บทที่ 2

เอกสารและงานวิจัยที่เกี่ยวข้อง

จากการศึกษาทฤษฎีและเอกสารที่เกี่ยวข้องเกี่ยวกับการพัฒนาระบบสารสนเทศเพื่อการบริหารโครงการระหว่าง หจก.จีเนียสไซเบอร์(ประเทศไทย) กับ บริษัท สแกนดีไซน์ (ประเทศเดนมาร์ก และ สวีเดน) ผู้วิจัยได้ทำการศึกษา ค้นคว้า และทบทวน แนวคิดทฤษฎีและงานวิจัยต่าง ๆ ที่เกี่ยวข้องทั้งหมด 11 ประเด็น ดังนี้

2.1 บริหารโครงการคืออะไร

2.2 โครงสร้างองค์กร

2.3 วิธีการเชิงระบบ

2.4 ขั้นตอนโครงการและวงจรชีวิตของโครงการ

2.5 วงจรชีวิตการพัฒนาซอฟต์แวร์

2.6 วงจรชีวิตของโครงการกับวงจรชีวิตการพัฒนาซอฟต์แวร์

2.7 กระบวนการพัฒนาซอฟต์แวร์

2.8 System Develop life cycle (SDLC)

2.9 Sequence Diagram ระบบการทำงานของ หจก.จีเนียสไซเบอร์

2.10 วิธีฟังก์ชันพอยท์ (Function Point)

2.11 แนวคิดเรื่องยูนิไฟเอ็ดโพรเซส (Unified Process, UP)

2.1 บริหารโครงการคืออะไร

วราภรณ์ จิรชีพพัฒนา (2551) อธิบายว่า การบริหารโครงการคือ การประยุกต์ความรู้ ทักษะ เครื่องมือ และเทคนิค เข้ากับกิจกรรมของโครงการเพื่อให้งานออกมาตรงกับความต้องการของโครงการ ผู้จัดการโครงการต้องอำนวยความสะดวกให้กระบวนการทั้งหมดทำงานให้ตรงกับความต้องการและความคาดหวังของผู้ใช้หรือลูกค้า รูปที่ 2.1 แสดงส่วนที่เกี่ยวข้องกับการบริหารโครงการซึ่งประกอบด้วยผู้มีส่วนได้ส่วนเสียกับโครงการ ความรู้การบริหารโครงการ เครื่องมือและเทคนิคการบริหารโครงการ

รูป 2.1 ส่วนที่เกี่ยวข้องกับการบริหารโครงการ (ปรับปรุงจาก Schwalbe, 2007)

2.2 โครงสร้างองค์กร

วราภรณ์ จิรชีพพัฒนา (2551) อธิบายว่า โครงสร้างองค์กรมี 3 แบบคือ โครงสร้างตามหน้าที่ (functional structure) โครงสร้างแบบโครงการ (project structure) และโครงสร้างแบบเมทริกซ์ (matrix structure) ดังแสดงในรูปที่ 2.3 โครงสร้างองค์กรมีอิทธิพลต่อโครงการสรุปในตารางที่ 2.1

โครงสร้างตามหน้าที่

โครงสร้างตามหน้าที่มีลักษณะเป็นลำดับขั้น โดยมีผู้บริหารสูงสุดอยู่ระดับบนสุด ที่รองประธาน หรือ ผู้จัดการฟังก์ชันต่างๆ ต้องรายงาน เช่น ผู้จัดการโรงงาน ผู้จัดการทรัพยากรมนุษย์ ผู้จัดการเทคโนโลยีสารสนเทศ เป็นต้น พนักงานแต่ละแผนกจะมีความเชี่ยวชาญเฉพาะ โครงสร้างตามหน้าที่เป็นโครงสร้างที่ยืดหยุ่นในการใช้บุคลากรทำงานให้โครงการ เนื่องจากบุคลากรสามารถกลับไปทำงานประจำของตนได้ทันทีที่โครงการสิ้นสุด โครงการสามารถใช้ผู้เชี่ยวชาญในแผนกทำงานนอกเวลาได้ ในทางตรงกันข้าม โครงสร้างองค์กรแบบนี้ส่งผลกระทบต่อโครงการคือ ผู้จัดการโครงการไม่มีอำนาจบังคับบัญชา การขอความร่วมมือจากแผนกอื่นๆ อาจทำได้ยาก แต่ละแผนกมุ่งเน้นแต่โครงการของตนเอง บุคลากรอาจให้ความสนใจน้อย เพราะคิดว่าเป็นการเพิ่มภาระ

