การสร้างแบบจำลองต้นทุนการพัฒนาซอฟต์แวร์จากผลิตภัณฑ์ซอฟต์แวร์โดยวิธีการจำลอง

นายภูมิ นวลจันทร์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2550

COTS-BASED DEVELOPMENT COST MODELING USING SIMULATION APPROACH

Mr.Poom Naunchan

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering Program in Computer Engineering
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2007

หัวข้อวิทยานิพนธ์	การสร้างแบบจำลองต้นทุนการพัฒนาซอฟต์แวร์จากผลิตภัณฑ์ซอฟต์แวร์โดยวิธีการจำลอง
โดย	นายภูมิ นวลจันทร์
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษา	อาจารย์ ดร.ดาริชา สุธีวงศ์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร.ดิเรก ลาวัณย์ศิริ)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(รองศาสตราจารย์ ดร. พรศิริ หมื่นไชยศรี)

..... อาจารย์ที่ปรึกษา
(อาจารย์ ดร.ดาริชา สุธีวงศ์)

..... กรรมการ
(อาจารย์ ดร.โปรดปราน บุณยพุกกณะ)

..... กรรมการ
(รองศาสตราจารย์ ดร.วีระ บุญจริง)

ภูมิ นวลจันทร์: การสร้างแบบจำลองต้นทุนการพัฒนาซอฟต์แวร์จากผลิตภัณฑ์ซอฟต์แวร์โดยวิธีการจำลอง (COTS-BASED DEVELOPMENT COST MODELING USING SIMULATION APPROACH) อ. ที่ปรึกษา : อ.ดร.ดาริชา สุธีวงศ์, 58 หน้า

งานวิจัยนี้ได้นำเสนอแบบจำลองต้นทุนของการพัฒนาซอฟต์แวร์จากผลิตภัณฑ์ซอฟต์แวร์โดยใช้วิธีการจำลอง โดยแบบจำลองนี้สามารถปรับเปลี่ยนให้เหมาะสมตามขั้นตอนการพัฒนาที่แตกต่างกันระหว่างองค์กร การศึกษานี้แบ่งเป็นสองส่วนโดยส่วนแรกเป็นแบบจำลองการประมาณต้นทุนเพื่อการวางแผนบุคคล ซึ่งศึกษาการจัดสรรคนให้แต่ละขั้นตอนการพัฒนาโดยอาศัยแนวทางจากสามหลักการ คือ แบบจำลองต้นทุน COCOTS หลักการจำลองแบบต่อเนื่องและกฎของบรูคส์ ซึ่งแบบจำลองนี้สามารถหาผลลัพธ์เป็นจำนวนคนที่ใช้ต่อเวลาและวิธีการจัดสรรคนเพื่อให้ได้เวลาและต้นทุนน้อยที่สุด ส่วนที่สองคือแบบจำลองการประมาณต้นทุนเพื่อการวางแผนขั้นตอนพัฒนา โดยได้ประยุกต์ใช้หลักการวิศวกรรมคู่ขนานซึ่งเหมาะสมสำหรับการพัฒนาซอฟต์แวร์จากผลิตภัณฑ์ซอฟต์แวร์ แบบจำลองนี้พิจารณาขั้นตอนการพัฒนาที่มีความสัมพันธ์กัน และหาผลลัพธ์เป็นความน่าจะเป็นของเวลาและต้นทุนที่ใช้สำหรับการพัฒนาซอฟต์แวร์จากผลิตภัณฑ์ซอฟต์แวร์

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต ภูมิ นวลจันทร์
สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่ออาจารย์ที่ปรึกษา ดาริชา สุธีวงศ์
ปีการศึกษา 2550

4970504421 : MAJOR COMPUTER ENGINEERING

KEY WORD : SOFTWARE COST ESTIMATION/ SIMULATION MODEL / COTS-BASED DEVELOPMENT/ BROOKS' LAW/ COCOTS MODEL/ CONCURRENT ENGINEERING

POOM NAUNCHAN : COTS-BASED DEVELOPMENT COST MODELING USING SIMULATION APPROACH. THESIS ADVISOR : DARICHA SUTIVONG, Ph.D., 58 pp.

This research proposes cost modeling for the COTS-BASED development using simulation approach, which allows users to adjust the models according to their organization's development and integration process that may vary across companies. The study is broken into two parts. The first part involves cost estimation model for workforce planning, which focuses on workforce allocation for each development process. The methodology integrates three existing approaches, namely COCOTS model for cost estimation, system dynamics for software process simulation, and Brooks' law. The workforce planning model estimates the required workforce for each time period and finds the minimal effort needed as well as the optimal group member allocation for the COTS development process. The second part involves cost estimation model for process arrangement. Utilizing concurrent engineering appropriate for COTS-BASED development, the model considers the relationship between interdependent processes, offers insights from process arrangement, and estimates probabilistically the development time and cost that are required for COTS-BASED development project.

Department Computer Engineering Student's signature Poom Naunchan

Field of study Computer Engineering Advisor's signature Daricha Sutivong

Academic year 2007

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จได้ด้วยความเมตตา ความอนุเคราะห์ และความช่วยเหลืออย่างยิ่งจาก อาจารย์ ดร.ดาริชา สุธีวงศ์ อาจารย์ที่ปรึกษา ซึ่งเป็นผู้ให้ข้อคิด แนวทาง และคำปรึกษา ตลอดจนเป็นผู้ตรวจทานแก้ไข ทำให้วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วง

ขอขอบพระคุณ รองศาสตราจารย์ ดร.พรศิริ หมื่นไชยศรี ประธานกรรมการสอบวิทยานิพนธ์ อาจารย์ ดร.โปรดปราน บุณยพุกกณะ รองศาสตราจารย์ ดร.วีระ บุญจริง กรรมการสอบวิทยานพนธ์ ที่ได้กรุณาให้คำแนะนำในการแก้ไขและตรวจสอบวิทยานิพนธ์ฉบับนี้ และขอขอบพระคุณคณาจารย์ในภาควิชาวิศวกรรมคอมพิวเตอร์ จุฬาลงกรณ์มหาวิทยาลัยทุกท่านที่ประสิทธิ์ประสาทความรู้อันมีค่ายิ่งแก่ผู้วิจัย

ท้ายที่สุดนี้ขอขอบพระคุณ บิดา มารดา ที่เป็นกำลังใจสำคัญให้แก่ข้าพเจ้าเสมอมา และขอขอบคุณเพื่อนๆ พี่ๆ ในห้องวิจัย ที่ผลักดันและให้ความช่วยเหลือในทุกๆ ด้านจนผู้วิจัยสามารถทำวิทยานิพนธ์ฉบับนี้สำเร็จลุล่วง

สารบัญ

หน้า

บทที่ หน้า

บทที่ หน้า

สารบัญภาพ

หน้า

หน้า

สารบัญตาราง

หน้า

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันการพัฒนาซอฟต์แวร์ยังมีข้อจำกัดหลายด้าน กล่าวคือในการพัฒนาซอฟต์แวร์ต้องใช้เวลานานในการพัฒนา อีกทั้งซอฟต์แวร์ที่ผลิตออกมานั้นยังมีความเสี่ยงสูงในการเกิดข้อผิดพลาดและอาจไม่สามารถใช้งานได้จริง ดังนั้นแนวโน้มการนำซอฟต์แวร์เพื่อมาใช้งานในปัจจุบัน จึงมักจะซื้อซอฟต์แวร์ที่มีอยู่แล้วมาดัดแปลงประกอบกับมีบริษัทจำนวนมากที่ทำการพัฒนาซอฟต์แวร์เพื่อการค้าทำให้มีทางเลือกสำหรับผู้ใช้มากขึ้น ดังนั้นจึงมีการพัฒนาซอฟต์แวร์รูปแบบใหม่ที่เรียกว่าการพัฒนาซอฟต์แวร์เชิงคอมโพเนนต์ (Component-based software development)

การพัฒนาซอฟต์แวร์เชิงคอมโพเนนต์คือการสร้างระบบที่เน้นการนำคอมโพเนนต์ (Component) ของซอฟต์แวร์กลับมาใช้ใหม่ โดยการนำแต่ละคอมโพเนนต์มาดัดแปลงเพิ่มเติมตามความต้องการซึ่งสามารถแยกได้เป็นสองประเภท คือ ประเภทแรกเป็นคอมโพเนนต์ที่เราสามารถจะทำการแก้ไขเพิ่มเติมในส่วนของรหัสโปรแกรม (Source code) ได้และอีกประเภทหนึ่งเป็นคอมโพเนนต์ที่เราไม่สามารถทำการแก้ไขเพิ่มเติมรหัสโปรแกรม (Black Box) ซึ่งมาจากการซื้อคอมโพเนนต์มาใช้งานจากผู้พัฒนาอีกทีหนึ่งที่เรียกกันว่าผลิตภัณฑ์ซอฟต์แวร์ (COTS :Commercial-Off-The-Shelf) ผลิตภัณฑ์ซอฟต์แวร์นั้นมีด้วยกันหลายประเภทขึ้นกับงานที่จะนำไปใช้ เช่น ผลิตภัณฑ์ซอฟต์แวร์ประเภทบัญชี ผลิตภัณฑ์ซอฟต์แวร์ประเภทฐานข้อมูล ผลิตภัณฑ์ซอฟต์แวร์ประเภทบริหารลูกค้าสัมพันธ์เป็นต้น ซึ่งในงานวิจัยนี้จะกล่าวเฉพาะการพัฒนาที่นำผลิตภัณฑ์ซอฟต์แวร์มาใช้งานและทำการรวมผลิตภัณฑ์ดังกล่าวจนเป็นระบบตามต้องการ ซึ่งมีข้อดีทั้งทางด้านผู้ผลิตซอฟต์แวร์ กล่าวคือการนำคอมโพเนนต์ของซอฟต์แวร์ที่ใช้แล้วให้นำกลับมาใช้ใหม่ เป็นการประหยัดและใช้ทรัพยากรที่มีอยู่อย่างคุ้มค่านำไปสู่การลดต้นทุนการผลิต อีกทั้งยังลดความผิดพลาดที่อาจจะเกิดขึ้นเพราะคอมโพเนนต์ที่ใช้แล้วย่อมผ่านการตรวจสอบมาแล้วเป็นอย่างดี สำหรับลูกค้าก็สามารถได้ซอฟต์แวร์มาใช้งานได้ภายในระยะเวลาอันสั้น

อย่างไรก็ดีการเตรียมการและการวางแผนสำหรับการพัฒนาซอฟต์แวร์เป็นสิ่งสำคัญ เนื่องจากการประมาณต้นทุนและระยะเวลาที่แม่นยำจะให้ผลดีทั้งผู้พัฒนาและผู้จ้างวาน มีงานวิจัยจำนวนมากที่เกี่ยวข้องกับการประมาณต้นทุนการพัฒนาซอฟต์แวร์ แต่งานวิจัยส่วนใหญ่จะมุ่งเน้นในการประมาณต้นทุนของการพัฒนาซอฟต์แวร์ที่เริ่มต้นตั้งแต่การเก็บความต้องการและพัฒนาจนเป็นระบบ มีงานวิจัยอยู่จำกัดที่เน้นไปที่การประมาณต้นทุนของการพัฒนาซอฟต์แวร์จากผลิตภัณฑ์ซอฟต์แวร์ ทั้งที่ปัจจุบันมีแนวโน้มการใช้ซอฟต์แวร์ประเภทนี้

เพิ่มขึ้น ซึ่งจากงานวิจัยของ Yang และคณะ (1) พบว่าการใช้ผลิตภัณฑ์ซอฟต์แวร์ เพิ่มขึ้น 28 เปอร์เซ็นต์ในปีค.ศ.1997 และสูงถึง 70 เปอร์เซ็นต์ในปีค.ศ. 2002 (2) และการใช้ผลิตภัณฑ์ซอฟต์แวร์มีปัจจัยความเสี่ยงที่แตกต่างจากการพัฒนาซอฟต์แวร์ทั่วไปคือ ผู้ใช้ผลิตภัณฑ์ไม่สามารถที่จะกำหนดฟังก์ชันการทำงานและประสิทธิภาพของตัวผลิตภัณฑ์ และอาจจะประสบปัญหาเมื่อทำการรวมระบบจากการที่ผลิตภัณฑ์ที่มีผู้พัฒนาต่างกันซึ่งอาจไม่ได้ถูกออกแบบมาใช้งานร่วมกัน (3)

งานวิจัยนี้จึงได้พัฒนาวิธีการสร้างแบบจำลองต้นทุนโดยใช้วิธีการจำลอง ที่เหมาะกับลักษณะการพัฒนาซอฟต์แวร์ในรูปแบบนี้ โดยเสนอแบบจำลองในการประมาณต้นทุนซึ่งส่งผลช่วยในการบริหารโครงการซอฟต์แวร์ในสองส่วนคือ แบบจำลองการประมาณต้นทุนเพื่อการจัดวางแผนบุคคลและแบบจำลองการประมาณต้นทุนเพื่อการจัดวางขั้นตอนกระบวนการพัฒนา

1.2 วัตถุประสงค์ของการวิจัย

งานวิจัยนี้มีวัตถุประสงค์เพื่อนำเสนอแบบจำลองสำหรับการประมาณต้นทุนการพัฒนาซอฟต์แวร์จากผลิตภัณฑ์ซอฟต์แวร์ ซึ่งแบ่งเป็นสองแบบจำลอง คือแบบจำลองการประมาณต้นทุนเพื่อการจัดวางแผนบุคคลและแบบจำลองการประมาณต้นทุนเพื่อการจัดวางขั้นตอนกระบวนการพัฒนา

1.3 ขอบเขตการวิจัย

1.3.1 ในส่วนของแบบจำลองการประมาณต้นทุนเพื่อการจัดวางแผนบุคคล ได้นำค่าของพารามิเตอร์ในแบบจำลอง COCOTS มาประยุกต์ใช้เป็นค่าปัจจัยผลิตภาพและกำหนดให้ไม่มีการเปลี่ยนขนาดของทีมในระหว่างการพัฒนา โดยใช้ระบบพลวัตเป็นวิธีการจำลองแบบต่อเนื่อง

1.3.2 ในส่วนของแบบจำลองการประมาณต้นทุนเพื่อการจัดวางขั้นตอนกระบวนการพัฒนา จะพิจารณาความไม่แน่นอนที่เกิดขึ้นจากการเปลี่ยนแปลงของความต้องการ (Requirement) โดยใช้ปัวส์ซองจำลองความน่าจะเป็นในการเกิด

1.4 ขั้นตอนและวิธีดำเนินการวิจัย

1.4.1 ศึกษาขั้นตอนและรูปแบบวิธีการต่างๆของการพัฒนาซอฟต์แวร์จากผลิตภัณฑ์ซอฟต์แวร์

1.4.2 ศึกษาแบบจำลองการวางแผนที่มีอยู่ นำมาวิเคราะห์ข้อดีข้อเสียของแต่ละแบบจำลองและนำหลักการมาประยุกต์ใช้เพื่อสร้างแบบจำลองใหม่

1.4.3 นำความรู้ที่ศึกษามาวิเคราะห์และสร้างแบบจำลองการประมาณต้นทุนของการพัฒนาระบบโดยใช้ผลิตภัณฑ์ซอฟต์แวร์

1.4.4 ทดลองประยุกต์ใช้แบบจำลองที่สร้างขึ้น

1.4.5 วิเคราะห์ผลการทดลอง

1.4.6 สรุปผลและเรียบเรียงวิทยานิพนธ์

1.5 ประโยชน์ที่คาดว่าจะได้รับจากการวิจัย

สามารถนำแบบจำลองไปช่วยในการตัดสินใจวางแผนคือ การจัดวางบุคคลและการจัดวางขั้นตอนของการพัฒนาซอฟต์แวร์จากผลิตภัณฑ์ซอฟต์แวร์ ซึ่งมีความยืดหยุ่นในการที่ผู้ใช้สามารถปรับเปลี่ยนแบบจำลองให้เหมาะสมกับองค์กรโดยให้ผลลัพธ์เป็นเวลาและต้นทุนที่ใช้ในการพัฒนา

1.6 ผลงานตีพิมพ์จากวิทยานิพนธ์

ส่วนหนึ่งของงานวิทยานิพนธ์ได้รับการตีพิมพ์เป็นบทความตีพิมพ์วิชาการในหัวเรื่อง “Adjustable Cost Estimation Model for COTS-Based Development” โดย Poom Naunchan และ Daricha Sutivong ในงานประชุมวิชาการ “18th Australian Software Engineering Conference, ASWEC 2007” ในวันที่ 10-13 เมษายน 2550

บทที่ 2

ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 ทฤษฎีที่เกี่ยวข้อง

2.1.1 แบบจำลองการประมาณต้นทุนการพัฒนาโดยใช้ผลิตภัณฑ์ซอฟต์แวร์ (COCOTS model)

แบบจำลอง COCOMO (COnstructive COst MOdel) เป็นแบบจำลองการประมาณต้นทุนของการพัฒนาซอฟต์แวร์โดยแบบจำลอง COCOMO รุ่นแรกได้ถูกออกแบบเพื่อนำมาใช้ประมาณต้นทุนการพัฒนาโปรแกรมที่สร้างตั้งแต่เริ่มต้นเพียงอย่างเดียว (New code) ต่อมาเมื่อรูปแบบการพัฒนาซอฟต์แวร์เปลี่ยนไปมีการนำรหัสโปรแกรมที่มีอยู่แล้วมาใช้ในการพัฒนา (Reuse code) แบบจำลอง COCOMO II (4) จึงถูกพัฒนาเพื่อให้เหมาะสมกับการพัฒนารูปแบบนี้ โดยแบบจำลองทั้งสองแบบเป็นการพัฒนาซอฟต์แวร์ที่ผู้พัฒนาสามารถแก้ไขรหัสโปรแกรมได้ แต่อย่างไรก็ตามแบบจำลองทั้งสองไม่ได้ถูกพัฒนามาเพื่อประมาณต้นทุนของการพัฒนาซอฟต์แวร์ที่ไม่สามารถแก้ไขรหัสโปรแกรมหรือผลิตภัณฑ์ซอฟต์แวร์ ซึ่งการพัฒนาในรูปแบบนี้ต้องใช้โปรแกรมประสาน (Glue code) เพื่อทำการเชื่อมโยงการทำงานซึ่งกันและกันระหว่างคอมโพเนนต์ ดังนั้นแบบจำลอง COCOTS (COnstructive COTS) (5) จึงถูกพัฒนามาเพื่อใช้ประมาณต้นทุนของการพัฒนาซอฟต์แวร์จากผลิตภัณฑ์ซอฟต์แวร์ โดยแบบจำลองนี้ได้ให้คำจำกัดความของลักษณะของผลิตภัณฑ์ซอฟต์แวร์ เป็นสองลักษณะคือ

1. ผลิตภัณฑ์ซอฟต์แวร์ไม่มีรหัสโปรแกรมให้กับผู้ที่ทำการพัฒนา กล่าวคือผู้พัฒนาไม่สามารถทำการแก้ไขรหัสโปรแกรมของผลิตภัณฑ์ซอฟต์แวร์ได้ ถ้าสามารถแก้ไขรหัสโปรแกรมในผลิตภัณฑ์ซอฟต์แวร์ได้จะจัดอยู่ในประเภทการใช้ซ้ำ (Reuse) ซึ่งอยู่ใน COCOMOII ในส่วนของ COCOMO reuse model

2. ผลิตภัณฑ์ซอฟต์แวร์มีการเปลี่ยนแปลงขึ้นกับผู้พัฒนาคอมโพเนนต์นั้น โดยผู้ใช้ไม่สามารถที่จะทำการควบคุมได้

แบบจำลอง COCOTS ได้แบ่งขั้นตอนของการพัฒนาเป็น 3 ขั้นตอน

1. การประเมิน (Assessment) เป็นขั้นตอนของการประเมินผลิตภัณฑ์ซอฟต์แวร์รวมถึงการคัดเลือกผลิตภัณฑ์ซอฟต์แวร์ที่เหมาะสมจากผู้จำหน่ายผลิตภัณฑ์เพื่อใช้ในการรวมระบบ (Integration) ซึ่งแบ่งขั้นตอนในการคัดเลือกออกเป็นสองขั้นตอน โดยในขั้นตอนแรกจะทำการคัดเลือกแบบหยาบ (Initial filtering) เพื่อตัดผลิตภัณฑ์ซอฟต์แวร์ ที่ไม่ตรงตามต้องการออกไป หลังเสร็จจากขั้นตอนที่หนึ่งก็นำผลิตภัณฑ์ซอฟต์แวร์ที่เหมาะสมมาทำการประเมินอย่าง

ละเอียด (Detailed assessment) ซึ่งความเพียรพยายามที่ใช้ (Effort) ของขั้นตอนการประเมิน คือ จำนวนความเพียรพยามที่ใช้ในการประเมินแบบหยาบรวมกับจำนวนความเพียรพยายามที่ใช้ในการประเมินแบบละเอียด

2. การปรับแต่ง (Tailoring) การปรับแต่งเป็นกิจกรรมในการปรับผลิตภัณฑ์ซอฟต์แวร์ให้ตรงกับความต้องการเพื่อเตรียมในการรวมระบบ (System integration) เป็นการปรับเปลี่ยนฟังก์ชันการทำงาน โดยไม่มีการแก้ไขหรือเขียนรหัสโปรแกรมเพื่อต่อเติมหรือขยายฟังก์ชันการทำงานของโปรแกรมเพราะส่วนนี้จะอยู่ในส่วนของการเขียนโปรแกรมประสาน โดยค่าความเพียรพยายามของการปรับแต่งมีค่าเท่ากับจำนวนของผลิตภัณฑ์ซอฟต์แวร์ ที่ต้องทำการปรับ (Configure) เพื่อเตรียมในการรวมระบบ คูณด้วยค่าเฉลี่ยความเพียรพยายามที่ใช้ในการปรับแต่งซึ่งสอดคล้องกับความซับซ้อนของผลิตภัณฑ์ซอฟต์แวร์นั้น

3. การเขียนโปรแกรมประสาน (Glue Code) เป็นส่วนของรหัสโปรแกรมที่ถูกเขียนขึ้นมาใหม่เพื่อใช้ในการรวมผลิตภัณฑ์ซอฟต์แวร์เข้าด้วยกัน ซึ่งมีการแบ่งประเภทของโปรแกรมประสานเป็นทั้งหมดสามประเภท

1. โปรแกรมประสานที่ถูกเขียนเพื่อใช้ในการเชื่อมต่อระหว่างคอมโพเนนต์ในการแลกเปลี่ยนและส่งผ่านข้อมูลระหว่างคอมโพเนนต์

2. โปรแกรมประสานที่ทำการเชื่อมต่อคอมโพเนนต์กับระบบ โปรแกรมนี้จะแตกต่างจากประเภทแรก ตรงที่โปรแกรมประสานที่เขียนขึ้นไม่มีส่วนในการแลกเปลี่ยนข้อมูลระหว่างคอมโพเนนต์เช่น การเขียนรหัสโปรแกรมเพื่อใช้สำหรับเปิดหรือเรียกใช้คอมโพเนนต์อื่นมาทำงาน

3. โปรแกรมประสานที่ถูกเขียนขึ้นเพื่อทำการเติมเต็มฟังก์ชันการทำงานของคอมโพเนนต์นั้น เนื่องจากขาดฟังก์ชันการทำงานในบางส่วน

โดยแบบจำลอง COCOTS ได้ให้สมการในการประมาณค่าเพียรพยายามในการเขียนโปรแกรมประสานดังสมการที่ 2.1

$$Glue\,Code\,Effort = A[Size(1+CREVOL)]^{B}\prod Effort\,multipliers \qquad (2.1)$$

โดยที่

A = ค่าคงที่เชิงเส้น (linear scaling constant) ที่ใช้สำหรับการปรับแบบจำลอง

$Size$ = ขนาดจำนวนบรรทัดของโปรแกรม (Line of code)

CREVOL = เปอร์เซ็นต์ของการเขียนโปรแกรมประสานใหม่ เมื่อมีการเปลี่ยนแปลงของความต้องการหรือการเปลี่ยนของผลิตภัณฑ์ซอฟต์แวร์

Effort multipliers = ค่าตัวคูณความเพียรพยายามทั้งหมด 13 ปัจจัย ซึ่งใช้ในการปรับค่าความเพียรพยายามให้มีค่าใกล้เคียงความเป็นจริงยิ่งขึ้นดังในรูปที่ 2.1

B = ค่าคงที่ไม่เชิงเส้น (Nonlinear scaling factor) จะมีค่าตามรูปแบบสถาปัตยกรรมของซอฟต์แวร์ที่ใช้ในการพัฒนา (Application architectural engineering)

โดยที่ค่าของตัวแปร A B และ Effort multipliers ทั้ง 13 ปัจจัยสามารถดูได้ที่ภาคผนวก

