การเปรียบเทียบความแม่นยำของวิธีการประมาณขนาดซอฟต์แวร์เชิงวัตถุด้วยวิธีฟังก์ชันพอยต์ ฟังก์ชันพอยต์เชิงวัตถุ และคลาสพอยต์

นางสาวชมพูนุช เผ่าประพัธน์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาการพัฒนาซอฟต์แวร์ด้านธุรกิจ ภาควิชาสถิติ
คณะพาณิชยศาสตร์และการบัญชี จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2555

A COMPARISON ON ACCURACY OF FUNCTION POINT, OBJECT-ORIENTED FUNCTION POINT AND CLASS POINT

Miss Chompoonuch Phoaprapat

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science Program in Business Software Development

Department of Statistics

Faculty of Commerce and Accountancy

Chulalongkorn University

Academic Year 2012

หัวข้อวิทยานิพนธ์	การเปรียบเทียบความแม่นยำของวิธีการประมาณขนาดซอฟต์แวร์เชิงวัตถุด้วยวิธีฟังก์ชันพอยต์ ฟังก์ชันพอยต์เชิงวัตถุ และคลาสพอยต์
โดย	นางสาวชมพูนุช เผ่าประพัธน์
สาขาวิชา	การพัฒนาซอฟต์แวร์ด้านธุรกิจ
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	รองศาสตราจารย์ ดร.อัษฎาพร ทรัพย์สมบูรณ์

คณะพาณิชยศาสตร์และการบัญชี จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

…….. คณบดีคณะพาณิชยศาสตร์และการบัญชี
(รองศาสตราจารย์ ดร.พสุ เดชะรินทร์)

คณะกรรมการสอบวิทยานิพนธ์

…….. ประธานกรรมการ
(ผู้ช่วยศาสตราจารย์ ดร. ถาวร อานุภาพไตรรงค์)

…….. อาจารย์ที่ปรึกษาวิทยานิพธ์หลัก
(รองศาสตราจารย์ ดร. อัษฎาพร ทรัพย์สมบูรณ์)

…….. กรรมการ
(ผู้ช่วยศาสตราจารย์ ดร. สมจารี ปรียานนท์)

…….. กรรมการภายนอกมหาวิทยาลัย
(ผู้ช่วยศาสตราจารย์ ดร. ธันวดี สุเนตนันท์)

ชมพูนุช เผ่าประพัธน์ : การเปรียบเทียบความแม่นยำของวิธีการประมาณขนาดซอฟต์แวร์เชิงวัตถุด้วยวิธีฟังก์ชันพอยต์ ฟังก์ชันพอยต์เชิงวัตถุ และคลาสพอยต์. (A Comparison on Accuracy of Function Point, Object-Oriented Function Point and Class Point Software Size Estimation Models.) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: รศ.ดร.อัษฎาพร ทรัพย์สมบูรณ์, 171 หน้า.

การประมาณการซอฟต์แวร์เป็นกิจกรรมหนึ่งที่จำเป็นในการวางแผนโครงการพัฒนาซอฟต์แวร์ เพื่อใช้บริหารจัดการทรัพยากรบุคคล เครื่องมือ ค่าใช้จ่ายและระยะเวลาในการพัฒนาซอฟต์แวร์ วิธีการประมาณการซอฟต์แวร์ที่เลือกมีผลต่อความแม่นยำต่อการประมาณการซอฟต์แวร์ ผู้บริหารโครงการซอฟต์แวร์ควรเลือกใช้ให้เหมาะสมกับประเภทของโครงการซอฟต์แวร์ การพัฒนาซอฟต์แวร์เชิงวัตถุเป็นกระบวนการพัฒนาซอฟต์แวร์หนึ่งที่เป็นที่นิยม โดยมีวิธีการประมาณการซอฟต์แวร์เชิงวัตถุที่ถูกนำเสนอขึ้น คือ ฟังก์ชันพอยต์ ฟังก์ชันพอยต์เชิงวัตถุ ฟังก์ชันพอยต์เชิงวัตถุสอง และคลาสพอยต์ แต่ยังไม่พบงานวิจัยเพื่อทดสอบเปรียบเทียบความแม่นยำจากวิธีการประมาณการทั้งสี่วิธีการ การศึกษานี้จึงเปรียบเทียบความแม่นยำทั้งสี่วิธีการ

การศึกษานี้เป็นการทดสอบข้อมูลที่ได้จากธุรกิจพัฒนาซอฟต์แวร์จริง ผลการวิเคราะห์ข้อมูลพบว่า การประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์และคลาสพอยต์ให้ผลความแม่นยำมากที่สุด โดยให้ผลความแม่นยำที่ใกล้เคียงกันอย่างมีนัยสำคัญ ส่วนวิธีประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุและฟังก์ชันพอยต์เชิงวัตถุสองไม่มีความแม่นยำอย่างมีนัยสำคัญ

ภาควิชา สถิติ ลายมือชื่อนิสิต .....

สาขาวิชา การพัฒนาซอฟต์แวร์ด้านธุรกิจ ลายมือชื่อ อ.ที่ปรึกษาวิทยานิพนธ์หลัก .....

ปีการศึกษา 2555

5281784226 : MAJOR BUSINESS SOFTWARE DEVELOPMENT
KEYWORDS : SOFTWARE ESTIMATION / SOFTWARE SIZE / OBJECT-ORIENTED SOFTWARE / SOFTWARE PROJECT MANAGEMENT

CHOMPOONUCH PHOAPRAPAT: A COMPARISON ON ACCURACY OF FUNCTION POINT, OBJECT-ORIENTED FUNCTION POINT AND CLASS POINT. ADVISOR: ASSOC. PROF. ASSADAPORN SAPSOMBOON, Ph.D., 171 pp.