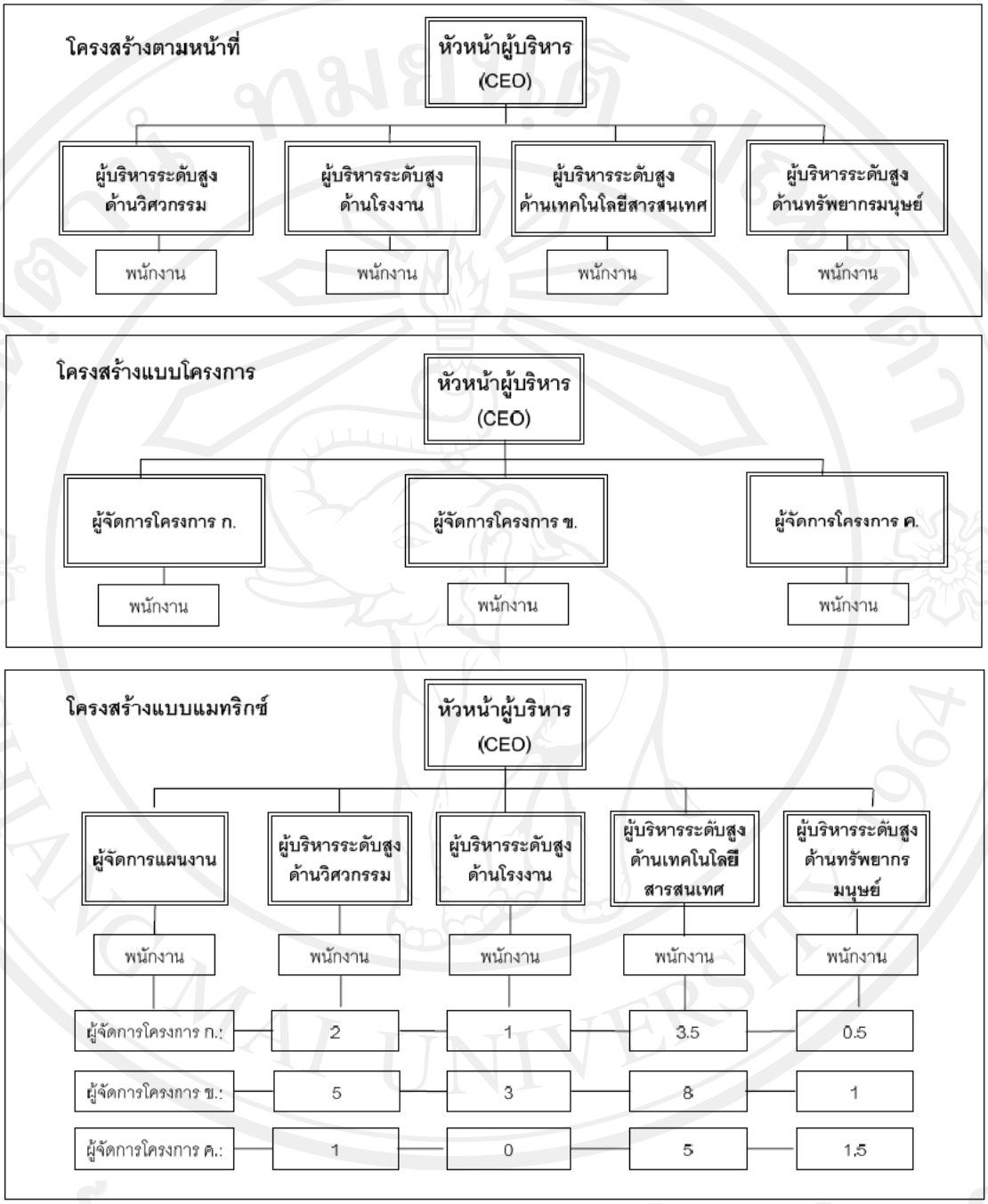
โครงสร้างแบบโครงการ

โครงสร้างแบบโครงการมีโครงสร้างเป็นลำดับขั้นเช่นเดียวกัน แต่แทนที่จะเป็นรองประธาน หรือผู้จัดการที่รายงานต่อผู้บริหารสูงสุด กลับเป็นผู้จัดการแผนงาน ที่รายงานต่อผู้บริหารสูงสุด ผู้ร่วมงานในแผนงานมีทักษะที่หลากหลายเพื่อทำงานโครงการต่างๆ ได้สมบูรณ์ โครงสร้างองค์กรแบบนี้สามารถทำโครงการได้เสร็จอย่างรวดเร็ว ผู้จัดการโครงการมีอำนาจเต็มที่และมีผู้ร่วมงานที่อุทิศเวลาให้กับโครงการอย่างเต็มที่ โดยมีเป้าหมายร่วมกัน การสื่อสารกับ

ผู้บริหารระดับสูงทำได้อย่างรวดเร็ว ถึงแม้ว่าผู้จัดการโครงการจะมีอำนาจสูงสุดในโครงสร้างแบบโครงการ แต่การจัดแบบนี้ไม่มีประสิทธิภาพ การมอบหมายพนักงานให้ทำงานเต็มเวลา เป็นการใช้ทรัพยากรไม่เหมาะสม เช่น ถ้าคนเขียนเอกสารเชิงเทคนิคถูกมอบหมายให้ทำงานเต็มเวลากับโครงการหนึ่ง แต่อาจไม่มีงานให้ทำในบางวัน องค์กรยังต้องจ่ายเงินให้พนักงานคนนั้นเต็มเวลา การจัดแบบโครงการอาจทำให้ไม่เกิดการประหยัด

โครงสร้างแบบเมทริกซ์

โครงสร้างแบบเมทริกซ์เป็นโครงสร้างที่ผสมของโครงสร้างทั้งสองที่กล่าวมาแล้ว มีลักษณะเป็นกริด บุคลากรของโครงการรายงานต่อผู้จัดการตามหน้าที่ และผู้จัดการโครงการ เช่น บุคลากรทางเทคโนโลยีสารสนเทศส่วนใหญ่จะทำงานหลายโครงการ แต่ต้องรายงานต่อผู้จัดการแผนกเทคโนโลยีสารสนเทศและผู้จัดการจากแผนกต่างๆ โครงสร้างแบบนี้ยังแบ่งออกเป็น 3 แบบย่อยคือ โครงสร้างเมทริกซ์แบบอ่อน (weak matrix), โครงสร้างเมทริกซ์แบบสมดุล (balanced matrix) และ โครงสร้างเมทริกซ์แบบเข้มข้น (strong matrix)



รูป 2.2 โครงสร้างองค์กร (Schwalbe, 2007)

ตาราง 2.1 อิทธิพลของโครงสร้างองค์กรต่อโครงการ (PMBOK Guide 2004)

ลักษณะโครงการ	ประเภทโครงสร้างองค์การ				
	ตามหน้าที่	แมทริกซ์			โครงการ
		แมทริกซ์แบบอ่อน ๆ	•	•	
อำนาจของผู้จัดการโครงการ	น้อย หรือ ไม่มี	จำกัด	ต่ำ – ปานกลาง	ปานกลาง – สูง	สูง - เกือบสมบูรณ์
ผู้ควบคุมงบประมาณโครงการ	ผู้จัดการตามหน้าที่	ผู้จัดการตามหน้าที่	ผสม	ผู้จัดการโครงการ	ผู้จัดการโครงการ
บทบาทของผู้จัดการโครงการ	ไม่เต็มเวลา	ไม่เต็มเวลา	เต็มเวลา	เต็มเวลา	เต็มเวลา
พนักงานบริหารงานของโครงการ	ไม่เต็มเวลา	ไม่เต็มเวลา	ไม่เต็มเวลา	เต็มเวลา	เต็มเวลา

2.3 วิธีการเชิงระบบ

วราภรณ์ จิรชีพพัฒนา (2551) อธิบายว่า ถึงแม้ว่าโครงการจะเป็นการรวมตัวกลุ่มคนชั่วคราวเพื่อสร้างผลิตผลหรือให้บริการ แต่ผู้จัด การโครงการไม่สามารถดำเนินโครงการแยกออกจากองค์กร โครงการจะต้องทำงานในสภาวะแวดล้อมขององค์กรในระดับกว้าง และผู้จัดการโครงการจำเป็นต้องพิจารณาโครงการภายใต้บริบทเชิงองค์กร เพื่อจัดการสถานการณ์ที่สลับซับซ้อนได้อย่างมีประสิทธิผล ผู้จัดการโครงการจำเป็นต้องใช้มุมมองแบบองค์รวม หรือการคิดเชิงระบบ (systems thinking)

วิธีการเชิงระบบคือ วิธีการเชิงวิเคราะห์แบบองค์รวมเพื่อการแก้ปัญหาที่สลับซับซ้อน เป็นวิธีการที่รวมการใช้ปรัชญาระบบ (systems philosophy) การวิเคราะห์ระบบ (systems analysis)

