Lecture 12 การประมาณการต้นทุนของซอฟต์แวร์

การประมาณการต้นทุนของซอฟต์แวร์ (Software Cost Estimation) เป็นกิจกรรมหนึ่งที่สำคัญที่สุดในการวางแผนโครงการ (Project Planning) ทีมงานจะต้องประมาณการค่าใช้จ่ายที่เกิดขึ้นทั้งหมดจากการผลิตซอฟต์แวร์ เพื่อนำมาคิดเป็นต้นทุนของซอฟต์แวร์ และนำไปใช้ประเมินราคาของซอฟต์แวร์ต่อไป ต้นทุนหรือค่าใช้จ่ายที่เกิดขึ้นจากการผลิตซอฟต์แวร์มีหลายรายการ ไม่ว่าจะเป็นค่าใช้จ่ายที่เกิดจากการซื้อวัตถุดิบทั่วไป ค่าใช้จ่ายในการเดินทาง หรือค่าใช้จ่ายเบ็ดเตล็ด แต่ค่าใช้จ่ายที่สำคัญที่สุด คือ “ค่าแรง (Effort)” ซึ่งผู้บริหารโครงการจะต้องกำหนดจำนวนของแรงงาน (ทีมงาน) ที่ต้องใช้ในแต่ละวัน (หรือเดือน) เพื่อการผลิตซอฟต์แวร์จนแล้วเสร็จ (จึงเรียกได้อีกอย่างหนึ่งว่า “ความเพียรพยายาม”) ค่าใช้จ่ายด้านแรงงาน เป็นองค์ประกอบหนึ่งของต้นทุนที่มีความไม่แน่นอนสูงมาก เนื่องจากปัจจัยหลายประการ เช่น ประสบการณ์ของทีมงาน โครงสร้างของโครงการ ความสามารถของแต่ละบุคคล เป็นต้น ปัจจัยดังกล่าวส่งผลให้ต้องใช้เวลาในการดำเนินงานมากขึ้น และทำให้ต้นทุนเพิ่มสูงขึ้นตามไปด้วย นอกจากนี้ ต้นทุนที่เกิดขึ้นบางครั้งอาจมีต้นทุนแฝง ซึ่งอาจเป็นค่าใช้จ่ายที่ไม่ชัดเจนเป็นส่วนประกอบด้วย อีกทั้งสภาพแวดล้อมในการผลิตซอฟต์แวร์ยังมีการเปลี่ยนแปลงตลอดระยะเวลาของโครงการ ทำให้ต้นทุนที่ประมาณการไว้ในตอนต้นมีการเปลี่ยนแปลง ผู้บริหารโครงการต้องปรับต้นทุนอยู่เสมอ การประมาณการต้นทุนจึงเป็นเรื่องยาก และจำเป็นต้องอาศัยเทคนิคในการประมาณการต่าง ๆ เข้ามาช่วย

12.1 การประมาณการต้นทุนของซอฟต์แวร์

Software Cost Estimation จัดทำขึ้นเพื่อนำไปประเมินราคาซอฟต์แวร์ ซึ่งเป็นสิ่งจำเป็นสำหรับการริเริ่มโครงการ สำหรับโครงการที่ต้องเข้าร่วมประมูล หากบริษัทผู้ผลิตซอฟต์แวร์ต้องการให้ชนะการประมูล จะต้องเสนอราคาซอฟต์แวร์หรือราคาโครงการที่ค่อนข้างต่ำกว่าคู่แข่งขัน แต่การเสนอราคาที่ต่ำกว่านั้นจะต้องไม่ทำให้บริษัทขาดทุนเมื่อเทียบกับต้นทุนที่ต้องจ่ายจริง ในอดีต การประเมินราคาซอฟต์แวร์ใช้สูตรแบบง่าย ๆ คือ ใช้ต้นทุนบวกด้วยกำไรที่ต้องการ แต่ไม่สามารถใช้ได้กับภาวะเศรษฐกิจในปัจจุบันที่มีปัจจัยหลายประการส่งผลกระทบต่อการกำหนดราคา ทำให้การประเมินราคาแบบเดิมคลาดเคลื่อน โดยอาจประเมินต่ำหรือสูงเกินไป การประเมินราคาซอฟต์แวร์นั้นเป็นสิ่งสำคัญสำหรับทุกโครงการ หากประเมินราคาต่ำทำให้บริษัทขาดทุน แต่หากประเมินสูงเกินไปก็ไม่สามารถชนะการประมูล หรือไม่ถูกเลือกให้ดำเนินงานต่อไปได้

สิ่งสำคัญในการประเมินราคาก็คือ “ต้นทุนของโครงการ (Project Cost)” ที่ไม่ได้หมายถึงเพียงต้นทุนที่ใช้ในการผลิตซอฟต์แวร์หรือต้นทุนทางตรงเท่านั้น แต่ยังหมายรวมถึงต้นทุนอื่น ๆ ด้วย โดยต้นทุนรวมของโครงการผลิตซอฟต์แวร์ ประกอบไปด้วย

1. ค่าใช้จ่ายด้านฮาร์ดแวร์ ซอฟต์แวร์ และการบำรุงรักษา (Hardware, Software and Maintenance Cost)
2. ค่าใช้จ่ายจากการเดินทาง และการฝึกอบรม (Travel and Training Cost)
3. ค่าใช้จ่ายในความเพียรพยายาม (Effort Cost: หรือค่าใช้จ่ายแรงงาน)

สำหรับค่าใช้จ่ายที่เกิดจาก Effort ที่ใช้ในการผลิตนั้น นอกจากจะหมายถึงค่าแรงหรือเงินเดือนบุคลากรในโครงการแล้ว ยังรวมถึงค่าใช้จ่ายอื่น ๆ เป็นส่วนประกอบด้วย ซึ่งก็คือ ค่าใช้จ่ายเบ็ดเตล็ด (Overhead) เช่น ค่าใช้จ่ายในการเตรียมงาน ค่าใช้จ่ายในการบริหารงาน การติดต่อสื่อสาร และสวัสดิการ เป็นต้น เมื่อโครงการเริ่มดำเนินการ ผู้บริหารโครงการจะต้องมีการปรับต้นทุนและตารางงานอย่างสม่ำเสมอ เพื่อจัดสรรการใช้ทรัพยากรให้เหมาะสมที่สุด

ปัจจัยที่ส่งผลกระทบต่อการกำหนดราคาซอฟต์แวร์ (Software Pricing) ในภาวะปัจจุบัน ได้แก่ โอกาสในด้านการตลาด (Market Opportunity) ข้อกำหนดในสัญญา (Contractual Term) ความต้องการของซอฟต์แวร์ที่เปลี่ยนแปลงได้ (Requirement Volatile) ฐานะทางการเงิน (Financial Health) และที่สำคัญที่สุดคือ การประมาณการต้นทุน (Cost Estimation) ซึ่งหากไม่มีความชัดเจนในตัวเลขต้นทุนที่ประมาณการได้ จะทำให้ราคาซอฟต์แวร์สูงเกินไป อาจทำให้เสียโอกาสในการประมูลได้

12.2 การประมาณขนาดของซอฟต์แวร์

สำหรับบางโครงการ ผู้บริหารโครงการอาจต้องใช้ค่าที่ได้จากการวัดประสิทธิผลในการทำงาน (Productivity) ซึ่งหมายถึงผลที่ได้จากการผลิตงานของบุคลากรในโครงการ ไปคำนวณหาต้นทุนหรือจัดตารางดำเนินงาน ตลอดจนอาจนำไปใช้ในการตัดสินใจประเมินกระบวนการหรือการนำเทคโนโลยีเข้ามาใช้ด้วย โดย Productivity วัดได้จากจำนวนหน่วยของงานที่ผลิตได้ หารด้วยจำนวนเวลาที่ต้องการใช้ในการผลิต (หรือ Effort นั่นเอง) อาจมีหน่วยเป็น Person-Hours หรือ Man-Day หรือ Man-Month ก็ได้ตามต้องการ

หากเป็นการผลิตสินค้าที่จับต้องได้ทั่วไป จำนวนหน่วยของงานจะเป็นรูปธรรมคือจำนวนสินค้าที่ผลิตได้ แต่สำหรับการผลิตซอฟต์แวร์ที่จับต้องไม่ได้ จำเป็นต้องวัดจากขนาดของซอฟต์แวร์ (Software Size) ดังนั้น จึงสามารถคำนวณ Productivity ในการผลิตซอฟต์แวร์ของบุคลากร ได้ดังนี้

Productivity = Size / Effort

สำหรับกรรมวิธีในการวัดขนาดของซอฟต์แวร์นั้น มี 2 ประเภท คือ วัดจากจำนวนบรรทัดของซอร์สโค้ด (Line of Code: LoC) และวัดจากจำนวนฟังก์ชันที่ใช้งานได้ (Function Point: FP) ดังรายละเอียดต่อไปนี้