Software estimation is one of the most important activities in software project management to manage resources, cost and time for software projects. Software estimation method had affected the accuracy of software size prediction. Project manager should choose appropriate method for characteristic software project. Object-oriented software became a popular development practice. Several techniques aimed at measuring object-oriented software have been proposed indicate Function point, Object-oriented function point, Object-oriented function point 2, and Class point. There are limit researches on comparison of accuracy of these methods. Therefore, this study focused on comparison of accuracy of Function point, Object-oriented function point and Class point.

This research is an experimental research on software industry projects. The analysis indicated that the most accurate software estimation method is Class point and Function point. But Object-oriented Function point and Object-oriented Function point 2 were not accurate statistically.

Department : Statistics Student's Signature .....
Field of Study : Business Software Development Advisor's Signature .....
Academic Year : 2012

กิตติกรรมประกาศ

วิทยานิพนธ์นี้สำเร็จได้ด้วยดี เนื่องจากได้รับแรงสนับสนุนจากรองศาสตราจารย์ ดร. อัษฎาพร ทรัพย์สมบูรณ์ อาจารย์ที่ปรึกษาวิทยานิพนธ์ที่กรุณาสละเวลาอันมีค่า ให้คำปรึกษา ตรวจทานแก้ไขข้อพบพร่อง คอยติดตามและให้กำลังใจ จนวิทยานิพนธ์นี้เสร็จสมบูรณ์ ขอขอบคุณอาจารย์ปิ่นหทัย เดชสิงหรัตน์ ที่ให้คำเสนอแนะ และแบ่งปันประสบการณ์ที่ช่วยส่งเสริมงานวิจัยนี้ ผู้วิจัยขอขอบพระคุณผู้ช่วยศาสตราจารย์ ดร.ถาวร อานุภาพไตรรงค์ ประธานกรรมการวิทยานิพนธ์ และ ผู้ช่วยศาสตราจารย์ ดร. สมจารี ปรียานนท์ กรรมการวิทยานิพนธ์ ที่มอบความรู้ ช่วยเสนอแนะ และแก้ไขวิทยานิพนธ์นี้จนเสร็จสมบูรณ์ และกรรมการภายนอกมหาวิทยาลัย ผู้ช่วยศาสตร์จารย์ ดร.ธันวดี สุเนตนันท์ ที่กรุณาเสียสละเวลาอันมีค่า ช่วยชี้แนะสิ่งต่าง ๆ ให้งานวิจัยลุล่วงไปได้ด้วยดี

ที่สำคัญยิ่งขอขอบพระคุณคุณพ่อและคุณแม่ ที่มอบทั้งกำลังใจ สนับสนุน และมอบทุนทรัพย์ในการเล่าเรียนตลอดมา ขอขอบคุณในความกรุณาของเพื่อน ๆ ที่ช่วยติดต่อให้ข้อมูลที่ใช้เป็นหน่วยตัวอย่าง สุดท้ายนี้ขอขอบเพื่อน ๆ ที่คอยเป็นแรงใจ อีกทั้งช่วยเหลือ และเป็นแรงสนับสนุนจนงานวิจัยนี้สำเร็จลุล่วงลงได้

สารบัญ

หน้า

หน้า

หน้า

หน้า

สารบัญตาราง

หน้า

หน้า

หน้า

หน้า

หน้า

สารบัญรูป

หน้า

หน้า

หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

การประมาณขนาดของซอฟต์แวร์ (Software size estimation) เป็นกิจกรรมที่สำคัญสำหรับการบริหารโครงการซอฟต์แวร์ เพื่อประเมินหาขนาดของซอฟต์แวร์ในการวางแผนโครงการซอฟต์แวร์ เพราะขนาดของซอฟต์แวร์ (Software size) เป็นตัวแปรที่ใช้ในตัวแบบของการคำนวณ (Model) ค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์ (Development effort) ต้นทุนของซอฟต์แวร์ (Cost) รวมไปถึงระยะเวลาที่ใช้ในการพัฒนาซอฟต์แวร์ นั่นหมายถึงขนาดของซอฟต์แวร์ที่ประมาณได้จะสามารถใช้ในการทำนายค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์และต้นทุนของซอฟต์แวร์ ได้อีกด้วย

ปัญหาการประเมินระยะเวลา ต้นทุน ค่าความพยายาม พบว่า เป็นปัญหาของโครงการซอฟต์แวร์ล้มเหลว ตั้งแต่ปี ค.ศ. 1994 เดอะ สแตนดิส กรุ๊ป พบว่า 53% ของโครงการซอฟต์แวร์ต้องท้าทายกับต้นทุน (Cost) และเวลาที่เพิ่มขึ้นถึงหนึ่งจุดแปดเท่าจากการประเมิน (The Standish Group, 1994) ปี ค.ศ. 1999 ลินเบิร์ก พบว่า 20% ของโครงการซอฟต์แวร์ต้องล้มเหลว และ 46% มาจากต้นทุนและตารางเวลาที่เกินกำหนด (cost & schedule overrun) (Linberg , 1999) ในปี ค.ศ. 2007 รายงานของเดอะ สแตนดิส กรุ๊ป ยังคงพบปัญหาต้นทุนและระยะเวลาในการพัฒนาซอฟต์แวร์ที่เกินกำหนด โดยพบว่า 46% ของโครงการที่ยังต้องท้าทายกับต้นทุนที่เพิ่มมากขึ้น และ 19% ที่โครงการต้องถูกยกเลิก (Rubenstein, 2007)