และการบริหารระบบ (system management) ปรัชญาระบบคือ ตัวแบบภาพรวมทั้งหมดสำหรับการคิดเกี่ยวกับสิ่งต่างๆ เสมือนระบบ การวิเคราะห์ระบบคือ วิธีการแก้ปัญหาที่ต้องมีการกำหนดขอบเขตของระบบ การแบ่งระบบเป็นส่วนๆ การระบุและการประเมินปัญหา โอกาส ข้อจำกัด และความต้องการ จากนั้น นักวิเคราะห์ระบบตรวจสอบคำตอบที่เป็นทางเลือกสำหรับการปรับปรุงสถานการณ์ปัจจุบัน กำหนดทางเลือกที่ดีที่สุด (optimum) หรืออย่างน้อยทางเลือกที่พึงพอใจ (satisfactory) หรือแผนการดำเนินการ รวมทั้งตรวจสอบแผนกับระบบทั้งหมด การบริหารระบบหมายถึงประเด็นเชิงธุรกิจ เทคโนโลยี และองค์กรที่เกี่ยวกับการสร้าง การบำรุงรักษา และการทำการเปลี่ยนแปลงให้กับระบบ

การใช้วิธีการเชิงระบบมีความสำคัญต่อความสำเร็จของการบริหารโครงการ ผู้บริหารระดับสูงและผู้จัดการโครงการต้องทำตามปรัชญาระบบเพื่อให้เข้าใจว่าโครงการมีความสัมพันธ์กับองค์กรทั้งหมดอย่างไร ผู้จัดการโครงการต้องใช้การวิเคราะห์ระบบเพื่อกำหนดความต้องการ ต้องใช้การบริหารระบบเพื่อระบุประเด็นหลักทางด้านธุรกิจ เทคโนโลยี และการวิเคราะห์ความสัมพันธ์ระหว่างองค์กรกับแต่ละโครงการเพื่อกำหนดผู้มีส่วนได้ส่วนเสียที่สำคัญ

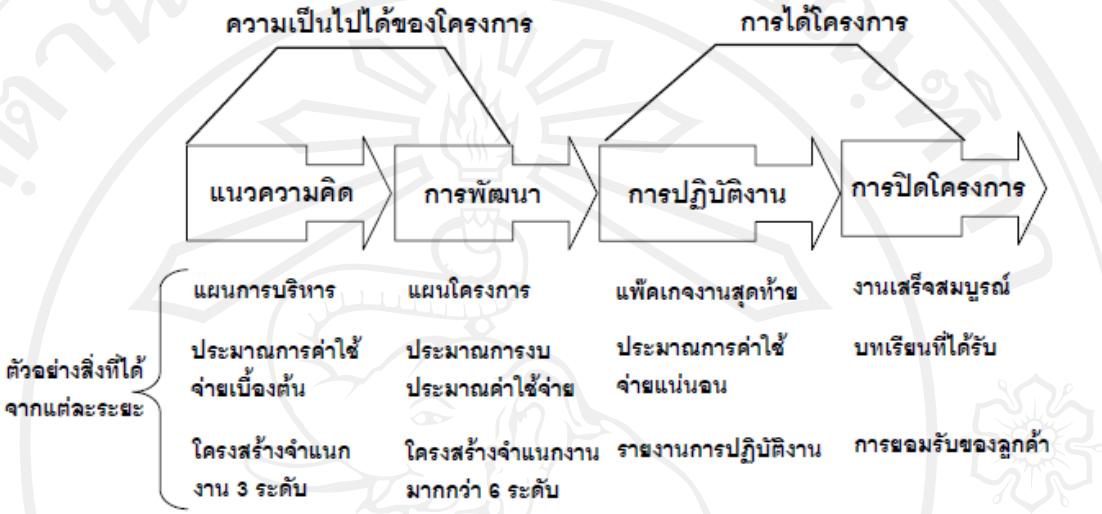
2.4 ขั้นตอนโครงการและวงจรชีวิตของโครงการ

วราภรณ์ จิรชีพพัฒนา (2551) อธิบายว่า วงจรชีวิตของโครงการคือ ชุดงานของขั้นตอนโครงการ โดยแต่ละขั้นตอนจะกำหนดงานที่ต้องทำ ทำเมื่อไร ใครเป็นคนทำ สิ่งที่ได้จากงานคืออะไร (deliverables) การควบคุมและอนุมัติงานจะทำอย่างไร สิ่งที่ได้จากงานคือ ผลิตผลหรือบริการ (เช่น รายงาน การอบรม ชิ้นส่วนฮาร์ดแวร์ หรือโปรแกรมบางส่วน)

ในขั้นตอนแรกของวงจรชีวิตของโครงการมีความต้องการใช้ทรัพยากรน้อย และมีระดับความไม่แน่นอนสูงสุด ผู้มีส่วนได้เสียของโครงการมีอิทธิพลสูงสุดต่อคุณลักษณะของผลิตผลสุดท้ายของโครงการ หรือแม้แต่ผลลัพธ์ที่เกิดขึ้นในช่วงแรกของโครงการ ระหว่างช่วงกลางของวงจรชีวิตของโครงการ ความแน่นอนจะเพิ่มขึ้น และความต้องการใช้ทรัพยากรเพิ่มขึ้นมากกว่าช่วงเริ่มต้นและช่วงสุดท้ายของโครงการ ในขั้นตอนสุดท้ายเป็นขั้นตอนที่เน้นความต้องการของโครงการตรงกับที่ผู้สนับสนุนโครงการอนุมัติหรือไม่

ขั้นตอนโครงการของแต่ละโครงการมีความแตกต่างกัน ขึ้นอยู่กับลักษณะของโครงการและอุตสาหกรรม แต่ขั้นตอนที่พบในการบริหารโครงการโดยทั่วไปประกอบด้วย 2 ขั้นตอนใหญ่ 4 ขั้นตอนย่อยคือ ความเป็นไปได้ของโครงการ (project feasibility) และการได้โครงการ (project acquisition) ขั้นตอนความเป็นได้ของโครงการยังประกอบด้วย แนวความคิด (concept) และการ

พัฒนา (development) ส่วนขั้นตอนการได้โครงการประกอบด้วย การปฏิบัติงาน (implementation) และปิดโครงการ (close-out) ขั้นตอนของโครงการได้แสดงในรูป 2.3



รูป 2.3 ขั้นตอนของวงจรชีวิตโครงการ (Schwalbe, 2007)

ในขั้นตอนแนวความคิดของโครงการ ผู้จัดการโครงการจะบรรยายโครงการอย่างคร่าวๆ พัฒนาแผนของโครงการแบบสรุป หรือระดับสูงมากๆ (high-level) ซึ่งแผนจะบรรยายถึงความต้องการโครงการ และแนวความคิดเบื้องต้น ประมาณการค่าใช้จ่ายอย่างหยาบๆ และกำหนดงานที่ต้องทำโดยรวม งานที่ต้องทำจะจำแนกเป็นงานย่อย กำหนดเอกสารที่ต้องผลิต เช่น ในโครงการคอมพิวเตอร์โน้ตบุ๊ค ทีมงานเริ่มศึกษาแนวความคิดที่จะเพิ่มการใช้เทคโนโลยี พัฒนาแผนการบริหารที่มีโครงการขนาดเล็กเพื่อสำรวจทางเลือกในการเพิ่มการใช้เทคโนโลยีสารสนเทศ ประมาณการค่าใช้จ่ายสำหรับโครงการสำรวจนี้ จำแนกงานโครงการสำรวจ สำรวจข้อมูลจากนักเรียนและพนักงานวิทยาลัย ประเมินการอย่างหยาบว่า การใช้เทคโนโลยีสารสนเทศจะมีผลกระทบต่อค่าใช้จ่ายและการขึ้นทะเบียนอย่างไร สุดท้ายขั้นตอนนี้จะได้รายงานที่นำเสนอผลการศึกษา