Personnel Drivers
1) ACIEP - COTS Integrator Experience with Product
2) ACIPC - COTS Integrator Personnel Capability
3) AXCIP - Integrator Experience with COTS Integration Processes
4) APCON - Integrator Personnel Continuity
COTS Component Drivers
5) ACPMT - COTS Product Maturity
6) ACSEW - COTS Supplier Product Extension Willingness
7) APCPX - COTS Product Interface Complexity
8) ACPPS - COTS Supplier Product Support
9) ACPTD - COTS Supplier Provided Training and Documentation
Application/System Drivers
10) ACREL - Constraints on Application System/Subsystem Reliability
11) AACPX - Application Interface Complexity
12) ACPER - Constraints on COTS Technical Performance
13) ASPRT - Application System Portability
Nonlinear Scale Factor
AAREN - Application Architectural Engineering

รูปที่ 2.1 ปัจจัยที่เกี่ยวข้องทั้งหมด 13 ปัจจัย (5)

2.1.2 กฎของบรูคส์ (Brooks' law)

กฎของบรูคส์ (6) ได้กล่าวไว้ว่าการเพิ่มจำนวนคนในโครงการพัฒนาซอฟต์แวร์ที่เสร็จล่าช้า จะกลับทำให้โครงการเกิดการล่าช้าขึ้น โดยบรูคส์ได้พัฒนากฎของตนเองจากการสังเกตโครงการหลายโครงการและสร้างกฎจากสิ่งที่พบเห็นทั่วไปในโครงการเหล่านั้น โดยกฎที่บรูคส์เสนอนั้นยังขาดการสนับสนุนจากการวิจัยเชิงปริมาณ (Quantitative research) จึงมีหลายงานวิจัยที่พยายามสร้างแบบจำลอง (Simulation model) เพื่อยืนยันความถูกต้องตามกฎของบรูคส์โดยศึกษาการเปลี่ยนไปของขนาดสมาชิกที่อยู่ในทีมซึ่งสัมพันธ์กับเวลาและต้นทุนที่ใช้ในการพัฒนา โดยผลสรุปจากงานวิจัยเหล่านี้ (7-12) พบว่ามีสองปัจจัยสำคัญที่ทำให้เกิดการล่าช้าคือ

1. เวลาที่สูญเสียไปเนื่องจากการเรียนรู้ของพนักงานใหม่ เมื่อพนักงานถูกเพิ่มเข้ามาในระหว่างการพัฒนาซอฟต์แวร์ พนักงานใหม่ต้องการเวลาในการฝึกเพื่อทำความคุ้นเคยกับงานและสภาพทางสังคมของทีม นอกจากนี้การฝึกพนักงานใหม่ทำให้อัตราผลิตภาพ (Productivity) ของทีมมีค่าลดลงเนื่องมาจากพนักงานที่มีอยู่เดิมต้องละทิ้งงานของตัวเองเพื่อมาใช้ในการฝึกอบรมพนักงานใหม่

2. เวลาที่สูญเสียเนื่องจากการติดต่อสื่อสารกันภายในทีม เมื่อทีมมีการเพิ่มขนาดเวลาที่ใช้ในการสื่อสารก็เพิ่มขึ้นตามไปด้วย ทำให้อัตราผลิตภาพโดยเฉลี่ยของแต่ละคนในทีมลดลงเนื่องจากสูญเสียไปในการติดต่อสื่อสารซึ่งได้แก่ การสื่อสารทางด้านวาจา การสื่อสารทางด้านเอกสาร เป็นต้น โดยค่าต้นทุนการสื่อสาร (Communication overhead) เพิ่มขึ้นมีความสัมพันธ์กับจำนวนสมาชิกของทีมยกกำลังสองตามสมมติฐานของบรูคส์ซึ่งเป็นที่ยอมรับโดยทั่วไป

2.1.3 การจำลองแบบไม่ต่อเนื่องและการจำลองแบบต่อเนื่อง (Discrete-event and continuous-time simulation)

ในทางปฏิบัตินั้นได้มีการนำการจำลอง (Simulation) มาใช้ในการศึกษาขั้นตอนของการพัฒนาซอฟต์แวร์ (Software process) ตัวอย่างเช่น การพัฒนากลยุทธ์เพื่อบริหารขั้นตอนการพัฒนาซอฟต์แวร์ การปรับปรุงขั้นตอนของการพัฒนาซอฟต์แวร์ การฝึกฝนการบริหารโครงการซอฟต์แวร์โดยใส่ข้อมูลที่มีอยู่และดูผลลัพธ์ที่เกิดขึ้น โดยสิ่งสำคัญในแง่ของการจำลองกระบวนการพัฒนาซอฟต์แวร์แบ่งเป็น 4 แง่มุมคือ (13)

1. จุดประสงค์ของการจำลอง (Purpose) ได้แก่ การวางแผน การปรับปรุงขั้นตอนการพัฒนา การศึกษาและเข้าใจขั้นตอนการพัฒนา เป็นต้น

2. ขอบเขตในการจำลอง (Scope) ได้แก่ การจำกัดขั้นตอน การจำกัดโครงการเดียวหรือหลายโครงการพร้อมกัน การจำกัดเวลาในการประมาณว่าเป็นระยะยาวหรือสั้น เป็นต้น

3. ตัวแปรผลลัพธ์ที่สำคัญ (Key results and variables) ได้แก่ปัจจัยต่างๆที่ส่งผลต่อโครงการ

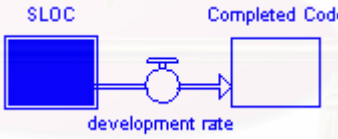
4. วิธีการในการจำลอง (Simulation approach) ได้แก่ การจำลองแบบไม่ต่อเนื่อง (Discrete event simulation) การจำลองแบบต่อเนื่อง (Continuous-time simulation) เป็นต้น

ระบบต่างๆนั้นสามารถแบ่งเป็นสองระบบคือ ระบบไม่ต่อเนื่อง (Discrete system) และระบบต่อเนื่อง (Continuous system) Law และ Kelton (14) ได้เสนอว่า ในทางปฏิบัตินั้นมี

เพียงไม่กี่ระบบที่เป็นระบบไม่ต่อเนื่องหรือต่อเนื่องทั้งระบบ ลักษณะที่เด่นชัดของระบบนั้นจะเป็นตัวแบ่งระบบว่าเป็นต่อเนื่องหรือไม่ต่อเนื่องโดยแต่ละระบบมีลักษณะสำคัญดังต่อไปนี้

1. ระบบไม่ต่อเนื่อง มีลักษณะที่สำคัญของระบบคือ ตัวแปรสถานะ (State variable) จะมีการเปลี่ยนแปลงเป็นบางจุดในช่วงเวลา ยกตัวอย่างเช่นกระบวนการพัฒนาซอฟต์แวร์ เวลาทั้งหมดที่ใช้ในแต่ละขั้นตอนเป็นตัวแปรสถานะ ซึ่งตัวแปรสถานะจะเกิดการเปลี่ยนแปลงไม่ต่อเนื่องคือจะเกิดเพียงเมื่อมีการทำงานใหม่ (Rework) เนื่องจากมีการเปลี่ยนแปลงการความต้องการ เป็นต้น

2. ระบบต่อเนื่อง มีลักษณะที่สำคัญของระบบคือ ตัวแปรสถานะ จะมีการเปลี่ยนแปลงอย่างต่อเนื่องตลอดช่วงเวลาเปรียบเสมือนการไหลของของเหลว ซึ่งสามารถจำลองได้โดยใช้ระบบพลวัต (System dynamic) ในการสร้างแบบจำลองและใช้เครื่องมือด้านกราฟิกเพื่อความสะดวกในการสร้างแบบจำลองและง่ายต่อการทำความเข้าใจ โดยแบบจำลองใช้ระดับน้ำในถังเก็บของเหลวเปรียบเสมือนปริมาณงานที่ต้องทำให้สำเร็จ ดังรูปที่ 2.2 ซึ่งแสดงให้เห็นว่างานในการพัฒนาซอฟต์แวร์ซึ่งมีหน่วยเป็นจำนวนบรรทัดของรหัสโปรแกรม (SLOC) ถูกแสดงเป็นปริมาณของเหลวที่ถูกเก็บในถังของเหลว SLOC และมีวาล์วเปรียบได้กับอัตราการพัฒนา (Development rate) ที่อยู่ระหว่างถังของเหลวทั้งสอง สิ่งซึ่งจำกัดการไหลของของเหลวโดยจะแปรตามจำนวนคนที่ใช้และค่าผลิตภาพ (Productivity) เปรียบเสมือนได้กับอัตราการพัฒนาซอฟต์แวร์ ดังนั้นการพัฒนาจะเสร็จสมบูรณ์ก็ต่อเมื่อระดับของเหลวในถังของเหลว SLOC ไหลไปยังถังของเหลว Completed code ทั้งหมด



รูปที่ 2.2 แบบจำลองการพัฒนาซอฟต์แวร์โดยใช้ระบบพลวัต

Kellner และคณะ (13) ได้ให้คำแนะนำในการเลือกวิธีการจำลองของกระบวนการพัฒนาซอฟต์แวร์ดังนี้

การจำลองแบบไม่ต่อเนื่อง (Discrete event simulation) มุ่งเน้นในรายละเอียดของกระบวนการในการพัฒนา (Detailed process analyses) การใช้ทรัพยากร (Resource utilization) หรือการประเมินแนวโน้มระยะสั้น (Shorter term analysis)

การจำลองแบบต่อเนื่อง (Continuous simulation) เช่น ระบบพลวัต (System dynamic) มุ่งเน้นไปในทางการวิเคราะห์กลยุทธ์ (Strategic analyses) การประมาณเริ่มต้น

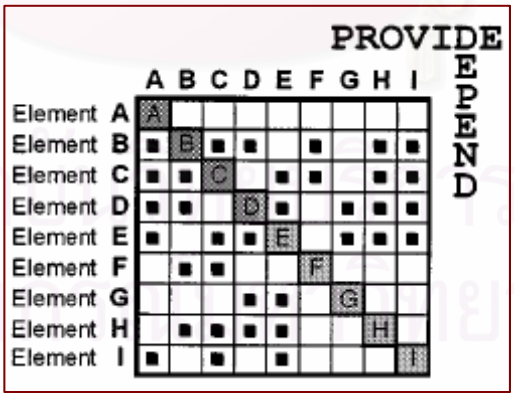
(Initial approximations) แนวโน้มระยะยาว (Long term trends) แง่มุมในระดับสูง (High-level perspectives) เป็นต้น ซึ่งเป็นการวิเคราะห์ที่ไม่มุ่งเน้นรายละเอียดของกระบวนการพัฒนา

2.1.4 การสร้างแบบจำลองขั้นตอนจากการไหลของข้อมูลโดยใช้เมทริกซ์โครงสร้างการออกแบบ (Information flow-based process modeling with design structure matrix) (15)

ขั้นตอนการพัฒนาซอฟต์แวร์ถือเป็นระบบที่มีความซับซ้อน วิธีที่ใช้อยู่ทั่วไปในการสร้างแบบจำลองเพื่อศึกษาระบบที่มีความซับซ้อนคือ

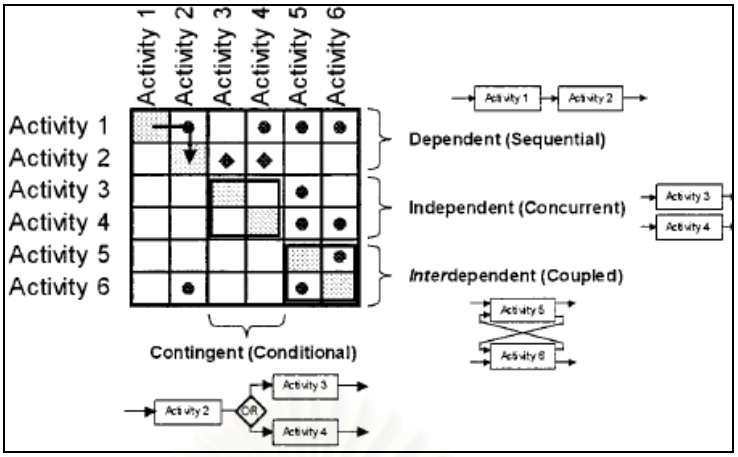
1. การแบ่งระบบย่อย (Decomposing) เพื่อลดความซับซ้อนของปัญหา
2. การพิจารณาความสัมพันธ์ระหว่างระบบย่อย สิ่งซึ่งมีผลต่อระบบโดยรวม
3. การพิจารณาข้อมูลที่นำเข้าจากภายนอกและผลลัพธ์ที่มีผลกระทบต่อระบบ

เมทริกซ์โครงสร้างการออกแบบ (DSM: Design Structure Matrix) เป็นเครื่องมือที่ช่วยในการวิเคราะห์ระบบที่มีความซับซ้อน ซึ่งสามารถแสดงความสัมพันธ์ระหว่างระบบย่อยในรูปแบบที่สามารถวิเคราะห์และศึกษาได้ง่าย ลักษณะของเมทริกซ์โครงสร้างการออกแบบจะเป็นตารางสี่เหลี่ยมจัตุรัส และมีชื่อของระบบย่อยอยู่แต่ละแกนของตารางโดยเรียงตามลำดับก่อนหลังดังรูปที่ 2.3 โดยแต่ละแถว (row) แสดงให้เห็นว่าผลลัพธ์ที่ได้จากระบบย่อยนั้นมีผลต่อระบบย่อยใดบ้างเช่น ระบบย่อย B มีผลต่อระบบย่อย A C D F H และ I และแต่ละคอลัมน์ (column) ตั้งแสดงว่าระบบย่อยขึ้นกับระบบย่อยอื่นใดบ้าง



รูปที่ 2.3 ลักษณะของเมทริกซ์โครงสร้างการออกแบบ (DSM)

ประเภทของเมทริกซ์โครงสร้างการออกแบบมีอยู่หลายประเภทตามการประยุกต์ใช้งาน สำหรับงานวิจัยนี้ได้ใช้เมทริกซ์โครงสร้างการออกแบบในการสร้างแบบจำลองเพื่อแสดงการขึ้นต่อกันระหว่างงาน (Coupled work) และผลป้อนกลับจากการส่งผ่านข้อมูลระหว่างกัน (Feedback) ซึ่งสามารถแสดงผลกระทบที่อาจจะเกิดขึ้นจากการขึ้นต่อกันระหว่างงาน



รูปที่ 2.4 ลักษณะความสัมพันธ์ของแต่ละงาน

สัญลักษณ์ที่อยู่ในเมทริกซ์ในรูป 2.4 แสดงลักษณะของงานว่าขึ้นต่อกันอย่างไร โดยสัญลักษณ์ “◆” แทน “หรือ (or)” และสัญลักษณ์ “●” แทน “และ (and)” จากการเขียนลักษณะความสัมพันธ์แบบนี้ทำให้สามารถแบ่งงานออกเป็น 4 ประเภทคือ

1. งานที่ขึ้นต่องานที่ทำมาก่อน (Dependent) ตัวอย่างเช่น สัญลักษณ์ “●” ที่อยู่ในช่อง (งานที่ 1, งานที่ 2) นั้นหมายความว่าต้องทำงานที่ 1 ก่อนที่จะทำงานที่ 2 ได้ อีกทั้งข้อมูลที่ได้จากงานที่ 1 จะมีผลกระทบต่องานที่ 2 ด้วย

2. งานที่เป็นอิสระต่อกัน (Independent) ตัวอย่างเช่น งานที่ 3 กับงานที่ 4 ซึ่งทั้งสองงานไม่มีความสัมพันธ์กันสามารถที่จะเลือกทำงานใดงานหนึ่งก่อนหลังได้ สังเกตได้จากกรอบสี่เหลี่ยมเล็กของงานที่ 3 และ 4 ไม่มีสัญลักษณ์ “◆” และ “●” อยู่เลย

3. งานที่ขึ้นต่อกันและกัน (Interdependent) ตัวอย่างเช่น งานที่ 5 และงานที่ 6 งานทั้งสองมีผลซึ่งกันและกันโดยที่ถ้ามีการเปลี่ยนแปลงที่งานใดงานหนึ่งต้องมีการเปลี่ยนแปลงอีกงานหนึ่งด้วย เนื่องจากงานที่ 5 มีความสัมพันธ์แบบ “และ” กับงานที่ 6 และงานที่ 6 มีความสัมพันธ์แบบ “และ” กับงานที่ 5 ด้วย

4. งานที่มีทางเลือก (Contingent) ตัวอย่างเช่น งานที่ 2 กับ งานที่ 3 และงานที่ 4 ผลลัพธ์ที่ได้จากงานที่ 2 อาจมีผลต่องานที่ 3 หรืองานที่ 4 เนื่องจากงานที่ 3 และ 4 มีความสัมพันธ์แบบ “หรือ” กับงานที่ 2

อย่างไรก็ตามเมทริกซ์โครงสร้างการออกแบบก็มีข้อจำกัดคือ ประการแรกเมทริกซ์โครงสร้างการออกแบบเพียงอันเดียวจะแสดงวิธีการส่งข้อมูลเพียงหนึ่งวิธีของแต่ละงาน ไม่สามารถแสดงเส้นทางการส่งข้อมูลทั้งหมดที่เป็นไปได้ถึงแม้จะมีสัญลักษณ์ “◆” และ “●” แล้วก็ตาม ประการที่สองเมทริกนี้ไม่ได้แสดงการซ้อนทับกัน (overlap) ของงานโดยผู้ใช้ไม่สามารถรู้

ได้ว่าแต่ละงานมีการซ้อนทับที่เวลาใดซึ่งเป็นสิ่งซึ่งแผนภูมิแกนต์ (Gantt chart) สามารถแสดงให้เห็นอย่างชัดเจน

2.1.5 แบบจำลองความไม่แน่นอนของการเปลี่ยนแปลงทางวิศวกรรม

ในวิศวกรรมคู่ขนาน (Concurrent engineering) งานที่ทำหลัง (Downstream) จะเริ่มก่อนที่งานที่ทำก่อนหน้า (Upstream) จะเสร็จสิ้น ดังนั้นงานที่ทำหลังจะได้รับข้อมูลเบื้องต้น (Preliminary information) จากงานที่ทำก่อนหน้า โดยที่ข้อมูลดังกล่าวยังไม่มีความสมบูรณ์และอาจมีการเปลี่ยนแปลงในเวลาต่อมา จะเห็นได้ว่าถ้าข้อมูลเบื้องต้นที่งานที่ทำหลังรับมามีการเปลี่ยนแปลง งานที่ทำหลังก็ต้องมีการเปลี่ยนแปลงด้วยโดย Loch และคณะ (16) ได้มองการเปลี่ยนแปลงทางวิศวกรรม (Engineering changes) ที่เกิดขึ้นในงานที่ทำก่อนหน้าในช่วงเวลาใดๆ เป็นกระบวนการสโทแคสติก (Stochastic process) ซึ่งสามารถจำลองในรูปแบบกระบวนการปัวส์ซองแบบไม่คงที่ (Nonstationary Poisson process) โดยที่โอกาสในการเกิดเหตุการณ์หรือการเปลี่ยนแปลงในแต่ละช่วงเวลาไม่เท่ากัน ซึ่งระดับความไม่แน่นอนนี้จะถูกระบุโดยค่าเฉลี่ยการเกิดเหตุการณ์ และสามารถลดลงได้จากการประชุมกำหนดข้อตกลงก่อนที่จะเริ่มทำงาน เช่น การระบุเทคโนโลยีที่ใช้ในการผลิต การแก้ปัญหาการออกแบบในอดีตที่ผ่านมา

ระดับของความไม่แน่นอนหรืออัตราการเกิดการเปลี่ยนแปลงสามารถระบุได้โดย μ_α โดยα แทนจำนวนของการประชุมร่วมกันก่อนการพัฒนาจะเริ่มขึ้น β แทนความสามารถขององค์กรที่สามารถลดความไม่แน่นอนจากการสื่อสารเบื้องต้น (Precommunication) และ μ_0 แทนอัตราการเปลี่ยนแปลงถ้าไม่มีการสื่อสารเบื้องต้น μ_α สามารถหาค่าได้ดังสมการที่ 2.2

$$\mu_\alpha = \mu_0 \exp\{-\beta\alpha\} \tag{2.2}$$

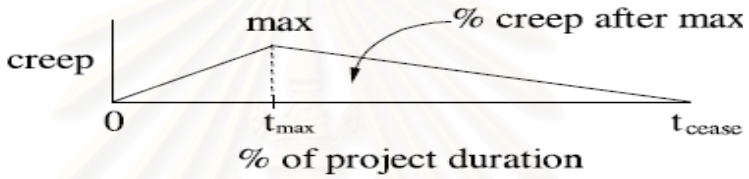
โดยที่ μ_α แทนความไม่แน่นอนที่เกิดขึ้นทางเทคนิคของโครงการหรืออัตราการเปลี่ยนแปลงแก้ไขความไม่สมบูรณ์ของแผนการ แต่เนื่องจากโอกาสในการเกิดขึ้นของเหตุการณ์แตกต่างกันในแต่ละช่วงเวลา เป็นกระบวนการปัวส์ซองแบบไม่คงที่ (Nonstationary Poisson process) ค่าเฉลี่ยของการเกิดเหตุการณ์จะมีค่าเป็น $\mu_\alpha(t^{up})$ เมื่อ t^{up} คือเวลา ณ ขณะใดๆ ในช่วงงานก่อนหน้า (T_1) หรือ $t^{up} \in [0, T_1]$ ดังแสดงในสมการที่ 2.3

$$\mu_\alpha(t^{up}) = \mu_\alpha\left[1 + e\left(2\,\frac{t^{up}}{T_1} - 1\right)\right] \tag{2.3}$$

พารามิเตอร์ $e \in [-1,1]$ คือลักษณะของการเปลี่ยนแปลงทางวิศวกรรมของโครงการ ถ้า e มีค่าน้อยกว่า 0 การเปลี่ยนแบบทางวิศวกรรมจะมีโอกาสเกิดขึ้นมากในช่วงเริ่มต้นและลดลงเมื่อใกล้จบงานนั้น ในทางกลับกันถ้า e มีค่ามากกว่า 0 การเปลี่ยนแปลงทางวิศวกรรมจะมีโอกาสเกิดขึ้นน้อยในช่วงเริ่มต้นและจะเพิ่มขึ้นในตอนใกล้จบงาน ถ้า e มีค่า

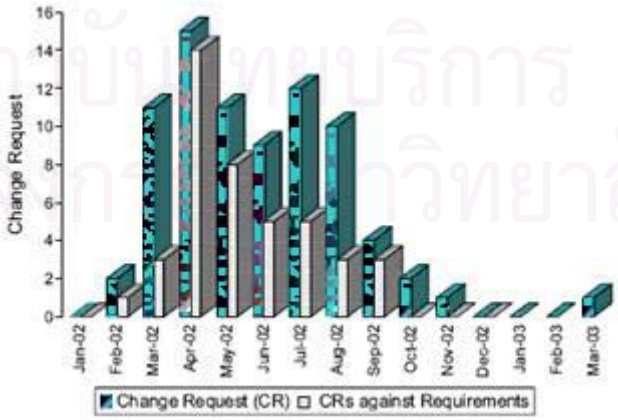
เท่ากับ 0 แสดงว่าการเปลี่ยนแปลงทางวิศวกรรมของโครงการมีโอกาสเกิดขึ้นเท่ากันทุกช่วงเวลา สำหรับที่มาของค่า e และความหมายโดยละเอียดดูได้จากงานวิจัยของ Krishnan และคณะ (17)

โดยในกระบวนการผลิตซอฟต์แวร์นั้นการเปลี่ยนแปลงทางวิศวกรรมเปรียบเทียบได้กับการเปลี่ยนแปลงของความต้องการที่เกิดขึ้นในช่วงเวลาของการพัฒนาซอฟต์แวร์โดย Houston และคณะ (18) ได้สร้างแบบจำลองต่อเนื่อง (Continuous model) ของการเปลี่ยนแปลงความต้องการ โดยมีการเพิ่มขึ้นของความต้องการ (Creep) เป็นเส้นตรงจนกระทั่งถึงเวลาที่ขึ้นสูงสุดและลดลงเป็นเส้นตรงแสดงได้ดังรูปที่ 2.5 จะเห็นได้ว่าการเปลี่ยนแปลงความต้องการจะเพิ่มขึ้นเป็นเชิงเส้นเมื่อถึง t_{max} ซึ่งเป็นเวลาที่เกิดการเปลี่ยนแปลงความต้องการเพิ่มขึ้นสูงที่สุด และจากนั้นจะลดลงจนเป็นศูนย์เมื่อถึงเวลา t_{cease}



รูปที่ 2.5 แบบจำลองต่อเนื่องของการเพิ่มขึ้นของความต้องการ (18)