ผู้วิจัยได้สำรวจจาก ผู้พัฒนาซอฟต์แวร์ในประเทศไทยมีประสบการณ์มากกว่า 1 ปีและมีความรู้เกี่ยวกับการประมาณขนาดซอฟต์แวร์ จำนวน 41 หน่วยตัวอย่าง 51.22% ทำงานอยู่ในองค์กรขนาดใหญ่(มากกว่า 200 คน) 26.83% ทำงานอยู่ในองค์กรขนาดกลาง(50-199 คน) และ 21.95% ทำงานอยู่ในองค์กรขนาดเล็ก(น้อยกว่า 50 คน) ผลการสำรวจ พบว่า 92.7% มองว่าการประมาณขนาดซอฟต์แวร์เป็นกิจกรรมที่มีความสำคัญต่อโครงการซอฟต์แวร์ ผลสำรวจยังพบว่า 90.2% ของหน่วยตัวอย่างมีการประมาณขนาดซอฟต์แวร์ และมักเกิดปัญหาประมาณระยะเวลาการพัฒนาซอฟต์แวร์ภายในองค์กรที่คลาดเคลื่อนเป็นประจำ 37.8% เป็นปกติ 24.3% เป็นบางครั้ง 35.1% และไม่เคยเกิดขึ้น 2.7%

การประมาณขนาดของซอฟต์แวร์จึงเป็นงานสำคัญที่จำเป็นต่อการวางแผนโครงการ ในสมัยก่อน ขนาดของซอฟต์แวร์จะถูกวัดด้วยจำนวนบรรทัดของโปรแกรม (Source line of code:

SLOC) แต่การวัดจากจำนวนบรรทัดของโปรแกรมไม่สามารถใช้วัดได้ในภาษาโปรแกรมบางประเภท (Programming language) เพราะจำนวนบรรทัดของโปรแกรมอาจแตกต่างกันซึ่งขึ้นอยู่กับผู้พัฒนา อีกทั้งการหาขนาดจากจำนวนบรรทัดของโปรแกรมจะสามารถหาได้ต่อเมื่อซอฟต์แวร์ถูกพัฒนาเสร็จสมบูรณ์ จึงไม่สามารถใช้ในการประมาณค่าความพยายามที่ใช้ในการพัฒนาในช่วงแรกของโครงการซอฟต์แวร์จะพัฒนาจริง

ในปี ค.ศ. 1979 Albrecht ได้นำเสนอ วิธีการประมาณขนาดซอฟต์แวร์ด้วยการวิธีฟังก์ชันพอยต์ (Function point Analysis: FPA) ซึ่งเป็นวิธีการคำนวณหาขนาดของซอฟต์แวร์จากฟังก์ชันของระบบที่จะพัฒนาในมุมมองของผู้ใช้งาน ในช่วงตั้งแต่ขั้นตอนการวิเคราะห์และออกแบบ จนไปถึงขั้นตอนของการพัฒนาซอฟต์แวร์เสร็จสมบูรณ์ จึงเป็นสาเหตุหนึ่งที่ทำให้วิธีการวิธีฟังก์ชันพอยต์ประสบความสำเร็จ (Albrecht, 1979) วิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์ยังเป็นวิธีการประมาณขนาดซอฟต์แวร์หนึ่งที่ได้รับการยอมรับว่า สามารถใช้ได้กับซอฟต์แวร์ทุกชนิดโดยไม่ขึ้นกับเทคโนโลยีหรือภาษาโปรแกรมใดๆ (Fetcke et al., 1997) เพราะฟังก์ชันพอยต์เป็นวิธีการวัดจำนวนฟังก์ชันที่มีอยู่ในซอฟต์แวร์โดยใช้ข้อกำหนดจากความต้องการของซอฟต์แวร์ (Software requirement specification) ซึ่งเป็นข้อมูลพื้นฐานในการวิเคราะห์คำนวณขนาดซอฟต์แวร์ อีกทั้งยังสามารถนำไปต่อยอดกับวิธีการวัดซอฟต์แวร์อื่นๆได้

ผลสำรวจวิธีการประมาณขนาดซอฟต์แวร์ในโครงการซอฟต์แวร์ของหน่วยตัวอย่างจำนวน 41 หน่วยตัวอย่าง มี 21 หน่วยตัวอย่างที่ทราบวิธีการประมาณขนาดซอฟต์แวร์ที่ถูกใช้จริงในโครงการพัฒนาซอฟต์แวร์ของตน พบว่า วิธีที่ถูกนำมาใช้อันดับหนึ่ง คือ การประมาณขนาดด้วยผู้เชี่ยวชาญ (Expert exjustment) จำนวน 95.2% ซึ่งเลือกใช้ควบคู่ไปกับวิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์ 14.29%

กระบวนการพัฒนาซอฟต์แวร์เชิงวัตถุ (Object-oriented software development methodology: OOSDM) ถูกนำเสนอขึ้นครั้งแรก ในปี ค.ศ. 1986 กระบวนการพัฒนาซอฟต์แวร์เชิงวัตถุเป็นกระบวนการที่เหมาะสมสำหรับซอฟต์แวร์ที่ซับซ้อน เพราะรองรับกระบวนการที่วิเคราะห์ ออกแบบ รวมไปถึงการพัฒนาระบบ ที่มีความยืดหยุ่น (Flexible) และสอดคล้องกับโลกแห่งความจริง (Robust real-world) โดยมีลักษณะพิเศษคือ การสืบทอด (Inheritance) การห่อหุ้ม (Encapsulation) การพ้องรูป (Polymorphism) และการนำกลับมาใช้ใหม่ (Reusability) โดยการถ่ายทอดลงแบบจำลองเชิงวัตถุ (object-oriented paradigm) (Ramsin & Paige, 2008; Agarwal & Sinha, 2003) โดยมีมาตรฐานในการออกแบบ คือ ยูเอ็มแอล (Unified Modeling Language: UML) ซึ่งได้รับการดูแลและควบคุมโดยอ็อบเจกต์ แมนเนจเมนต์ กรุ๊ป (Object

Management Group: OMG) สำหรับการออกแบบที่นิยมใช้กันคือ แผนภาพยูสเคส(Use case diagram) แผนภาพคลาส (Class diagram), แผนภาพสเตต (State diagram) และแผนภาพซีเควนซ์ (Sequence diagram) (Agarwal & Sinha, 2003)