12.2.1 การนับจำนวนบรรทัดของซอร์สโค้ด (Line of Code)

เป็นวิธีวัดขนาดซอฟต์แวร์วิธีแรกที่นำมาใช้ในยุคก่อน เนื่องจากภาษาโปรแกรมมิ่งในยุคนั้น มีลักษณะการเขียนแบบลำดับเป็นบรรทัดต่อเนื่องกันไป เช่น COBOL, ASSEMBLY, FORTRAN เป็นต้น การนับจำนวนบรรทัดของโค้ด จึงเป็นวิธีวัดขนาดซอฟต์แวร์ที่ง่ายและให้ผลชัดเจนที่สุด แต่เนื่องจากใน 1 โปรแกรมนั้น นอกจากจะมีบรรทัดที่เป็นชุดคำสั่งให้คอมพิวเตอร์ทำงานจริง ๆ แล้ว (Source Line Code) ยังประกอบไปด้วยบรรทัดที่เป็น Comment โปรแกรม การประกาศตัวแปรและฟังก์ชัน วิธีการนับจำนวนบรรทัดจึงแบ่งออกเป็นหลายวิธี ดังนี้

Simple Line Count เป็นวิธีการนับโค้ดที่ง่ายที่สุด เนื่องจากนับทุกบรรทัดที่อยู่ใน Source File โดยจะนับรวมบรรทัดว่างหรือบรรทัดที่เป็น Comment โปรแกรมด้วย

Physical Lines (LINES) เป็นวิธีการนับจำนวนบรรทัดของโค้ดที่ยังคงใช้กันอยู่ในการบริหารโครงการผลิตซอฟต์แวร์ โดยจะนับโค้ดทุกบรรทัด (ยกเว้นบรรทัดที่เป็นการนิยามตัวแปรของโค้ดที่เขียนด้วย)

Physical Line Of Code เป็นวิธีนับจำนวนบรรทัด แต่ไม่นับรวมบรรทัดว่างและบรรทัดที่เป็น Comment โปรแกรม บางครั้งจึงเรียกวิธีแบบนี้ว่า “Source Line Code (sLOC)”

Logical Lines of Code (LLOC) วิธีการนับแบบ Logical มีลักษณะคล้ายกับ Physical แตกต่างกันคือ Logical Lines นั้น จะนับบรรทัดที่มีการเชื่อมต่อกันด้วยอักขระ “_” รวมเป็นบรรทัดเดียว

Statements (STMT) วิธีการนี้ไม่ใช่การนับจำนวนบรรทัด แต่เป็นการนับจำนวนประโยคคำสั่ง โดยภาษาโปรแกรมมิ่งส่วนใหญ่จะมีเพียง 1 ประโยคคำสั่งต่อ 1 บรรทัด

การวัดขนาดซอฟต์แวร์ด้วยวิธีการนับจำนวนบรรทัด ปัจจุบันพบว่ายังคงถูกนำมาใช้เป็นดัชนีชี้วัดโครงการ (Project Metric) อยู่ แต่ด้วยวิธีการนับจำนวนบรรทัดในยุคก่อนนั้นยังมีข้อจำกัดอยู่บ้าง หากต้องเปรียบเทียบโค้ดที่เขียนด้วยภาษาโปรแกรมมิ่งที่แตกต่างกัน ดังนั้น จึงพบว่ามีการแบ่งแยกวิธีการนับออกเป็นหลายแบบข้างต้น อย่างไรก็ตาม วิธีการนับจำนวนบรรทัดยังแตกต่างออกไป ขึ้นอยู่กับผู้คิดค้นวิธีนับเหล่านั้น ทั้งนี้ ก็เพื่อให้ผลที่ได้จากการนับสามารถอธิบายขนาดของซอฟต์แวร์ได้ถูกต้องและชัดเจนที่สุด (วิธีนับจำนวนบรรทัดข้างต้น ในที่นี้อ้างอิงมาจากเครื่องมือช่วยวัดที่ชื่อว่า “Project Analyzer 7.1” ซึ่งเป็นเครื่องมือจัดทำดัชนีวัดโครงการผลิตซอฟต์แวร์) ปัจจุบันถึงแม้ว่าจะมีการพัฒนาเครื่องมือนับจำนวนบรรทัดแบบอัตโนมัติขึ้นมาหลายผลิตภัณฑ์ก็ตาม แต่ทุกผลิตภัณฑ์ไม่สามารถรองรับภาษาโปรแกรมมิ่งได้ครบถ้วน หรือบางผลิตภัณฑ์สามารถใช้ได้กับภาษาโปรแกรมมิ่งเพียงภาษาเดียวเท่านั้น *(ค้นหาเครื่องมือนับจำนวนบรรทัดสำหรับระบบปฏิบัติการ UNIX ได้ที่ http://sarovar.org/projects/kloc ส่วนระบบปฏิบัติการ Windows ค้นที่ได้ที่ http://www.anologx.com/contents/download/program/kloc.htm ส่วนเครื่องมืออื่น ๆ ที่ใช้นับจำนวนบรรทัดแบบ Physical และสนับสนุนภาษาโปรแกรมมิ่งที่หลายหลายเช่น C, Java, COBL, Fortran และ C# ค้นหาได้ที่ http://www.geronesoft.com)*

ข้อเสียของการประมาณการขนาดของซอฟต์แวร์ด้วยการนับจำนวนบรรทัด คือ จำนวนบรรทัดที่นับได้ ขึ้นอยู่กับภาษาโปรแกรมมิ่งที่ใช้และคุณภาพในการออกแบบโปรแกรม หากภาษาโปรแกรมมิ่งที่ใช้แตกต่างกันจะไม่สามารถเปรียบเทียบกันได้ บางภาษาต้องเขียนโค้ดหลายบรรทัดในการสร้างฟังก์ชันงานเดียว กับภาษาโปรแกรมมิ่งอื่นที่เขียนเพียงไม่กี่บรรทัด หากออกแบบโปรแกรมได้ดี อาจทำให้จำนวนบรรทัดของโค้ดน้อยกว่าการออกแบบโปรแกรมในลักษณะอื่นได้ ดังนั้นค่าที่ได้จากการนับจำนวนบรรทัด จึงไม่สามารถนำมาใช้ประมาณขนาดของซอฟต์แวร์ได้อย่างสมบูรณ์ ยกตัวอย่างโปรแกรม “Hello World” ที่เขียนจากภาษาโปรแกรมมิ่งที่แตกต่างกัน ดังนี้

C Programming Language	COBOL Programming Language

#include<stdio.h>
#include<conio.h>
void main()
{
printf(“Hello, World”);
getch();
}
	100 IDENTIFICATION DIVISION.
	200 PROGRAM-ID. EXAMHELLO.
	300 ENVIRONMENT DIVISION.
	400 CONFIGURATION SECTION.
	500 SOURCE-COMPUTER. KTP.
	600 OBJECT-COMPUTER. KTP.
	700 DATA DIVISION.
	800 FILE SECTION.
	900 PROCEDURE DIVISION.
	1000 MAIN-DISPLAY SECTION.
	2000 BEGIN.
	3000 DISPLAY “Hello World” LINE 10
	POSITION 10
	4000 STOP RUN.

เมื่อประมาณการขนาดของซอฟต์แวร์ได้แล้ว ก็สามารถนำมาคำนวณหาประสิทธิผลในการผลิตซอฟต์แวร์ได้ โดยนำขนาดของซอฟต์แวร์ (ในที่นี้คือจำนวนบรรทัดของโค้ดที่นับได้) มาหารด้วย Effort ยกตัวอย่างดังนี้

ตัวอย่างที่ 12.1 แสดงการคำนวณหา Productivity ของโปรแกรมเมอร์

	วิเคราะห์	ออกแบบ	เขียนโปรแกรม	ทดสอบ	จัดทำเอกสาร
โค้ด Assembly	4 สัปดาห์	6 สัปดาห์	10 สัปดาห์	12 สัปดาห์	2 สัปดาห์
โค้ดภาษาระดับสูง	4 สัปดาห์	6 สัปดาห์	5 สัปดาห์	7 สัปดาห์	2 สัปดาห์

	Size	Effort	Productivity
โค้ด Assembly	6,000 บรรทัด	34 สัปดาห์	705 บรรทัด/เดือน
โค้ดภาษาระดับสูง	2,500 บรรทัด	24 สัปดาห์	416 บรรทัด/เดือน
		• ***Effort ต้องหารด้วย 4 ก่อน เพื่อทำให้หน่วยเป็นเดือน***	