ต่อมา Nurmuliani และคณะ (19) ได้ทำการวัดการเปลี่ยนแปลงความต้องการดังกล่าว โดยวัดจากอัตราการร้องขอการเปลี่ยนแปลงที่เกิดขึ้น (Change request arrival rate) ณ เวลาใดๆ โดยเป็นการวิเคราะห์ข้อมูลของโครงการซอฟต์แวร์ที่ใช้เวลาทั้งสิ้น 16 เดือนซึ่งมีการร้องขอการเปลี่ยนแปลงทั้งสิ้น 78 ครั้ง ตามรูปที่ 2.6 แผนภูมิแท่งมีลายคือการร้องขอการเปลี่ยนแปลงทั้งหมด ส่วนแผนภูมิแท่งไม่มีลายคือ การร้องขอการเปลี่ยนแปลงที่เกี่ยวข้องกับความต้องการซึ่งได้กราฟที่มีรูปร่างใกล้เคียงกันกับแบบจำลองที่ Houston และคณะได้เสนอไว้



รูปที่ 2.6 การร้องขอการเปลี่ยนแปลง ณ เวลาใดๆ (19)

2.2 งานวิจัยที่เกี่ยวข้อง

2.2.1 งานวิจัยที่เกี่ยวข้องกับแบบจำลองการประมาณต้นทุนการพัฒนาซอฟต์แวร์

แบบจำลองการประมาณต้นทุนโดยส่วนใหญ่สร้างจากพื้นฐานหลักการทางสถิติ โดยใช้ข้อมูลที่มีอยู่เดิมเป็นตัวพยากรณ์คือ แบบจำลองที่ใช้หลักการเชิงถดถอย (Regression based model) ได้แก่แบบจำลองในตระกูลของ COCOMO โดยงานวิจัย (20, 21) ได้เสนอวิธีในการปรับเทียบ (Calibrate) เพื่อให้แบบจำลองมีความแม่นยำและใกล้เคียงกับองค์กร วิธีการดังกล่าวมีข้อดีคือง่ายสำหรับการใช้งานแบบจำลองแต่ขณะเดียวกันก็มีข้อเสียเนื่องจากแบบจำลองมีที่มาจากการเก็บข้อมูลและใช้หลักการทางสถิติ เพื่อหาสมการประมาณต้นทุนและค่าตัวคูณความเพียรพยายาม (Effort multiplier) ดังนั้นการนำแบบจำลองมาใช้ต้องยึดหลักเกณฑ์มาตรฐานการวัดและขั้นตอนการพัฒนาตามแบบจำลองซึ่งทำให้ขาดความยืดหยุ่น เนื่องจากในความเป็นจริงทุกองค์กรย่อมมีขั้นตอนการพัฒนาและปัจจัยที่ส่งผลกระทบแตกต่างกัน อีกทั้งแบบจำลองบางที่พัฒนาจาก COCOMO เช่น COCOTS ไม่ได้ให้สมการการคำนวณระยะเวลาในการพัฒนา

สำหรับการใช้การจำลอง (Simulation) เพื่อประมาณต้นทุนซอฟต์แวร์นั้นโดยส่วนใหญ่ (22-24) ทำเพื่อศึกษาปัจจัยที่มีผลต่อการพัฒนาซอฟต์แวร์ ข้อดีของการใช้การจำลองคือมีความยืดหยุ่นผู้ใช้สามารถทำการปรับเปลี่ยนค่าที่ใช้ จัดเรียงขั้นตอนการพัฒนาที่ใกล้เคียงกับองค์กรและใช้การจำลองเพื่อดูผล สำหรับข้อเสียของการใช้การจำลองคือ ต้องการรายละเอียดของข้อมูลกระบวนการพัฒนามากกว่าแบบจำลองที่ใช้หลักการทางสถิติ และข้อมูลบางอย่างมีความซับซ้อนในการระบุค่า เช่นอัตราการเปลี่ยนแปลงพนักงานในทีมเมื่อเวลาเปลี่ยนไป อัตราการเรียนรู้ของพนักงานใหม่ ซึ่งสิ่งเหล่านี้มีผลต่อความแม่นยำที่ใช้ประมาณต้นทุนและเวลา

งานวิจัยนี้มุ่งเน้นการสร้างแบบจำลองในการประมาณต้นทุนของการพัฒนาซอฟต์แวร์โดยใช้ผลิตภัณฑ์ซอฟต์แวร์ จากการศึกษางานวิจัยที่เกี่ยวข้องกับการประมาณต้นทุนของการพัฒนารูปแบบนี้พบว่าขั้นตอนหลักในการพัฒนาระบบโดยใช้ผลิตภัณฑ์ซอฟต์แวร์ประกอบด้วยการประเมินผลิตภัณฑ์ซอฟต์แวร์และการรวมผลิตภัณฑ์ซอฟต์แวร์เข้าด้วยกัน ซึ่งในขั้นตอนการประเมินผลิตภัณฑ์นั้นจะมุ่งเน้นการเลือกผลิตภัณฑ์มาใช้งานเพราะส่งผลต่อจำนวนและชนิดของโปรแกรมประสานที่ต้องใช้ซึ่งจะทำให้ต้นทุนในการรวมระบบแตกต่างกัน ด้วยโดย Yakimovich และคณะ (25) ได้บรรยายตัวแปรสำคัญที่เป็นคุณสมบัติของผลิตภัณฑ์ซอฟต์แวร์ที่ส่งผลต่อต้นทุนในการรวมระบบ ส่วนขั้นตอนการรวมระบบจะเริ่มขึ้นหลังจากเลือกผลิตภัณฑ์ซอฟต์แวร์ที่เหมาะสมแล้วโดยแบบจำลองการประมาณต้นทุนการพัฒนาระบบโดยใช้ผลิตภัณฑ์ซอฟต์แวร์ที่ใช้ในการอ้างอิงคือแบบจำลอง COCOTS (5) ซึ่งแบ่งการประมาณความเพียรพยายามเป็นสามขั้นตอนคือ การประเมินผลิตภัณฑ์ซอฟต์แวร์ การปรับแต่ง และการพัฒนาโปรแกรมประสาน อย่างไรก็ตามแบบจำลองนี้ไม่สามารถประมาณเวลาได้โดยตรงและไม่

สามารถให้ผู้ใช้ปรับเปลี่ยนขั้นตอนกระบวนการรวมระบบที่แตกต่างกันในแต่ละองค์กรหรือโครงการ ต่อมา Baik และคณะ. (23) ได้พัฒนาแบบจำลองพลวัตสำหรับการพัฒนาโปรแกรมประสานและการรวมในระดับระบบ โดยได้ประยุกต์ใช้ตัวคูณความเพียรพยายามของแบบจำลอง COCOTS และใช้แบบจำลองทรัพยากรมนุษย์ (Human resource model) เพื่อศึกษาว่าทั้งสองกระบวนการมีผลกระทบซึ่งกันและกันอย่างไร และมีผลกระทบต่อเวลาในการพัฒนาอย่างไรด้วย โดยการจำลองนี้มีความซับซ้อนและต้องการรายละเอียดของข้อมูลอย่างมาก จึงเหมาะสำหรับการศึกษามากกว่าการนำไปใช้งานจริง

2.2.2 งานวิจัยที่เกี่ยวข้องกับวิศวกรรมคู่ขนาน (Concurrent engineering)

หลักการทางวิศวกรรมคู่ขนานได้ถูกนำเสนอมาก่อนต้นคริสตศักราชที่ 19 โดยมีจุดประสงค์ในการเพิ่มศักยภาพในการแข่งขัน สร้างหลักการให้กับผู้ออกแบบขั้นตอนการผลิตและลดเวลาการผลิตให้สั้นลง (26) การพัฒนาซอฟต์แวร์จากผลิตภัณฑ์ซอฟต์แวร์ก็มีความใกล้เคียงกับการผลิตฮาร์ดแวร์ กล่าวคือ เป็นการพัฒนาซึ่งนำคอมโพเนนต์ต่างๆ มาทำงานร่วมกันเป็นระบบที่ต้องการ โดยมองเปรียบเทียบได้กับการประกอบฮาร์ดแวร์ ดังนั้นการพัฒนาแต่ละคอมโพเนนต์จึงสามารถที่จะแยกพัฒนาไปพร้อมกันเพื่อลดเวลาทั้งหมดที่ใช้ในการพัฒนา

แต่ในการพัฒนาซอฟต์แวร์นั้นยังมีการนำทฤษฎีของวิศวกรรมคู่ขนานมาใช้ไม่แพร่หลาย อีกทั้งส่วนใหญ่ใช้เพียงเฉพาะกิจ (Adhoc) (27) สำหรับวิศวกรรมการพัฒนาซอฟต์แวร์คู่ขนาน (Concurrent software engineering) ได้ถูกนำเสนอครั้งแรกโดย Blackburn และคณะ (27) งานวิจัยนี้พบว่าการพัฒนาคู่ขนานเป็นปัจจัยที่มีผลสำคัญสำหรับการลดเวลาในการพัฒนาและได้นำหลักการวิศวกรรมคู่ขนานมาประยุกต์ใช้ในการพัฒนาซอฟต์แวร์ โดยเสนอว่าการคู่ขนานกันควรแบ่งเป็นสองส่วนคือ การคู่ขนานด้านกิจกรรม (Activity concurrency) และการคู่ขนานด้านข้อมูล (Information concurrency) โดยได้แยกประเภทของแต่ละชนิดและการใช้งานที่เหมาะสม แต่หลักการที่เสนอมาเป็นในเชิงทฤษฎีไม่มีการวิเคราะห์ทางตัวเลข

การจำลองจะทำให้สามารถวิเคราะห์เชิงตัวเลขได้ ซึ่งการจำลองมีทั้งการจำลองแบบต่อเนื่องและไม่ต่อเนื่อง เช่น Padberg (28-30) ได้สร้างแบบจำลองสำหรับการจำลองแบบไม่ต่อเนื่องโดยแบ่งเป็นสองส่วนคือ ทีมที่พัฒนาและคอมโพเนนต์ทั้งหมดที่ต้องพัฒนา โดยในการจำลองแต่ละทีมจะไปทำงานในแต่ละคอมโพเนนต์เมื่อมีการรายงานการผิดพลาดบนคอมโพเนนต์ ทีมที่พัฒนาต้องกลับไปแก้ไขกลุ่มของคอมโพเนนต์ที่เกี่ยวข้องทั้งหมด แบบจำลองสามารถแสดงลำดับการพัฒนาคอมโพเนนต์ที่ให้เวลาการพัฒนาน้อยที่สุดแต่อย่างไรก็ตามแบบจำลองไม่สามารถแสดงจุดเวลาที่ดีที่สุดในการซ้อนทับ (Overlap) ของขั้นตอนการพัฒนา ต่อมา Iida และคณะ (31) ได้เสนอแบบจำลองขั้นตอนการพัฒนาที่มีการซ้อนทับกันโดยใช้แบบจำลองก้าวหน้า (Progress model) ซึ่งแบบจำลองนี้สามารถบอกได้ว่าการซ้อนทับหลังจาก

จุดเวลาหนึ่งไม่สามารถทำให้เวลาที่ใช้ในการพัฒนาลดลงแต่แบบจำลองไม่ได้พิจารณาผลกระทบที่เกิดขึ้นระหว่างขั้นตอน

ในส่วนการจำลองแบบต่อเนื่อง Baik และคณะ. (23) ได้พัฒนาแบบจำลองพลวัตสำหรับการพัฒนาระบบโดยใช้ผลิตภัณฑ์ซอฟต์แวร์ โดยศึกษาผลกระทบของจุดเริ่มต้นเวลาในการรวมระหว่างโปรแกรมที่พัฒนาเอง (Custom code development) กับโปรแกรมประสาน (Glue code) ซึ่งทั้งสองส่วนเป็นขั้นตอนที่สามารถซ้อนทับกัน ผลการวิจัยพบว่าการกำหนดจุดเริ่มต้นของการรวมระหว่างโปรแกรมที่พัฒนาเองกับโปรแกรมประสานที่ 80-90% ของงานในขั้นตอนโปรแกรมที่พัฒนาเองและ 30-40 % ของงานในขั้นตอนการพัฒนาโปรแกรมประสานจะให้เวลารวมน้อยที่สุดอย่างไรก็ตามแบบจำลองมีความซับซ้อนมากและไม่ได้คำนึงถึงลักษณะเฉพาะของโครงการ เช่นบางโครงการมีการเปลี่ยนแปลงความต้องการมากในตอนเริ่มต้นหรือในทางกลับกันบางโครงการมีการเปลี่ยนแปลงความต้องการมากในตอนใกล้จบ ซึ่งปัจจัยเหล่านี้เป็นสิ่งสำคัญที่ทำให้จุดเวลาที่ดีที่สุดในการซ้อนทับต่างกัน

ในส่วนของงานวิจัยที่เกี่ยวข้องกับหลักการวิศวกรรมคู่ขนานของอุตสาหกรรมการผลิตคือ การเปลี่ยนแปลงการออกแบบ (Design change) Krishnan และคณะ (17) ได้นำเสนอระดับการเปลี่ยนแปลงของการออกแบบ (Degree of evolution) โดยค่านี้จะแปรผันตามอัตราการเปลี่ยนแปลงของการออกแบบตามเวลา ซึ่งถ้าอัตราการเปลี่ยนแปลงของการออกแบบลดลงเมื่อเวลาผ่านไปเรียกว่า การเปลี่ยนแปลงเร็ว (Fast evolution) หรือถ้าอัตราการเปลี่ยนแปลงของการออกแบบเพิ่มขึ้นเมื่อเวลาผ่านไปเรียกว่า การเปลี่ยนแปลงช้า (Slow evolution) ต่อมา Loch และคณะ (16) ได้เสนอแบบจำลองการเปลี่ยนแปลงทางวิศวกรรมด้วยกระบวนการปัวส์ซองแบบไม่คงที่โดยมีระดับการเปลี่ยนแปลงของการออกแบบเป็นตัวกำหนดรูปร่างของการแจกแจงปัวส์ซอง ซึ่งงานวิจัยได้กำหนดให้ระดับการเปลี่ยนแปลงของการออกแบบเป็นค่าคงที่ และเมื่อมีการเปลี่ยนแปลงการออกแบบเกิดขึ้นในงานก่อนหน้า (Upstream task) งานที่ทำหลัง (Downstream task) จะได้รับผลกระทบจากการที่งานที่ทำหลังเริ่มก่อนที่การออกแบบในงานก่อนหน้าจะสมบูรณ์

สำหรับการหาความสัมพันธ์ระหว่างงานที่ทำนั้น Browning และคณะ (32) ได้ใช้เมทริกซ์โครงสร้างการออกแบบ (Design Structure Matrix) และการจำลองแบบไม่ต่อเนื่องเพื่อหาผลกระทบของเวลาและต้นทุนจากความสัมพันธ์ของขั้นตอนการพัฒนาแต่การจำลองกำหนดไว้ว่าจะมีการเปลี่ยนแปลงก็ต่อเมื่อจบขั้นตอนการทำงานนั้น ซึ่งไม่สามารถจำลองการซ้อนทับของเวลาแต่ละงานได้

บทที่ 3

การสร้างแบบจำลอง

งานวิจัยนี้มีแนวคิดเพื่อสร้างเครื่องมือที่ช่วยในการประมาณต้นทุน การตัดสินใจและวางแผนการพัฒนาซอฟต์แวร์จากผลิตภัณฑ์ซอฟต์แวร์ โดยมีการแบ่งแบบจำลองออกเป็นสองส่วน ส่วนแรกเป็นแบบจำลองการประมาณต้นทุนเพื่อการจัดวางแผนบุคคลให้ทำงานในแต่ละขั้นตอนของการพัฒนาซึ่งให้ผลลัพธ์เป็นเวลาและต้นทุน โดยแบบจำลองที่สร้างขึ้นมีความยืดหยุ่นในการปรับเปลี่ยนให้เหมาะสมตามปัจจัยที่ส่งผลกระทบและกระบวนการพัฒนาที่แตกต่างกันในแต่ละองค์กร ส่วนที่สองเป็นแบบจำลองการประมาณต้นทุนเพื่อการจัดวางขั้นตอนกระบวนการพัฒนา เนื่องจากในแบบจำลองแรกนั้นเป็นการพิจารณาในแง่ของการจัดวางบุคคลเป็นหลักซึ่งในการพัฒนาซอฟต์แวร์จากผลิตภัณฑ์ซอฟต์แวร์บางขั้นตอนสามารถที่จะทำคู่ขนานกันได้ทำให้เวลาที่ใช้ลดน้อยลง ดังนั้นงานวิจัยนี้จึงได้เสนอแบบจำลองในส่วนที่สองสำหรับเป็นเครื่องมือช่วยตัดสินใจการจัดเรียงขั้นตอนกระบวนการพัฒนาโดยให้ผลลัพธ์เป็นเวลาและต้นทุนที่ใช้ ซึ่งได้นำหลักการจากงานส่วนที่หนึ่งและหลักการวิศวกรรมคู่ขนาน (Concurrent engineering) มาประยุกต์ใช้ในการสร้างแบบจำลองที่สองนี้

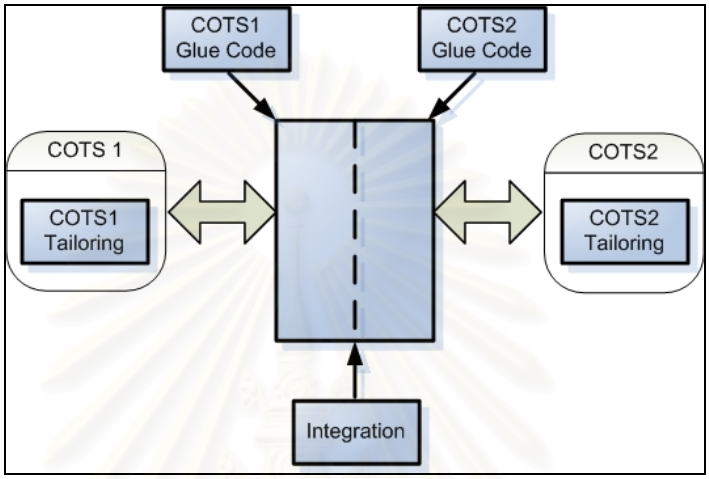
3.1 แบบจำลองการประมาณต้นทุนเพื่อการจัดวางแผนบุคคล

แบบจำลองการประมาณต้นทุนในส่วนนี้ถูกสร้างขึ้นเพื่อใช้ในการตัดสินใจจัดวางแผนบุคคลในแต่ละขั้นตอนของการพัฒนาซอฟต์แวร์จากผลิตภัณฑ์ซอฟต์แวร์ โดยได้ประยุกต์ใช้กฎของบรูคส์ในหัวข้อที่ 2.1.2 และค่าความเพียรพยายามของแบบจำลอง COCOTS มาใช้เป็นค่าปัจจัยผลิตภาพ (Productivity factor) ของแต่ละขั้นตอนการพัฒนาและใช้การจำลองโดยระบบพลวัตเป็นวิธีการจำลอง การสร้างแบบจำลองนี้มีการใช้ทั้งหลักการทางสถิติ ได้แก่ แบบจำลอง COCOTS และหลักการจำลอง ได้แก่ กฎของบรูคส์ และระบบพลวัต เพื่อให้ได้แบบจำลองที่มีความยืดหยุ่นและลดความซับซ้อนในการระบุค่า ทำให้ผู้ใช้สามารถวางแผนบุคคลในแต่ละขั้นตอนการพัฒนา โดยระบุปัจจัยที่ส่งผลกระทบต่อการทำงานในแต่ละขั้นตอนและจัดเรียงกระบวนการพัฒนาได้โดยแสดงผลลัพธ์เป็นเวลาและต้นทุนที่ใช้ นอกจากนี้แบบจำลองยังสามารถนำไปจำลองเพื่อหารูปแบบการจัดวางแผนที่ใช้เวลาและต้นทุนน้อยที่สุดได้อีกด้วย

การสร้างแบบจำลองการประมาณต้นทุนเพื่อการจัดวางแผนบุคคลของการพัฒนาซอฟต์แวร์จากผลิตภัณฑ์ซอฟต์แวร์ สามารถแบ่งขั้นตอนสำคัญเป็น 4 ขั้นตอนซึ่งแต่ละขั้นตอนมีรายละเอียดดังต่อไปนี้

3.1.1 แบบจำลองงานที่ต้องทำในการรวมผลิตภัณฑ์ซอฟต์แวร์เข้าด้วยกัน

งานวิจัยนี้มีแนวคิดในการแบ่งขั้นตอนการรวมผลิตภัณฑ์ซอฟต์แวร์ เป็นสามขั้นตอนด้วยกันโดยทั้งสามขั้นตอนนี้เป็นขั้นตอนหลักของการพัฒนาซอฟต์แวร์จากผลิตภัณฑ์ซอฟต์แวร์ ซึ่งจะยกกรณีการรวมสองผลิตภัณฑ์ซอฟต์แวร์เข้าด้วยกันเป็นตัวอย่างประกอบดังรูปที่ 3.1 โดยมีรายละเอียดดังนี้



รูปที่ 3.1 งานที่ต้องทำในการรวมผลิตภัณฑ์ซอฟต์แวร์สองผลิตภัณฑ์

1. การปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ (Tailoring) เป็นการปรับแต่งฟังก์ชันการทำงานของผลิตภัณฑ์ให้เข้ากับความต้องการและเพื่อเตรียมพร้อมสำหรับการรวมระบบ ดังนั้นงานที่ทำการปรับแต่งจะมีสองส่วนคือ การปรับแต่งของผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่ง (COTS1) และการปรับแต่งของผลิตภัณฑ์ซอฟต์แวร์ที่สอง (COTS2)

2. การเขียนโปรแกรมประสาน (Glue code) เป็นการเขียนโปรแกรมเพื่อเชื่อมต่อการทำงานระหว่างผลิตภัณฑ์ซอฟต์แวร์เข้าด้วยกัน หรือเขียนเพื่อที่จะเพิ่มฟังก์ชันที่ไม่สมบูรณ์ของผลิตภัณฑ์ซอฟต์แวร์ เพื่อความสะดวกในการสร้างแบบจำลองได้แบ่งงานเป็นสองส่วนคือ การพัฒนาโปรแกรมประสานของผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่งและการพัฒนาโปรแกรมประสานของผลิตภัณฑ์ซอฟต์แวร์ที่สอง โดยโปรแกรมประสานของผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่ง จะเป็นโปรแกรมประสานที่ติดต่อกับส่วนต่อประสาน (Interface) ของผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่ง และโปรแกรมประสานของผลิตภัณฑ์ซอฟต์แวร์ที่สองก็เป็นเช่นเดียวกัน

3. การรวมและทดสอบระดับระบบ (System-level integration and test) เป็นการรวมทุกส่วนของระบบเข้าด้วยกันและทดสอบระบบ โดยขนาดของงานที่จะต้องทำคือผลรวมของโปรแกรมประสานทั้งหมดที่เกิดขึ้นในแต่ละส่วนต่อประสานนั้น

โดยสรุปในขั้นตอนที่หนึ่งจะสามารถแบ่งงานในการรวมระบบในแต่ละสองผลิตภัณฑ์ได้เป็น 5 งานดังนี้ 1) การปรับแต่งของผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่ง (COTS1 tailoring) 2) การปรับแต่งของผลิตภัณฑ์ซอฟต์แวร์ที่สอง (COTS2 tailoring) 3) การพัฒนาโปรแกรมประสานของผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่ง (COTS1 glue code) 4) การพัฒนาโปรแกรมประสานของผลิตภัณฑ์ซอฟต์แวร์ที่สอง (COTS2 glue code) และ 5) การรวมและทดสอบระดับระบบ (System-level integration) โดยงานที่หนึ่งและสองมาจากขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ ส่วนงานที่สามและสี่มาจากขั้นตอนการเขียนโปรแกรมประสาน และงานที่ห้ามาจากขั้นตอนการรวมและทดสอบระดับระบบ ซึ่งงานที่ระบุในข้างต้นเป็นงานที่เกิดขึ้นโดยทั่วไปของการรวมผลิตภัณฑ์ซอฟต์แวร์ แต่ผู้ใช้สามารถที่จะปรับเปลี่ยนขั้นตอนได้ตามความเหมาะสม ยกตัวอย่างเช่น อาจจะมีการเพิ่มขั้นตอนการออกแบบ (Design phase) หรือละเว้นขั้นตอนบางอย่างที่ไม่สนใจออกได้

3.1.2 การสร้างแบบจำลองพลวัตสำหรับขั้นตอนการรวมผลิตภัณฑ์ซอฟต์แวร์

เนื่องจากระบบพลวัตสามารถใช้สมการกฎของบรูคส์ในการจำลองได้ และตัวแปรสถานะในแบบจำลองไม่ได้เปลี่ยนแปลงตามเหตุการณ์แต่มีการเปลี่ยนแปลงที่ต่อเนื่องไปตามเวลาที่เปลี่ยนไป การจำลองโดยใช้ระบบพลวัตจึงมีความเหมาะสมต่อแบบจำลองดังกล่าว นอกจากนี้ระบบพลวัตยังง่ายต่อผู้ใช้ในการออกแบบการจัดเรียงกระบวนการพัฒนาและง่ายต่อการจำลองกระบวนการพัฒนาทั้งแบบเป็นลำดับและขนานกัน