ในปัจจุบัน กระบวนการพัฒนาซอฟต์แวร์เชิงวัตถุนั้นได้รับความสนใจมากขึ้น เนื่องจากซอฟต์แวร์ในธุรกิจมีความซับซ้อนมากยิ่งขึ้นเรื่อยๆ สำหรับวิธีการประมาณขนาดซอฟต์แวร์เชิงวัตถุนั้นมีหลายวิธีที่ไม่สามารถทำได้ เนื่องจากกระบวนการและเอกสารการออกแบบระบบของซอฟต์แวร์เชิงวัตถุนั้นแตกต่างกันออกไป โดยพบว่า วิธีการประมาณขนาดซอฟต์แวร์ด้วย COSMIC-FFP (Abran et al., 2001) ซึ่งได้รับการยอมรับนั้น ไม่สามารถใช้ประมาณขนาดสำหรับซอฟต์แวร์เชิงวัตถุได้ (Poels, 2003)

สำหรับวิธีการประมาณขนาดสำหรับซอฟต์แวร์เชิงวัตถุนั้นมีหลากหลายวิธีที่ถูกนำเสนอขึ้น วิธีการประมาณขนาดด้วยมาตรวัด (Object-oriented metric) เป็นการประมาณขนาดในระดับคลาสด้วยจำนวนของลักษณะประจำ (attribute) เมท็อดภายนอก (external method) และมาตรวัดของซีเค (CK) อีก 3 ตัว คือ ดับเบิ้ลยูเอ็มซี (WMC: Weight method per class) ดีไอที (DIT: Average depth of class in hierarchy tree) เอ็นโอซี (NOC: Average number of children per class) แต่วิธีการคำนวณดังกล่าวสามารถวัดขนาดได้ที่ระดับของคลาสเท่านั้น (Henderson-Sellers, 1996) ซึ่งยากที่จะนำมาประมาณค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์จริงในระบบใหญ่ๆ หรือโครงการใหญ่ ซึ่งมาตรวัดดังกล่าวถูกนำมาเพื่อใช้เป็นตัวแปรในการคำนวณด้วยวิธีการอื่นๆ

ในปี ค.ศ. 1995 วิธีการประมาณขนาดซอฟต์แวร์ด้วยอ็อบเจกต์พอยต์ (Object point) เกิดขึ้น ซึ่งเป็นการประมาณขนาดด้วยวิธีที่คล้ายกับฟังก์ชันพอยต์ (Boehm, et al., 1995) แต่วิธีการดังกล่าวไม่เหมาะสำหรับซอฟต์แวร์เชิงวัตถุ เนื่องจากเป็นการนับที่เอกสารของโครงการซอฟต์แวร์ทั่วไป คือ เอกสารการออกแบบหน้าจอ อีกทั้งวิธีการดังกล่าวยังให้ผลลัพธ์ที่ใกล้เคียงกับฟังก์ชันพอยต์ ซึ่งในปี ค.ศ. 1997 วิธีการใหม่ชื่อการทำนายอ็อบเจกซ์ (Predictive object point) เพื่อให้สามารถประมาณซอฟต์แวร์เชิงวัตถุได้ โดยอ้างอิงจากมาตรวัดเชิงวัตถุของซีเค (CK) จำนวน 3 ตัว คือ ดับเบิ้ลยูเอ็มซี (WMC: Weight method per class) ดีไอที (DIT: Average depth of class in hierarchy tree) เอ็นโอซี (NOC: Average number of children per class) รวมกับ ทีแอลซี (TLC: Number of top level class) (Minkiewicz, 1997) แต่วิธีการดังกล่าวต้องการรายละเอียดระดับลึกของการออกแบบ ซึ่งสามารถหาได้ในช่วงการออกแบบ

เสร็จสิ้นแล้วเท่านั้น จึงไม่เหมาะสมสำหรับการนำมาใช้จริงเพื่อประมาณซอฟต์แวร์ในช่วงก่อนการพัฒนาจริง

วิธีการประมาณขนาดด้วยยูสเคสพอยต์ (Use case point) (Karner, 1993) เป็นการคำนวณขนาดจากแผนภาพยูสเคส (Use case diagram) โดยใช้วิธีอ้างอิงจากฟังก์ชันพอยต์ (Function point) โดยจะดูที่ยูสเคส (Use case) แอกเตอร์ (actor) และรายละเอียดของยูสเคส (Use case description) ซึ่งเป็นวิธีที่สามารถทำได้ตั้งแต่ต้นๆ แต่วิธีการดังกล่าวต้องอ้างอิงจากผู้เชี่ยวชาญในการสร้างยูสเคสที่ละเอียดเพียงพอ ซึ่งค่าที่ประมาณได้จะขึ้นอยู่กับผู้เชี่ยวชาญในการลงรายละเอียด และพบว่าให้ค่าความแม่นยำน้อยกว่าวิธีการประมาณขนาดในช่วงการออกแบบระบบ (Zivkovic et al., 2005)

การประมาณขนาดซอฟต์แวร์เชิงวัตถุถูกนำเสนออย่างต่อเนื่อง ในปี ค.ศ. 2004 Abrahao และคณะ นำเสนอวิธีการประมาณขนาดซอฟต์แวร์ฟังก์ชันพอยต์เมท็อดเชิงวัตถุ (Object-oriented method function point: OOmFP) เป็นการวัดด้วยแบบจำลองเชิงวัตถุ 4 ตัว คือ อ็อบเจกต์โมเดล (Object model) ไดนามิกโมเดล (Dynamic model) ฟังก์ชันนอลโมเดล (Functional model) และพรีเซนเทชันโมเดล (Presentation model) ซึ่งพบว่าวิธีการดังกล่าวให้ผลลัพธ์ที่ดีกว่าฟังก์ชันพอยต์ (Abrahao et al., 2004) แต่วิธีการดังกล่าวไม่เหมาะสำหรับการประมาณซอฟต์แวร์ในธุรกิจจริงเพราะต้องใช้เอกสารจำนวนมากในการคำนวณ

วิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุ (Object-Oriented Function Point: OOFP) (Antoniol et al., 1998) เป็นวิธีการวัดขนาดของระบบจากการออกแบบแผนภาพคลาส (Class diagram) ด้วยวิธีการต่อยอดจากการประมาณขนาดฟังก์ชันพอยต์ ซึ่งสามารถวัดได้ในช่วงที่ไม่ต้องอาศัยการออกแบบที่ละเอียดมาก และสามารถปรับปรุงได้เมื่อมีรายละเอียดมากขึ้น การประมาณขนาดซอฟต์แวร์เชิงวัตถุจะคำนวณจากคลาสและเมท็อด และในปี ค.ศ. 2005 ได้มีนักวิจัยนำวิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุมาปรับปรุงต่อยอดเพื่อวัดซอฟต์แวร์เชิงวัตถุ (OOFP2) โดยการเปลี่ยนการคิดค่าความซับซ้อนของเมท็อดให้ลดน้อยลงเนื่องจากซอฟต์แวร์เชิงวัตถุ เนื่องจากซอฟต์แวร์เชิงวัตถุมีการส่งค่าพารามิเตอร์ระหว่างคลาสมากกว่าการส่งค่าภายในการออกแบบซอฟต์แวร์ทั่วไป (Zivkovic et al., 2005) วิธีการประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุจึงเป็นหนึ่งวิธีการประมาณขนาดที่น่าสนใจและพบว่าให้ผลลัพธ์ที่ดีกว่าการประมาณขนาดด้วยวิธีการยูสเคสพอยต์

ในปี ค.ศ. 2005 คอสตากลอล่า และคณะ (Costagliola et al., 2005) ได้นำเสนอวิธีการประมาณขนาดซอฟต์แวร์เชิงวัตถุ คลาสพอยต์ โดยการคำนวณหาขนาดของซอฟต์แวร์จาก

แผนภาพคลาส (Class diagram) อ้างอิงจากมาตรวัดสามตัว คือ จำนวนลักษณะประจำ (Attribute) จำนวนเมท็อดสาธารณะ (Number of External Method: NEM) และจำนวนการเรียกใช้บริการจากภายนอก (Number of Services Requested: NSR) และพบว่าให้ผลดีกว่าการใช้มาตรวัดเชิงวัตถุ อีกทั้งคอสตากลอล่า กล่าวว่า วิธีการนี้สามารถใช้ประมาณขนาดของซอฟต์แวร์เชิงวัตถุได้ดีกว่าวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์ แต่ในงานวิจัยดังกล่าวยังไม่มีการทดสอบใดๆ

ในปี ค.ศ. 2010 วิธีการประมาณขนาดซอฟต์แวร์ด้วยแพทเทิร์นพอยต์ (Pattern point) เป็นอีกหนึ่งวิธีที่ถูกนำเสนอขึ้นเพื่อใช้วัดขนาดซอฟต์แวร์เชิงวัตถุ ด้วยการอ้างอิงกระบวนการคิดคำนวณจากคลาสพอยต์ เพื่อประมาณขนาดของซอฟต์แวร์ตามดีไซน์แพทเทิร์น (Design pattern) (Adekile et al., 2010) แต่วิธีการดังกล่าวอาจจะไม่เหมาะสมสำหรับซอฟต์แวร์ในธุรกิจ เนื่องจาก การออกแบบด้วยแพทเทิร์นจะต้องใช้ผู้เชี่ยวชาญในการออกแบบ ซึ่งปัจจุบันผู้ออกแบบที่มีประสบการณ์ความเชี่ยวชาญเท่านั้นที่สามารถออกแบบระบบได้ตามแพทเทิร์น

การประมาณขนาดซอฟต์แวร์ที่ให้ความแม่นยำที่ดีที่สุด คือ การประมาณขนาดในช่วงการออกแบบระบบ ซึ่งจะมีวิธีการที่เหมาะสมและสามารถนำมาใช้จริงได้ คือ ฟังก์ชันพอยต์ ฟังก์ชันพอยต์เชิงวัตถุ ฟังก์ชันพอยต์เชิงวัตถุสอง และคลาสพอยต์ โดยพบ งานวิจัยทดสอบความแม่นยำของการประมาณขนาดซอฟต์แวร์เชิงวัตถุด้วยวิธีการดังกล่าว เปรียบเทียบความแม่นยำด้วยจำนวนบรรทัดของโปรแกรม พบว่า วิธีการประมาณขนาดด้วยฟังก์ชันพอยต์ให้ค่าคลาดเคลื่อนเกินจริงประมาณ 16% วิธีการประมาณขนาดฟังก์ชันพอยต์เชิงวัตถุให้ค่าต่ำกว่าขนาดจริง 13% และวิธีการประมาณขนาดของคลาสพอยต์ต่ำกว่าขนาดจริง 15% ซึ่งนักวิจัยยอมรับว่า ขนาดซอฟต์แวร์ควรเปรียบเทียบกับค่าความพยายามที่ในการพัฒนาซอฟต์แวร์จริง แต่ในงานวิจัยดังกล่าวไม่สามารถทำได้ เนื่องจากใช้โครงงานของนักศึกษา จึงไม่สามารถเก็บข้อมูลในส่วนนี้ได้ (Bianco & Lavazza, 2005)

ความถูกต้องของวิธีการประมาณขนาดซอฟต์แวร์จะสามารถวัดได้จากค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์ เพราะการใช้จำนวนบรรทัดของโปรแกรมนั้นขึ้นอยู่กับความสามารถและเทคนิคของโปรแกรมเมอร์ ซึ่งไม่สามารถวัดได้โดยตรงกับทุกภาษา โดยเฉพาะการพัฒนาซอฟต์แวร์เชิงวัตถุ

