรือปัจจัยป้อน** (Input Evaluation: I) การประเมินเพื่อพิจารณา ความเป็นไปได้ของโครงการอีกทั้งยังประเมินทรัพยากรที่จะใช้ในการดำเนินโครงการ เช่น บุคลากร วัสดุอุปกรณ์ งบประมาณ เวลา รวมทั้งเทคโนโลยีและแผนการดำเนินงาน เป็นต้น (สุพักตร์ พิบูลย์, 2544)

3. **การประเมินกระบวนการ** (Process Evaluation: P) การประเมินระหว่างการดำเนินงานโครงการ เพื่อหาข้อบกพร่องของการดำเนินโครงการ สำหรับใช้เป็นข้อมูลในการพัฒนาแก้ไข ปรับปรุง ให้การดำเนินการช่วงต่อไปมีประสิทธิภาพมากขึ้น และตรวจสอบกิจกรรม เวลา ภาวะผู้นำ การมีส่วนร่วมของผู้ที่เกี่ยวข้องในโครงการทรัพยากรที่ใช้ในโครงการ

การประเมินกระบวนการมีจุดมุ่งหมาย คือ

3.1 เพื่อตรวจสอบข้อบกพร่อง ระหว่างที่มีการปฏิบัติการ หรือการดำเนินงานตามแผน

3.2 เพื่อจัดเก็บข้อมูลต่างๆ สำหรับใช้ในการตัดสินใจเกี่ยวกับการดำเนินงานของโครงการ

3.3 เพื่อรวบรวมข้อมูลต่างๆ ที่ได้รับจากการดำเนินงานของโครงการ (สุพักตร์ พิบูลย์, 2544)

4. **การประเมินผลผลิต** (Product Evaluation: P) การประเมินเพื่อเปรียบเทียบผลลัพธ์ที่เกิดขึ้นกับวัตถุประสงค์ของโครงการ หรือเป้าหมายที่กำหนดไว้ รวมทั้งการประเมินผล เรื่องผลลัพธ์ (Outcomes) และผลกระทบ (Impact) ของโครงการ โดยนำข้อมูลจากการประเมินสภาวะแวดล้อม และปัจจัยเบื้องต้นและกระบวนการร่วมด้วย (สุพักตร์ พิบูลย์, 2544)

งานวิจัยที่เกี่ยวข้อง

นับตั้งแต่นักวิศวกรรมซอฟต์แวร์ได้นำเสนอแนวคิดการพัฒนาซอฟต์แวร์ด้วยวิธีการอาไจล์ (Agile) กลายเป็นหัวข้อที่มีการถกเถียงกันอย่างมากเกี่ยวกับประโยชน์ของการประยุกต์ใช้เทคนิคการพัฒนาซอฟต์แวร์แบบ อาไจล์ (Agile) ผู้พัฒนาและนักวิจัย ได้อ้างเหตุผลสนับสนุนเกี่ยวกับผลดีที่จะได้จากการพัฒนาด้วยวิธีการอาไจล์ (Agile) และมีข้อโต้แย้งเกี่ยวกับอาไจล์ (Agile) ผู้สนับสนุนเทคนิคการพัฒนาซอฟต์แวร์แบบ อาไจล์ (Agile) แนะนำให้มีการประยุกต์ใช้แนวคิดของอาไจล์ (Agile) กับการวางแผนงานและพัฒนาระบบ เนื่องจากมีการปรับเปลี่ยนระบบให้เข้ากับความต้องการที่เปลี่ยนแปลงได้ง่าย และถือเป็นเครื่องมือที่ช่วยในการประสานงานระหว่างลูกค้ากับนักพัฒนาระบบ ซึ่งช่วยให้สามารถส่งงานได้ตามภายใต้เงื่อนไขของเวลาและงบประมาณที่จำกัด และ ช่วยให้สามารถเข้าถึงความต้องการที่ปรับเปลี่ยนอย่างรวดเร็วของลูกค้า ส่งผลให้ซอฟต์แวร์ที่พัฒนาขึ้นเป็นไปตามมาตรฐานความต้องการของลูกค้า

Ming Huo, June Verner, Liming Zhu, and Muhammad Ali Babar (2004) ได้ทำการศึกษาเกี่ยวกับคุณภาพของซอฟต์แวร์และวิธีการ อาไจล์ (Agile) โดยมีวัตถุประสงค์ของงานวิจัย คือ เปรียบเทียบแบบจำลองน้ำตก (Waterfall Model) กับแบบจำลองอาไจล์ (Agile) ภายใต้ความกดดันด้วยเงื่อนไขของเวลา และความต้องการที่เปลี่ยนแปลงเสมอๆ จากผลการวิจัยพิสูจน์ได้ว่าวิธีการอาไจล์ (Agile) ช่วยให้คุณภาพการพัฒนาซอฟต์แวร์ดีกว่า Non-Agile เนื่องจาก