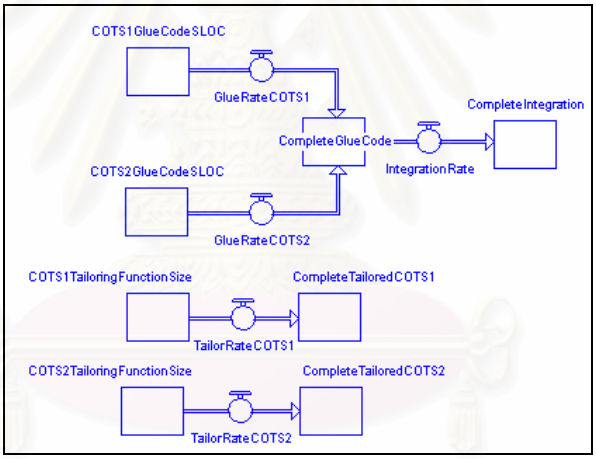
ขั้นตอนที่ผ่านมาได้ระบุงานที่เกิดขึ้นในการรวมสองผลิตภัณฑ์ซอฟต์แวร์ ในขั้นตอนนี้จะทำการจัดเรียงขั้นตอนเหล่านี้โดยใช้แบบจำลองระบบพลวัต ซึ่งแสดงได้ดังรูปที่ 3.2 โดยปริมาณงานของการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่ง จะแสดงได้โดยระดับของเหลวในถังของเหลว COTS1TailoringFunctionSize และปริมาณงานในการเขียนโปรแกรมประสานของผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่ง จะสามารถแสดงได้โดยระดับของเหลวในถังของเหลว COTS1GlueCodeSLOC โดยที่งานทั้งสองส่วนนั้นแยกจากกันเพราะในขั้นตอนการปรับแต่งจะไม่มีการแก้ไขหรือเขียนรหัสโปรแกรมเพื่อต่อเติมหรือขยายฟังก์ชันการทำงานของโปรแกรมซึ่งจะอยู่ในส่วนของการเขียนโปรแกรมประสาน และมีลักษณะงานเช่นเดียวกันกับผลิตภัณฑ์ซอฟต์แวร์ที่สอง โดยปริมาณงานในการรวมระดับระบบจะเท่ากับระดับของเหลวในถังของเหลว CompleteGlueCode โดยมีวาล์วซึ่งจำกัดการไหลของของเหลวเปรียบเสมือนอัตราการพัฒนา นอกจากนั้นแบบจำลองยังให้ผู้ใช้ทำการระบุเวลาเริ่มต้นของกระบวนการที่สามารถทำควบคู่กันได้ ตัวอย่างเช่น ขั้นตอนการเขียนโปรแกรมประสานจะเริ่มต้นเมื่อขั้นตอนการปรับแต่งแล้วเสร็จไป 75 เปอร์เซ็นต์ของงานทั้งหมด ซึ่งก็คือวาล์ว GlueRateCOTS1 และ GlueRateCOTS2 จะถูกเปิดเมื่องานทั้งหมดของการปรับแต่งเสร็จไปแล้ว 75 เปอร์เซ็นต์ เช่นเดียวกันถ้าขั้นตอนการรวมและทดสอบระบบเริ่มต้นเมื่อขั้นตอนการเขียนโปรแกรมประสานเสร็จสิ้นไปแล้ว 80 เปอร์เซ็นต์นั้นคือวาล์ว IntegrationRate จะถูกเปิดเมื่องานในขั้นตอนการเขียนโปรแกรมเสร็จ

สิ้นแล้ว 80 เปอร์เซ็นต์ โดยเราสามารถคำนวณเวลาการพัฒนาทั้งหมดเมื่อการจำลองครบตามเงื่อนไขดังต่อไปนี้

1. ของเหลวในถังเก็บของเหลว COTS1TailoringFunctionSize ได้ไหลลงไปยังถังเก็บของเหลว CompleteTailoredCOTS1 ทั้งหมด ซึ่งหมายถึงงานในการปรับแต่งผลิตภัณฑ์เสร็จสมบูรณ์แล้ว

2. ของเหลวในถังเก็บของเหลว COTS2TailoringFunctionSize ได้ไหลลงไปยังถังเก็บของเหลว CompleteTailoredCOTS2 ทั้งหมด ซึ่งหมายถึงงานในการปรับแต่งผลิตภัณฑ์เสร็จสมบูรณ์แล้ว

3. ของเหลวในถังเก็บของเหลว COTS1GlueCodeSLOC และถังเก็บของเหลว COTS2GlueCodeSLOC ได้ไหลลงไปยังถังเก็บของเหลว CompleteIntegration ทั้งหมดซึ่งหมายถึงงานในการพัฒนาโปรแกรมประสานของผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่งและสอง และงานการรวมและทดสอบระดับระบบเสร็จสมบูรณ์แล้ว



รูปที่ 3.2 แบบจำลองพลวัตของการรวมแต่ละสองผลิตภัณฑ์ซอฟต์แวร์เข้าด้วยกัน

3.1.3 ปัจจัยผลิตภาพในขั้นตอนการรวมผลิตภัณฑ์ซอฟต์แวร์

แบบจำลองที่นำเสนอนี้ได้นำค่าตัวคูณความเพียรพยายาม (Effort multiplier) ในแบบจำลอง COCOTS มาเพื่อใช้ในการประยุกต์หาค่าปัจจัยผลิตภาพ (Productivity factor) โดยในงานวิจัยของ Minkiewicz (33) ได้ทำการจัดแบ่งขั้นตอนในการพัฒนาซอฟต์แวร์จากผลิตภัณฑ์ซอฟต์แวร์ และได้ระบุปัจจัยที่ส่งผลกระทบในแต่ละขั้นตอน ดังนั้นผู้วิจัยจึงได้นำองค์ความรู้ดังกล่าวมาแยกประเภทตัวคูณความเพียรพยายามของแบบจำลอง COCOTS ที่มีผลในแต่ละขั้นตอนของกระบวนการพัฒนาได้ดังนี้

1. ขั้นตอนการปรับแต่ง ปัจจัยผลิตภาพที่มีผลในขั้นตอนนี้คือ

1.1. ปัจจัยความซับซ้อนของการปรับแต่ง (Tailoring complexity factor)

2. ขั้นตอนการเขียนโปรแกรมประสาน: ปัจจัยผลิตภาพที่มีผลในขั้นตอนนี้คือ

2.1. ปัจจัยข้อกำหนดการทำงาน (Operating specification factors): ACREL, ASPRT

2.2. ปัจจัยความซับซ้อนของตัวผลิตภัณฑ์และปัจจัยเกี่ยวกับผู้ขาย (Vendor and product complexity factors): ACPMT, ACSEW, APCPX, ACPPS, ACPTD

2.3. ปัจจัยข้อบังคับทางด้านเทคนิคของโครงการ (Project constraints factor): ACPER

2.4. ปัจจัยความสามารถของทีมที่ทำการรวม (Integration team maturity factors): ACIEP, ACIPC, AXCIP, APCON

3. การรวมระดับระบบและการทดสอบ ปัจจัยผลิตภาพที่มีผลในขั้นตอนนี้คือ

3.1. ปัจจัยข้อกำหนดการทำงาน (Operating specification factors): ACREL, ASPRT

3.2. ปัจจัยข้อบังคับทางด้านเทคนิคของโครงการ (Project constraints factor): ACPER

3.3. ปัจจัยความสามารถของทีมที่ทำการรวม (Integration team maturity factors): ACIEP, ACIPC, AXCIP, APCON

สำหรับรายละเอียดของแต่ละปัจจัยอยู่ในงานวิจัย (5, 33) และค่าของแต่ละปัจจัยอยู่ในภาคผนวก ส่วนความหมายของตัวย่อของปัจจัยสามารถดูได้จากรูปที่ 2.1 อนึ่งแบบจำลองที่นำเสนอนี้มีการประยุกต์ใช้ตัวคูณความเพียรพยายามในแบบจำลอง COCOTS เพื่อหาค่าของปัจจัยผลิตภาพ ซึ่งใช้หลักการเดียวกันกับหลักการที่เสนอโดย Boehm (34) โดยได้ใช้ค่าความเพียรพยายามในการปรับอัตราผลิตภาพ (Productivity) ยกตัวอย่างเช่น อัตราผลิตภาพของพนักงานแต่ละคนมีค่าเท่ากับ 250 SLOC/man-month โดยมีตัวคูณความเพียรพยายาม AXCIP (Integrator Experience with COTS Integration Processes) อยู่ในระดับปกติ (มีค่าเท่ากับ 1 ดูได้จากภาคผนวก) ถ้าประสบการณ์ของคนในทีมถูกพัฒนาสูงขึ้น ทำให้ปัจจัย AXCIP อยู่ในระดับสูง (มีค่าเท่ากับ 0.89 ดูได้จากภาคผนวก) ดังนั้นค่าปัจจัยผลิตภาพจึงมีค่าเท่ากับ 1/0.89

= 1.124 ซึ่งค่าอัตราผลิตภาพที่ถูกปรับใหม่จะมีค่าเป็น 250 x 1.124 = 280.90 SLOC/man-month ซึ่งจะถูกนำไปใช้ในขั้นตอนต่อไป

3.1.4 การประมาณความเพียรพยายามและระยะเวลาที่ใช้การรวมผลิตภัณฑ์ซอฟต์แวร์

งานวิจัยนี้มีสมมติฐานว่าขนาดของทีมไม่มีการเปลี่ยนแปลงระหว่างการพัฒนาระบบ และพนักงานทุกคนมีค่าอัตราผลิตภาพที่เท่ากัน ดังนั้นภายในทีมจะเสียเฉพาะต้นทุนค่าสื่อสารระหว่างสมาชิก (Communication overhead) เช่น การสื่อสารทางวาจา การสื่อสารทางเอกสารเป็นต้น ตามสมมติฐานดังกล่าวเมื่อนำไปใช้ในแบบจำลอง (8,10-12) จะได้สมการความสัมพันธ์ระหว่างจำนวนคนกับเวลาที่ใช้ในการพัฒนาดังสมการที่ 3.1

$$Time = \frac{Size}{Wf \times (Productivity - R_{CO})} \tag{3.1}$$

$Size$ = ขนาดของโครงการซอฟต์แวร์ซึ่งวัดในรูปแบบของจำนวนบรรทัดของโปรแกรม

Wf = จำนวนพนักงาน (Workforce)

R_{CO} = อัตราการพัฒนาซอฟต์แวร์ที่ลดลงจากการสื่อสารโดยใช้ความสัมพันธ์จากสมมติฐานของบรูคส์

$Productivity$ = อัตราผลิตภาพเฉลี่ยของพนักงานแต่ละคนมีหน่วยเป็นจำนวนบรรทัดต่อเดือนหรือขนาดฟังก์ชันต่อเดือน

ถ้ารวมการทำงานใหม่ (Rework) เนื่องจากการเปลี่ยนแปลงของผลิตภัณฑ์ซอฟต์แวร์ ในระหว่างการพัฒนาและมีการประยุกต์ใช้ปัจจัยผลิตภาพ (Productivity factor) ในขั้นตอนที่สามเราจะได้สมการประมาณเวลาดังสมการที่ 3.2

$$Time = \frac{Size(1 + Rework\%/100)}{Wf \times (AdjustedProductivity(1 - R_{CO}))} \tag{3.2}$$

$Size$ = ขนาดของโครงการซอฟต์แวร์โดยวัดในรูปแบบของจำนวนบรรทัดของโปรแกรมในขั้นตอนการเขียนโปรแกรมประสานและการรวมระดับระบบ และในรูปแบบของขนาดฟังก์ชันสำหรับขั้นตอนของการปรับแต่ง

$Rework\%$ = เปอร์เซ็นต์ของการทำงานใหม่เนื่องจากการเปลี่ยนผลิตภัณฑ์ซอฟต์แวร์ ในระหว่างการพัฒนา

Wf = จำนวนพนักงาน

R_{CO} = อัตราการพัฒนาซอฟต์แวร์ที่ลดลงจากการสื่อสาร ในที่นี้ได้คิดเป็นร้อยละของอัตราผลิตภาพที่เสียไป ($0.0006n^2$) (8)

$$AdjustedProductivity = Productivity \times \prod_i Productivity\ factors_i \quad (3.3)$$

$Productivity\ factors$ = ค่าปัจจัยผลิตภาพของแต่ละปัจจัยที่ผู้ใช้ระบุแต่ละขั้นตอน

สมการมีข้อจำกัดที่ว่าค่า R_{CO} < 1 ดังนั้นค่า n ซึ่งแทนด้วยจำนวนคนต้องมีค่าไม่เกิน 40 คน ซึ่งถ้าจำนวนคนที่ใช้ในแต่ละขั้นตอนเกิน 40 คนจะไม่สามารถใช้สมการดังกล่าวได้

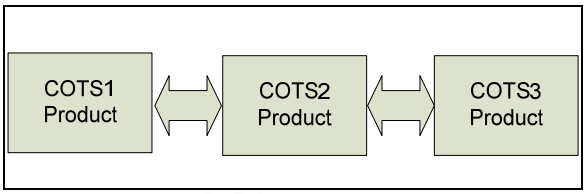
ถ้าขั้นตอนการพัฒนาดำเนินการอย่างเป็นลำดับ เวลาที่ได้จะเท่ากับผลรวมในแต่ละขั้นตอน แต่เนื่องจากในบางขั้นตอนสามารถที่จะทำงานอย่างควบคู่กันได้ ดังนั้นจะหาระยะเวลาในการพัฒนาได้จากการแบ่งพนักงานที่มีอยู่ไปทำงานยังขั้นตอนต่างๆ และทำการจำลองเพื่อหาระยะเวลาที่โครงการจะแล้วเสร็จ โดยผลลัพธ์ที่ได้จะให้ความสัมพันธ์ระหว่างจำนวนคนที่ใช้ในแต่ละเวลา โดยพื้นที่ใต้กราฟดังกล่าวก็คือความเพียรพยายามซึ่งสามารถนำค่านี้ไปคำนวณต้นทุนของการพัฒนาซอฟต์แวร์ต่อไป

3.2 แบบจำลองการประมาณต้นทุนเพื่อการจัดวางขั้นตอนกระบวนการพัฒนา

เนื่องจากแบบจำลองการประมาณต้นทุนเพื่อการจัดวางแผนบุคคลนั้นไม่ได้พิจารณาผลกระทบที่อาจจะเกิดขึ้นจากการทำคู่ขนานกันในบางขั้นตอน ถึงแม้การทำคู่ขนานจะลดเวลาพัฒนาน้อยลงแต่ถ้าขั้นตอนหนึ่งมีการแก้ไขก็อาจจะส่งผลกระทบต่ออีกขั้นตอนหนึ่งซึ่งเป็นผลทำให้ต้นทุนสูงขึ้น ดังนั้นงานวิจัยนี้จึงนำเสนอแบบจำลองการประมาณต้นทุนเพื่อการจัดวางขั้นตอนกระบวนการพัฒนาซึ่งแสดงผลลัพธ์เป็นเวลาและต้นทุนที่ใช้จากการจัดวางขั้นตอนโดยผู้วิจัยได้นำเอาหลักการวิศวกรรมคู่ขนานมาใช้วิเคราะห์การพัฒนาซอฟต์แวร์โดยใช้ผลิตภัณฑ์ซอฟต์แวร์ ซึ่งเป็นกระบวนการพัฒนาที่สามารถทำการแบ่งแยกส่วนกันพัฒนาได้อย่างชัดเจน โดยได้พิจารณาการขึ้นต่อกันระหว่างขั้นตอนเมื่อมีการทำคู่ขนานกัน (Concurrent dependency) และใช้การจำลองแบบไม่ต่อเนื่องเพื่อจำลองความไม่แน่นอนที่เกิดจากการเปลี่ยนแปลงของความต้องการ โดยได้ประยุกต์ใช้แบบจำลองความไม่แน่นอนของการเปลี่ยนแปลงทางวิศวกรรมในหัวข้อ 2.1.5 เพื่อหาผลกระทบของการจัดวางขั้นตอนการพัฒนาในรูปแบบต่างๆ

กรณีตัวอย่างของการพัฒนาซอฟต์แวร์จากผลิตภัณฑ์ซอฟต์แวร์ หลังจากผ่านการคัดเลือกและประเมินแล้วแสดงได้ดังรูปที่ 3.3 ซึ่งในกรณีนี้ประกอบด้วยผลิตภัณฑ์ซอฟต์แวร์สามผลิตภัณฑ์ โดยผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่งอาจเป็นซอฟต์แวร์ที่เกี่ยวข้องกับระบบจัดการ

หน้าร้าน ผลิตภัณฑ์ซอฟต์แวร์ที่สองเป็นซอฟต์แวร์ที่เกี่ยวข้องกับระบบจัดการฐานข้อมูล และผลิตภัณฑ์ซอฟต์แวร์ที่สามเป็นซอฟต์แวร์ที่เกี่ยวข้องกับระบบจัดการหลังร้าน ซึ่งแต่ละส่วนมีความสัมพันธ์ซึ่งกันและกัน

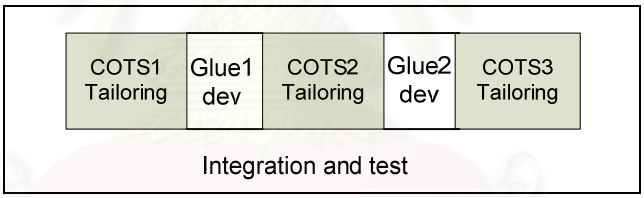


รูปที่ 3.3 ภาพจำลองของการรวมผลิตภัณฑ์ซอฟต์แวร์เข้าด้วยกัน

การสร้างแบบจำลองเพื่อใช้ในการวางแผนกระบวนการพัฒนาและประมาณต้นทุนที่ใช้ในการพัฒนาซอฟต์แวร์จากผลิตภัณฑ์ซอฟต์แวร์ สามารถแบ่งเป็นขั้นตอนต่างๆ ตามนี้

3.2.1 แบบจำลองงานที่ต้องทำในการรวมผลิตภัณฑ์ซอฟต์แวร์เข้าด้วยกัน

ในขั้นตอนนี้ผู้ใช้ต้องทำการระบุงานที่เกิดขึ้นจากการพัฒนาซอฟต์แวร์โดยใช้ผลิตภัณฑ์ซอฟต์แวร์ ซึ่งในกรณีตัวอย่างนี้ เราสามารถวิเคราะห์งานที่เกิดขึ้นจากการรวมผลิตภัณฑ์ซอฟต์แวร์ทั้งสาม ได้ตามรูปที่ 3.4



รูปที่ 3.4 รายละเอียดของการรวมผลิตภัณฑ์ซอฟต์แวร์ทั้งสามผลิตภัณฑ์

1. การปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ (Tailoring) เป็นการปรับแต่งฟังก์ชันการทำงานของผลิตภัณฑ์ให้เข้ากับความต้องการและเพื่อเตรียมพร้อมสำหรับการรวมระบบ ดังนั้นงานที่ทำการปรับแต่งจะมีทั้งหมดสามส่วนคือ ส่วนที่ปรับแต่งของผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่ง ส่วนที่ปรับแต่งของผลิตภัณฑ์ซอฟต์แวร์ที่สอง และส่วนที่ปรับแต่งของผลิตภัณฑ์ซอฟต์แวร์ที่สาม

2. การเขียนโปรแกรมประสาน (Glue code) เป็นการเขียนโปรแกรมเพื่อเชื่อมต่อการทำงานระหว่างผลิตภัณฑ์ซอฟต์แวร์หรือเขียนเพื่อที่จะเพิ่มฟังก์ชันที่ไม่สมบูรณ์ของผลิตภัณฑ์ซอฟต์แวร์ ในที่นี้เราได้แบ่งงานเป็นสองส่วนคือ การพัฒนาโปรแกรมประสานส่วนที่หนึ่งเป็นโปรแกรมประสานที่เชื่อมการทำงานระหว่างผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่งกับสอง การพัฒนาโปรแกรมประสานส่วนที่สองเป็นโปรแกรมประสานที่เชื่อมการทำงานระหว่างผลิตภัณฑ์ซอฟต์แวร์ที่สองและสาม

3. การรวมและทดสอบระดับระบบ (System-level integration and test) เป็นการทดสอบระบบรวมทั้งหมดเพื่อทดสอบการทำงานระหว่างผลิตภัณฑ์ซอฟต์แวร์ หลังจากพัฒนาระบบจนใกล้เสร็จสมบูรณ์

โดยสรุปในขั้นตอนที่หนึ่งเราสามารถแบ่งงานในการรวมระบบในแต่ละสองผลิตภัณฑ์ได้เป็น 6 งานได้ตามนี้คือ

1. การปรับแต่งของผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่ง (T1)
2. การปรับแต่งของผลิตภัณฑ์ซอฟต์แวร์ที่สอง (T2)
3. การปรับแต่งของผลิตภัณฑ์ซอฟต์แวร์ที่สาม (T3)
4. การพัฒนาโปรแกรมประสานส่วนที่หนึ่ง (G1)
5. การพัฒนาโปรแกรมประสานส่วนที่สอง (G2)
6. การรวมและทดสอบระบบ (I)

โดยงานที่หนึ่งสองและสามมาจากส่วนการปรับแต่งของผลิตภัณฑ์ ส่วนงานที่สี่และงานที่ห้ามาจากส่วนการเขียนโปรแกรมประสาน ส่วนงานที่หกมาจากการรวมและทดสอบระบบ ซึ่งงานที่ระบุในข้างต้นเป็นงานที่เกิดขึ้นโดยทั่วไปของการรวมผลิตภัณฑ์ซอฟต์แวร์ แต่ผู้ใช้สามารถที่จะปรับเปลี่ยนขั้นตอนได้ตามความเหมาะสมยกตัวอย่างเช่น อาจจะมีการเพิ่มขั้นตอนการออกแบบ (Design phase) หรือละเว้นขั้นตอนบางอย่างที่ไม่สนใจออกได้

3.2.2 ปัจจัยผลิตภาพในขั้นตอนการรวมผลิตภัณฑ์ซอฟต์แวร์

ใช้หลักการเดียวกันกับหัวข้อ 3.1.3

3.2.3 การประมาณต้นทุนและระยะเวลาที่ใช้การรวมผลิตภัณฑ์ซอฟต์แวร์ในแต่ละขั้นตอน

1. การประมาณระยะเวลาการพัฒนาของแต่ละงาน เราจะประมาณเวลาที่ใช้โดยใช้หลักการจากหัวข้อที่ 3.1.4

2. การประมาณต้นทุนที่ใช้ในการพัฒนาของแต่ละงาน ขึ้นอยู่กับพนักงานและเวลาในการพัฒนา โดยเราจะประมาณต้นทุนดังสมการ 3.4

$$Cost = Duration \times Wf \times Salary \tag{3.4}$$

โดยที่

$Duration$ = ระยะเวลาในการพัฒนาซอฟต์แวร์ (สัปดาห์)

Wf = จำนวนพนักงาน (คน)

$Salary$ = ค่าจ้างพนักงานต่อวันสำหรับขั้นตอนนั้น (บาท)

นอกจากนี้ผู้ใช้ยังต้องระบุเวลาเริ่มต้นของแต่ละงานและค่าใช้จ่ายถ้าเกิดมีการทำงานใหม่ (Rework) ต่อสัปดาห์ โดยข้อมูลทั้งหมดจะถูกแสดงดังตารางที่ 3.1

ตารางที่ 3.1 เวลาเริ่มต้น เวลากับต้นทุนที่ใช้ และต้นทุนที่เสียถ้ามีการทำงานใหม่

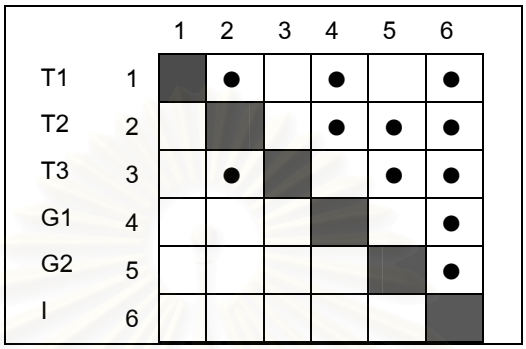
Activities		**Start time**	**Duration(weeks)**	**Cost(฿)**	**Rework cost per week(฿)**
ID	**Name**				
1	T1	0	6	40,000	10,000
2	T2	First Month	7	60,000	8,000
3	T3	0	6	50,000	9,000
4	G1	$0.4\,t_2$	10	150,000	11,000
5	G2	$0.5\,t_2$	9	170,000	12,000
6	I	$0.6\,t_4$	9	45,000	6,000