ดังนั้นเพื่อเป็นการทดสอบความถูกต้องของวิธีการประมาณขนาดซอฟต์แวร์เชิงวัตถุ ในงานวิจัยนี้ ผู้วิจัยจึงสนใจที่จะทดสอบประสิทธิภาพของการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์เชิงวัตถุ (Development effort) โดยเลือกการประมาณขนาดซอฟต์แวร์ในช่วง

การออกแบบระบบ เนื่องจากระดับการออกแบบระบบถือว่าวัดได้ง่ายและจะเห็นมุมมองที่ชัดเจนกว่าเมื่อต้องการประมาณค่าความพยายามที่จะใช้ในการพัฒนาซอฟต์แวร์ (Moser et al, 1997) อีกทั้งยังพบอีกว่า ความถูกต้องของการประมาณจะเพิ่มขึ้นในช่วงการออกแบบ เนื่องจากข้อมูลมีมากยิ่งขึ้น และพบว่า วิธีการวัดจากแผนภาพคลาส (Class diagram) ดีกว่าการวัดที่แผนภาพยูสเคส และแผนภาพซีเควนซ์ (Zivkovic et al., 2005)

ดังนั้นในงานวิจัยนี้ ผู้วิจัยจึงเลือกเปรียบเทียบวิธีการประมาณขนาดซอฟต์แวร์เชิงวัตถุด้วย ฟังก์ชันพอยต์เชิงวัตถุ (Antoniol et al., 1998) ฟังก์ชันพอยต์เชิงวัตถุสอง (Zivkovic et al., 2005) คลาสพอยต์ (Costagliola et al., 2005) และนำมาเปรียบเทียบกับวิธีการที่ยอมรับและมีใช้อยู่จริง คือ ฟังก์ชันพอยต์ (IFPUG, 1999) ด้วยข้อมูลในธุรกิจการพัฒนาซอฟต์แวร์จริง

สาเหตุที่ผู้วิจัยเลือกวิธีการฟังก์ชันพอยต์มาเทียบกับวิธีการประมาณขนาดซอฟต์แวร์เชิงวัตถุวิธีอื่นๆ เนื่องจากในงานวิจัยหลายๆงานได้ให้ผลการวิจัยเกี่ยวกับการประมาณขนาดซอฟต์แวร์เชิงวัตถุด้วยฟังก์ชันพอยต์ที่ไม่ตรงกัน โดยในปี ค.ศ. 1998 (Fetcke et al., 1997) ได้ทดสอบด้วยวิธีประมาณซอฟต์แวร์เชิงวัตถุด้วยฟังก์ชันพอยต์ สรุปได้ว่า วิธีการประมาณขนาดด้วยฟังก์ชันพอยต์สามารถใช้วัดซอฟต์แวร์ได้โดยไม่ขึ้นกับเทคโนโลยีและสามารถวัดซอฟต์แวร์เชิงวัตถุได้เช่นเดียวกัน แต่ในงานวิจัยในปี ค.ศ. 1992 กล่าวว่า วิธีการประมาณซอฟต์แวร์ด้วยฟังก์ชันพอยต์ไม่ควรนำประมาณซอฟต์แวร์เชิงวัตถุ (Kemerer & Porter, 1992)

1.2 วัตถุประสงค์ของการวิจัย

เปรียบเทียบความแม่นยำในการประมาณค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์เชิงวัตถุด้วยวิธีการประมาณขนาดซอฟต์แวร์เชิงวัตถุ คือ ฟังก์ชันพอยต์ ฟังก์ชันพอยต์เชิงวัตถุ ฟังก์ชันพอยต์เชิงวัตถุสอง และคลาสพอยต์

1.3 ขอบเขตของการวิจัย

งานวิจัยนี้ มีขอบเขตดังนี้

1. ศึกษาเฉพาะระบบที่พัฒนาด้วยภาษาจาวา (JAVA)
2. ศึกษาระบบที่มีการพัฒนาใหม่ทั้งหมดเท่านั้น
3. ศึกษาขนาดของระบบจากเอกสารการออกแบบระบบ
4. ศึกษาวิธีการประมาณการซอฟต์แวร์เชิงวัตถุเพียงสี่วิธี คือ ฟังก์ชันพอยต์ ฟังก์ชันพอยต์เชิงวัตถุ ฟังก์ชันพอยต์เชิงวัตถุสอง และคลาสพอยต์

1.4 ขั้นตอนและวิธีดำเนินการวิจัย

1. ศึกษาปัญหาของการพัฒนาซอฟต์แวร์ทางธุรกิจ
2. ศึกษาปัญหาการประมาณขนาดซอฟต์แวร์ที่มีอยู่ปัจจุบัน
3. ศึกษาแนวคิดและวิธีการประมาณขนาดซอฟต์แวร์เชิงวัตถุ
4. คัดกรองหน่วยตัวอย่างที่สามารถใช้เก็บข้อมูลเพื่อนำมาวิเคราะห์วิธีการประมาณขนาดซอฟต์แวร์เชิงวัตถุ
5. เก็บข้อมูลเพื่อใช้เป็นหน่วยตัวอย่างในธุรกิจการพัฒนาซอฟต์แวร์
6. คำนวณขนาดของซอฟต์แวร์ด้วยวิธีการของฟังก์ชันพอยต์ ฟังก์ชันพอยต์เชิงวัตถุ ฟังก์ชันพอยต์เชิงวัตถุสอง และคลาสพอยต์
7. เปรียบเทียบความแม่นยำของวิธีการประมาณขนาดซอฟต์แวร์ทั้งสี่วิธี
8. วิเคราะห์ผลของความแม่นยำจากแต่ละวิธีการประมาณขนาดซอฟต์แวร์
9. สรุปผลและรายงาน