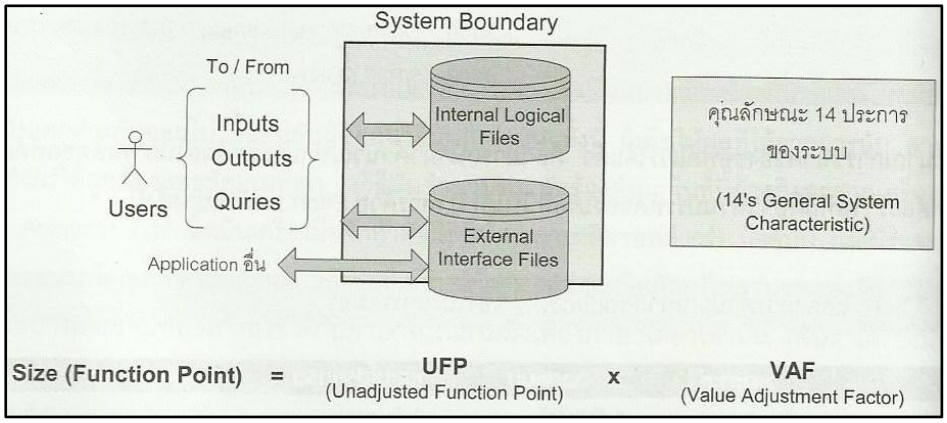
LoC ถูกคิดค้นขึ้นในยุคแรกของภาษาระดับสูง เช่น COBOL, FORTRAN เป็นต้น แต่ปัจจุบัน ภาษาโปรแกรมมิ่งถูกพัฒนาไปอย่างมาก ทำให้เกิดปัญหาความไม่เท่าเทียมกันของผลจากการวัดประสิทธิผลการผลิตงานซอฟต์แวร์ เนื่องจากภาษาโปรแกรมมิ่งที่โปรแกรมเมอร์ใช้แตกต่างกันมาก ในงานเดียวกันที่ให้ผลลัพธ์เหมือนกัน เมื่อเขียนด้วยภาษาโปรแกรมมิ่งที่แตกต่างกัน จำนวนบรรทัดจะแตกต่างกันมาก ดังตัวอย่างที่ 12.1 ทำให้ต้องมีการคิดค้นวิธีการวัดขนาดของซอฟต์แวร์ขึ้นมาใหม่ นั่นคือ “นับจำนวนฟังก์ชั่น (Function Point: FP)”

12.2.2 การนับจำนวนฟังก์ชั่น (Function Point: FP)

การนับจำนวนฟังก์ชัน (Function Point: FP) เป็นการวัดขนาดของซอฟต์แวร์ด้วยการนับจำนวนฟังก์ชันการทำงานของโปรแกรมจากข้อกำหนดความต้องการของซอฟต์แวร์ ดังนั้น วิธีการนับจำนวนฟังก์ชันจึงลดปัญหาด้านความแตกต่างของภาษาโปรแกรมมิ่งที่ใช้ และความแตกต่างในเทคโนโลยีด้านอื่น ๆ ด้วย วิธีการนับจำนวนฟังก์ชันถูกปรับปรุงโดยคณะกรรมการกำหนดและรวบรวมกรรมวิธีในการวัดขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยท์ (International Function Point User Group: IFPUG) โดยมีสูตรการคำนวณดังนี้

$$FP = UFP * VAF$$

จากสูตรการคำนวณ จำนวนของฟังก์ชันจะหาได้จากค่า FP ที่ยังไม่ได้ถูกปรับแต่ง (Unadjusted Function Point: UFP) คูณกับค่าปัจจัยคุณลักษณะของระบบ (Value Adjustment Factor: VAF) ดังรูป



การนำค่าปัจจัยคุณลักษณะของระบบมาคำนวณด้วยนั้น เนื่องจากคุณลักษณะเด่นของแต่ละระบบแตกต่างกัน ทำให้ความยากง่ายในการผลิตซอฟต์แวร์แตกต่างกัน หากไม่นำปัจจัยเหล่านี้มาคำนวณด้วย ค่า FP ที่ได้ในแต่ละระบบจะไม่เท่าเทียมกัน ทำให้ค่า FP ไม่สามารถบ่งบอกถึงประสิทธิผลในการทำงานของโปรแกรมเมอร์แต่ละคน ได้อย่างแท้จริง จากสูตรการคำนวณ ในที่นี้ขอแบ่งขั้นตอนการคำนวณออกเป็น 3 ขั้นตอน คือ คำนวณหา FP ที่ยังไม่ได้ปรับแต่ง (UFP) คำนวณค่าปัจจัยคุณลักษณะของระบบ (VAF) และคำนวณค่า FP ที่ปรับแต่งแล้ว

1. **คำนวณหาค่า FP ที่ยังไม่ได้ปรับแต่ง (UFP)**

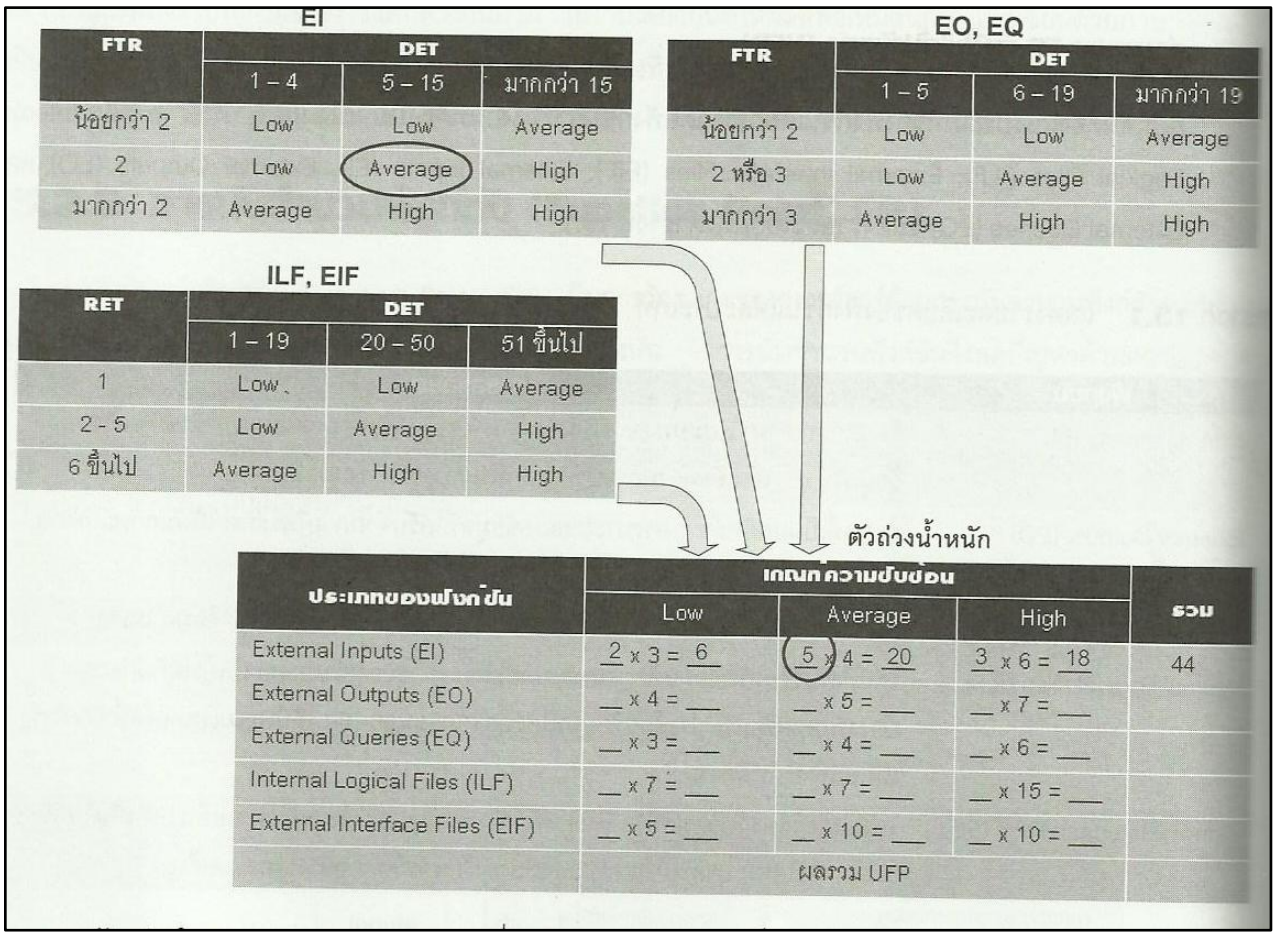
   ก่อนอื่นต้องแบ่งประเภทของฟังก์ชันเสียก่อน โดยฟังก์ชันการทำงานที่ต้องนับ แบ่งเป็น 5 ประเภท ได้แก่ Internal Logical Files (ILF), External Interface Files (EIF), External Inputs (EI), External Outputs (EO) และ External Queries (EQ) แสดงรายละเอียดดังตารางที่ 12.1