3.2.4 การวิเคราะห์การขึ้นต่อกันระหว่างงานโดยใช้เมทริกซ์โครงสร้างการออกแบบ

จากกรณีตัวอย่างได้เริ่มต้นที่ขั้นตอนของการปรับแต่งโดยที่ขั้นตอนการออกแบบได้เสร็จสิ้นลงแล้ว T1, T2, และ T3 จึงเริ่มได้ทันที ส่วนงาน G1 และ G2 ต้องรอให้งานในส่วนของการปรับแต่งบางส่วนเสร็จสมบูรณ์ก่อนจึงจะพัฒนาได้อย่างเต็มรูปแบบ และงาน I สามารถเริ่มได้ก็ต่อเมื่อระบบเกือบจะเสร็จสมบูรณ์แล้ว แม้กระนั้นก็ตามอาจมีการเปลี่ยนแปลงความต้องการในระหว่างการพัฒนา ดังนั้นเราต้องวิเคราะห์การขึ้นแก่กันระหว่างงานเพื่อวิเคราะห์ผลกระทบที่เกิดขึ้นเมื่องานใดงานหนึ่งมีการเปลี่ยนแปลง

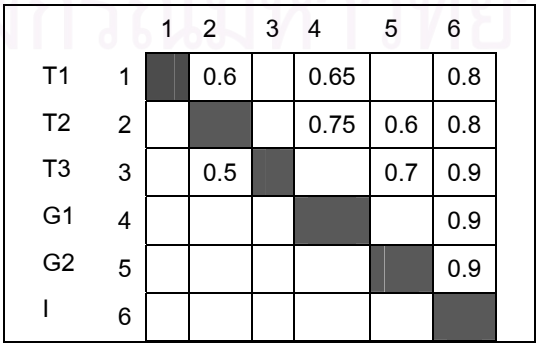
ตามกรณีตัวอย่างพิจารณาความสัมพันธ์ระหว่าง งาน T1 งาน T2 และงาน G1มีความสัมพันธ์กันเมื่อมีการเปลี่ยนแปลงฟังก์ชันการทำงานในผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่ง ต้องมีการแก้ไขในส่วนของฐานข้อมูล (งาน T2) และส่วนของโปรแกรมประสานระหว่างผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่งและสอง (งาน G1) เนื่องจากค่าที่ใช้ส่งพารามิเตอร์ของแต่ละส่วนต่อประสานมีการเปลี่ยนแปลง ซึ่งเป็นเช่นเดียวกันกับความสัมพันธ์ระหว่าง งาน T3 งาน T2 และงาน G2

ส่วนถ้ามีการแก้ไขงานในส่วนฐานข้อมูลต้องมีการแก้ไขโปรแกรมประสานทั้งสอง (งาน G1,G2) และงานในส่วนของการปรับแต่งและการเขียนโปรแกรมประสานจะมีผลต่อการรวมและทดสอบระบบ เพราะถ้ามีการเปลี่ยนแปลงในงานดังกล่าวก็ต้องทำการทดสอบใหม่ จะได้ความสัมพันธ์ดังรูปที่ 3.5



รูปที่ 3.5 ตารางเมทริกซ์โครงสร้างการออกแบบการส่งของข้อมูลระหว่างกัน

นอกจากความสัมพันธ์ระหว่างงานแล้ว ระดับการขึ้นแก่กันระหว่างงาน (Dependency level) ก็มีความสำคัญเช่นกัน ถ้าระหว่างงานสองงานใดมีระดับการขึ้นแก่กันสูงก็จะมีโอกาสที่การแก้ไขในงานหนึ่งจะส่งผลกระทบต่ออีกงานหนึ่งสูงด้วยโดยเรียกว่า ความน่าจะเป็นของการเกิดการทำงานใหม่ (Rework probability) (32) ของงานนั้น ในการพัฒนาซอฟต์แวร์โดยใช้ผลิตภัณฑ์ซอฟต์แวร์ระดับการขึ้นแก่กันเราสามารถพิจารณาได้จากจุดเชื่อมต่อส่วนประสาน (Interface point) ซึ่งแต่ละจุดเชื่อมต่อจะประกอบด้วยจำนวนของค่าพารามิเตอร์เข้า (Input parameter) และค่าพารามิเตอร์ออก (Output parameter) โดยการคู่ควบ (Coupling) สามารถวัดได้จากชนิดของข้อมูลและวิธีการส่งข้อมูล (35) ถ้าระหว่างผลิตภัณฑ์ใดมีการคู่ควบสูงงานที่เกิดจากการรวมผลิตภัณฑ์นั้นก็จะมีระดับการขึ้นแก่กันระหว่างงานสูงด้วย โดยเราระบุค่าความน่าจะเป็นของการเกิดการทำงานใหม่ในเมทริกซ์โครงสร้างการออกแบบได้ดังรูปที่ 3.6



รูปที่ 3.6 ตารางเมทริกซ์โครงสร้างการออกแบบความน่าจะเป็นของการเกิดการทำงานใหม่

3.2.5 แบบจำลองความไม่แน่นอนของการเปลี่ยนแปลงความต้องการและผลกระทบที่เกิดขึ้นของแต่ละงาน

1. ความไม่แน่นอนของการเกิดการเปลี่ยนแปลงการความต้องการ (Requirement change)

งานวิจัยนี้ได้สร้างแบบจำลองการเปลี่ยนแปลงความต้องการซึ่งมีลักษณะรูปแบบการเปลี่ยนแปลงความต้องการตามงานวิจัย (18) โดยใช้หลักการจำลองในรูปแบบกระบวนการปัวส์ซองแบบไม่คงที่ซึ่งเป็นหลักการเดียวกับการจำลองการเปลี่ยนแปลงทางวิศวกรรมที่นำเสนอโดย Loch และคณะ (16) จะได้สมการค่าเฉลี่ยจำนวนการเปลี่ยนแปลงความต้องการที่เวลาใดๆ ได้ดังสมการที่ 3.5 ซึ่งจะให้กราฟการเปลี่ยนแปลงความต้องการมีรูปร่างดังรูปที่ 2.5

$$\mu(t) = \begin{cases} \dfrac{\mu_{\max}}{t_{\max}}(t) & 0 \le t \le t_{\max} \\ \dfrac{\mu_{\max}}{(t_{\max} - t_{cease})}(t - t_{cease}) & t_{\max} < t \le t_{cease} \end{cases} \quad (3.5)$$

โดยที่

$\mu(t)$ คือค่าเฉลี่ยจำนวนการเปลี่ยนแปลงความต้องการที่เวลาใด ๆ และ t คือเวลา ณ ขณะใดๆ $t \in [0, t_{cease}]$

$\mu_{\max}$ คือค่าเฉลี่ยจำนวนการเปลี่ยนแปลงความต้องการสูงสุด

$t_{\max}$ คือระยะเวลาที่เกิดจำนวนการเปลี่ยนแปลงความต้องการสูงสุด

สำหรับจำนวนการเปลี่ยนแปลงความต้องการที่เกิดขึ้นทั้งหมดและจำนวนการเปลี่ยนแปลงความต้องการที่เกิดขึ้นเฉลี่ยในเวลาหนึ่งมีค่าดังสมการที่ 3.6 กับ 3.7

จำนวนการเปลี่ยนแปลงความต้องการที่เกิดขึ้นทั้งหมด = $\frac{1}{2}(\mu_{\max})(t_{cease})$ (3.6)

จำนวนการเปลี่ยนแปลงความต้องการที่เกิดขึ้นเฉลี่ยในเวลาหนึ่ง = $\frac{\mu_{\max}}{2}$ (3.7)

จากกรณีตัวอย่างได้กำหนดจำนวนการเปลี่ยนแปลงความต้องการที่เกิดขึ้นบนงานต่างๆ และช่วงเวลาของการเกิดการเปลี่ยนแปลงที่มากสุดดังตารางที่ 3.2

ตารางที่ 3.2 การเปลี่ยนแปลงความต้องการทั้งหมดที่เกิดขึ้นและช่วงเวลาที่เกิดการเปลี่ยนแปลงสูงสุดของแต่ละงาน

Activities		**Total change**	**Max change time (% of task duration)**
NO	**Name**		
1	T1	45	60%
2	T2	Depend on T1 and T3	-
3	T3	60	40%
4	G1	40	40%
5	G2	32	20%
6	I	Depend on T1,T2,T3,G1,G2	-

2. ผลกระทบที่เกิดขึ้นเนื่องจากการเปลี่ยนแปลงที่เกิดขึ้นบนงานอื่น

เมื่อพิจารณาจากขั้นตอนที่ 3.2.4 จะสามารถรู้ได้ว่างานใดบ้างที่ส่งผลกระทบซึ่งกันและกัน โดยให้ขนาดของผลกระทบขึ้นอยู่กับปริมาณงานที่ทำไป ถ้าปริมาณงานที่ทำเสร็จไปแล้วมากเมื่อมีการแก้ไขก็ย่อมต้องแก้ไขมากเช่นกัน ดังสมการที่ 3.8 (16)

$$f_{impact}(t) = kt \tag{3.8}$$

t เป็น ณ ขณะเวลาที่ทำงานอยู่บนงานนั้นมีค่า $0 \le t \le t_{finish}$

k เป็นเปอร์เซ็นต์ของจำนวนเวลาที่ต้องทำงานใหม่เนื่องจากการเปลี่ยนแปลงความต้องการของงานที่เกี่ยวข้อง โดยค่าของ k ที่ใช้จะมีค่าตั้งแต่ 0 ถึง 10 (36) ดังแสดงได้รูปที่ 3.7

		1	2	3	4	5	6
T1	1		3%	0	5%	0	3%
T2	2	0		0	6%	7%	4%
T3	3	0	4%		0	8%	3.5%
G1	4	0	0	0		0	4.5%
G2	5	0	0	0	0		5%
I	6	0	0	0	0	0	

รูปที่ 3.7 เปอร์เซ็นต์ของฟังก์ชันผลกระทบในแต่ละงาน

3.2.6 ขั้นตอนวิธีการจำลอง (Simulation approach)

ในขั้นตอนนี้จะนำค่าจากข้อมูลในตารางที่ 3.1 ตารางที่ 3.2 รูปที่ 3.6 และ รูปที่ 3.7 มาทำการจำลองโดยใช้การจำลองแบบไม่ต่อเนื่อง (Discrete- event simulation) โดยได้ทำการจำลองจุดเวลาที่เกิดเหตุการณ์การเปลี่ยนแปลงความต้องการของแต่ละขั้นตอน จากนั้นพิจารณาเหตุการณ์ที่เกิดขึ้นทั้งหมดในโครงการเป็นอันดับ เมื่อเกิดการเปลี่ยนแปลงในแต่ละเหตุการณ์ก็พิจารณาว่าส่งผลกระทบต่อขั้นตอนใดบ้างจากนั้นนำขั้นตอนที่ได้รับผลกระทบมาพิจารณาว่ามีผลต่อขั้นตอนใดบ้างอีกครั้งหนึ่ง ซึ่งในหนึ่งรอบของการจำลองสามารถเขียนเป็นขั้นตอนที่ใช้จำลองได้ดังนี้

1. กำหนดค่าเริ่มต้น

1.1. กำหนดค่าเริ่มต้นให้กับตัวแปรต่างๆ ตามตารางที่ 3.1

1.2. กำหนดค่าตารางเมทริกซ์โครงสร้างการออกแบบ

1.2.1 ความน่าจะเป็นของการเกิดการทำงานใหม่ ($[R_{i,j}]$)

1.2.2 เปอร์เซ็นต์ของฟังก์ชันผลกระทบในแต่ละงาน ($[I_{i,j}]$)

2. สุ่มเวลาของการเกิดการเปลี่ยนแปลงด้วยการแจกแจงปัวส์ซองแบบไม่คงที่ของแต่ละขั้นตอนจากข้อมูลในตารางที่ 3.2

3. หาจุดเวลาที่เกิดการเปลี่ยนแปลงการออกแบบ

4. เมื่อมีการเปลี่ยนแปลงเกิดขึ้นในงานที่ i (W_i) โดยให้ $WorkDone_i$ คือจำนวนงานที่ทำไปแล้วบนงานที่ i

4.1 ตรวจสอบทุก j ของ $[R]$แถว W_i ให้ W_y เป็นงานที่ค่าใน $[R]$ ที่ช่องดังกล่าวมีค่าไม่เท่ากับศูนย์และสุ่มค่าตัวเลขที่อยู่ระหว่าง 0 ถึง 1 ถ้าค่าที่ได้น้อยกว่าหรือเท่ากับให้ทำขั้นตอนถัดไป แต่ถ้ามากกว่าให้ข้ามกลับไปหางาน W_y ถัดไป ถ้าไม่มี W_y ให้กลับไปทำดังข้อ 3 เพื่อหาจุดเวลาที่เกิดการเปลี่ยนแปลงถัดไป

4.1.1 ระยะเวลารวมของ W_y (D_y)

D_y = ระยะเวลารวมของงาน W_y ก่อนหน้า $+(I_{i,y} \times WorkDone_y)$

4.1.2 ต้นทุนรวมของ W_y (C_y)

C_y = ต้นทุนรวมของงาน W_y ก่อนหน้า $+(I_{i,y} \times WorkDone_y \times reworkcost_y)$

4.2. ตรวจสอบทุก j ของ $[R]$แถว W_y ให้ W_z เป็นงานที่ค่าใน $[R]$ ที่ช่องดังกล่าวมีค่าไม่เท่ากับศูนย์และสุ่มค่าตัวเลขที่อยู่ระหว่าง 0 ถึง 1 ถ้าค่าที่ได้น้อยกว่าหรือเท่ากับให้คำนวณระยะเวลารวมและต้นทุนรวมของ W_z จากสมการในข้อ 4.1.1 และ 4.1.2 แต่ถ้ามากกว่าให้ข้ามกลับไปหางาน W_z ถัดไป ทำจนครบทุก W_z และกลับไปทำ W_y ถัดไป (กลับไป 4.1 ใหม่)

5. เมื่อทำจนครบทุกจุดเวลาที่มีการเปลี่ยนแปลงเกิดขึ้นแล้วให้แสดงเวลาและต้นทุนรวม

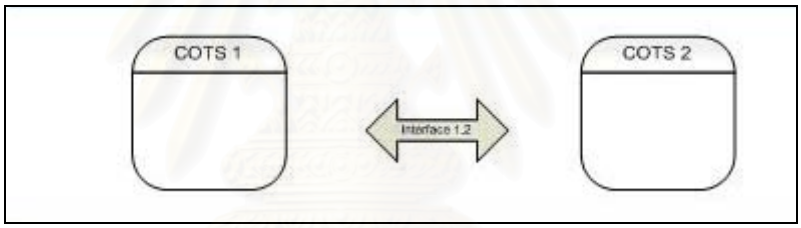
บทที่ 4

การทดลอง

งานวิจัยนี้ได้เสนอแบบจำลองการประมาณต้นทุนเพื่อการจัดวางแผนบุคคลและการจัดวางขั้นตอนกระบวนการพัฒนาในการพัฒนาซอฟต์แวร์โดยใช้ผลิตภัณฑ์ซอฟต์แวร์ ซึ่งจะใช้แบบจำลองในการวิเคราะห์กรณีตัวอย่างที่กำหนดขึ้นมา โดยในบทนี้จะกล่าวถึงรายละเอียดต่างๆ ของการทดลอง ซึ่งจะแบ่งการทดลองเป็นสองส่วนคือการทดลองของแบบจำลองการประมาณต้นทุนเพื่อการจัดวางแผนบุคคล และการทดลองของแบบจำลองการประมาณต้นทุนเพื่อการจัดวางขั้นตอนกระบวนการพัฒนา

4.1 การทดลองของแบบจำลองการประมาณต้นทุนเพื่อการจัดวางแผนบุคคล

การทดลองจะกำหนดกรณีขึ้นมาเพื่อใช้แบบจำลองวิเคราะห์เวลาและต้นทุนที่ใช้ในการพัฒนาโครงการโดยพิจารณาจากการจัดวางแผนบุคคลที่จะต้องใช้ โดยกรณีตัวอย่างเป็นไปตามดังรูปที่ 4.1



รูปที่ 4.1 กรณีตัวอย่างของการทดลองแบบจำลองการจัดวางแผนบุคคล

กรณีตัวอย่างเป็นการสร้างระบบตรวจสุขภาพซึ่งมีการใช้ผลิตภัณฑ์ซอฟต์แวร์สองผลิตภัณฑ์มาทำการเชื่อมต่อจนเป็นระบบโดยค่าต่างๆ ที่พิจารณาได้ระบุไว้เป็นกรณีตัวอย่างที่ใช้วิเคราะห์ดังตารางที่ 4.1

ตารางที่ 4.1 ค่าพารามิเตอร์ของโครงการที่ใช้ในแบบจำลอง

	COTS1 Product	**COTS2 Product**
Application type	Health Monitoring Systems	
COTS Product type	CRM	Database
Workforce	20	
Expected Change in COTS product during development	Yes	No

Total work of each activity		
Tailoring (Function size)	2000+25%	2000
Glue Code (SLOC)	3500+30%	3000
System Integration(SLOC)	7550	
Productivity in each phase		
Productivity in tailoring	250 Function size/man-month	
Productivity in glue code development	300 SLOC/man-month	
Productivity in system integration	500 SLOC/man-month	
ACPPS (COTS supplier product support)	High	Nominal
ACPTD (COTS supplier provided training and documentation)	Low	Nominal
Productivity factors		
ACIPC (COTS integrator personnel capability)	High	
AXCIP (Integrator experience with COTS integration processes)	High	
APCON (Integrator personnel continuity)	Nominal	
ACREL (Constraints on application system reliability)	High	
ASPRT (Application system portability)	Nominal	
ACPER (Constraints on COTS technical performance)	Very High	
Tailoring complexity	High	Nominal
APCPX (COTS product interface complexity)	Nominal	

ACPMT (COTS product maturity)	High	Nominal
ACSEW (Supplier product extension willingness)	Nominal	
ACIEP (COTS integrator experience with product)	Low	Nominal
ACPPS (COTS supplier product support)	High	Nominal
ACPTD (COTS supplier provided training and documentation)	Low	Nominal

นอกจากนี้ได้กำหนดให้ขั้นตอนของการพัฒนาโปรแกรมประสานเริ่มต้น เมื่องานในส่วนของขั้นตอนการปรับแต่งเสร็จสิ้นไปแล้ว 75% ของงานทั้งหมดในขั้นตอนการปรับแต่ง และกำหนดให้ขั้นตอนของการรวมและทดสอบระบบเริ่มต้นเมื่องานในส่วนของขั้นตอนการพัฒนาโปรแกรมประสานเสร็จไปแล้ว 80 % ของงานทั้งหมดในขั้นตอนนี้ และกำหนดให้เมื่อเกิดการปรับเปลี่ยนระหว่างการพัฒนาผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่งคือระบบลูกค้าสัมพันธ์จะส่งผลให้ต้องทำงานเพิ่มในขั้นตอนของการปรับแต่ง 25 % และทำงานเพิ่มในเฟสของการเขียนโปรแกรมประสาน 30% จากงานที่มีอยู่เดิมในขั้นตอนนั้น

4.1.1 การศึกษาการใช้บุคคลในแต่ละช่วงเวลาของการพัฒนาซอฟต์แวร์โดยใช้ผลิตภัณฑ์ซอฟต์แวร์

ในการทดลองกำหนดให้ใช้คนพัฒนาทั้งสิ้น 20 คน โดยแต่ละคนทำงานเพียงหนึ่งงาน ซึ่งจัดแบ่งบุคคลเป็น 5 กลุ่มไปทำงานได้ดังนี้ พนักงาน 3 คนทำงานในส่วนการปรับแต่งของผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่ง พนักงาน 2 คนทำงานในส่วนการปรับแต่งของผลิตภัณฑ์ซอฟต์แวร์ที่สอง พนักงาน 5 คนทำงานในส่วนของการพัฒนาโปรแกรมประสานของผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่ง พนักงาน 4 คนทำงานในส่วนของการพัฒนาโปรแกรมประสานของผลิตภัณฑ์ซอฟต์แวร์ที่สอง พนักงาน 6 คนทำงานในส่วนของการรวมและทดสอบระบบโดยใช้แบบจำลองการประมาณต้นทุนและเวลาที่ใช้ในการพัฒนาจากการจัดวางบุคคลซึ่งอาจเขียนย่อเป็น 3,2,5,4,6

4.1.2 การศึกษารูปแบบการจัดบุคคลที่ใช้เวลาและต้นทุนน้อยที่สุด

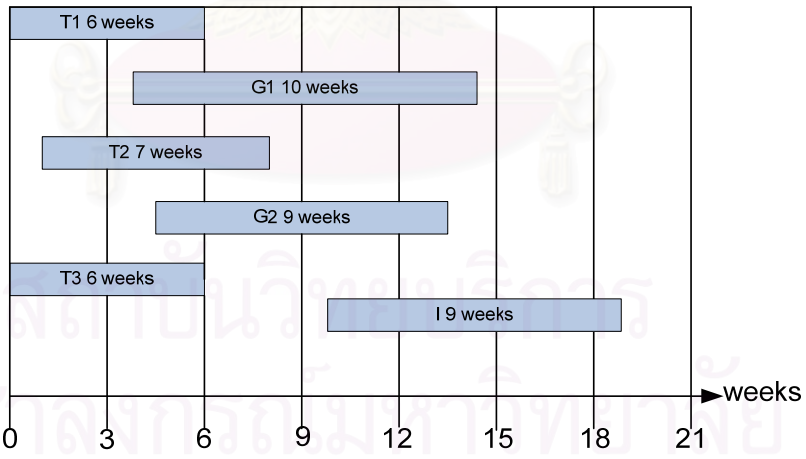
ในการทดลองนี้กำหนดให้ใช้คนทั้งหมด 20 คน โดยจำลองการแบ่งกลุ่มพนักงานเป็น 5 กลุ่ม เช่น 4,4,4,4,4 หรือ 3,4,5,3,4 เป็นต้น โดยจะจำลองทุกรูปแบบของการจัด และหารูปแบบการจัดบุคคลที่ใช้เวลาน้อยที่สุดในการพัฒนา และการจัดบุคคลที่ใช้ต้นทุนน้อยที่สุดในการพัฒนา

4.1.3 การศึกษาความสัมพันธ์ระหว่างจำนวนบุคคลที่ใช้กับระยะเวลาของการพัฒนาที่น้อยที่สุด

การจำลองเพื่อหาความสัมพันธ์ระหว่างจำนวนคนที่ใช้กับเวลาที่น้อยที่สุด โดยจำลองการแบ่งกลุ่มของพนักงานตั้งแต่ 10 ถึง 70 คนเพื่อหาเวลาที่ใช้น้อยที่สุดของในแต่ละจำนวนพนักงานที่ใช้

4.2 การทดลองของแบบจำลองการประมาณต้นทุนเพื่อการจัดวางขั้นตอนกระบวนการพัฒนา

การทดลองจะกำหนดกรณีศึกษาขึ้นมาเพื่อใช้แบบจำลองวิเคราะห์เวลาและต้นทุนที่ใช้ในการพัฒนาโครงการโดยพิจารณาในแง่การจัดวางขั้นตอนกระบวนการพัฒนาโดยที่แต่ละขั้นตอนของการพัฒนามีความสัมพันธ์กัน ซึ่งได้นำกรณีตัวอย่างดังรูปที่ 3.3 และข้อมูลที่ต้องใช้ในแบบจำลองทั้งหมดสรุปได้ในตารางที่ 3.1 ตารางที่ 3.2 รูปที่ 3.6 และรูปที่ 3.7 โดยนำมาเขียนเป็นแผนภูมิแกนต์ได้ดังรูปที่ 4.2



รูปที่ 4.2 แผนภูมิแกนต์ของกรณีศึกษาโครงการพัฒนาซอฟต์แวร์

4.2.1 การประมาณระยะเวลาและต้นทุนที่ใช้ในการพัฒนา

นำข้อมูลที่กำหนดให้ในเบื้องต้นมาทำการจำลองโดยใช้แบบจำลองการจัดวางขั้นตอนกระบวนการพัฒนา โดยทำการจำลองทั้งหมด 50,000 ครั้งและนำผลที่ได้มาวิเคราะห์หา

ค่าเฉลี่ย ส่วนเบี่ยงเบนมาตรฐานของระยะเวลาและต้นทุนทั้งหมดที่จะต้องใช้ในโครงการและพิจารณาการทำงานใหม่ที่ได้รับผลกระทบจากขั้นตอนอื่นในแต่ละขั้นตอน

4.2.2 การประมาณระยะเวลาและต้นทุนที่ใช้ในการทำงานใหม่เนื่องจากผลกระทบจากขั้นตอนอื่น

การทดลองนี้ได้นำผลลัพธ์ที่ได้จากขั้นตอนที่ 4.2.1 มาทำการวิเคราะห์ต้นทุนและเวลาที่เสียไปเนื่องจากการทำงานใหม่ซึ่งเป็นผลมาจากการเปลี่ยนแปลงความต้องการของขั้นตอนอื่น โดยวิเคราะห์แยกเป็นแต่ละขั้นตอนเพื่อดูผลกระทบที่เกิดขึ้นจากการจัดวางขั้นตอนการพัฒนาดังกล่าว