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1. นักพัฒนาซอฟต์แวร์จะมีแนวทางสำหรับการเลือกใช้วิธีประมาณขนาดของซอฟต์แวร์เชิงวัตถุ โดยสามารถเปรียบเทียบและเลือกใช้วิธีการที่ให้ความแม่นยำที่ดีกว่าสำหรับงานการพัฒนาซอฟต์แวร์เชิงวัตถุ
2. ผู้บริหารโครงการซอฟต์แวร์สามารถเลือกเป็นแนวทางในการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์จริง ทำให้สามารถบริหารโครงการด้วยความเสี่ยงที่ลดลง
3. ผู้บริหารโครงการซอฟต์แวร์สามารถนำไปปรับปรุงต่อยอดเพื่อใช้สำหรับการประเมินต้นทุนในการพัฒนาซอฟต์แวร์ (Cost estimation) ได้
4. ผลการทดสอบของการเปรียบเทียบวิธีประมาณขนาดซอฟต์แวร์เชิงวัตถุในงานวิจัยนี้จะเป็นประโยชน์ต่อนักวิจัยที่สนใจวิธีการประมาณขนาดซอฟต์แวร์เชิงวัตถุ และสามารถนำไปต่อยอดความรู้ต่อไปได้

1.6 คำจำกัดความที่ใช้ในการวิจัย

การประมาณโครงการซอฟต์แวร์ คือ การประมาณคุณลักษณะของซอฟต์แวร์ที่จะพัฒนาในแง่ของการประมาณขนาดซอฟต์แวร์ การประเมินระยะเวลาโครงการซอฟต์แวร์ การประมาณค่าความพยายามที่จะต้องใช้ในการพัฒนาซอฟต์แวร์ รวมไปถึงต้นทุนในการพัฒนาซอฟต์แวร์

การประมาณขนาด (Size estimation) คือ การคิดคำนวณเพื่อประเมินหาขนาดของซอฟต์แวร์ด้วยวิธีคำนวณด้วยวิธีการในแง่มุมต่างๆ เช่น การวัดจำนวนออกมาด้วยจำนวนบรรทัดของโปรแกรม, การคิดคำนวณโดยอ้างอิงจากจำนวนและรายละเอียดฟังก์ชัน, การคิดคำนวณโดยอ้างอิงจากจำนวนและรายละเอียดยูสเคส เป็นต้น ซึ่งขนาดที่ประมาณได้จะสามารถนำไปใช้ในการประเมินระยะเวลาในการพัฒนา, การทำนายค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์ รวมไปถึงการประเมินต้นทุนซอฟต์แวร์

ขนาดของซอฟต์แวร์ (Software size) คือ ลักษณะของซอฟต์แวร์ที่สามารถประเมินได้ด้วยหลากหลายมุมมองและวิธีการ เช่น การวัดได้จากจำนวนบรรทัดของโปรแกรม ฟังก์ชันพอยต์ คลาสพอยต์ อ็อบเจกต์พอยต์ ซึ่งในงานวิจัยนี้จะสนใจขนาดของซอฟต์แวร์ในมุมมองของจำนวนค่าความพยายามที่ต้องใช้ในการพัฒนาซอฟต์แวร์

ค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์ (Development effort) คือ จำนวนแรงงานที่จะต้องทำเพื่อกำจัดงานให้หมดเสร็จสมบูรณ์ ซึ่งในงานวิจัยนี้จะวัดค่าความพยายามที่ใช้ในการพัฒนาเพราะค่าความพยายามที่ใช้ในการพัฒนา จะสามารถวัดประสิทธิภาพของวิธีการประมาณขนาดซอฟต์แวร์ได้เหมาะสมที่สุด เนื่องจากค่าความพยายามที่ใช้ในการพัฒนาแต่ละระบบจะขึ้นแปรผันโดยตรงกับขนาดของแต่ละระบบ ซึ่งในงานวิจัยนี้จะใช้หน่วย คน-ชั่วโมง (Man-hours)

คน-ชั่วโมง (Man-hours) คือ หน่วยของค่าความพยายามของหนึ่งคนในการทำงานด้วยจำนวนชั่วโมงเพื่อทำให้งานเสร็จสิ้น

ซอฟต์แวร์เชิงวัตถุ (Object-oriented) คือ ลักษณะการของซอฟต์แวร์ ซึ่งอ้างอิงและให้ความสำคัญกับวัตถุ ซึ่งวัตถุจะประกอบไปด้วยสองสิ่งต่อไปนี้ คือ ลักษณะประจำ (Attribute) และพฤติกรรม (Behavior) โดยวัตถุจะประกอบไปด้วยคุณสมบัติของซอฟต์แวร์เชิงวัตถุ ดังนี้ (1) ความสามารถในการปกปิดหรือซ่อนเร้น (Encapsulation) (2) ความสามารถการสืบทอด (Inheritance) (3) ความสามารถภาวะพหุสัณฐาน (Polymorphism) และ (4) ความสามารถในการรวมกันขององค์ประกอบ (Composition)

บทที่ 2
เอกสารและงานวิจัยที่เกี่ยวข้อง

2.1 การประมาณขนาดซอฟต์แวร์

การประมาณขนาดซอฟต์แวร์เป็นหนึ่งกิจกรรมที่สำคัญในการบริหารจัดการโครงการซอฟต์แวร์ เพื่อใช้ในการวางแผนในการบริหารต้นทุน ทรัพยากรรวมไปถึงแรงงานคน โดยมีหลากหลายวิธีที่ถูกนำเสนอขึ้นเพื่อใช้ในการประมาณขนาดของซอฟต์แวร์ในแง่มุมที่แตกต่างกันดังนี้