ขั้นตอนการพัฒนาเป็นขั้นตอนที่ทีมงานสร้างแผนโครงการที่ละเอียดมากขึ้น การประมาณการค่าใช้จ่ายถูกต้องมากขึ้น และการจำแนกงานลงรายละเอียดมากกว่าเดิม สมมุติว่าในขั้นตอนแนวความคิดได้เสนอแนะว่า การให้นักเรียนมีคอมพิวเตอร์โน้ตบุ๊คเป็นแนวทางที่จะเพิ่มการใช้เทคโนโลยีสารสนเทศ ทีมงานต้องขยายแนวความคิดต่อในขั้นตอนนี้ ถ้านักเรียนซื้อหรือเช่าคอมพิวเตอร์โน้ตบุ๊คแล้ว ทีมงานอาจต้องตัดสินใจว่าฮาร์ดแวร์และซอฟต์แวร์ที่ต้องใช้มีอะไร จะคิดเงินกับนักเรียนเท่าไร จะจัดการอบรมและบำรุงรักษาเครื่องอย่างไร จะบูรณาการการใช้

เทคโนโลยีใหม่กับวิชาปัจจุบันอย่างไร ถ้ารายงานจากขั้นตอนแรกเสนอว่า คอมพิวเตอร์โน้ตบุ๊คเป็นแนวความคิดที่ไม่ดีสำหรับวิทยาลัย ทีมงานก็ไม่ต้องทำงานในขั้นตอนที่สอง

ขั้นตอนที่สามของวงจรชีวิตของโครงการคือ การปฏิบัติงาน ในขั้นตอนนี้ ทีมงานประมาณการค่าใช้จ่ายได้อย่างถูกต้อง และส่งงานที่ต้องการ มีการทำรายงานผลการทำงานต่อผู้มีส่วนได้เสีย สำหรับโครงการคอมพิวเตอร์โน้ตบุ๊คนั้น ทีมงานต้องหาฮาร์ดแวร์และซอฟต์แวร์ ติดตั้งอุปกรณ์เครือข่าย ส่งมอบคอมพิวเตอร์โน้ตบุ๊คให้นักเรียนและพนักงาน

ขั้นตอนสุดท้ายคือ ปิดโครงการ งานทุกอย่างเสร็จสมบรูณ์ ได้รับการยอมรับจากผู้ใช้ทั้งโครงการ ทีมงานบันทึกประสบการณ์ที่ได้จากโครงการ ทีมงานอาจสำรวจความคิดเห็นของนักเรียนและพนักงานวิทยาลัย ตรวจสอบสัญญากับผู้ขายสินค้าว่าเสร็จสิ้นสมบรูณ์หรือไม่ จ่ายเงินเรียบร้อยหรือไม่ ส่งมอบโครงการให้กับหน่วยงานอื่นต่อไป

2.5 วงจรชีวิตการพัฒนาซอฟต์แวร์

วราภรณ์ จิรชีพพัฒนา (2551) อธิบายว่า วงจรชีวิตการพัฒนาซอฟต์แวร์โดยปกติประกอบด้วยขั้นตอน การวางแผน การวิเคราะห์ การออกแบบ การสร้างและติดตั้ง และการบำรุงรักษา ขั้นตอนดังกล่าวได้ถูกจัดการหลายรูปแบบ จึงทำให้เกิดวงจรชีวิตการพัฒนาซอฟต์แวร์รูปแบบต่างๆ ส่วนใหญ่วงจรชีวิตการพัฒนาซอฟต์แวรที่ใช้กันมีดังนี้

วงจรชีวิตแบบน้ำตก (waterfall life cycle model) เป็นการพัฒนาที่มีการกำหนดขั้นตอนการพัฒนาที่ชัดเจนคือ วิเคราะห์ระบบ ออกแบบระบบ พัฒนาระบบ ทดสอบระบบ ติดตั้งระบบ และบำรุงรักษาระบบ ขั้นตอนต่างๆ ทำแบบเรียงลำดับ วิธีการนี้มีสมมุติฐานว่าความต้องการไม่เปลี่ยนแปลงหลังจากได้กำหนดแล้ว

วงจรชีวิตแบบก้นหอย (spiral life cycle model) ได้พัฒนาขึ้นจากประสบการณ์การใช้วิธีการพัฒนาแบบน้ำตก ในความเป็นจริง วงจรชีวิตการพัฒนาซอฟต์แวร์เป็นแบบซ้ำๆ หรือก้นหอยมากกว่าแบบเรียงลำดับ โดยแต่ละงานที่ทำแบ่งออกเป็น 4 ส่วนคือ วางแผน วิเคราะห์และขจัดความเสี่ยง พัฒนางาน

วงจรการพัฒนาแบบเพิ่ม (incremental development life cycle model) เป็นการพัฒนาซอฟต์แวร์แบบก้าวหน้า ส่งมอบซอฟต์แวร์ทีละส่วน แล้วค่อยๆ เพิ่มความสามารถของระบบไปเรื่อยๆ จนระบบมีความสามารถสมบูรณ์

วงจรชีวิตแบบต้นแบบ (prototyping life cycle model) เป็นการพัฒนาซอฟต์แวร์โดยการใช้ต้นแบบในการทำความเข้าใจความต้องการของผู้ใช้ให้ชัดเจน ทีมงานต้องพัฒนาต้นแบบให้ผู้ใช้เห็นและทดลองใช้ ถ้าผู้ใช้ยังไม่พอใจ ทีมงานจะทำการแก้ไข แล้วนำมาให้ผู้ใช้พิจารณาใหม่

ถ้าผู้ใช้พอใจต้นแบบนั้น ทีมงานจะนำต้นแบบไปใช้ในการออกแบบรายละเอียดของระบบที่แท้จริง หรืออาจนำต้นแบบนั้นไปติดตั้งให้กับผู้ใช้เลยก็ได้ ทั้งนี้ขึ้นอยู่กับโครงการ