4.2.3 การวิเคราะห์ความเป็นไปได้ของโครงการที่จะสำเร็จตามเป้าหมายที่ตั้งไว้

การทดลองนี้ต้องการนำผลลัพธ์ที่ได้จากการจำลองมากำหนดเป้าหมายด้านเวลาและต้นทุนที่ใช้ในการพัฒนา โดยมีความน่าจะเป็นที่จะเสร็จตรงตามเป้าหมายที่วางไว้ 95 เปอร์เซ็นต์

4.2.4 การศึกษาผลกระทบที่เกิดจากระยะเวลาการเริ่มต้นขั้นตอนกระบวนการพัฒนาที่แตกต่างกัน

ขั้นตอนนี้จะวิเคราะห์ผลกระทบจากระยะเวลาที่ใช้ในการเริ่มต้นของขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สอง โดยจะทดลองระยะเวลาที่ใช้การเริ่มต้นขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สองที่แตกต่างกันตั้งแต่เดือนที่เริ่มต้นโครงการจนถึงเดือนที่หก โดยวิเคราะห์ระยะเวลาและต้นทุนที่ใช้เพื่อช่วยในการตัดสินใจของผู้พัฒนา

นอกจากนี้ได้ทำการวิเคราะห์ขั้นตอนอื่นๆ โดยยกตัวอย่างขั้นตอนการรวมและทดสอบระบบเพราะเป็นขั้นตอนที่กำหนดระยะเวลาทั้งหมดที่ใช้ในการพัฒนาของโครงการ โดยจะให้มีการเริ่มต้นขั้นตอนที่ 50% 60% และ 70% ของขั้นตอนการพัฒนาโปรแกรมประสานที่หนึ่ง โดยนำผลที่ได้มาวิเคราะห์ระยะเวลาและต้นทุนที่ใช้

บทที่ 5

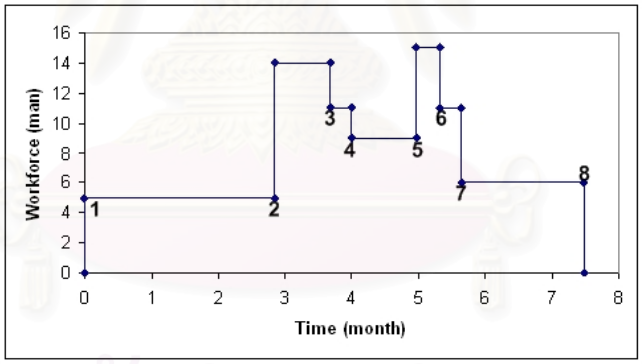
ผลการทดลองและวิเคราะห์ผล

ในบทนี้จะกล่าวถึงผลการทดลองของการออกแบบการทดลองในบทที่ 4 รวมถึงการวิเคราะห์ผลการทดลองโดยแยกพิจารณาเป็นแต่ละการทดลอง ซึ่งแบ่งเป็นสองส่วนคือผลการทดลองในส่วนแบบจำลองการประมาณต้นทุนเพื่อการจัดวางแผนบุคคลและผลการทดลองในส่วนแบบจำลองการประมาณต้นทุนเพื่อการจัดวางขั้นตอนกระบวนการพัฒนา

5.1 ผลการทดลองของแบบจำลองการประมาณต้นทุนเพื่อการจัดวางแผนบุคคล

5.1.1 การศึกษาการใช้บุคคลในแต่ละช่วงเวลาของการพัฒนาซอฟต์แวร์โดยใช้ผลิตภัณฑ์ซอฟต์แวร์

การทดลองนี้มีจุดประสงค์เพื่อหาผลลัพธ์ของเวลาและต้นทุนที่ใช้ในการพัฒนาจากข้อมูลที่ผู้ใช้ได้ระบุไว้ในตารางที่ 4.1 โดยจากการออกแบบการทดลองในหัวข้อ 4.1.1 ซึ่งใช้คนในการพัฒนาทั้งสิ้น 20 คนและแบ่งคนไปทำงานในรูปแบบ 3,2,5,4,6 หลังจากทำการจำลองจะได้ผลลัพธ์ดังรูปที่ 5.1



รูปที่ 5.1 จำนวนคนที่ใช้ในแต่ละช่วงเวลาของการพัฒนา

จากรูปจะพบว่ามี 8 เหตุการณ์ที่เกิดการเปลี่ยนแปลงจำนวนคนระหว่างทำการพัฒนาโดยสามารถวิเคราะห์เหตุการณ์ทั้ง 8 ได้ดังนี้

เหตุการณ์ที่หนึ่ง โครงการอยู่ในขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์โดยที่งานการปรับแต่งผลิตภัณฑ์ทั้งสองได้เริ่มต้นขึ้นซึ่งใช้คนรวมทั้งหมด 5 คน

เหตุการณ์ที่สอง จะเห็นได้ว่ามีการเพิ่มจำนวนคนในโครงการ ทั้งนี้เนื่องจากงานของการปรับแต่งเสร็จสิ้นไปแล้ว 75 % ดังนั้นงานในส่วนการพัฒนาโปรแกรมประสานทั้งสอง จึงได้เริ่มต้นขึ้นโดยที่งานในขั้นตอนการปรับแต่งยังไม่เสร็จสมบูรณ์ช่วงเวลานี้ใช้คนทั้งหมด 14 คน

เหตุการณ์ที่สาม จะเห็นได้ว่าจำนวนคนที่ใช้ลดจำนวนลงทั้งนี้เกิดจากการที่งานในส่วนปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่งเสร็จสมบูรณ์แล้ว

เหตุการณ์ที่สี่ จะเห็นได้ว่าจำนวนคนที่ใช้ลดลง ทั้งนี้เกิดจากการที่งานในส่วนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สองเสร็จสมบูรณ์แล้ว

เหตุการณ์ที่ห้า จะเห็นได้ว่ามีการใช้จำนวนคนเพิ่มขึ้น ทั้งนี้เนื่องมาจากงานในส่วนของการพัฒนาโปรแกรมประสานเสร็จสมบูรณ์ไปแล้ว 80 % จึงได้เริ่มต้นทำงานในขั้นตอนการรวมและทดสอบระบบซึ่งจำนวนคนที่ใช้ทั้งหมดในช่วงเวลานี้คือ 15 คน

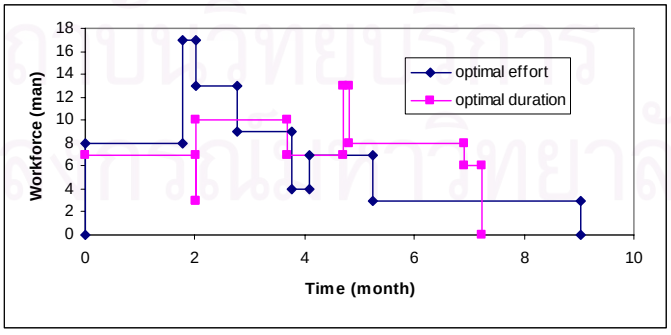
เหตุการณ์ที่หก จะเห็นได้ว่ามีการใช้จำนวนคนลดลง ทั้งนี้เนื่องมาจากการพัฒนาโปรแกรมประสานที่สองได้เสร็จสมบูรณ์ จำนวนคนที่ใช้ในขณะนี้มีทั้งหมด 11 คน

เหตุการณ์ที่เจ็ด จะเห็นได้ว่ามีการใช้จำนวนคนลดลง ทั้งนี้เนื่องมาจากการพัฒนาโปรแกรมประสานที่หนึ่งได้เสร็จสมบูรณ์ จำนวนคนที่ใช้ในขณะนี้มีทั้งหมด 6 คน

เหตุการณ์ที่แปด ระบบได้พัฒนาจนเสร็จสมบูรณ์โดยใช้เวลาทั้งสิ้น 7.49 เดือนและใช้ต้นทุนในการพัฒนาโดยคำนวณจากพื้นที่ใต้กราฟจะได้ค่าความเพียรพยายามคือ 57.90 (Man-month)

5.1.2 การศึกษารูปแบบการจัดบุคคลที่ใช้เวลาและต้นทุนน้อยที่สุด

การทดลองนี้ต้องการทราบรูปแบบการจัดคนเพื่อไปทำงานในขั้นตอนต่างๆ ของกระบวนการพัฒนาโดยที่ใช้เวลาและต้นทุนน้อยที่สุด โดยจากการออกแบบหัวข้อการทดลองที่ 4.1.2 ได้ทำการจำลองแบ่งกลุ่มของพนักงานจำนวน 20 คนออกเป็นห้ากลุ่ม โดยเปรียบเทียบเวลาและต้นทุนที่ใช้ของแต่ละการแบ่งกลุ่ม ซึ่งผลลัพธ์ที่ออกมาได้ดังรูปที่ 5.2



รูปที่ 5.2 เวลาและต้นทุนที่ใช้น้อยที่สุดรวมผลิตภัณฑ์ซอฟต์แวร์เข้าด้วยกัน

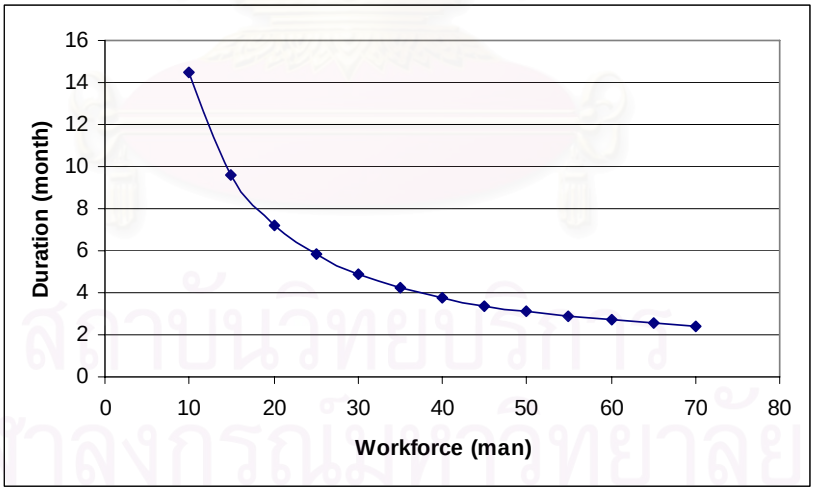
เส้นกราฟแสดงเวลาที่ใช้น้อยที่สุดซึ่งเท่ากับ 7.23 เดือนโดยมีการแบ่งคนทั้งหมดเป็นดังนี้ พนักงาน 3 คนในการปรับแต่งผลิตภัณฑ์ที่หนึ่ง พนักงาน 4 คนในการปรับแต่ง

ผลิตภัณฑ์ที่สอง พนักงาน 5 คนในการพัฒนาโปรแกรมประสานส่วนที่หนึ่ง พนักงาน 2 คนในการพัฒนาโปรแกรมประสานส่วนที่สอง และพนักงาน 6 คนในการรวมและทดสอบระบบ

เส้นกราฟแสดงต้นทุนที่ใช้น้อยที่สุดซึ่งใช้ค่าความเพียรพยายามเท่ากับ 57.67 โดยมีการแบ่งคนทั้งหมดเป็นดังนี้ พนักงาน 4 คนในการปรับแต่งผลิตภัณฑ์ที่หนึ่ง พนักงาน 4 คนในการปรับแต่งผลิตภัณฑ์ที่สอง พนักงาน 4 คนในการพัฒนาโปรแกรมประสานส่วนที่หนึ่ง พนักงาน 5 คนในการพัฒนาโปรแกรมประสานส่วนที่สอง และพนักงาน 3 คนในการรวมและทดสอบระบบ

5.1.3 การศึกษาความสัมพันธ์ระหว่างจำนวนบุคคลที่ใช้กับระยะเวลาของการพัฒนาที่น้อยที่สุด

การทดลองนี้ต้องการทราบเวลาที่ใช้น้อยที่สุดของแต่ละจำนวนคนที่ใช้ในโครงการจากการออกแบบการทดลองที่ 4.1.3 เพื่อหาความสัมพันธ์ระหว่างเวลาที่น้อยที่สุดและจำนวนคนที่ใช้ทั้งหมดโดยทำการแบ่งกลุ่มพนักงานจำนวนตั้งแต่ 10 คนถึง 70 คนซึ่งให้ผลลัพธ์ดังรูปที่ 5.3 โดยพื้นที่ส่วนที่อยู่เหนือกราฟนั้นเป็นส่วนที่มีความเป็นไปได้ในการทำโครงการให้สำเร็จตามเป้าหมายสูง และพื้นที่ที่อยู่ใต้กราฟนั้นเป็นส่วนที่มีความเป็นไปได้ในการทำโครงการให้สำเร็จตามเป้าหมายต่ำ ดังนั้นผู้บริหารโครงการจึงสามารถใช้กราฟนี้ในการวิเคราะห์ให้โครงการมีความเป็นไปได้สูงในการพัฒนาให้สำเร็จโดยสัมพันธ์กับจำนวนคนที่ใช้

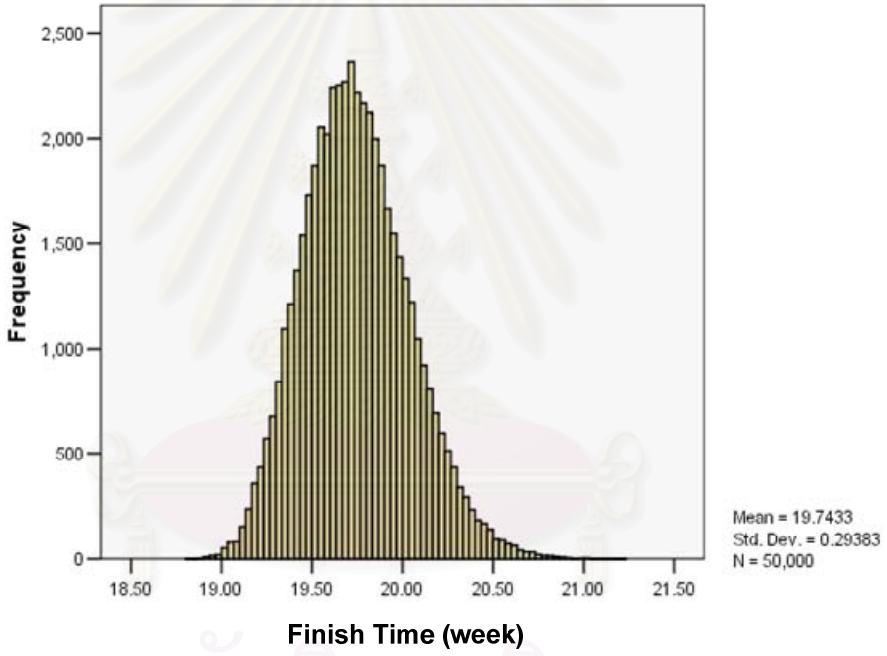


รูปที่ 5.3 ความสัมพันธ์ระหว่างเวลาที่ใช้น้อยที่สุดกับจำนวนคนที่ใช้

5.2 ผลการทดลองของแบบจำลองการประมาณต้นทุนเพื่อการจัดวางขั้นตอนกระบวนการพัฒนา

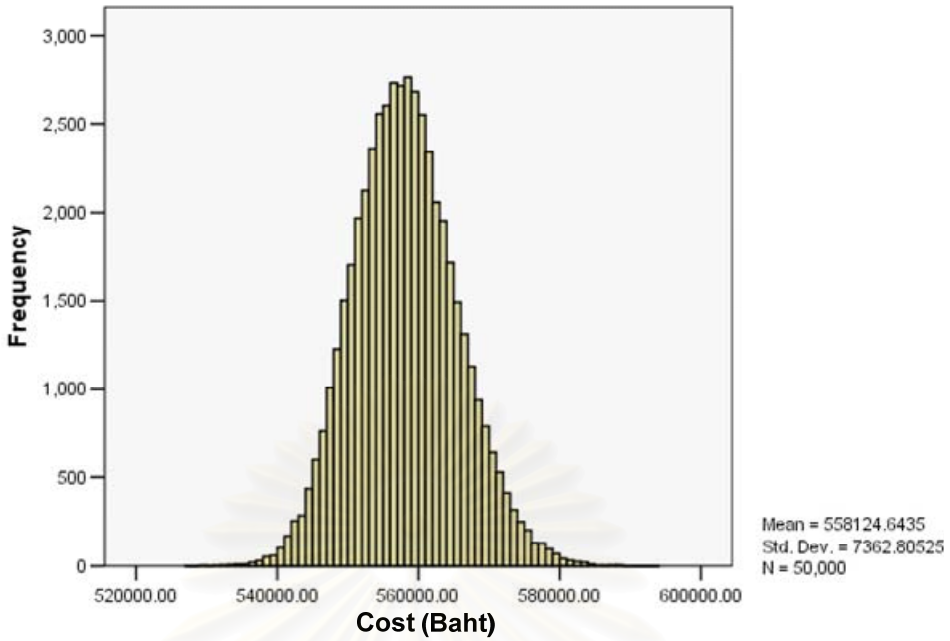
5.2.1 การประมาณระยะเวลาและต้นทุนที่ใช้ในการพัฒนา

การจำลองเพื่อหาเวลาและต้นทุนทั้งหมดที่ต้องใช้ในการพัฒนาตามการออกแบบการทดลองที่ 4.2.1 ซึ่งได้จำลองทั้งสิ้น 50,000 ตัวอย่างได้ผลดังรูปที่ 5.4 และ 5.5 ตามลำดับ โดยพบว่าค่าเฉลี่ยของเวลาที่ใช้ในการพัฒนาคือ 19.74 สัปดาห์ โดยมีส่วนเบี่ยงเบนมาตรฐานคือ 0.29 ซึ่งถ้าไม่รวมส่วนของการทำงานใหม่ที่ได้รับผลกระทบจากขั้นตอนอื่นจะพบว่าใช้เวลาทั้งสิ้น 18.8 สัปดาห์ ซึ่งมีความแตกต่างเพียง 0.94 สัปดาห์คิดเป็นร้อยละ 5 ของระยะเวลาที่ใช้ทั้งหมด



รูปที่ 5.4 กราฟแจกแจงความถี่ของเวลาทั้งหมดที่ใช้ในการพัฒนา

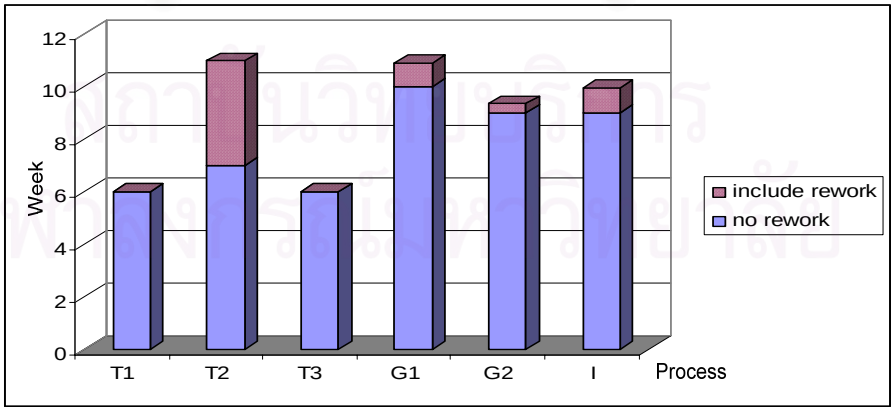
สำหรับต้นทุนการพัฒนาพบว่าค่าเฉลี่ยของต้นทุนที่ใช้ทั้งหมดคือ 558,124.64 บาท โดยมีส่วนเบี่ยงเบนมาตรฐานคือ 7362.80 ซึ่งถ้าไม่รวมต้นทุนในส่วนของการทำงานใหม่ที่ได้รับผลกระทบจากขั้นตอนอื่นจะใช้ต้นทุนคือ 515,000 บาท โดยต้นทุนที่เกิดจากการทำงานใหม่คิดเป็นร้อยละ 8.73 จากต้นทุนที่ใช้ทั้งหมด



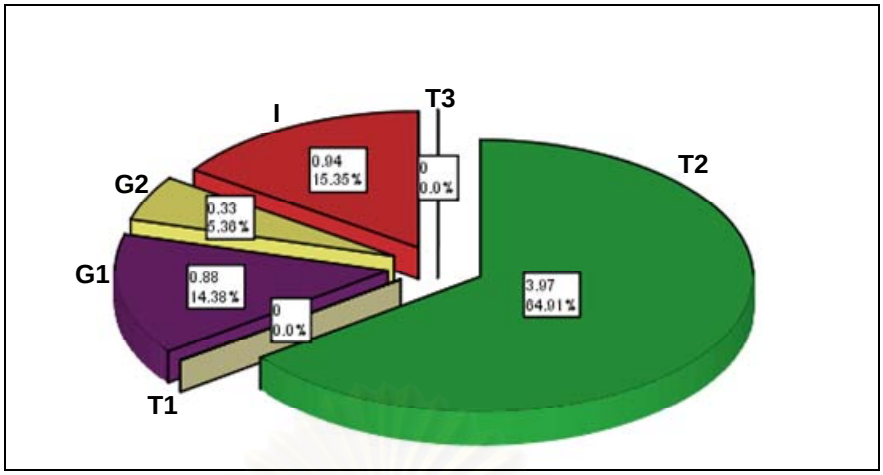
รูปที่ 5.5 กราฟแจกแจงความถี่ของต้นทุนทั้งหมดที่ใช้ในการพัฒนา

5.2.2 การประมาณระยะเวลาและต้นทุนที่ใช้ในการทำงานใหม่เนื่องจากการเปลี่ยนแปลงความต้องการในขั้นตอนอื่น

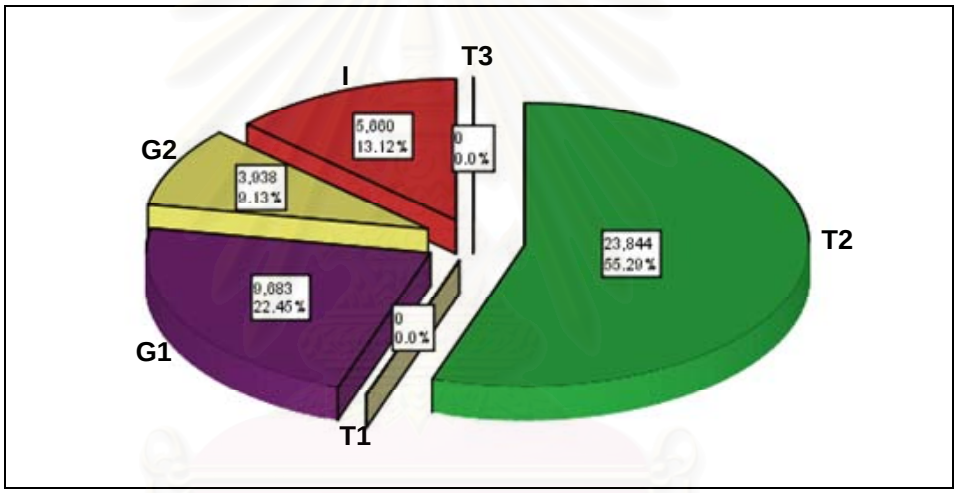
การทดลองนี้ต้องการพิจารณาเวลาและต้นทุนที่ใช้ในส่วนของการทำงานใหม่ที่เกิดขึ้นเนื่องจากผลกระทบที่ได้รับจากขั้นตอนอื่นในแต่ละขั้นตอน เพื่อศึกษาสาเหตุของเวลาและต้นทุนที่เพิ่มขึ้น รูปที่ 5.6 แสดงเวลาที่ใช้ทั้งหมดในแต่ละขั้นตอนถ้ามีการทำงานใหม่เกิดขึ้น และรูปที่ 5.7 กับรูปที่ 5.8 แสดงผลการวิเคราะห์เวลาและต้นทุนที่ใช้ในการทำงานใหม่ของแต่ละขั้นตอน



รูปที่ 5.6 แผนภูมิแท่งแสดงเวลาที่ใช้ในแต่ละขั้นตอนถ้ามีการทำงานใหม่เนื่องจากการเปลี่ยนแปลงในขั้นตอนอื่น



รูปที่ 5.7 แผนภูมิวงกลมแสดงร้อยละของเวลาที่ใช้ในการทำงานใหม่ที่เกิดขึ้นเนื่องจากผลกระทบที่ได้รับจากขั้นตอนอื่นของแต่ละขั้นตอน



รูปที่ 5.8 แผนภูมิวงกลมแสดงร้อยละของต้นทุนการทำงานใหม่ที่เกิดขึ้นเนื่องจากผลกระทบที่ได้รับจากขั้นตอนอื่นของแต่ละขั้นตอน

จะเห็นได้ว่าขั้นตอนในส่วนของการปรับแต่งผลิตภัณฑ์ที่สองเป็นขั้นตอนที่ใช้เวลาและต้นทุนในการทำงานใหม่เนื่องจากได้รับผลกระทบจากการเปลี่ยนขั้นตอนอื่นมากที่สุดโดยคิดเป็นร้อยละ 64.91 ของเวลาทั้งหมดในการทำงานใหม่ที่เกิดขึ้นและร้อยละ 55.29 ของต้นทุนทั้งหมดที่ใช้ในการทำงานใหม่ ซึ่งถ้าพิจารณาจากการจัดเรียงขั้นตอนและความสัมพันธ์ระหว่างงานจะพบว่าขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สองจะได้รับผลกระทบถ้ามีการเปลี่ยนแปลงเกิดขึ้นในส่วนของขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่งและการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สาม