ตารางที่ 12.1 แสดงรายละเอียดของฟังก์ชันแต่ละประเภท

ฟังก์ชัน	รายละเอียด
External Inputs (EI)	ข้อมูลที่รับเข้ามาในระบบ (อาจเป็นข้อมูลทางธุรกิจหรือข้อมูลควบคุม) เพื่อนำไปอัพเดทข้อมูลใน ILF เช่น ข้อมูลในกระบวนการ เพิ่ม ลบ แก้ไข ข้อมูล เป็นต้น
External Outputs (EO)	ข้อมูลที่เป็นผลลัพธ์จากการประมวลผลข้อมูลที่ได้รับจากภายในระบบ ให้นับการแสดงผลข้อมูลที่มีรูปแบบแตกต่างกัน
External Queries (EQ)	กระบวนการดึงข้อมูลและประมวลผลเพื่อแสดงผลต่อผู้ใช้ (คือ Query ข้อมูลนั่นเอง)
Internal Logical Files (ILF)	ไฟล์ที่เกี่ยวข้องกับข้อมูลที่อยู่ในระบบตลอดช่วงอายุของระบบ และเป็นไฟล์ที่มักจะถูกบำรุงรักษาหรือปรับปรุงด้วยข้อมูลที่ได้รับจากภายนอก (EI) ให้นับรวมเรคคอร์ดที่ทำหน้าที่เทียบเท่ากับไฟล์ด้วย
External Interface Files (EIF)	ไฟล์ที่เกี่ยวข้องกับข้อมูลที่ใช้เพื่อการอ้างอิงเท่านั้น และใช้ร่วมกับระบบอื่น EIF เป็นไฟล์ที่ถูกเรียกใช้โดยระบบที่จะพัฒนา แต่จะบำรุงรักษาหรือถูกสร้างโดยระบบอื่น

ฟังก์ชันแต่ละประเภทเกิดจากการทำรายการข้อมูล (Transaction) ของผู้ใช้ จึงมีความซับซ้อนแตกต่างกันตามจำนวนของข้อมูล (Data Element Type: DET) เรคคอร์ด (Record Element Type: RET) และไฟล์ที่เกี่ยวข้อง (File Type Reference: FTR) ที่ประกอบขึ้นเป็น Transaction แต่ละรายการ ดังนั้น การนับฟังก์ชันแต่ละประเภท จึงต้องนับที่จำนวนของ DET, RET และ FTR ที่เกี่ยวข้องกับฟังก์ชันแต่ละประเภท แล้วนำมาเทียบกับตารางเกณฑ์ระดับความซับซ้อนของฟังก์ชัน ซึ่งแบ่งเป็น 3 ระดับ คือ ระดับต่ำ (Low) ปานกลาง (Average) และสูง (High) ในฟังก์ชันแต่ละประเภทจะถูกวัดขนาดความซับซ้อนออกมาได้หลายระดับ ให้นับว่าแต่ละฟังก์ชันมีระดับ ต่ำ กลาง และสูงจำนวนเท่าใด แล้วนำมาคูณกับตัวถ่วงน้ำหนักของแต่ละระดับในฟังก์ชันแต่ละประเภท ตามตารางตัวถ่วงน้ำหนัก จากนั้น หาผลรวมฟังก์ชันทั้งหมดที่นับได้ก็จะได้ค่า UFP (ค่า FP ที่ยังไม่ได้ปรับแต่ง)

ตารางที่ 12.2 แสดงตารางเกณฑ์ระดับความซับซ้อนของฟังก์ชันแต่ละประเภท ตารางถ่วงน้ำหนัก และการคำนวณหาค่า UFP



EI

FTR	DET		
	1 – 4	5 – 15	มากกว่า 15
น้อยกว่า 2	Low	Low	Average
2	Low	Average	High
มากกว่า 2	Average	High	High

EO, EQ

FTR	DET		
	1 – 5	6 – 19	มากกว่า 19
น้อยกว่า 2	Low	Low	Average
2 หรือ 3	Low	Average	High
มากกว่า 3	Average	High	High

ILF, EIF

RET	DET		
	1 – 19	20 – 50	51 ขึ้นไป
1	Low	Low	Average
2 - 5	Low	Average	High
6 ขึ้นไป	Average	High	High

ตัวถ่วงน้ำหนัก

ประเภทของฟังก์ชัน	เกณฑ์ความซับซ้อน			รวม
	Low	Average	High	
External Inputs (EI)	2 x 3 = 6	5 x 4 = 20	3 x 6 = 18	44
External Outputs (EO)	__ x 4 = ___	__ x 5 = ___	__ x 7 = ___	
External Queries (EQ)	__ x 3 = ___	__ x 4 = ___	__ x 6 = ___	
Internal Logical Files (ILF)	__ x 7 = ___	__ x 7 = ___	__ x 15 = ___	
External Interface Files (EIF)	__ x 5 = ___	__ x 10 = ___	__ x 10 = ___	
	ผลรวม UFP			

จากตัวอย่างในตาราง สมมติ ข้อมูลสินค้าที่จะนำเข้าสู่ระบบ (EI) เกี่ยวข้องกับไฟล์ 2 ชนิด (FTR) และข้อมูลสินค้านี้ประกอบไปด้วยฟิลด์ข้อมูลไม่เกิน 15 ฟิลด์ (DET) เมื่อเทียบกับตาราง EI แล้วพบว่าฟังก์ชัน EI นี้มีระดับความซับซ้อนอยู่ที่ Average แต่ข้อมูลที่จะนำเข้าสู่ระบบทั้งหมดมี 10 ชนิด (คือ นอกจากข้อมูลสินค้าแล้วยังมีข้อมูลอื่น ๆ ด้วย) เมื่อประเมินแล้วพบว่า อยู่ในระดับ “Low” 2 ชนิด อยู่ในระดับ “Average” 5 ชนิด และระดับ “High” 3 ชนิด เมื่อนำไปคูณกับตัวถ่วงน้ำหนักแล้วรวมกันทั้งหมดได้ค่าความซับซ้อนของ EI เท่ากับ 44 จึงดำเนินการหาค่าความซับซ้อนของข้อมูลชนิดต่อไปจนครบเพื่อหาผลรวม UFP ของข้อมูลทุกชนิด

2. **คำนวณหาค่าปัจจัยคุณลักษณะของระบบ (VAF)**

ก่อนคำนวณหาค่า FP สุดท้าย ให้คำนวณค่าปัจจัยที่ส่งผลต่อความแตกต่างกันของแต่ละระบบ ปัจจัยดังกล่าวคือ คุณลักษณะเด่นของระบบทั้งหมด 14 ด้านตามข้อกำหนดความต้องการที่ลูกค้าต้องการ โดยให้กำหนดระดับอิทธิพลของคุณลักษณะในแต่ละด้านว่ามีความเกี่ยวข้องกับระบบมากน้อยเพียงใด โดยมีค่าตั้งแต่ 0 (ไม่เกี่ยวข้อง) ถึง 5 (เกี่ยวข้องมาก)

ตารางที่ 12.3 คุณลักษณะเด่นของระบบทั้ง 14 ด้าน ค่าที่ประเมินตั้งแต่ 0 (ไม่เกี่ยวข้อง) ถึง 5 (เกี่ยวข้องมาก)

	คุณลักษณะ	ค่า		คุณลักษณะ	ค่า
1	การติดต่อสื่อสารข้อมูล (Data Communication)		8	การปรับปรุงข้อมูลแบบออนไลน์ (Online Update)	
2	การประมวลผลข้อมูลแบบกระจาย (Distribution Data Processing)		9	ความซับซ้อนของการประมวลผล (Complex Processing	
3	ประสิทธิภาพของระบบ (Performance)		10	การนำไปใช้ซ้ำได้ (Reusability)	
4	การแก้ไขค่าของระบบ (Configuration)		11	ความง่ายในการติดตั้ง (Installation Ease)	
5	ปริมาณรายการข้อมูล (Transaction)		12	ความง่ายในการดำเนินงาน (Operational Ease)	
6	การป้อนข้อมูลเข้าสู่ระบบแบบออนไลน์ (Online Data Entry)		13	การใช้งานได้หลายไซต์ (Multiple Sites)	
7	ประสิทธิภาพการใช้งานของผู้ใช้ (End-user Effciency)		14	รองรับการเปลี่ยนแปลงความต้องการของผู้ใช้ (Change Requirement)	

จากนั้น ให้รวมระดับอิทธิพลทั้ง 14 ด้านเข้าด้วยกัน แล้วนำมาคำนวณหา VAF ตามสูตรคำนวณ ดังนี้