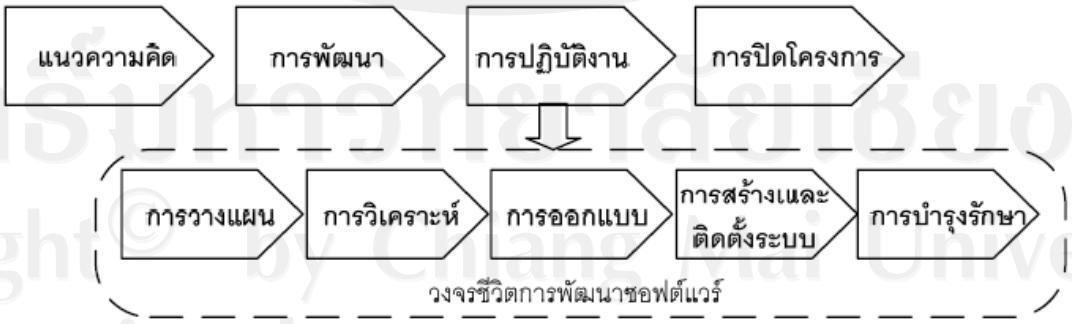
วงจรชีวิตการพัฒนาระบบงานแบบรวดเร็ว (rapid application development life cycle model) เป็นการพัฒนาโดยการใช้เครื่องมือที่ช่วยให้ทีมงานพัฒนาซอฟต์แวร์ได้อย่างรวดเร็ว เช่น CASE JAD ใช้ภาษา 4th GL การพัฒนาซอฟต์แวร์ด้วยวิธีการนี้ต้องแลกเปลี่ยนกับคุณภาพของซอฟต์แวร์ เนื่องจากต้องจำกัดกิจกรรมที่ต้องดำเนินการให้เหลือน้อยที่สุด

การโปรแกรมแบบเข้มข้น (extreme programming (XP)) เป็นการพัฒนาซอฟต์แวร์ที่สอดคล้องกับความต้องการที่สภาวะแวดล้อมเปลี่ยนอย่างรวดเร็ว การพัฒนาด้วยวิธีนี้จะใช้ทีมพัฒนาคู่เพื่อส่งเสริมการทำงานประสานกัน และเพิ่มประสิทธิภาพ ผู้พัฒนาซอฟต์แวร์ต้องเขียนและทดสอบโปรแกรมเอง

วงจรชีวิตแบบสกรัม (scrum life cycle model) เป็นการพัฒนาแบบซ้ำๆ ที่เน้นความต้องการที่เปลี่ยนแปลง แต่ละวันทีมงานทั้งหมดจะประชุมกันช่วงเวลาสั้นๆ ว่าวันนี้จะต้องทำอะไรให้สำเร็จ สมาชิกจะระบุอุปสรรค และผู้จัดการโครงการจะต้องจัดการกับอุปสรรคนั้น วิธีการนี้ต้องการผู้นำที่เข้มแข็ง

2.6 วงจรชีวิตของโครงการกับวงจรชีวิตการพัฒนาซอฟต์แวร์

วราภรณ์ จิรชีพพัฒนา (2551) อธิบายว่า จะเห็นได้ว่าวงจรชีวิตของโครงการและวงจรชีวิตการพัฒนาซอฟต์แวร์ดูเหมือนว่าจะคล้ายคลึงกัน แต่วงจรชีวิตของโครงการเน้นที่กระบวนการการจัดการโครงการ ขณะที่วงจรการพัฒนาซอฟต์แวร์เน้นที่การสร้างและการทำให้เกิดระบบสารสนเทศ จากรูป 2.4 เราจะเห็นว่าจริงๆ แล้ว วงจรชีวิตการพัฒนาซอฟต์แวร์เป็นส่วนหนึ่งของวงจรชีวิตของโครงการ เพราะหลายๆ กิจกรรมสำหรับการพัฒนาระบบสารสนเทศเกิดขึ้นระหว่างเฟสการปฏิบัติงาน



รูป 2.4 วงจรชีวิตของโครงการกับวงจรชีวิตการพัฒนาซอฟต์แวร์

2.7 กระบวนการพัฒนาซอฟต์แวร์

สารานุกรมไทยสำหรับเยาวชนฯ อธิบายว่า กระบวนการการพัฒนาซอฟต์แวร์ที่ดีควรหาข้อผิดพลาดให้ได้ แต่ต้องไม่สับสนเรื่องขั้นตอนในการพัฒนา ถ้าผลิตอย่างมีขั้นตอน ก็ควรมีความยืดหยุ่นพอสมควร และไม่ยึดติดกับขั้นตอนมากเกินไป จนทำให้โครงการล่าช้าหรือล้มเหลว เพราะเลือกใช้ขั้นตอนที่ไม่เหมาะสมกับ ประเภทของซอฟต์แวร์

ถ้าพบข้อผิดพลาดช่วงแรกๆ ก็จะช่วยลดระยะเวลาและค่าใช้จ่ายในการพัฒนาซอฟต์แวร์ได้มาก การศึกษาเรื่องค่าใช้จ่ายด้านซอฟต์แวร์ของบริษัทไอบีเอ็ม (IBM) บริษัทจีทีอี (GTE) และบริษัททีอาร์ดับเบิลยู (TRW) โดยนายแบรี บีม(Barry Boehm) ในปี ค.ศ. 1936 พบว่า ถ้าแก้ไขข้อผิดพลาดเมื่อพัฒนาซอฟต์แวร์เสร็จแล้วแทนที่จะแก้ไขตั้งแต่ตอนที่ กำหนดคุณลักษณะของซอฟต์แวร์ ก็จะต้องเสียค่าใช้จ่ายเพิ่มขึ้นเป็น 100 เท่า ตัวอย่างของปัญหานี้เห็นได้อย่างชัดเจนในการแก้ปัญหาคอมพิวเตอร์ ปี ค.ศ. 2000 หรือที่เรียกว่า ปัญหาวายทูเค (Y2K) บริษัที่ออกแบบซอฟต์แวร์ให้ใช้กับปีที่มีเลข 4 หลักตั้งแต่ต้นแทบจะไม่เดือดร้อนในการแก้ไขเลย แต่ซอฟต์แวร์อื่นๆ ที่ทั้งผู้ผลิตและผู้ใช้ต่างละเลยปัญหานี้ โดยยังคงใช้ปีเป็นเลข 2 หลักอยู่ บางรายใช้อยู่เป็นสิบๆ ปี เมื่อถึงเวลาแก้ไขข้อผิดพลาดของซอฟต์แวร์ ก็ปรากฏว่าต้องใช้เงินเป็นจำนวนมากถึงหลายสิบล้านบาท ร้อยล้านบาท หรือมากกว่านั้น

การพัฒนาซอฟต์แวร์อย่างมีขั้นตอน มีหลายแบบ และยังมีวิวัฒนาการอยู่อย่างต่อเนื่อง เพราะสาขาทางวิศวกรรมซอฟต์แวร์นี้ เพิ่งเกิดขึ้นในครึ่งหลังของศตวรรษที่ 20 หากเทียบกับระยะเวลาที่มนุษยชาติศึกษากระบวนการสร้างที่อยู่อาศัยที่มีมา เป็นพันๆ ปีแล้ว ก็นับได้ว่าสาขาวิชานี้ยังใหม่อยู่มาก ดังนั้น ในการเลือกกระบวนการการพัฒนาซอฟต์แวร์จะต้องตั้งคำถามว่ากระบวนการใด "เหมาะ" ที่สุดสำหรับโจทย์ปัญหาและประเภทของซอฟต์แวร์ ไม่ใช่กระบวนการใด "ดี" ที่สุด