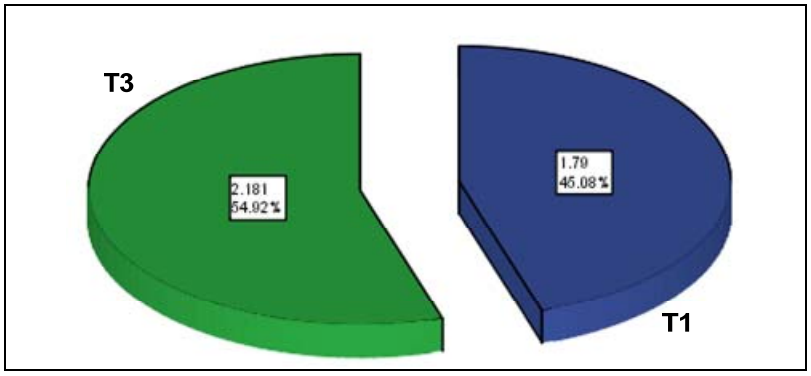
ขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่งและขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สาม ไม่มีการทำงานใหม่เนื่องจากผลกระทบจากการเปลี่ยนแปลงของขั้นตอนอื่นทั้งนี้เนื่องมาจากสองขั้นตอนนี้เป็นขั้นตอนเริ่มต้น

ขั้นตอนในส่วนของพัฒนาโปรแกรมประสานที่หนึ่ง ใช้เวลาและต้นทุนในการทำงานใหม่เนื่องจากได้รับผลกระทบที่เกิดจากขั้นตอนอื่นคิดเป็นร้อยละ 14.38 ของเวลาทั้งหมดในการทำงานใหม่และร้อยละ 22.45 ของต้นทุนทั้งหมดที่ใช้ในการทำงานใหม่ ซึ่งถ้าพิจารณาจากการจัดเรียงขั้นตอนและความสัมพันธ์จะพบว่าขั้นตอนการพัฒนาโปรแกรมประสานที่หนึ่งจะได้รับผลกระทบถ้ามีการเปลี่ยนแปลงเกิดขึ้นในส่วนของการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่งและการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สอง ซึ่งจะเห็นว่าร้อยละของต้นทุนมีค่าสูงกว่าร้อยละของเวลาที่ใช้ในการทำงานใหม่ทั้งนี้เนื่องมาจากต้นทุนที่ใช้ต่อสัปดาห์ของการพัฒนาโปรแกรมประสานมีค่าสูง

ขั้นตอนในส่วนของพัฒนาโปรแกรมประสานที่สอง ใช้เวลาและต้นทุนในการทำงานใหม่เนื่องจากได้รับผลกระทบที่เกิดจากขั้นตอนอื่นคิดเป็นร้อยละ 5.36 ของเวลาทั้งหมดในการทำงานใหม่และร้อยละ 9.13 ของต้นทุนทั้งหมดที่ใช้ในการทำงานใหม่ ซึ่งถ้าพิจารณาจากการจัดเรียงขั้นตอนและความสัมพันธ์จะพบว่าขั้นตอนการพัฒนาโปรแกรมประสานที่สองจะได้รับผลกระทบถ้ามีการเปลี่ยนแปลงเกิดขึ้นในส่วนของการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สามและการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สอง ซึ่งจะเห็นว่าร้อยของต้นทุนมีค่าสูงกว่าร้อยละของเวลาที่ใช้ในการทำงานใหม่ทั้งนี้เนื่องมาจากต้นทุนที่ใช้ต่อสัปดาห์ของการพัฒนาโปรแกรมประสานมีค่าสูง

ขั้นตอนในส่วนของการรวมและทดสอบระบบ ใช้เวลาและต้นทุนในการทำงานใหม่เนื่องจากได้รับผลกระทบที่เกิดจากขั้นตอนอื่นคิดเป็นร้อยละ 15.35 ของเวลาทั้งหมดในการทำงานใหม่และร้อยละ 13.12 ของต้นทุนทั้งหมดที่ใช้ในการทำงานใหม่ ซึ่งถ้าพิจารณาจากการจัดเรียงขั้นตอนและความสัมพันธ์จะพบว่าขั้นตอนการรวมและทดสอบระบบจะได้รับผลกระทบถ้ามีการเปลี่ยนแปลงเกิดขึ้นในส่วนของการพัฒนาโปรแกรมประสานที่หนึ่งและการพัฒนาโปรแกรมประสานที่สอง ทั้งนี้เนื่องจากในกรณีศึกษาการรวมและทดสอบเริ่มต้น หลังจากขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่ง การปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สองและการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สาม ดังนั้นขั้นตอนทั้งสามจึงไม่มีผลต่อการทำงานใหม่ จะเห็นได้ว่าร้อยของต้นทุนมีค่าต่ำกว่าร้อยละของเวลาที่ใช้ในการทำงานใหม่ทั้งนี้เนื่องมาจากต้นทุนที่ใช้ต่อสัปดาห์ของการทดสอบระบบมีค่าต่ำ

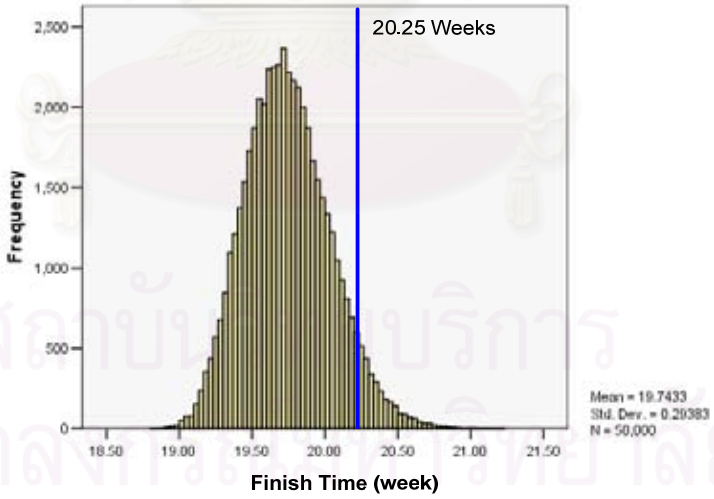
จะเห็นได้ว่าขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สองเกิดการทำงานใหม่เนื่องจากผลกระทบที่เกิดขึ้นจากขั้นตอนอื่นมากที่สุด เมื่อพิจารณาที่มาของผลกระทบพบว่ามาจากขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่งคิดเป็นร้อยละ 54.92 ของเวลาที่ใช้ในการทำงานใหม่และมาจากขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สามคิดเป็นร้อยละ 45.08 ของเวลาที่ใช้ในการทำงานใหม่ ดังรูปที่ 5.9



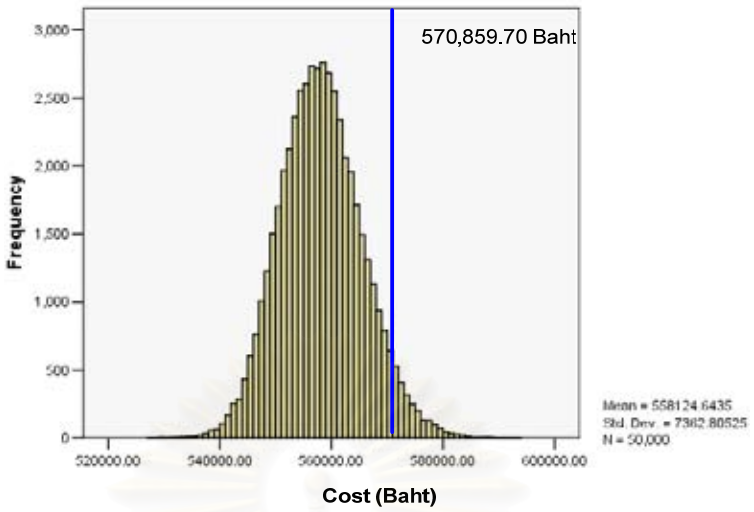
รูปที่ 5.9 แผนภูมิวงกลมแสดงร้อยละของต้นทุนการทำงานใหม่ที่เกิดขึ้นเนื่องจากผลกระทบที่ได้รับจากขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่งและที่สาม

5.2.3 การวิเคราะห์ความเป็นไปได้ของโครงการที่จะสำเร็จตามเป้าหมายที่ตั้งไว้

สำหรับการทดลองนี้ต้องการหาเวลาและต้นทุนที่ใช้ซึ่งมีโอกาสสำเร็จ 95 เปอร์เซ็นต์ โดยหลังจากได้ทำการจำลองได้ผลดังรูปที่ 5.10 และ 5.11 พบว่าเวลาที่ใช้ในการพัฒนาโดยมีโอกาสสำเร็จที่ 95 เปอร์เซ็นต์คือ 20.25 สัปดาห์และต้นทุนที่ใช้ในการพัฒนาโดยมีโอกาสสำเร็จที่ 95 เปอร์เซ็นต์คือ 570,859.70 บาท ซึ่งการวิเคราะห์และผลลัพธ์เหล่านี้สามารถนำไปช่วยในการตัดสินใจวางแผนโครงการ



รูปที่ 5.10 เวลาที่ใช้ในการพัฒนาโดยมีโอกาสสำเร็จ 95 เปอร์เซ็นต์



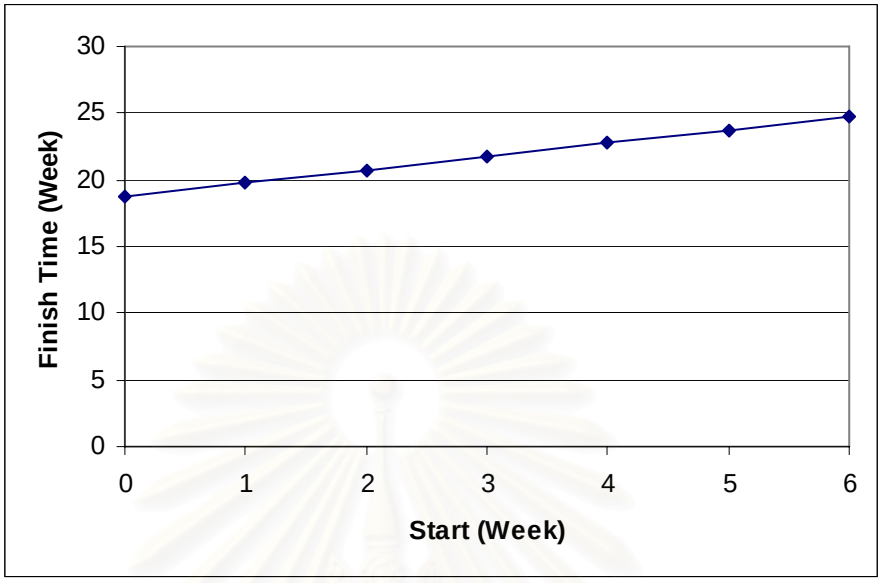
รูปที่ 5.11 ต้นทุนที่ใช้ในการพัฒนาโดยมีโอกาสสำเร็จ 95 เปอร์เซ็นต์

5.2.4 การศึกษาผลกระทบที่เกิดขึ้นเมื่อพิจารณาจากระยะเวลาเริ่มต้นของขั้นตอนที่แตกต่างกัน

การทดลองได้พิจารณาการเริ่มต้นของขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สองเนื่องจากมีผลต่อต้นทุนโดยรวมมากที่สุด ซึ่งการเริ่มต้นทำงานในขั้นตอนการพัฒนาโปรแกรมประสานที่หนึ่ง การพัฒนาโปรแกรมประสานที่สองและขั้นตอนการรวมและทดสอบระบบขึ้นอยู่กับระยะเวลาของขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สองด้วยเช่นกัน ดังนั้นถ้ามีการเปลี่ยนแปลงเวลาการเริ่มต้นในขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สองก็ย่อมมีผลในการเปลี่ยนแปลงเวลาเริ่มต้นของขั้นตอนดังกล่าวด้วย ซึ่งจากการจำลองหาความสัมพันธ์ระหว่างเวลาและต้นทุนทั้งหมดที่ใช้ในการพัฒนากับเวลาในการเริ่มต้นขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สองที่แตกต่างกัน ได้ผลดังรูปที่ 5.12 และ รูปที่ 5.13

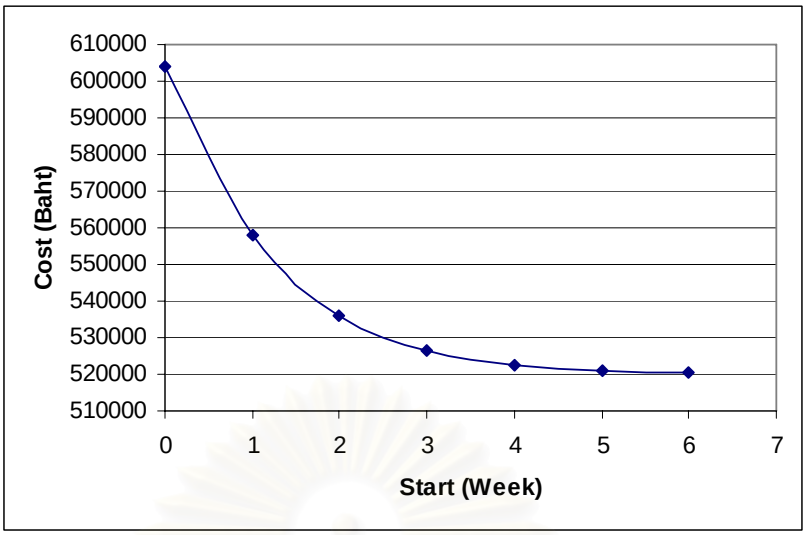
จะเห็นได้ว่าความสัมพันธ์ระหว่างระยะเวลาเริ่มต้นทำงานของขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สองกับเวลาที่ใช้ทั้งหมดมีความสัมพันธ์เป็นแบบเชิงเส้นตรง ทั้งนี้เนื่องมาจากถ้าพิจารณาในเชิงของเวลาทั้งหมดที่ใช้ในการพัฒนานั้น ขั้นตอนที่เสร็จช้าที่สุดจะเป็นตัวกำหนดเวลาที่ใช้ทั้งหมดซึ่งในที่นี้ก็คือขั้นตอนการรวมและทดสอบระบบ แต่เมื่อพิจารณาในขั้นตอนการรวมและทดสอบระบบพบว่าจากการจำลองถึงแม้ขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สองจะเริ่มต้นที่เวลาใดก็ตามขั้นตอนที่ทำให้เกิดการทำงานใหม่ในขั้นตอนการรวมและทดสอบระบบมีเพียงขั้นตอนการพัฒนาโปรแกรมประสานที่หนึ่งและขั้นตอนการพัฒนาโปรแกรมประสานที่สองเท่านั้น ทั้งนี้เป็นเพราะขั้นตอนอื่นนอกจากสองขั้นตอนดังกล่าวจะเสร็จก่อนที่ขั้นตอนการรวมและทดสอบระบบจะเริ่มขึ้น ดังนั้นการเปลี่ยนแปลงระยะเวลาเริ่มต้น

ขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สองจึงไม่มีผลกระทบต่อขั้นตอนการรวมและทดสอบระบบกราฟที่ได้จึงเป็นเส้นตรง



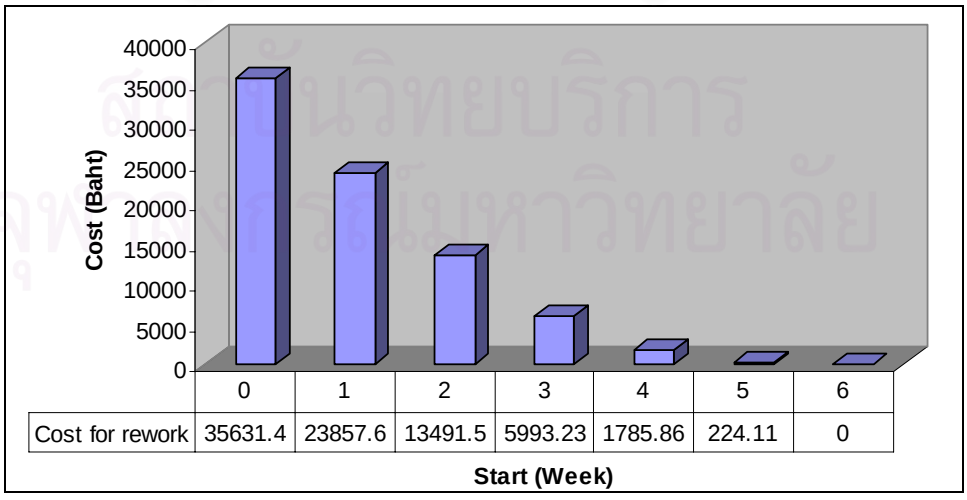
รูปที่ 5.12 ความสัมพันธ์ระหว่างการเริ่มต้นทำการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สองกับเวลาทั้งหมดที่ใช้ในการพัฒนาโครงการ

เมื่อพิจารณาความสัมพันธ์กับต้นทุนที่ใช้จะเห็นได้ว่าระยะเวลาการเริ่มต้นของขั้นตอนการปรับแต่งผลิตภัณฑ์ที่สองจะมีผลอย่างมากต่อต้นทุนในช่วงต้นและลดลงจนเกือบคงที่ในช่วงปลาย ทั้งนี้เป็นผลมาจากในการเริ่มต้นขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สองพร้อมกันกับการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่งและผลิตภัณฑ์ซอฟต์แวร์ที่สาม จะทำให้มีการทำงานใหม่เกิดขึ้นสูงมากเพราะได้รับผลกระทบเนื่องจากการเปลี่ยนแปลงที่ยังไม่แน่นอนบนขั้นตอนดังกล่าว อีกทั้งการเริ่มต้นขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สองเร็วมีผลทำให้มีการเริ่มต้นขั้นตอนการพัฒนาโปรแกรมประสานที่หนึ่งและขั้นตอนการพัฒนาโปรแกรมประสานที่สองเร็วขึ้นตามไปด้วย การที่ขั้นตอนการพัฒนาโปรแกรมประสานทั้งสองเริ่มต้นเร็วขึ้นทำให้ได้ผลกระทบจากการเปลี่ยนแปลงจากขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่งและขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สามด้วยจึงทำให้ต้นทุนโดยรวมสูงขึ้น แต่ในช่วงของการเริ่มต้นที่เดือนหลังๆกับพบว่าต้นทุนมีค่าลดลงและมีแนวโน้มที่จะลดลงเข้าสู่ค่าหนึ่ง ทั้งนี้มาจากการที่ระยะเวลาในการเริ่มต้นช้าทำให้ได้รับผลกระทบจากขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่งและขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สามลดลงและลดลงเรื่อยจนไม่มีผลเมื่อขั้นตอนการปรับแต่งผลิตภัณฑ์ที่สอง เริ่มต้นหลังจากขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่งและที่สามได้เสร็จสิ้นสมบูรณ์แล้ว

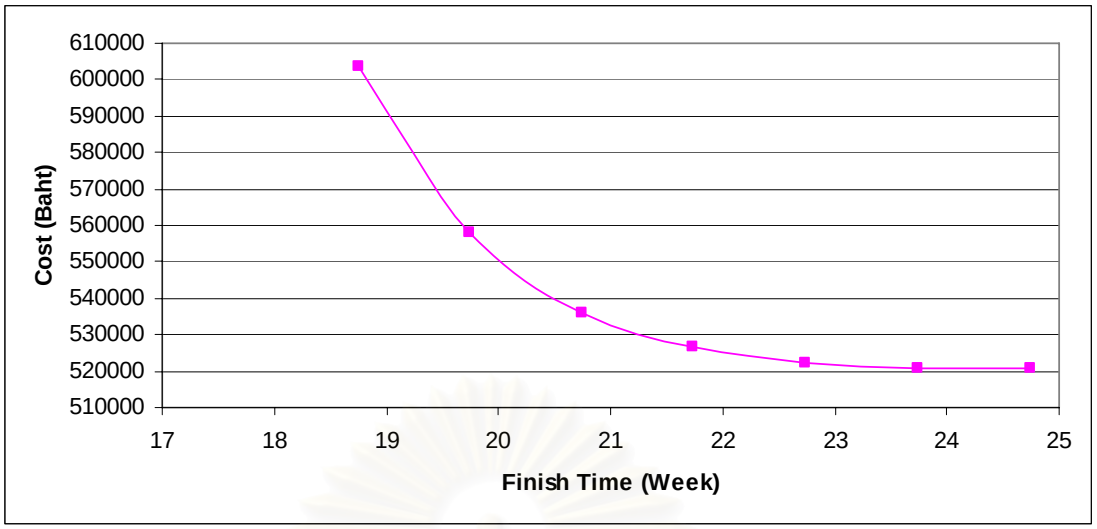


รูปที่ 5.13 ความสัมพันธ์ระหว่างการเริ่มต้นทำการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สองกับต้นทุนทั้งหมดที่ใช้ในการพัฒนาโครงการ

นอกจากนี้เมื่อพิจารณาขั้นตอนที่เกิดการทำงานใหม่มากที่สุดนั้นก็คือขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สองตามรูปที่ 5.14 ซึ่งพบว่าถ้าเริ่มต้นขั้นตอนดังกล่าวพร้อมกับขั้นตอนการปรับแต่งผลิตภัณฑ์ที่หนึ่งและที่สามจะพบว่าเกิดต้นทุนเนื่องจากการทำงานใหม่สูงถึง 35,631.40 บาท หลังจากนั้นจะมีการลดลงอย่างรวดเร็วถ้าชะลอการเริ่มต้นในขั้นตอนดังกล่าวเพื่อหลีกเลี่ยงความไม่แน่นอนของการเปลี่ยนแปลงที่อาจจะส่งผลกระทบต่อต้นทุนที่ใช้ นอกจากนี้จะเห็นว่าในการเริ่มต้นในเดือนที่หกจะไม่มีต้นทุนการทำงานใหม่เลยทั้งนี้เนื่องจากขั้นตอนที่ส่งผลกระทบคือขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่หนึ่งและขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สามเสร็จสิ้นสมบูรณ์แล้วแต่ก็มีข้อเสียคือทำให้ระยะเวลารวมที่ใช้ในการพัฒนาล่าช้า



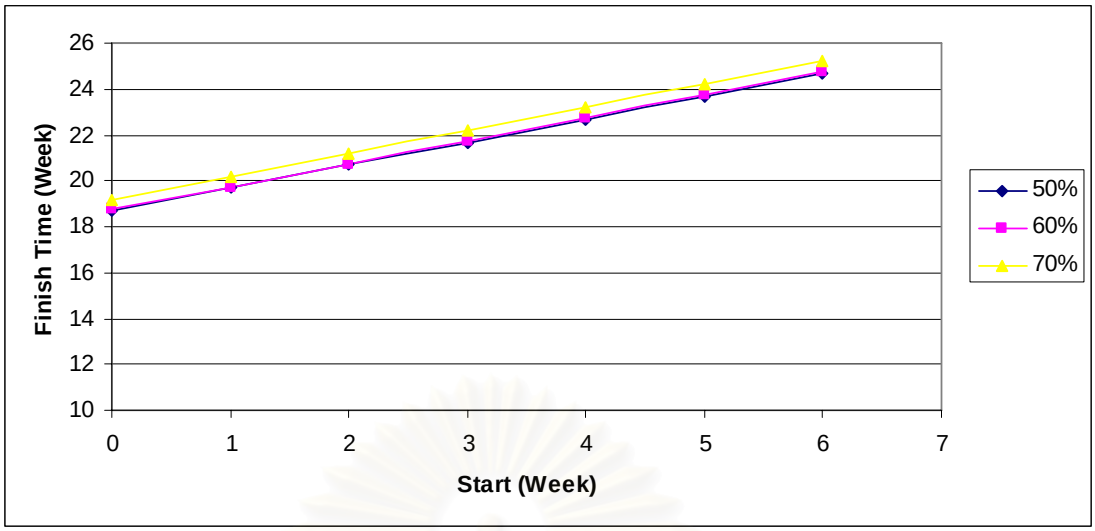
รูปที่ 5.14 ความสัมพันธ์ระหว่างเวลาที่เริ่มต้นกับต้นทุนทั้งหมดที่ใช้สำหรับการทำงานใหม่ในขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สอง