VAF = 0.65 + [0.01 * ผลรวมค่าคุณลักษณะ 14 ด้าน]

3. **คำนวณหาค่า FP ที่ปรับแต่งแล้ว**

เมื่อคำนวณหา UFP และ VAF แล้ว นำมาคูณกัน จะได้ผลลัพธ์เป็นค่า FP ที่ปรับแต่งตามคุณลักษณะเด่นของระบบ ตามสูตร FP = UFP * VAF

ทั้ง LoC และ FP คือ วิธีวัดขนาดของซอฟต์แวร์ โดยสามารถนำไปคำนวณหา Productivity และ Effort ต่อไปได้ อย่างไรก็ตาม สูตรคำนวณเพื่อประมาณการ Effort นั้นบางครั้งอาจต้องการใช้ขนาดของซอฟต์แวร์ที่เป็น LoC ดังนั้น ค่า FP ที่หาได้ อาจต้องแปลงไปเป็น LoC ซึ่งมีตารางเปรียบเทียบ ดังนี้

ตารางที่ 12.4 ตารางเปรียบเทียบค่า FP เพื่อแปลงไปเป็น LoC ตามมาตรฐานของ QSM (Quantitiative Software Management: www.qsm.com)

ภาษาโปรแกรมมิ่ง	LoC ต่อ 1 FP			
	Average	Medium	Low	High
Access	35	38	15	47
Ada	154	-	104	205
ASP	59	62	32	127
C	148	104	9	704
C++	60	53	29	178
C#	59	59	51	66
Clipper	38	39	27	70

COBOL	73	77	8	400
FoxPro	32	35	25	35
HTML	43	42	35	53
Informix	42	31	24	57
J2EE	61	50	50	100
Java	60	59	14	97
JavaScript	56	54	44	65
JSP	59	-	-	-
Oracle	38	29	4	122
Perl	60	-	-	-
PL/1	59	59	22	92
PL/SQL	46	31	14	110
Powerbuilder	30	24	7	105
SQL	39	35	15	143
VBScript	45	34	27	50
Visual Basic	50	42	14	276

12.3 เทคนิคการประมาณการต้นทุนและ Effort

การประมาณการต้นทุน และ Effort ที่ดีจะต้องใกล้เคียงกับความเป็นจริง แต่การประมาณการให้ใกล้เคียงกับค่าจริงนั้นเป็นสิ่งที่ทำได้ยาก จึงได้มีการคิดค้นเทคนิคในการประมาณการต้นทุนและ Effort ขึ้นมาหลายเทคนิค ดังตัวอย่างต่อไปนี้

ตารางที่ 12.5 ตารางแสดงตัวอย่างเทคนิคในการประมาณการต้นทุนและ Effort

เทคนิค	รายละเอียด
Algorithm Cost Modeling	การใช้แบบจำลองทางคณิตศาสตร์เพื่อการประมาณการ โดยแบบจำลองนั้นถูกพัฒนาจากการรวบรวมข้อมูลต้นทุนที่ใช้จริงในอดีตที่มีความสัมพันธ์กับหน่วยวัดบางอย่างของซอฟต์แวร์ เช่น ขนาดของซอฟต์แวร์ เพื่อใช้เป็นตัวแปรที่ส่งผลต่อการประมาณการต้นทุน
Expert Judgment	การใช้ผู้เชี่ยวชาญประมาณการ โดยใช้ความเห็นของผู้เชี่ยวชาญ เปรียบเทียบกับข้อมูลในอดีต ปรึกษา และตกลงกำหนดต้นทุนร่วมกัน
Estimation by Analogy	การประมาณการด้วยการวิเคราะห์ โดยอาศัยข้อมูลจากโครงการในธุรกิจเดียวกันที่ดำเนินการประสบความสำเร็จแล้วมาเป็นข้อมูลหลักในการวิเคราะห์
Parkinson's Law	กฎของพาร์คินสัน คือ การกระจายงานให้กับบุคลากรตามระยะเวลาที่มีอยู่ เช่น ถ้าต้องส่งมอบซอฟต์แวร์ภายใน 12 เดือน และมีบุคลากรอยู่เพียง 5 คน จะต้องจัดสรรบุคลากรทั้ง 5 ให้ทำงานได้เท่ากับ 60 Person-Month
Pricing to Win	การประมาณการเพื่อให้ชนะการประมูล กล่าวคือ ต้องประมาณการต้นทุนให้ต่ำที่สุดเพื่อจะได้กำหนดราคาซอฟต์แวร์ให้ต่ำกว่าคู่แข่งขัน โดยไม่สนใจว่าการประมาณการนี้จะถูกต้องหรือไม่

ตัวอย่างเทคนิคในตารางข้างต้น เป็นเทคนิคที่อาศัยความรู้และประสบการณ์หรือข้อมูลในอดีตเป็นส่วนใหญ่ แต่โครงการในปัจจุบันนั้นมีความแตกต่างจากในอดีตเป็นอย่างมาก วิธีการดังกล่าวจึงคลาดเคลื่อนสูง ในที่นี้จึงขอ

กล่าวถึงการประมาณการด้วยแบบจำลอง หรือสูตรทางคณิตศาสตร์ที่เรียกว่า “COCOMO Model” ซึ่งมีการนำปัจจัยที่ส่งผลต่อต้นทุนหลายประการมาคำนวณด้วย

12.4 เทคนิคการประมาณการแบบ COCOMO

COCOMO (Constructive Cost Model) เป็นแบบจำลองประมาณการ Effort ต้นทุน และจัดตารางการทำงาน ที่คิดค้นโดย Dr. Barry Boehm ในปี ค.ศ. 1981 โดยการพิจารณาจากขนาดของซอฟต์แวร์และคุณลักษณะของซอฟต์แวร์ที่ผู้ใช้ต้องการ เนื่องจากความซับซ้อนและคุณลักษณะของซอฟต์แวร์เริ่มมีมากขึ้น อันเป็นผลมาจากแนวทางเครื่องมือ และเทคโนโลยีที่ใช้ผลิตซอฟต์แวร์นั้นถูกพัฒนาขีดความสามารถมากขึ้น ไม่ว่าจะเป็นเทคโนโลยีการออกแบบเชิงวัตถุ (Object-oriented Design: OOD) การผลิตซอฟต์แวร์เป็นคอมโพเน้นท์ รวมถึงเครื่องมือทดสอบโปรแกรมแบบอัตโนมัติ

การประมาณการต้นทุนและ Effort ด้วยแบบจำลอง COCOMO นั้น เป็นการคำนวณจากขนาดของซอฟต์แวร์ร่วมกับปัจจัยแวดล้อมอื่น ๆ ที่เกี่ยวข้อง เช่น ความแน่นอนของกระบวนการ และความสามารถในการผลิตซอฟต์แวร์ของทีมงาน ความยืดหยุ่น ความเสี่ยง และวิธีจัดการกับความเสี่ยง เป็นต้น นอกจากนี้ แบบจำลอง COCOMO ยังมีการคำนวณแบบเอ็กซ์โปเนนเชียล (Exponential) เนื่องจาก Dr.Barry Boehm พบว่า Effort กับขนาดของซอฟต์แวร์มีความสัมพันธ์กันแบบไม่เป็นเส้นตรง กล่าวคือ Effort แปรผันตามขนาดของซอฟท์แวร์แบบยกกำลัง เช่น หากเพิ่มขนาดของซอฟต์แวร์ นอกจากจะทำให้จำนวนบุคลากรในทีมงานเพิ่มขึ้นแล้ว ยังอาจทำให้ค่าใช้จ่ายอื่น ๆ เพิ่มขึ้นตามไปด้วย ไม่ว่าจะเป็นค่าใช้จ่ายในการติดต่อสื่อสาร ค่าใช้จ่ายในการบริหาร ตลอดจนค่าใช้จ่ายในการรวมระบบ (เรียกค่าใช้จ่ายเหล่านี้ว่า “Overhead”) เป็นต้น นอกจากขนาดของซอฟต์แวร์และปัจจัยแวดล้อม ที่ส่งผลต่อ Effort ทั้งการเพิ่มขึ้นหรือลดลงของ Effort แล้ว ยังมีปัจจัยอื่นที่ต้องพิจารณาเพิ่มเติม เช่น คุณลักษณะของซอฟต์แวร์ คุณลักษณะของ Platform คุณลักษณะของทีมงาน และคุณลักษณะของการบริหารโครงการ เป็นต้น แล้วให้น้ำหนักกับคุณลักษณะดังกล่าวเพื่อปรับค่าจำนวน Effort ที่เหมาะสมที่สุด