การพัฒนาซอฟต์แวร์ไม่ใช่กระบวนการที่มีจุดเริ่มต้นและจุดสิ้นสุดที่แน่นอน เหมือนเช่นผลิตภัณฑ์อื่นๆ เนื่องจากซอฟต์แวร์ใช้สำหรับสั่งให้คอมพิวเตอร์ช่วยแก้โจทย์ปัญหาบางอย่างให้แก่มนุษย์ เมื่อสภาพของโจทย์ปัญหาหรือคอมพิวเตอร์เปลี่ยนแปลงไป ซอฟต์แวร์ก็ต้องเปลี่ยนแปลงตาม ซอฟต์แวร์ที่ขาดการทำนุบำรุงจึงเสื่อมคุณภาพ ทั้งที่ไม่ได้สึกหรอ ทั้งนี้เพราะไม่สามารถนำไปใช้แก้ปัญหาได้อย่างมีประสิทธิภาพอีกต่อไป

ในสมัยแรกๆ การผลิตซอฟต์แวร์มักไม่มีขั้นตอน คือ เริ่มต้นด้วยการเขียนซอฟต์แวร์เลย เมื่อมีปัญหาก็แก้ไข เขียนแล้วแก้สลับกันไปจนกว่าจะได้คุณลักษณะที่ต้องการ ผลก็คือ จะได้ซอฟต์แวร์ที่ซับซ้อนมาก หากต้องมีการปรับปรุงแก้ไขในภายหลัง และผู้ที่แก้ไขไม่ใช่ผู้เขียน

ซอฟต์แวร์นั้นเอง ก็จะมีปัญหามาก มักทำให้ต้องเสียค่าใช้จ่ายในการทำนุบำรุงซอฟต์แวร์เกินกว่างบประมาณที่ กำหนดไว้

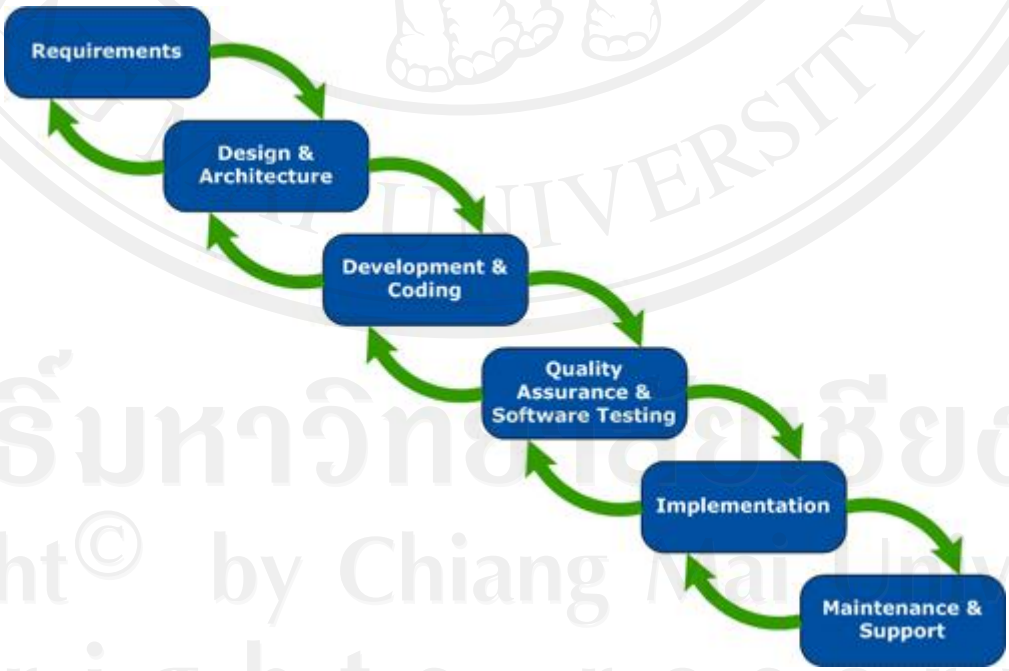
ต่อมาได้มีการนำหลักวิศวกรรมมาประยุกต์ใช้ในการพัฒนาซอฟต์แวร์ การพัฒนาซอฟต์แวร์จึงแบ่งได้เป็น 3 ระยะ คือ

1) กำหนดคุณลักษณะซอฟต์แวร์ (Definition) เน้นว่าจะ "สร้างอะไร" โดยให้คำตอบว่า "โจทย์ปัญหาที่ต้องการแก้คืออะไร" และ "สิ่งที่จะใช้แก้ปัญหานี้คืออะไร"

2) สร้างซอฟต์แวร์ (Development) เน้นว่าจะ "สร้างอย่างไร" โดยให้คำตอบเรื่อง "ทำอย่างไรจึงจะสร้างสิ่งที่นำมาใช้แก้ปัญหาได้" และ "ทำอย่างไรจึงจะตรวจสอบหาข้อบกพร่องของสิ่งที่สร้างขึ้นได้ ตลอดจนสิ่งที่นำมาใช้แก้ปัญหารวมทั้งซอฟต์แวร์และเอกสารอธิบายซอฟต์แวร์"

3) วิวัฒนาการของซอฟต์แวร์ (Evolution)เน้นว่าจะ "เปลี่ยนแปลงอย่างไร" โดยให้คำตอบเรื่อง "เมื่อสภาพการณ์หรือปัญหาเปลี่ยนแปลงไปต้องทำอย่างไรจึงจะสามารถปรับปรุง สิ่งนั้นให้ยังคงใช้แก้ปัญหาได้"

แม่แบบของกระบวนการการพัฒนาซอฟต์แวร์ที่เป็นขั้นตอนที่เก่าแก่ที่สุดนั้น เรียกกันว่า แม่แบบแบบขั้นน้ำตก (Waterfall Model) ซึ่งเมื่อลากเส้นเชื่อมต่อแต่ละขั้นตอนลงไปจนถึงขั้นตอนสุดท้ายแล้ว ก็จะมีลักษณะคล้ายน้ำตก



รูป 2.5 waterfall life cycle model

2.8 System Develop life cycle (SDLC)

Tayntor อธิบายว่า System Develop life cycle เป็นวงรอบหรือขั้นตอนของการออกแบบและพัฒนา Software ที่นิยมใช้กันมาก เพื่อสร้าง Software ที่สนองตอบต่อความต้องการในระบบเทคโนโลยีสารสนเทศขององค์กร โดยมีขั้นตอนพื้นฐานที่กำหนดไว้เป็นไปโดยลำดับขั้น ดังนี้

1) การเริ่มต้นโครงการ (Project Initiation)