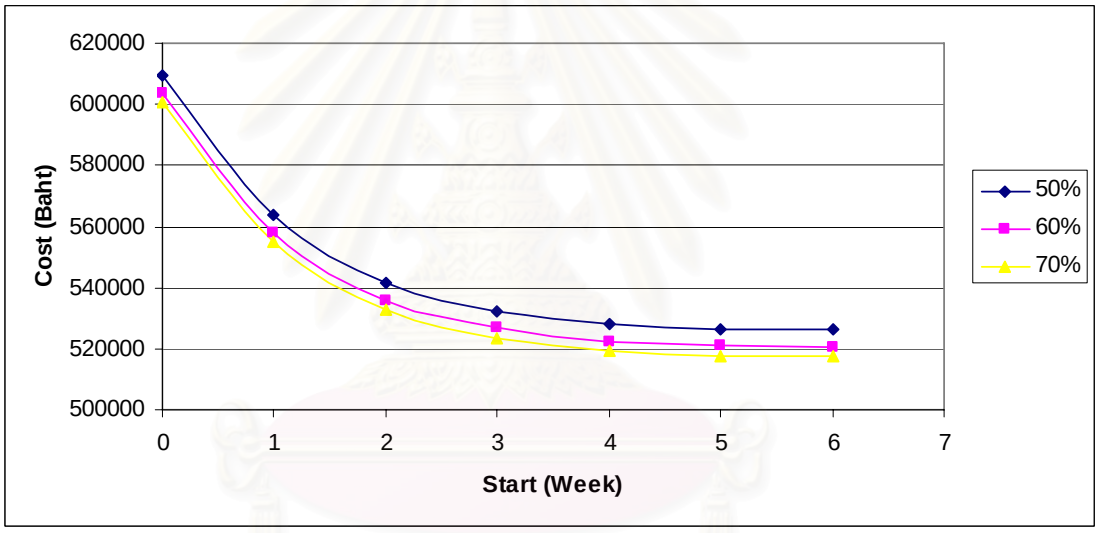
รูปที่ 5.15 ความสัมพันธ์ระหว่างเวลาและต้นทุนที่เปลี่ยนแปลงตามเวลาการเริ่มต้นของขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สอง

เมื่อพิจารณาในแง่ของความสัมพันธ์ของเวลาและต้นทุนทั้งหมดที่ใช้ได้ดังรูปที่ 5.15 จะเห็นได้ว่าในแต่ละจุดนั้นจะแสดงระยะเวลาที่เริ่มต้นของขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สองที่แตกต่างกันโดยเวลาที่ใช้ลดลงต้นทุนที่ใช้ก็จะสูงขึ้น ซึ่งต้นทุนที่ใช้จะลดลงอย่างรวดเร็วในช่วงแรกเมื่อเทียบกับเวลาที่ใช้ ถ้าสามารถเลื่อนเวลาในการพัฒนาให้เสร็จช้าลงได้จะช่วยลดต้นทุนลงได้มาก ในทางกลับกันช่วงท้ายของกราฟต้นทุนที่ใช้จะลดลงเล็กน้อยเมื่อเทียบกับเวลาที่ใช้ การเลื่อนระยะเวลาการพัฒนามีผลกระทบเพียงเล็กน้อยต่อต้นทุนโดยรวม ทั้งนี้ผู้บริหารโครงการสามารถเลือกระยะเวลาที่ใช้ให้เหมาะสมกับต้นทุนโดยการตัดสินใจเวลาเริ่มต้นของขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สอง

นอกจากนี้ถ้าพิจารณาในขั้นตอนที่กำหนดระยะเวลาการเสร็จสมบูรณ์ของโครงการ คือ ขั้นตอนการรวมและทดสอบระบบซึ่งกำหนดให้มีการเริ่มต้นที่ 50% 60% และ 70% ของงานในส่วนของขั้นตอนการพัฒนาโปรแกรมประสานที่หนึ่ง จะได้เส้นกราฟของเวลาและต้นทุนที่ใช้สามเส้นที่แตกต่างกันดังรูปที่ 5.16 และรูปที่ 5.17 ตามลำดับ เมื่อพิจารณาเวลาที่ใช้พบว่าการที่ขั้นตอนการรวมและทดสอบระบบเริ่มต้นที่ 50% และ 60% ของขั้นตอนการพัฒนาโปรแกรมประสานที่หนึ่งนั้นเวลาที่ใช้ทั้งหมดมีค่าใกล้เคียงกัน ทั้งที่การเริ่มต้นที่ 50% น่าจะใช้เวลาน้อยกว่าอย่างชัดเจนเหมือนกับการเริ่มต้นที่ 60% เมื่อเทียบกับ 70% แต่เมื่อกลับไปพิจารณาต้นทุนที่ใช้ในรูปที่ 5.17 พบว่าต้นทุนที่ใช้ของการเริ่มต้นขั้นตอนการรวมและทดสอบระบบที่ 50% กับ 60% ของขั้นตอนการพัฒนาโปรแกรมประสานที่หนึ่งนั้นมีความแตกต่างกันมาก ซึ่งต้นทุนที่แตกต่างกันนี้เป็นผลมาจากการทำงานใหม่นั้นเอง เมื่อต้นทุนของการทำงานใหม่สูงนั้นหมายความว่าเวลาที่ใช้ในการทำงานใหม่ก็สูงเช่นกัน

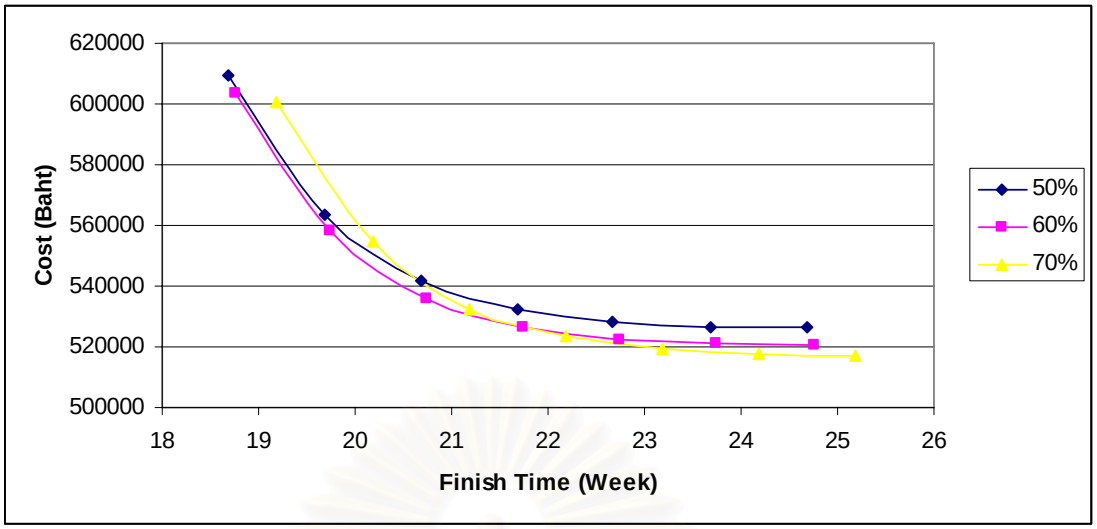


รูปที่ 5.16 ความสัมพันธ์ระหว่างเวลาเริ่มต้นของขั้นตอนการรวมและทดสอบระบบกับเวลาที่ใช้



รูปที่ 5.17 ความสัมพันธ์ระหว่างเวลาเริ่มต้นขั้นตอนการรวมและทดสอบระบบกับต้นทุนที่ใช้

ดังนั้นจึงสามารถสรุปได้ว่าการที่เริ่มต้นขั้นตอนการรวมและทดสอบระบบที่ 50% ทำให้ใช้ระยะเวลาเสร็จใกล้เคียงกับการเริ่มต้นที่ 60% ทั้งนี้เป็นผลมาจากเวลาที่ใช้ในการทำงานใหม่ของการเริ่มต้นที่ 50% มากกว่า 60% ทำให้ผลลัพธ์เวลารวมของขั้นตอนการรวมและทดสอบระบบมีค่าใกล้เคียงกัน ดังนั้นเราจึงไม่ควรเลือกที่จะเริ่มต้นขั้นตอนการรวมและทดสอบระบบที่ 50% เพราะใช้ต้นทุนที่สูงกว่าแต่กลับไม่ทำให้โครงการเสร็จเร็วขึ้น



รูปที่ 5.18 ความสัมพันธ์ระหว่างเวลากับต้นทุนที่ใช้ของระยะเวลาเริ่มต้นขั้นตอนการรวมและทดสอบระบบที่แตกต่างกัน

จากนั้นได้ทำการวิเคราะห์โดยเปรียบเทียบเวลาและต้นทุนที่ใช้เมื่อระยะเวลาเริ่มต้นของขั้นตอนการทดสอบและรวมระบบที่แตกต่างกันดังรูปที่ 5.18 โดยช่วงระยะเวลาการเริ่มต้นของขั้นตอนการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สองในสัปดาห์ที่ 0 ถึงสัปดาห์ที่ 2 (สามจุดแรกในแต่ละกราฟ)พบว่าระยะเวลาของการเริ่มต้นขั้นตอนการทดสอบและรวมระบบที่ 60 % ของขั้นตอนการพัฒนาโปรแกรมประสานที่หนึ่งใช้ต้นทุนน้อยที่สุดที่ระยะเวลาการพัฒนาที่เท่ากัน แต่ในช่วงตั้งแต่การเริ่มต้นของการปรับแต่งผลิตภัณฑ์ซอฟต์แวร์ที่สองในสัปดาห์ที่ 3 ถึงสัปดาห์ที่ 6 (สี่จุดหลังของแต่ละกราฟ)พบว่าระยะเวลาของการเริ่มต้นขั้นตอนการทดสอบและรวมระบบที่ 70 % ของขั้นตอนการพัฒนาโปรแกรมประสานที่หนึ่งใช้ต้นทุนน้อยที่สุดที่ระยะเวลาการพัฒนาที่เท่ากัน ซึ่งจะเห็นได้ว่าการจัดเรียงขั้นตอนที่แตกต่างกันจะใช้เวลาและต้นทุนไม่เท่ากัน โดยสามารถนำผลที่ได้มาช่วยในการตัดสินใจวางแผนการจัดวางขั้นตอนการพัฒนาเพื่อให้เกิดประสิทธิภาพสูงสุด

บทที่ 6

สรุปผลการวิจัย

จากการทดลองโดยใช้แบบจำลองทั้งสองคือแบบจำลองการประมาณต้นทุนเพื่อการจัดวางแผนบุคคลและแบบจำลองการประมาณต้นทุนเพื่อการจัดวางแผนขั้นตอนกระบวนการพัฒนาสามารถสรุปผลการวิจัยและข้อเสนอแนะได้ดังนี้

6.1 สรุปผลการวิจัย

6.1.1 แบบจำลองการประมาณต้นทุนเพื่อการจัดวางแผนบุคคล

1. แบบจำลองนี้สามารถพยากรณ์เวลาและต้นทุนที่ใช้ในการพัฒนาตามค่าที่ผู้ใช้ได้ระบุเอาไว้โดยแบบจำลองได้ใช้หลักการทางสถิติในการประมาณ เช่น ค่าอัตราผลิตภาพ ค่าปัจจัยผลิตภาพซึ่งได้จากแบบจำลองต้นทุน COCOTS และหลักการจำลองซึ่งมีความยืดหยุ่นในการจัดเรียงรูปแบบการพัฒนาทำให้ได้แบบจำลองที่มีความเฉพาะเจาะจงเหมาะสมกับองค์กร

2. แบบจำลองนี้สามารถที่จะหาผลลัพธ์ของเวลาและต้นทุนที่ดีที่สุดของการจัดวางบุคคลในแต่ละขั้นตอนการพัฒนาโดยการวิเคราะห์การแบ่งคนไปทำงานในแต่ละขั้นตอนของการพัฒนาที่ได้ออกแบบไว้จากนั้นหาผลลัพธ์และนำมาเปรียบเทียบเพื่อได้เวลาและต้นทุนที่ดีที่สุด

6.1.2 แบบจำลองการประมาณต้นทุนเพื่อการจัดวางขั้นตอนกระบวนการพัฒนา

1. แบบจำลองนี้สามารถพยากรณ์เวลาและต้นทุนที่ใช้ในการพัฒนา จากการจัดเรียงขั้นตอนที่ผู้ใช้ระบุไว้ ซึ่งได้ใช้หลักการทางวิศวกรรมคู่ขนานในการหาความสัมพันธ์ของแต่ละขั้นตอน โดยเวลาและต้นทุนที่ใช้เพิ่มจะขึ้นอยู่การทำงานใหม่ที่เกิดจากการผลกระทบที่ได้จากการเปลี่ยนแปลงในขั้นตอนอื่น

2. แบบจำลองสามารถวิเคราะห์และแสดงผลลัพธ์ของเวลาและต้นทุนที่ใช้จากวิธีการจัดเรียงขั้นตอนการพัฒนาที่แตกต่างกัน โดยสามารถนำผลลัพธ์ที่ได้มาทำการเปรียบเทียบเพื่อช่วยในการตัดสินใจการเลือกวิธีการจัดเรียงขั้นตอนกระบวนการพัฒนาที่เหมาะสม

6.2 ปัญหาและข้อกำจัดที่ได้พบจากการวิจัย

6.2.1 แบบจำลองการประมาณต้นทุนเพื่อการจัดวางแผนบุคคล

1. ข้อจำกัดด้านข้อมูลที่ใช้ในการทดลอง เนื่องจากข้อมูลที่มีอยู่ไม่เพียงพอและข้อมูลที่เก็บจากการพัฒนาซอฟต์แวร์อาจมีการคลาดเคลื่อนสูงเนื่องจากเป็นการเก็บข้อมูลที่เกี่ยวข้องกับคนไม่ใช่เครื่องจักร

2. แบบจำลองกำหนดให้แต่ละคนทำงานได้เพียงอย่างเดียวต่อเวลา เมื่อทำงานเสร็จก็จะหยุดทำงานทันที ในความเป็นจริงบุคคลเหล่านี้สามารถที่จะไปทำงานอย่างอื่นต่อได้ซึ่งสามารถที่จะนำประเด็นดังกล่าวไปพัฒนาแบบจำลองให้สามารถบริหารยอดรวมของโครงการกล่าวคือถ้าบุคคลใดทำงานเสร็จในโครงการหนึ่งก็สามารถส่งไปทำงานในโครงการอื่นๆ ที่อยู่ในบริษัทได้

3. แบบจำลองดังกล่าวไม่ได้พิจารณาความสัมพันธ์ระหว่างแต่ละขั้นตอนการพัฒนา ถ้ามีการเปลี่ยนแปลงแก้ไขในขั้นตอนหนึ่งขั้นตอนใดในโครงการก็จะไม่มีผลต่อขั้นตอนอื่น ซึ่งประเด็นดังกล่าวสามารถใช้แบบจำลองการประมาณต้นทุนเพื่อการจัดวางขั้นตอนกระบวนการพัฒนาในการช่วยวิเคราะห์และทำการตัดสินใจ

6.2.2 แบบจำลองการประมาณต้นทุนเพื่อการจัดวางขั้นตอนกระบวนการพัฒนา

1. เนื่องจากงานวิจัยนี้ได้ทำการเสนอแบบจำลองสำหรับช่วยตัดสินในการจัดวางขั้นตอนกระบวนการพัฒนา แต่ไม่ได้เสนอวิธีการจัดเรียงขั้นตอนการพัฒนาที่ให้ต้นทุนต่ำที่สุด ซึ่งประเด็นตรงจุดนี้สามารถที่จะนำมาศึกษาหาวิธีการแก้ปัญหาต่อไป

2. แบบจำลองที่มาเสนอนี้เป็นแบบจำลองแบบง่ายที่สามารถนำไปปรับปรุงให้เหมาะสมกับการใช้งานได้ยกตัวอย่างเช่น

2.1.ระยะเวลาของแต่ละขั้นตอนไม่จำเป็นต้องเป็นค่าใดค่าหนึ่ง ผู้ใช้สามารถทำการประยุกต์โดยใช้กราฟการแจกแจงความน่าจะเป็นในกรณีที่เวลาแต่ละขั้นตอนไม่แน่นอน

2.2.การเกิดการเปลี่ยนแปลงความต้องการอาจจะไม่เหมือนกันในแต่ละขั้นตอน โดยที่งานวิจัยนี้ได้ใช้ปัวส์ซองในการจำลองการเกิดการเปลี่ยนแปลง ผู้ใช้สามารถที่จะใช้ข้อมูลที่เก็บได้จริงแทนการใช้ปัวส์ซองจำลองการเปลี่ยนแปลงซึ่งจะใกล้เคียงความเป็นจริงมากขึ้น

2.3.ฟังก์ชันผลกระทบในงานวิจัยนี้ได้กำหนดให้ผลกระทบมีค่าแปรตามงานที่ได้ทำมาแล้ว ซึ่งผู้ใช้สามารถที่จะกำหนดฟังก์ชันนี้ให้เหมาะสมตามลักษณะของแต่ละขั้นตอนได้ด้วยตนเอง

2.4.การประยุกต์ใช้หลักการอื่นๆ เช่น การประยุกต์ใช้กราฟการเรียนรู้ (Learning curve) เนื่องจากเมื่อเวลาผ่านไปทีมงานจะมีประสบการณ์ในการพัฒนามากขึ้น ดังนั้นผลกระทบ

ที่ได้รับจากการเปลี่ยนแปลงความต้องการจะลดลงเพราะทีมงานที่มีประสบการณ์จะสามารถแก้ไขโปรแกรมได้เร็วขึ้น เป็นต้น

รายการอ้างอิง

[1] Yang, Y., J. Bhuta, B Boehm and D. Port. 2005. Value-based processes for COTS-based applications. IEEE Software 22(4): 54-62.

[2] Ye, Y., B. Boehm, and D. Wu. 2006. COCOTS risk analyzer. Proceedings of the Fifth International Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems.

[3] Boehm, B. and C. Abts. 1999. COTS integration: plug and pray? IEEE Computer 32(1): 135-138.

[4] Boehm, B., B. Clark, E. Horowitz, C. Westland, R. Madachy and R. Selby. 1995. Cost Models for Future Software Life Cycle Processes. COCOMO 2.0 Annals of Software Engineering : 57-94.

[5] Abst, C., B. Boehm, and E.B. Clark. 2000. COCOTS: A COTS Software Integration Lifecycle Cost Model-Model Overview and Preliminary Data Collection Findings. Proceeding of the 11th ESCOM Conference.

[6] Brooks, F.P. 1975. The Mythical Man-Month: Essays on Software Engineering. Boston: Addison-Wesley.

[7] Abdel-Hamid, T.K. 1989. The dynamics of software project staffing: a system dynamics based simulation approach. Software Engineering, IEEE Transactions on 15(2): 109-119.

[8] Abdel-Hamid, T.K. and S.E. Madnick. 1991. Software Project Dynamics: An Integrated Approach. Englewood Cliffs: Prentice-Hall.

[9] Gordon, R.L. and J.C. Lamb. 1997. A Close Look at Brooks' Law. Datamation 23 : 81–83.

[10] Madachy, R.J. and B. Boehm. 2005. Software Process Dynamics. IEEE Computer Society Press.

[11] Pei, H., H. Chih-Tung, and D.C. Kung. 1999. Brooks' law revisited: a system dynamics approach. Computer Software and Applications Conference.

[12] Zhang, H., M. Huo, B. Kitchenham, and R. Jeffery. 2006. Qualitative simulation model for software engineering process. Software Engineering Conference.

[13] Kellner, M.I., R.J. Madachy, and D.M. Raffo. 1999. Software Process Modeling and Simulation: Why, What, How. Journal of Systems and Software 46 (No.2/3).

[14] Law, A.M. and W.D. Kelton. 2000. Simulation Modeling and Analysis 3ed. New York: McGraw-Hill.

[15] Browning, T.R. 2001. Applying the design structure matrix to system decomposition and integration problems: a review and new directions. Engineering Management, IEEE Transactions on 48(3): 292-306.

[16] Loch, C.H. and C. Terwiesch. 1998. Communication and Uncertainty in Concurrent Engineering. Managment Science 44(8): 1032-1048.

[17] Krishnan, V., S.D. Eppinger, and D.E. Whitney. 1997. A Model-Based Framework to Overlap Product Development Activities. Managment Science 43(4): 437-451.

[18] Houston, D.X., G.T. Mackulak, and J.S. Collofello. 2001. Stochastic simulation of risk factor potential effects for software development risk management. Journal of Systems and Software 59(3): 247-257.

[19] Nurmuliani, N., D. Zowghi, and S. Powell. 2004. Analysis of requirements volatility during software development life cycle. Software Engineering Conference.

[20] Chulani, S., B. Boehm, and B. Steece. 1999. Bayesian analysis of empirical software engineering cost models. Software Engineering, IEEE Transactions on 25(4): 573-583.

[21] Smith, R.K., J.E. Hale, and A.S. Parrish. 2001. An empirical study using task assignment patterns to improve the accuracy of software effort estimation. Software Engineering, IEEE Transactions on 27(3): 264-271.

[22] Andersson, C., et al. 2002. Understanding software processes through system dynamics simulation: a case study. Proceeding of the 9th IEEE International Conference on Engineering of Computer-Based Systems.

[23] Baik, J., N. Eickelmann, and C. Abts. 2001. Empirical software simulation for COTS glue code development and integration. Proceedings of the 25th International Computer Software and Applications Conference on Invigorating Software Development.

[24] Madachy, R.J. 1996. System dynamics modeling of an inspection-based process. Proceedings of the 18th international conference on Software engineering.

[25] Yakimovich, D., J.M. Bieman, and V.R. Basili. 1999. Software architecture classification for estimating the cost of COTS integration. Proceedings of the 21st international conference on Software engineering.

[26] Smith, R.P. 1997. The historical roots of concurrent engineering fundamentals. Engineering Management, IEEE Transactions on 44(1): 67-78.

[27] Blackburn, J.D., G. Hoedemaker, and L.N. van Wassenhove. 1996. Concurrent software engineering: prospects and pitfalls. Engineering Management, IEEE Transactions on 43(2): 179-188.

[28] Padberg, F. 2001. Scheduling software projects to minimize the development time and cost with a given staff. Proceedings of the Eighth Asia-Pacific on Software Engineering Conference.

[29] Padberg, F. 2002. Using process simulation to compare scheduling strategies for software projects. Proceedings of the Ninth Asia-Pacific Software Engineering Conference.

[30] Padberg, F. 2004. Computing optimal scheduling policies for software projects. Proceedings of the 11th Asia-Pacific Software Engineering Conference.

[31] Iida, H., J. Eijima, S. Yabe, K. Matsumoto and K. Torii. 1996. Simulation model of overlapping development process based on progress of activities. Proceedings of the Third Asia-Pacific Software Engineering Conference.

[32] Browning, T.R. and S.D. Eppinger. 2002. Modeling impacts of process architecture on cost and schedule risk in product development. Engineering Management, IEEE Transactions on 49(4): 428-442.

[33] Minkiewicz, A.F. 2004. Are software COTS solutions an affordable alternative. Aerospace Conference.

[34] Boehm, B. 2000. Safe and simple software cost analysis. IEEE Software 17(5): 14-17.

[35] Ballurio, K., B. Scalzo, and L. Rose. 2002. Risk Reduction in COTS Software Selection with BASIS in COTS-Based Software Systems. Proceedings of the First International Conference on COTS-Based Software Systems.

[36] Xiao, R. and S. Si. 2003. Research on the process model of product development with uncertainty based on activity overlapping. Journal of Manufacturing Technology Management 14(7): 567-574.

ภาคผนวก

ตาราง ค่าความความเพียรพยายามของปัจจัยในแบบจำลอง COCOTS

Rating	Driver													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	ACIEP	ACIPC	AXICP	APCON	ACPMT	ACSEW	APCPX	ACPPS	ACPTD	APVOL	ACREL	AACPX	ACPER	ASPRT
VL	1.34	1.60		1.58	1.45				1.20	0.71				
L	1.16	1.27	1.12	1.26	1.20	1.07	0.82	1.14	1.09	0.84	0.88	0.84		
N	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
H	0.86	0.79	0.89	0.80	0.83	0.94	1.22	0.88	0.91	1.19	1.14	1.19	1.11	1.07
VH	0.75	0.62	0.79	0.63	0.69	0.88	1.48	0.77	0.84	1.33	1.30	1.42	1.22	1.14

Linear Scaling Factor
A
12.0

Nonlinear Scaling Factor				
AAREN				
VL	L	N	H	VH
4.00	3.00	2.00	1.00	0.00

Reference: University of Southern California, Center for Software Engineering, “USC COCOTS Software” (available at http://sunset.usc.edu/research/COCOTS/cocots_main.html), 1997.

ประวัติผู้เขียนวิทยานิพนธ์

นายภูมิ นวลจันทร์เกิดเมื่อวันที่ 18 มกราคม พ.ศ. 2527 ที่นครศรีธรรมราช สำเร็จการศึกษาปริญญาวิศวกรรมศาสตร์บัณฑิต สาขาวิศวกรรมคอมพิวเตอร์ จากภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรม จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2548 และเข้าศึกษาในหลักสูตรวิศวกรรมศาสตร์มหาบัณฑิต สาขาวิศวกรรมคอมพิวเตอร์ ที่ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ปีการศึกษา 2549