แบบจำลอง COCOMO ถูกพัฒนาเป็นเวอร์ชั่น 2 คือ COCOMO II ตั้งแต่ปี ค.ศ. 1997 โดยรวบรวมข้อมูลจากโครงการทั้งหมด 161 โครงการ จำแนกตามขนาดของโครงการ และใช้หลักการวิเคราะห์ทางสถิติ (Bayesian Statistical Analysis) วิเคราะห์ข้อมูลของโครงการที่ประสบความสำเร็จเหล่านั้น รวมทั้งความคิดเห็นของผู้เชี่ยวชาญต่าง ๆ ด้วย COCOMO II แบ่งแบบจำลองออกเป็น 3 ชนิด เพื่อใช้ประมาณการในระยะต่าง ๆ ของกระบวนการพัฒนาซอฟต์แวร์

1. Application-composition Model เหมาะกับการผลิตซอฟต์แวร์ตามแนวทางคอมโพเน้นท์ (Component-based Development) และอยู่ในระยะสรุป Concept ในการดำเนินงาน ใช้ Object Point แทนขนาดของซอฟต์แวร์

2. Early Design Model ใช้ประมาณการในระยะก่อนการออกแบบซอฟต์แวร์ แต่หลังจากการกำหนดความต้องการเรียบร้อยแล้ว ใช้ค่า FP แทนขนาดของซอฟต์แวร์

3. Post-architecture Model ใช้ประมาณการในระยะหลังการออกแบบซอฟต์แวร์ เป็นการประมาณการอีกครั้งเพื่อความถูกต้องของค่าประมาณการที่ได้

อย่างไรก็ตาม ในระบบงานหรือโครงการขนาดใหญ่ อาจแบ่งการประมาณการออกเป็นส่วนย่อย โดยในแต่ละส่วนสามารถใช้เทคนิคในการประมาณการที่แตกต่างกันได้ จากนั้นจึงนำผลที่ได้ในแต่ละส่วนมารวมกัน โดยที่ผู้บริหารโครงการไม่จำเป็นต้องประเมินทุกส่วนก็ได้ ขึ้นอยู่กับความเหมาะสมและข้อตกลง

Application-composition Model

Application-composition Model เป็นแบบจำลอง COCOMO II ที่เหมาะกับการผลิตซอฟต์แวร์ด้วยแนวทางคอมโพเน้นท์ โดยแต่ละคอมโพเน้นท์สามารถอธิบายแทนด้วย Object Point ได้ นั่นคือ ขนาดของซอฟต์แวร์จะต้องนับเป็นอ็อบเจ็กพอยท์ เป็นการนับจำนวนอ็อบเจ็กต์ ที่หมายถึงคอมโพเน้นท์3 ส่วน ได้แก่ หน้าจอ (Screen) รายงาน (Report) และโมดูลที่เขียนด้วยภาษาโปรแกรมมิ่งในยุค 3GL (Third Generation Language) ขึ้นไป ดังนั้นอ็อบเจ็กต์ในที่นี้จึงต่างจากอ็อบเจ็กต์ตามการนิยามของเทคโนโลยีเชิงวัตถุ(Object-oriented Technology) อ็อบเจ็กต์คอมโพเน้นท์เหล่านี้ จะมีจำนวนอ็อบเจ็กต์พอยท์แตกต่างกัน ขึ้นอยู่กับระดับความซับซ้อน โดยแบ่งออกเป็น3 ระดับ แต่ละระดับมีคะแนนความซับซ้อน ดังตารางต่อไปนี้

ตารางที่ 12.6 แสดงระดับความซับซ้อนของอ็อบเจ็กต์พอยท์

	ง่าย (Simple)	**ซับซ้อน (Complex)**	**ซับซ้อนมาก (Very Complex)**
Screen	1	2	3
Reports	2	5	6
3GL Modules	4	10	-

กรณีคอมโพเน้นท์ของซอฟต์แวร์ถูกออกแบบให้มีการนำกลับมาใช้ใหม่ และมีการใช้งานไลบรารีด้วย จะต้องนำอัตราการนำไปใช้ใหม่มาลบออกจากจำนวนอ็อบเจ็กต์พอยท์ที่นับได้ทั้งหมด ผลลัพธ์ที่ได้คือ ค่าของอ็อบเจ็กต์พอยท์ที่ถูกปรับลดแล้ว หรือเรียกว่า “Revised Object Point (ROP)” มีสูตรคำนวณ ดังนี้

Revised Object Point (ROP) = Object Point x [(100 - %reuse)/100]

จากนั้นให้นำ ROP ที่ได้ไปคำนวณหา Effort ดังนี้

MME (Man Month Effort) = [ROP / Productivity Constant]

โดย Effort ที่ได้ จะมีหน่วยเป็น Man-Month หาได้จากอัตราส่วนระหว่าง ROP กับค่าคงที่ของประสิทธิผลในการผลิตซอฟต์แวร์ (Productivity Constant) ซึ่งคิดจากประสบการณ์และความสามารถของทีมงาน มีค่าคงที่ดังตารางต่อไปนี้

ตารางที่ 12.7 แสดงค่าคงที่ของประสิทธิผลในการผลิตซอฟต์แวร์ โดยพิจารณาจากระดับประสบการณ์และความสามารถของทีมงาน

ระดับประสบการณ์และความสามารถ	**ต่ำมาก (Very Low)**	**ต่ำ (Low)**	**ปานกลาง (Nominal)**	**สูง (High)**	**สูงมาก (Very High)**
Productivity Constant (NOP per Month)	4	7	13	25	50

ประสิทธิผลในการผลิตซอฟต์แวร์ จะมีหน่วยเป็นจำนวนของอ็อบเจ็กต์พอยท์ต่อเดือน(Number of Object Point per Month)

ยกตัวอย่างเช่น ในระยะการก่อนการดำเนินงาน **โครงการ ก** นับจำนวนอ็อบเจ็กต์พอยท์ได้ 40 OP มีอัตราการนำโค้ดไปใช้ใหม่ 10% และเมื่อประเมินประสบการณ์และความสามารถของทีมงานแล้ว พบว่าอยู่ในระดับปานกลาง (Nominal/Normal) สามารถคำนวณหา Effort ที่ต้องการใช้ในโครงการได้ ดังนี้

$$ROP = 40 \times \frac{(100 - 10)}{100} = 40 \times 0.90 = 36$$

$$MME(ManMonthEffort) = \frac{36}{13} = 3\ Man\ Months$$

การประมาณการ Effort ด้วยแบบจำลองชนิดนี้ นอกจากจะใช้ในระยะการหา Concept ในการดำเนินงานแล้ว ยังสามารถใช้ประมาณการจำนวนแรงงานจากโปรโตไทป์ที่สร้างขึ้นมา เพื่อเสนอลูกค้า ในกรณีที่ข้อมูลความต้องการของลูกค้ายังไม่ชัดเจนได้อีกด้วย

Early Design Model

เป็นแบบจำลอง COCOMO II ที่ใช้ในระยะก่อนการออกแบบซอฟต์แวร์ โดยมีสูตรคำนวณพื้นฐาน ดังนี้

$$MME = A \times (Size)^B$$

โดยที่ MME คือ Effort ที่มีหน่วยเป็น Man-Month (Man Month Effort)

A คือ ค่าคงที่ของประสิทธิผลในการผลิตซอฟต์แวร์ คิดที่ระดับปานกลาง (Nominal)

B คือ ค่าของปัจจัยที่ส่งผลกระทบให้ Effort และขนาดแปรผันตรงต่อกันแบบไม่เป็นเส้นตรง (Exponential) เรียกปัจจัยเหล่านี้ว่า “Scaling Factor” หรือ “Economics Scale” หรือ “Cost Driver” ในที่นี้ขอเรียกว่า “ปัจจัยขับ”

Size คือ ขนาดของซอฟต์แวร์ มีหน่วยเป็น KLoC (Kilo of Line of Code = Loc x 1000)

ตามที่เคยกล่าวไปแล้วว่า COCOMO II ได้นำปัจจัยอื่น ๆ มาคำนวณด้วย เนื่องจากพบว่า ปัจจัยดังกล่าวทำให้ขนาดและ Effort แปรผันต่อกันแบบไม่เป็นเส้นตรง ปัจจัยดังกล่าวเรียกว่า “Scaling Factor” หรือ ค่าของปัจจัยขับแสดงแทนด้วย B จากสูตรคำนวณ จะสังเกตว่า B เป็นเลขชี้กำลังของขนาดซอฟต์แวร์ ดังนั้น B จะส่งผลให้ Size of Software เปลี่ยนแปลง โดยจะส่งผลให้ Effort เปลี่ยนแปลงไปด้วย ดังนี้

- ถ้า B = 1 หมายถึง Scaling Factor ไม่ส่งผลกระทบต่อขนาดซอฟต์แวร์ (ไม่ทำให้ขนาดซอฟต์แวร์เปลี่ยนแปลง)