2) การวิเคราะห์ระบบ (System Analysis)

3) การออกแบบระบบ (System Design)

4) การก่อสร้างระบบ (Construction)

5) การทดสอบระบบและการประกันคุณภาพ (Testing and Quality Assurance)

6) การติดตั้งระบบ (Implementation)

การเริ่มต้นโครงการ (Project Initiation) ในขั้นตอนนี้ประกอบด้วย 6 ขั้นตอนย่อยในการดำเนินการ คือ

1) พิจารณาคัดเลือกโครงการ (Identify the problem)

2) สร้างทีมงาน (From the team)

3) คัดเลือกความต้องการขั้นต้น (Identify preliminary requirements)

4) หาสาเหตุและวิธีแก้ไข เพื่อสนองตอบต่อความต้องการ (Validate the requirements)

5) พัฒนาและศึกษาความเป็นไปได้ (Develop a feasibility study)

6) อนุมัติโครงการ (Obtain project approval)

การวิเคราะห์ระบบ (System Analysis) ในการวิเคราะห์ระบบนี้ ก็เพื่อที่จะวิเคราะห์โครงการตามขั้นตอนที่แล้ว ให้มีความเป็นไปได้ โดยมีขั้นตอนย่อยที่แนะนำให้ต้องดำเนินการตามลำดับ ดังนี้

1) เข้าใจแนวโน้มต่างๆ ของกระบวนการในโครงการโดยถ่องแท้

2) คัดเลือกความต้องการของลูกค้า

3) จัดลำดับความสำคัญของความต้องของการลูกค้า

4) คัดเลือกความต้องการของลูกค้า ตามศักยภาพขององค์กรที่สามารถดำเนินการได้

5) ตัดสินใจที่จะปรับปรุงกระบวนการที่มีผลกระทบต่อโครงการมากที่สุด โดยเป็นไปตามลำดับความต้องการสูงสุดของลูกค้า

6) จัดทำรายละเอียดที่จะทำ ลงในแผนที่ของโครงการ

7) พิจารณากระทบและความเสี่ยงที่จะทำ ในกระบวนการที่ตัดสินใจปรับปรุง

8) สรุปแนวทางการพัฒนาของการออกแบบ

9) สรุปรายงานความต้องการตามคุณลักษณะเฉพาะที่ต้องการ

10) ได้รับความเห็นชอบร่วมกันทั้งหมด

การออกแบบระบบ (System Design) ในขั้นตอนนี้จะแบ่งออกเป็นขั้นตอนย่อย 3 ขั้นตอน อันได้แก่

1) การออกแบบเกี่ยวกับลักษณะงาน (Functional Design)

2) การออกแบบทางเทคนิค (Technical Design)

3) การออกแบบโปรแกรม (Program Design)

การสร้างระบบ (Construction) ในการสร้างระบบต้องคำนึงถึงจุดประสงค์ต่างๆ ที่ต้องการ โดยมีข้อพิจารณาดังนี้

1) แน่ใจว่าจะไม่เกิดข้อบกพร่องเมื่อใช้งานจริง

2) แน่ใจว่าการเขียนโปรแกรมต้องเป็นไปตามมาตรฐาน เป็นไปตามข้อตกลง มีเหตุมีผล มีขั้นตอนที่เป็นระบบ และระเบียบ

3) ขั้นตอนการดูแลรักษาระบบ ต้องสร้างให้ผู้ใช้ มีส่วนร่วมในการดูแลรักษา ผู้ใช้สามารถเข้าใจและรับรู้การทำงานของระบบ สามารถช่วยแกปัญหาในโปรแกรมเมื่อต้องการเปลี่ยนแปลง และผู้ใช้สามารถรักษาดูแลระบบได้ในระยะยาว มีต้นทุนที่ต่ำ

การทดสอบระบบและการประกันคุณภาพ (Testing and Quality Assurance) วัตถุประสงค์ในขั้นนี้ คือ เป็นการยืนยันระหว่างผู้ออกแบบระบบ กับความต้องการของลูกค้า ว่า ระบบที่ออกแบบนี้สามารถใช้งานได้จริง ในขั้นนี้อาจแบ่งการทดสอบออกเป็นส่วนๆ ได้ดังนี้

1) การทดสอบแผน (The Test Plan)

2) การทดสอบเป็นหน่วยหรือแผนก (The Unit Test)

3) การทดสอบทั้งระบบ (The System Test)

4) การทดสอบแบบบูรณการ (The Integration Test)

5) การทดสอบความทนทานของระบบ (The Stress Test)

6) การทดสอบการยอมรับของผู้ใช้ (The Acceptance Test)

การติดตั้งระบบ (Implementation) ในขั้นตอนนี้เป็นขั้นตอนสุดท้ายของ SDLC ซึ่งมี 4 ขั้นตอนหลัก คือ

1) การสอนและฝึกผู้ใช้ระบบ (Customer Training)

2) ส่งมอบคู่มือและเอกสารต่างๆ ให้กับผู้ใช้

3) วางแผนการเปลี่ยนแปลงของข้อมูลต่างๆ ของระบบ (Data conversion)

4) การประเมินผลของระบบ

2.9 วิธีฟังก์ชันพอยต์ (Function Point)

A.J. Albrecht (1979) เนื่องจากปัญหาเกี่ยวกับความแตกต่างของภาษาโปรแกรมทำให้การวัดจำนวนบรรทัดการเขียนโค้ด (Line of Code) ไม่เท่ากัน จึงมีการแก้ปัญหาโดยใช้ฟังก์ชันพอยต์โดย A.J. Albrecht จากบริษัทไอบีเอ็ม (IBM) ทำการวิเคราะห์ฟังก์ชันพอยต์ (Function Point Analysis: FPA) ซึ่งเป็นส่วนหนึ่งของวิธีการของฟังก์ชันพอยต์ โดยที่ไม่สนใจว่าซอฟต์แวร์นั้นพัฒนาด้วยภาษาโปรแกรมใด แต่จะนับจากผลลัพธ์ของฟังก์ชัน ส่วนประกอบของระบบ ฟังก์ชันพอยต์แต่ละประเภทจะมีความซับซ้อนต่างกัน ดังนั้นจึงมีการให้ค่าน้ำหนัก (Complexity Weight) เพื่อแบ่งความซับซ้อนของฟังก์ชันพอยต์แต่ละประเภท

ตัวแปรที่มีผลกระทบในการพัฒนาซอฟต์แวร์ (Value Adjustment Factor, VAF) หรือสภาพแวดล้อมในของการพัฒนาซอฟต์แวร์นั้น อาจจะมีหรือไม่มีผลกระทบจากตัวแปรต่าง ๆ เหล่านี้ผู้ประมาณการจะเป็นผู้ให้ค่าน้ำหนักของตัวแปรเพื่อหาค่าความซับซ้อนของตัวแปร (Technical Complexity Factor, TCF) ที่มีผลกระทบต่อไปโดย ตัวแปรที่มีผลกระทบในการพัฒนาซอฟต์แวร์ประกอบ 14 ปัจจัยดังนี้