1. วิธีการอาไจล์ (Agile) มีแนวปฏิบัติหลักซึ่งช่วยให้เกิดการรับรองคุณภาพอยู่ในขั้นตอนการพัฒนา และขั้นตอนอื่นๆ

2. วิธีการรับรองคุณภาพของอาไจล์ (Agile) ปรากฏขึ้นบ่อยกว่าการพัฒนาแบบ waterfall

3. วิธีการรับรองคุณภาพของซอฟต์แวร์ด้วยการพัฒนาแบบ อาไจล์ (Agile) สามารถที่จะใช้ประโยชน์ได้อย่างแท้จริงในขั้นตอนของกระบวนการในช่วงต้น เนื่องจาก คุณลักษณะของกระบวนการอาไจล์ (Agile)

หากพิจารณาถึงการวิจัยของ Ming Huo, June Verner, Liming Zhu, and Muhammad Ali Babar (2004) พบว่า งานวิจัยมีสนับสนุนแนวคิดการพัฒนาซอฟต์แวร์แบบอาไจล์ (Agile) เนื่องจากผลงานวิจัยชี้ให้เห็นว่าแนวคิดการพัฒนาซอฟต์แวร์แบบอาไจล์ (Agile) ช่วยให้กระบวนการพัฒนาซอฟต์แวร์มีคุณภาพเร็วขึ้นและเป็นเครื่องมือที่ช่วยให้ความต้องการของการพัฒนาซอฟต์แวร์แน่นอนขึ้น ซึ่งต่างจากการศึกษาครั้งนี้มีจุดมุ่งหมายเปรียบเทียบวิธีการพัฒนาซอต์ฟแวร์แบบไม่ใช่อาไจล์ (Non-Agile Method) และ แนวคิดของ อาไจล์ (Agile) ในไทย เพื่อ

1. มิติด้านวงจรเวลาในการพัฒนา (Cycle time)
2. มิติด้านการบริหารงบประมาณ (Cost and benefit Effective)
3. มิติด้านคุณภาพของกระบวนการพัฒนาซอฟต์แวร์ (Quality of process)
4. มิติด้านการทำงานร่วมกันของผู้ที่เกี่ยวข้องกับโครงการพัฒนาซอฟต์แวร์ (Inter-supplier performance)

Brian Henderson-Sellers (2000) ได้วิจัยเกี่ยวกับการพัฒนาโครงการพัฒนาซอฟต์แวร์แบบเน้นการทำงานร่วมกันไว้ดังนี้ โดยสร้างเป้าหมายหลักของการพัฒนาในรูปแบบที่เน้นการทำงานร่วมกัน และมีการเปิดเผยเป็นสิ่งที่ทำให้เกิดประโยชน์ด้านมาตรฐานกระบวนการพัฒนาซอฟต์แวร์ซึ่งมีการทำงานร่วมกัน ประกอบด้วย การปฏิบัติงานทางธุรกิจทำได้อย่างไร การพัฒนาในรูปแบบที่เน้นการทำงานร่วมกัน และมีการเปิดกว้าง มีหลายส่วน: กระบวนการ แบบจำลองการบริหารจัดการ การวัด และด้วยเหตุนี้ มี 3 ระดับของกระบวนการในการพัฒนาในรูปแบบที่เน้นการทำงานร่วมกัน กระบวนการวิศวกรรมซอฟต์แวร์ และ ธุรกิจถูกให้ความสนใจไปที่วงจรการพัฒนาผลิตภัณฑ์ แบบจำลองกระบวนการช่วยให้ระบุว่าเหตุการณ์เปลี่ยนแปลงไปกับเวลาเช่นไร และผลิตภัณฑ์ควรจะถูกพัฒนาให้เป็นผลสำเร็จอย่างไร และเมื่อไรที่แบบจำลองกระบวนการเป็นส่วนประกอบสำคัญของวิธีการ ที่อยู่บนพื้นฐานของความเป็นจริง

กระบวนการวิศวกรรมซอฟต์แวร์ประกอบเข้ากับส่วนประกอบของวิธีการในสภาพแวดล้อมที่เกี่ยวข้องกับคนเพียง 1 คน หรือมากกว่านั้นกับทีมพัฒนา และคำนึงถึงมาตรฐานวัฒนธรรมองค์กรและความสามารถในการใช้ประโยชน์จากเทคโนโลยี ตัวอย่างเช่น การใช้วิธีการแบบใช้อำนาจเผด็จการในองค์กรเกี่ยวกับวิทยาลัยสามารถที่จะพบกับความเหตุการณ์ที่ร้ายแรงได้ ถ้าการดำเนินการซอฟต์แวร์ล้มเหลว เช่นเดียวกับเตรียมโครงการไปจนถึงผลิตภัณฑ์