- ถ้า B > 1 หรือ **B < 1** หมายถึง Scaling Factor ส่งผลกระทบให้ขนาดของซอฟต์แวร์เปลี่ยนแปลงขึ้นหรือลง

ปัจจัยขับที่นำมาใช้ใน Early Design Model มีทั้งหมด 5 ปัจจัย โดยเบื้องต้นจะต้องประเมินระดับและให้คะแนนแต่ละปัจจัย (Rating) จากนั้นหาผลรวมของคะแนนที่ประเมินได้ในปัจจัยทั้งหมด แล้วนำมาคำนวณหาค่าของปัจจัยที่แทนด้วย B จากสูตรคำนวณ ดังนี้

$$B = 0.91 + 0.01(\sum_{1}^{5} Ratings)$$

ระดับในการประเมินปัจจัยทั้ง 5 มี 4 ระดับ ได้แก่ ต่ำมาก(Very Low) ต่ำ (Low) ปานกลาง (Nominal) และ สูง (High) แต่ละระดับมีคะแนน ดังตารางต่อไปนี้

ตารางที่ 12.8 แสดงคะแนนของปัจจัยแต่ละระดับ (Value of Rating for Scaling Factor)

Factor Code	ต่ำมาก (Very Low)	ต่ำ (Low)	ปานกลาง (Nominal)	สูง (High)	Factor Name
PREC	6.20	4.96	3.72	2.48	Precedentness
FLEX	5.07	4.05	3.04	2.03	Flexibility
RESL	7.07	5.65	4.24	2.83	Risk Resolution
TEAM	5.48	4.38	3.29	2.19	Team Cohesion
PMAT	7.80	6.24	4.68	3.12	Process Maturity

สำหรับความหมายของ ปัจจัยขับ ทั้ง 5 มีดังนี้

ตารางที่ 12.9 แสดงปัจจัยขับสำหรับ COCOMO II ในระยะ Early Design

ปัจจัย	รายละเอียด
PREC	ความเหมือนของซอฟต์แวร์ใหม่กับซอฟต์แวร์เดิมที่เคยพัฒนามาแล้ว (เหมือนมาก คะแนนน้อยอยู่ในระดับสูง แปลว่า ผลกระทบน้อย แต่ถ้าเหมือนน้อย อยู่ในระดับต่ำ คะแนนสูง เพราะส่งผลกระทบมาก)
FLEX	การวัดระดับความยืดหยุ่นในการบริหารจัดการและดำเนินโครงการ (เช่น การใช้เครื่องมือ)
RESL	การวัดระดับความสามารถในการจัดการหรือควบคุมความเสี่ยงขององค์กรหรือทีมงานของโครงการ
TEAM	การวัดระดับของการทำงานเป็นทีมขององค์กรหรือทีมงานโครงการ
PMAT	การวัดระดับวุฒิภาวะความสามารถขององค์กร หรือทีมงานโครงการ ตั้งแต่ระดับต่ำสุดคือ 1 จนถึงระดับสูงสุดคือ 5

ตัวอย่างที่ 12.2 สมมติว่า ปัจจัยขับทั้ง 5 ข้อ ถูกจัดอันดับให้อยู่ในระดับต่ำมาก (Very Low) ทั้งหมด และกำหนดให้ขนาดของซอฟต์แวร์ที่นับแบบฟังก์ชันพอยท์และแปลงมาเป็น LoC แล้วมีค่าเท่ากับ 10 KLoC สามารถคำนวณหาแรงงานโดยประมาณ บนพื้นฐานของค่าคงที่ของประสิทธิผลในการผลิตที่ระดับปกติ(Nominal) ได้ดังนี้

หาค่าระดับคะแนนรวมของปัจจัยขับ หรือค่า B ก่อน ดังนี้

$$B = 0.91 + 0.01 \times (6.20 + 5.07 + 7.07 + 5.48 + 7.80)$$
$$= 0.91 + 0.01 \times 31.62$$
$$= 1.2262$$

สังเกต B มีค่ามากกว่า 1 ทำให้ทราบในเบื้องต้นว่า ปัจจัยขับส่งผลกระทบต่อขนาดของซอฟต์แวร์และ Effort แน่นอน ต่อไปคำนวณหา Effort โดยประมาณ ดังนี้

$$MME = A \times (Size)^{B}$$
$$= 13 \times (10)^{1.2262} \quad ** \ A = 13 \text{ เทียบจากตารางที่ } 12.7 \ **$$
$$= 13 \times 16.8344$$
$$= 218.84 \text{ หรือ ประมาณ } 219 \text{ Man Month}$$

ตัวอย่างที่ 12.3 ในทางตรงกันข้ามกับตัวอย่างที่ 12.2 สมมติ ปัจจัยขับทั้ง 5 ข้อ ถูกจัดอันดับให้อยู่ในระดับสูง (High) ทั้งหมด และกำหนดให้ขนาดของซอฟต์แวร์ที่นับแบบฟังก์ชันพอยท์มีค่าเท่าเดิม สามารถคำนวณหาแรงงานโดยประมาณ บนพื้นฐานของค่าคงที่ของประสิทธิผลในการผลิตที่ระดับปกติ(Nominal) ได้ดังนี้

B = 0.91 + 0.01 x (2.48 + 2.03 + 2.83 + 2.19 + 3.12)
= 0.91 + 0.01 x 12.65
= 1.0365 หรือประมาณ 1.04

MME = A x (Size)^{B}
= 13 x (10)$^{1.04}$ ** *A = 13 เทียบจากตารางที่ 12.7* **
= 13 x 10.9647
= 142.54 หรือ ประมาณ 143 Man Month

จะเห็นว่า ค่า B มากกว่า 1 เช่นกัน แต่มากกว่าเพียงเล็กน้อยเท่านั้น ดังนั้น จึงส่งผลให้ขนาดของซอฟต์แวร์เพิ่มขึ้นเพียงเล็กน้อย กล่าวคือ เพิ่มจาก 10 เป็น 10.9647 ทำให้ใช้แรงงานหรือความพยายามโดยประมาณเพียง 143 Man Month

Post Architecture Model

ในระยะหลังการออกแบบ จะพบว่านอกจากปัจจัยขับทั้ง 5 ในระยะก่อนออกแบบ ที่มีผลต่อ Effort ที่ต้องใช้โดยประมาณแล้ว ยังมีปัจจัยที่มีผลกระทบร่วมด้วย ทั้งในด้านคุณลักษณะของผลิตภัณฑ์หรือซอฟต์แวร์(Product Factor) ด้าน Platform (Platform Factor) ด้านบุคลากร (Personnel Factor) และด้านโครงการ (Project Factor) รวมทั้งสิ้น 16 ปัจจัย เรียกปัจจัยเหล่านี้ว่า “Effort Multiplier” ดังนั้น จึงต้องปรับค่า Effort โดยประมาณที่คำนวณได้จากระยะก่อนออกแบบใหม่เป็น **MME (Modified)** ด้วยการประเมินระดับการส่งผลกระทบต่อ Effort ของทั้ง 16 ปัจจัย ออกมาเป็นค่าคะแนนในแต่ละระดับ ได้แก่ ต่ำมาก (Very Low) ต่ำ (Low) ปานกลาง (Nominal) และสูง (High) นำคะแนนระดับที่ประเมินได้ทั้ง 16 มาคูณกัน เพื่อให้ได้เป็น Effort Multiplier คือ Effort ตามสูตรคำนวณต่อไปนี้

MME (Modified) = MME x (EM)

โดย EM คือ Effort Multiplier เป็นผลคูณของปัจจัยที่ส่งผลให้จำนวน Effort เปลี่ยนแปลงไป นั่นคือ EM1 x EM2 x Em3 x ..... x EM16