1) Data communication ข้อมูลและการควบคุมข้อมูลที่ใช้ในซอฟต์แวร์

2) Distributed function การประมวลผลข้อมูลแบบกระจาย

3) Performance ประสิทธิภาพด้านต่างๆของซอฟต์แวร์

4) Heavily used configuration โปรแกรมที่ต้องมีการคอนฟิกบ่อยครั้ง

5) Transaction rates อัตราการทำงานของข้อมูล

6) On-line data entry สามารถที่จะควบคุมข้อมูลแบบออนไลน์

7) Design for end-user efficiency ประสิทธิภาพในการออกแบบสำหรับผู้ใช้

8) On-line update คุณสมบัติในการปรับปรุงข้อมูลแบบออนไลน์

9) Complex processing ความซับซ้อนในการประมวลผล

10) Usability in other applications มีความยืดหยุ่นที่จะใช้ร่วมกับแอปพลิเคชันอื่น

11) Installation ease ความง่ายในการติดตั้ง

12) Operational ease ความง่ายในการใช้งาน

13) Multiple sites มีการแบ่งการทำงานเป็นหลายที่หรือไม่

14) Facilitate change สามารถที่เปลี่ยนแปลงซอฟต์แวร์ได้ง่าย

การตัดสินใจในเรื่องของขนาดโดยใช้ส่วนประกอบในการประมวลผลของซอฟต์แวร์ โดยถือเอาว่าส่วนประกอบของระบบคือสิ่งที่ผู้ใช้ใช้งานหรือมองเห็นได้ (เรียก User Function Types by IBM) และมีการกำหนดการวัดขนาดที่ใช้วัดดังนี้

1) ข้อมูลเข้าจากภายนอก (External Input)

2) ข้อมูลที่ส่งออกสู่ภายนอก (External Output)

3) ข้อมูลที่ดึงมาจากภายนอก (External Inquiries)

4) ข้อมูลที่ต้องการจากภายนอก (External Interface Files)

5) ข้อมูลเชิงตรรกะภายใน (Internal Logical Files)

เป็นเหตุทำให้มีการศึกษาการวิเคราะห์ค่าน้ำหนัก ยกตัวอย่างระบบที่เป็นแบบ Batch ที่จำนวนอินพุทและเอาท์พุท (Input / Output) ที่เกี่ยวข้องกับฟังก์ชั่นพอยท์ที่สามารถนับค่าได้ หากพัฒนาระบบนี้เป็นแบบออนไลน์มีข้อมูลอินพุท/เอาท์พุทเท่าเดิมจะเกิดปัญหาขึ้น เพราะหากใช้ FPA แบบ Albrecht ค่าก็ย่อมออกมาเท่ากัน กรณีปัญหานี้บอกให้รู้ว่า FPA ยังไม่คลอบคลุมไปในเรื่องที่เกี่ยวกับความสัมพันธ์ทางด้านเทคโนโลยี

2.10 แนวคิดยูนิไฟเอ็ดโพรเซส (Unified Process, UP)

A.J. Albrecht (1979) ยูนิไฟเอ็ดโพรเซสหรือยูพี (Unified Process, UP) โดยความหมายแล้ว คือ กระบวนการทางวิศวกรรมซอฟต์แวร์ที่เกิดจากการรวมเอาสิ่งที่ผู้เชี่ยวชาญทางการพัฒนาซอฟต์แวร์ (Software engineer) ที่เคยกำหนดไว้และใช้แล้วได้ผลดีในการพัฒนาซอฟต์แวร์นั้นมารวมกัน โดยเลือกแต่เทคนิคที่ดีและขั้นตอนหลักที่เหมือนๆกันและใช้ได้อย่างมีประสิทธิภาพของแต่ละผู้เชี่ยวชาญดังกล่าวมารวมกัน (Unify) และกำหนดให้มีชื่อใหม่ว่า “กระบวนการพัฒนาซอฟต์แวร์แบบรวมเป็นหนึ่งเดียว (Unified Process)” บางครั้งเราจะพบว่ามีกระบวนการคล้ายกันนี้

ในแวดวงทางวิศวกรรมซอฟต์แวร์ เช่น Rational Unified Process (RUP) ซึ่งเป็นของบริษัทยักษ์ใหญ่ในวงการพัฒนาซอฟต์แวร์ที่ชื่อว่า “Rational Rose Corporation” และถูกจดลิขสิทธิ์ด้วย หลักการหรือแนวคิดจะคล้ายกันแต่จะแตกต่างที่รายละเอียดของกระบวนการมากกว่า สำหรับแนวคิดที่มีลักษณะร่วมกันหรือเหมือนกันของยูพี ได้แก่ การพัฒนาแบบวนกลับ (Iterative Development), การจัดการกับความต้องการ (Requirement Management) และการใช้เครื่องมือช่วยทางวิศวกรรมซอฟต์แวร์(CASE Tools) เป็นต้น

เป้าหมายหรือวัตถุประสงค์ของยูพี คือ การที่ให้ได้ซอฟต์แวร์ที่มีคุณภาพและสอดคล้องกับความต้องการของผู้ใช้ โดยอยู่ภายใต้งบประมาณและเวลาที่สามารถคาดเดาได้ ยูพีนั้นจะเน้นการกำหนดบทบาท (Role) ไปที่ทีมพัฒนางานมากกว่าแต่ละบุคคล กล่าวคือจะมีการกำหนดว่าในแต่ละช่วง (Phase) ของการพัฒนานั้นว่า ควรประกอบไปด้วยใคร (Who) แต่ละคนมีหน้าที่รับผิดชอบอะไร (What) จะทำงานที่รับผิดชอบนั้นเมื่อไหร่ (When) และปฏิบัติอย่างไร (How) ที่กล่าวมาเป็นลักษณะที่เป็นนามธรรมกระบวนการยูพี ซึ่งอาจจะทำให้เข้าใจภาพยังไม่ชัดเจน กลยุทธ์ที่ใช้ในยูพีรวมแล้วจะเรียกมันว่า “Best Practice Model” หรือ “Best Practice” ก็ได้ มีการปฏิบัติ 6 อย่าง ดังนี้

1) การพัฒนาซอฟต์แวร์ควรเป็นการพัฒนาแบบวนกลับ (Iterative Development)

2) การพัฒนาซอฟต์แวร์ใดๆ ควรมีการจัดการความต้องการได้ (Requirement Management)

3) การใช้แนวคิดสถาปัตยกรรมแบบองค์ประกอบ (Component–based Model Architecture)

4) การสร้างต้นแบบของระบบที่สามารถมองเห็นได้ (Visual Model) ด้วยภาษา UML

5) การตรวจสอบคุณภาพของซอฟต์แวร์ที่พัฒนาอย่างต่อเนื่อง (Continuously Verify)

6) การจัดการการเปลี่ยนแปลง (Change Management)