ส่วนประกอบของกระบวนการที่สาม เป็นส่วนประกอบภายนอกของขอบเขตการพัฒนาซอฟต์แวร์ มันอธิบายถึงธุรกิจว่าทำงานเช่นไร การประเมินโอกาสและอุปสรรคทางธุรกิจลงไปในกระบวนการวิศวกรรมซอฟต์แวร์ และการจัดการส่วนประกอบซึ่งถูกดำเนินการไปจนถึงการส่งมอบผลิตภัณฑ์ไปยังลูกค้า

จากการศึกษางานวิจัยของ Brian Henderson-Sellers พบว่า กระบวนการพัฒนาซอฟต์แวร์ ซึ่งเน้นการทำงานร่วมกันควรเปิดกว้างใน 3 ระดับของ 1) กระบวนการระดับธุรกิจ 2) กระบวนการระดับวิศวกรรมซอฟต์แวร์ 3) กระบวนการระดับแบบจำลอง จากแนวคิดดังกล่าว ผู้วิจัยได้ใช้ความรู้ที่ได้จากการศึกษามาช่วยในการสร้างเครื่องมือในการเก็บข้อมูล เพื่อให้เข้าใกล้คำตอบของการเปรียบเทียบประสิทธิภาพของกระบวนการพัฒนาซอฟต์แวร์ในแต่ละแบบ

A. Qumer, B. Henderson-Sellersd (2006) ได้ทำการศึกษาเกี่ยวกับการประเมินระดับความเป็นอาไจล์ (Agile) ภายในวิธีการ อาไจล์ (Agile) ทั้ง 6 วิธีคือ

1. เอ็กซ์ทรีม โปรแกรมมิ่ง-เอ็กซ์พี (Extreme Programming – XP)
2. สครัม (Scrum)
3. วิธีการพัฒนาระบบไม่หยุดนิ่ง-ดีเอสดีเอ็ม (Dynamic System Development Method-DSDM)
4. การพัฒนาที่ขับเคลื่อนด้วยคุณลักษณะของซอฟต์แวร์-เอฟดีดี (Feature Driven Development-FDD)
5. คริสตัล (Crystal)
6. การพัฒนาซอฟต์แวร์แบบปรับตัว- เอเอสดี (Adaptive Software Development - ASD)

จากแถลงการณ์อาไจล์ (Agile Manifesto) ทั้ง 4 ข้อ คือ

1. การให้ความสำคัญกับบุคคลและการสื่อสารมากกว่ากระบวนการและเครื่องมือ
2. ให้ความสำคัญแก่การทำงานของซอฟต์แวร์ที่ทำงานได้จริงมากกว่าการจัดทำเอกสาร
3. การให้ความสำคัญแก่การทำงานร่วมกับลูกค้า โดยยอมรับลูกค้ามาเป็นส่วนหนึ่งของทีมมากกว่าข้อตกลงตามสัญญา
4. ให้ความสำคัญแก่การยอมรับและตอบสนองต่อการเปลี่ยนแปลงมากกว่าการปฏิบัติตามขั้นตอนหรือแผนงานที่ได้กำหนดไว้

ผลที่ได้ คือ เครื่องมือนี้เรียกว่า 4-DAT มีการจัดเตรียมขอบข่ายการประเมินค่าระดับความเป็นอาไจล์ (Agile) ทั้ง 4 มิติ ตามแถลงการณ์อาไจล์ (Agile) เพื่อประเมินหลักการอาไจล์ (Agile)

ผู้วิจัยใช้การประเมินค่าระดับความเป็นอาไจล์ (Agile) สำหรับวิธีการ Agile โดยอ้างอิงจากงานวิจัยของ A. Qumer, B. Henderson-Sellersd ซึ่งให้คะแนนเท่ากับ 1 สำหรับกรณีศึกษาที่ประยุกต์ใช้แนวปฏิบัติหลักตามระเบียบวิธีการพัฒนาซอฟต์แวร์ตามแบบอาไจล์ (Agile) และให้คะแนนเท่ากับ 0 สำหรับกรณีศึกษาที่มีการประยุกต์ใช้แนวปฏิบัติหลักที่ระเบียบวิธีการพัฒนาซอฟต์แวร์ตามแบบไม่ใช่อาไจล์ (Non-Agile)