สำหรับปัจจัยในระยะหลังการออกแบบ ทั้ง 16 ปัจจัย จะแบ่งออกเป็น 4 กลุ่ม ดังนี้

ตารางที่ 12.10 แสดงปัจจัยสำหรับ COCOMO II ในระยะ Post Architecture

กลุ่มปัจจัย	ปัจจัย	รายละเอียด
ผลิตภัณฑ์ซอฟต์แวร์ (Product)	**RELY**: Software Reliability	ระดับความน่าเชื่อถือและไว้วางใจได้ของซอฟต์แวร์ที่ต้องการ
	DATA: Database Size	ขนาดของฐานข้อมูล
	CPLX: Software Complexity	ระดับความซับซ้อนของซอฟต์แวร์
	RUSE: Required Reusability	ความต้องการในการนำโค้ดไปใช้ซ้ำ
	DOCU: Documentation	ระดับมาตรฐานของเอกสาร
แพลตฟอร์ม (Platform)	**TIME**: Time Constraint on Execution	ข้อจำกัดด้านเวลาในการรันซอฟต์แวร์
	STOR: Main Storage Constraint	ข้อจำกัดด้านเนื้อที่การเก็บข้อมูล
	PVOL: Platform Volatility	ความถี่ในการเปลี่ยนแพลทฟอร์มหรือระบบปฏิบัติการ
บุคลากร (Personnel)	**ACAP**: Analyst Capability	ความสามารถของนักวิเคราะห์ระบบ
	PCAP: Programmer Capability	ความสามารถของโปรแกรมเมอร์
	PCON: Personnel Continuity	ความถี่ในการเปลี่ยนแปลงพนักงานหรือทีมงาน
	AEXP: Analyst Experience	ประสบการณ์ของนักวิเคราะห์ระบบ
	PEXP: Programmer Experience	ประสบการณ์ของโปรแกรมเมอร์
	LTEX: Language and Tools Experience	ประสบการณ์ในการใช้ภาษาโปรแกรมมิ่งและเครื่องมือ
โครงการ (Project)	**TOOL**: Use of Software Tools	การใช้เครื่องมือในการบริหารโครงการ
	SITE: Site Environment	จำนวนของไซต์งาน

เมื่อประเมินระดับการส่งผลกระทบต่อ Effort ของทั้ง 16 ปัจจัยจนครบแล้ว ให้นำระดับที่ประเมินได้ มาเทียบเป็นคะแนน ตามเกณฑ์ในตารางต่อไปนี้

ตารางที่ 12.11 แสดงระดับการส่งผลกระทบต่อ Effort ของปัจจัยทั้ง 16 ประการในระยะ Post Architecture

Factor	Levels and Ratings			
	ต่ำมาก (Very Low)	ต่ำ (Low)	ปานกลาง (Nominal)	สูง (High)
Product Factor				
1. RELY	0.82	0.92	1.00	1.10
2. DATA	0.80	0.90	1.00	1.14
3. CPLX	0.73	0.87	1.00	1.17
4. RUSE	0.85	0.95	1.00	1.07
5. DOCU	0.81	0.91	1.00	1.11
Platform Factors				
1. TIME	-	-	1.00	1.11
2. STOR	-	-	1.00	1.05
3. PVOL	-	-	1.00	1.15
Personnel Factors				
1. ACAP	1.42	1.19	1.00	0.85
2. PCAP	1.34	1.15	1.00	0.88
3. AEXP	1.22	1.10	1.00	0.88
4. PEXP	1.19	1.09	1.00	0.91
5. LTXP	1.20	1.09	1.00	0.91

6. PCON	1.29	1.12	1.00	0.90
Project Factors				
1. TOOL	1.17	1.09	1.00	0.90
2. SITE	1.22	1.09	1.00	093

<u>ตัวอย่างที่ 12.4</u> จากตัวอย่างที่ 12.2 และ 12.3 เป็นการหาค่า MME (Effort) โดยประมาณในระยะก่อนการออกแบบ เมื่อมาถึงระยะหลังการออกแบบ ต้องนำค่า Effort ที่ได้มาปรับค่าใหม่ตามปัจจัยเพิ่มเติม 16 ปัจจัย โดยยังคงให้ขนาดของซอฟต์แวร์มีค่าเท่ากับ 10 KLoC เช่นเดิม แต่ในตัวอย่างนี้ จะคำนวณหา MME ใน 2 กรณี คือ กรณีที่ 1 ปัจจัยขับทั้ง 16 ประการอยู่ในระดับ “Very Low” และกรณีที่ 2 คือ ปัจจัยขับทั้ง 16 ประการอยู่ในระดับ “High”

<u>กรณีที่ 1</u> ปัจจัยขับทั้ง 16 ประการ ถูกจัดให้อยู่ในระดับ “Very Low” ทั้งหมด

Effort Multiplier กรณีที่ 1 = EM1 x EM2 x EM16

= 0.82 x 0.80 x 0.73 x 0.85 x 0.81 x 1.42 x 1.34 x 1.22 x 1.19 x 1.20 x 1.29 x 1.17 x 1.22

= **2.01**

<u>กรณีที่ 2</u> ปัจจัยขับทั้ง 16ประการ ถูกจัดให้อยู่ในระดับ “High” ทั้งหมด

Effort Multiplier กรณีที่ 2 = EM1 x EM2 x EM16

= 1.10 x 1.14 x 1 17 x 1.07 x 1.11 x 1.11 x 1.05 x 1.15 x 0.85 x

= 0.88 x 0.88 x 0.91 x 0.91 x 0.90 x 0.90 x 0.93

= **0.96**

ดังนั้น **MME (Modified)** ของทั้ง 2 กรณี คำนวณได้ ดังนี้

MME (Modified) = MME x (EM)

MME (จากตัวอย่างที่ 12.2) = 219 x 2.01 ≈ 440 Man Month

MME (จากตัวอย่างที่ 12.3) = 143 x 0.96 ≈ 137 Man Month

<u>หากต้องการคำนวณค่าใช้จ่ายของ Effort (Man Month Cost) โดยประมาณ ให้นำ MME โดยประมาณคูณด้วยอัตราค่าแรงต่อคน (Man Month Rate)</u>

<u>สรุป</u>

การประมาณการต้นทุนของซอฟต์แวร์ (Software Cost Estimation) จัดทำขึ้นเพื่อนำต้นทุนที่ประมาณการได้มาประเมินราคาโครงการหรือประเมินราคาของซอฟต์แวร์ การประมาณการต้นทุนซอฟต์แวร์จึงเป็นอีกกิจกรรมหนึ่งที่สำคัญเทียบเท่ากับกิจกรรมอื่น เนื่องจากหากต้นทุนที่ประมาณการได้นำไปสู่การประมาณราคาของซอฟต์แวร์ไม่ถูกต้องอาจทำให้องค์กรต้องขาดทุนหรือกำไรที่ได้ไม่คุ้มค่า หรืออาจไม่ชนะการแข่งขันประมูลโครงการ ในระหว่างการประมาณการต้นทุน ย่อมต้องมีปัจจัยที่ทำให้ต้นทุนเปลี่ยนแปลง ผู้บริหารโครงการมีหน้าที่ในการปรับค่าต้นทุนให้เหมาะสมกับปัจจัยที่เปลี่ยนแปลงไปด้วย เพื่อให้ต้นทุนโดยประมาณใกล้เคียงกับต้นทุนจริงมากที่สุด และจะส่งผลให้

การประมาณราคาซอฟต์แวร์ถูกต้องที่สุดด้วย

ต้นทุนของโครงการผลิตซอฟต์แวร์ประกอบไปด้วยค่าใช้จ่ายหลายประเภท เช่น ค่าเดินทาง ค่าฝึกอบรม ค่าฮาร์ดแวร์ ซอฟต์แวร์ และแรงงาน เป็นต้น แต่การประมาณการแรงงาน (Effort) เพื่อให้ได้ประสิทธิผลการทำงานที่ต้องการ (Productivity) นั้น จำเป็นต้องอาศัย “ขนาด (Size)” ของซอฟต์แวร์ในการประมาณการ ดังนั้น จึงต้องมีการคิดค้นวิธีการวัดขนาดของซอฟต์แวร์ขึ้นมา โดยนิยมใช้วิธี “นับจำนวนบรรทัด (Line of Code: LoC)” และวิธี “นับฟังก์ชัน (Function Point: FP)” โดยวิธีนับจำนวนบรรทัดเป็นวิธีที่ง่ายและชัดเจนมากกว่า แต่ข้อเสียคือ ไม่สามารถนำมาใช้วัดประสิทธิผลการผลิตซอฟต์แวร์ของโปรแกรมเมอร์ที่ใช้ภาษาโปรแกรมมิ่งที่แตกต่างกันได้ เนื่องจากภาษาโปรแกรมมิ่งที่แตกต่างกัน จะใช้จำนวนบรรทัดที่ไม่เท่ากัน เมื่อต้องเขียนฟังก์ชันงานเดียวกัน ดังนั้นจึงต้องใช้วิธี FP แทน เนื่องจากเป็นการนับที่จำนวนฟังก์ชั่นงาน และยังมีการนำค่าคุณลักษณะของซอฟต์แวร์มาคำนวณร่วมด้วย ทำให้ลดปัญหาความแตกต่างของภาษาโปรแกรมมิ่งได้

เทคนิคในการคำนวณหาต้นทุนและแรงงานที่นิยมใช้กันมากในปัจจุบัน คือ “COCOMO (Constructive Cost Model)” เนื่องจากเป็นวิธีที่มีการเก็บข้อมูลทางสถิติในการผลิตซอฟต์แวร์ของโครงการทั้งสิ้น161 โครงการ แล้วนำมาสร้างเป็นสูตรคำนวณหาต้นทุน ทำให้ได้ค่าที่น่าเชื่อถือมากขึ้น