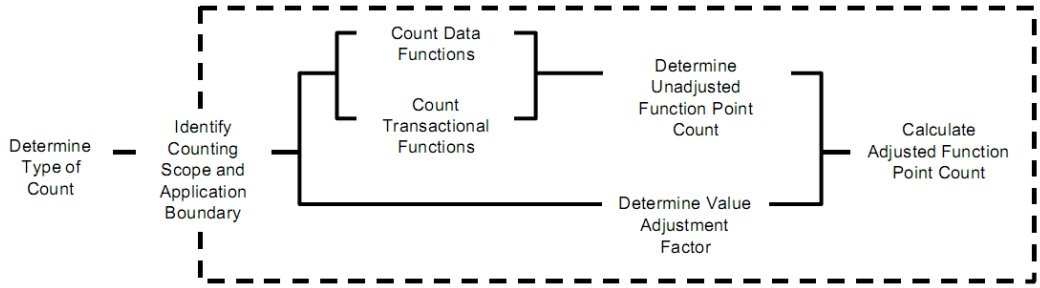
2.1.1 การประมาณขนาดซอฟต์แวร์ด้วยจำนวนบรรทัดของโปรแกรม (Source line of code: SLOC)

วิธีการประมาณขนาดของซอฟต์แวร์ด้วยจำนวนบรรทัดเป็นวิธีการนับจำนวนบรรทัดของซอร์สโค้ดทั้งหมดของภาษาโปรแกรม โดยไม่รวมจำนวนความคิดเห็น (Comment) และบรรทัดว่าง การวัดด้วยจำนวนบรรทัดของโปรแกรมนั้นสามารถวัดได้ง่าย ไม่ซับซ้อน

วิธีการวัดซอฟต์แวร์ด้วยจำนวนบรรทัดจะขึ้นภาษาโปรแกรมที่ใช้ (Programming language) และทักษะของผู้พัฒนา ขนาดของซอฟต์แวร์ที่วัดจากจำนวนบรรทัดของโปรแกรมจะสามารถวัดได้เมื่อระบบพัฒนาเสร็จสมบูรณ์แล้ว ทำให้จำนวนบรรทัดของโปรแกรมจึงไม่สามารถใช้ประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ในก่อนการพัฒนาเสร็จสมบูรณ์ได้ (Carleton, et al., 1992)

2.1.2 การประมาณขนาดซอฟต์แวร์ด้วยการวิธีฟังก์ชันพอยต์ (Function point: FP) (IFPUG, 1999)

วิธีการประมาณขนาดซอฟต์แวร์ด้วยการวิธีห์ฟังก์ชันพอยต์ (Function point: FP) ถูกนำเสนอในปี ค.ศ. 1979 โดย Allan Albrecht เพื่อประมาณขนาดซอฟต์แวร์จากการนับฟังก์ชันการทำงานของซอฟต์แวร์จากมุมมองของผู้ใช้งานระบบ ซึ่งสามารถวัดได้ตั้งแต่ช่วงการวิเคราะห์ระบบ โดยไม่ยึดติดกับประเภทของภาษาและวิธีการพัฒนาซอฟต์แวร์จึงทำให้ฟังก์ชันพอยต์เป็นวิธีการประมาณซอฟต์แวร์ชนิดหนึ่งที่ได้การยอมรับ โดยมีขั้นตอนแบ่งเป็น 7 ขั้นตอนได้ดังรูปที่ 2.1



รูปที่ 2.1 แสดงแผนผังกิจกรรมของการประมาณขนาดซอฟต์แวร์ด้วยวิธีฟังก์ชันพอยต์ (IFPUG, 1999)

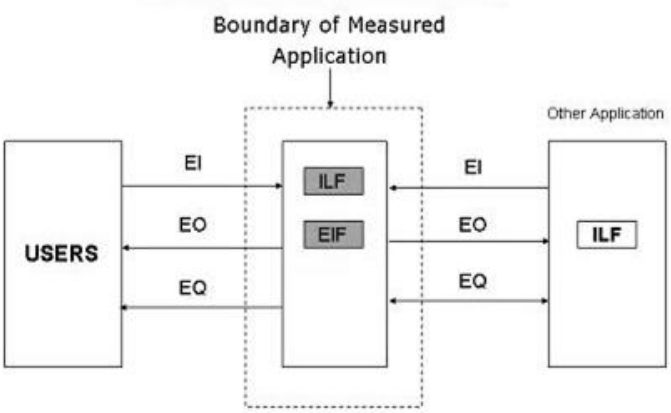
ขั้นตอนที่ 1 แบ่งประเภทของซอฟต์แวร์ที่จะนับ (Determine type of count)

วิธีการประมาณขนาดด้วยฟังก์ชันพอยต์สามารถนำไปใช้กับโครงการพัฒนาซอฟต์แวร์กับสามประเภทโครงการ คือ

1. การนับฟังก์ชันพอยต์ในช่วงการพัฒนาโครงการซอฟต์แวร์ (Development project function point count) ประเภทของการนับฟังก์ชันพอยต์นี้เหมาะกับโครงการที่พัฒนาขึ้นมาใหม่ ขอบเขตงานที่เพิ่มขึ้นสามารถติดตาม และตรวจสอบขนาดฟังก์ชันการทำงานในทุกขั้นตอนของโครงการ ซึ่งประเภทการนับในรูปแบบนี้จะถูกเรียกว่าเป็น “A Baseline Function Point Count”
2. การนับฟังก์ชันพอยต์ในส่วนที่ต้องการเพิ่มเติมส่วนที่เสร็จแล้ว(Enhancement project function point count) เป็นเรื่องปกติเมื่อซอฟต์แวร์ได้รับการพัฒนาเสร็จสิ้นแล้ว และมีการปรับปรุงคุณภาพหรือปรับปรุงประสิทธิภาพเพิ่มเติม ประเภทของการนับฟังก์ชันพอยต์นี้จะรับในส่วนของโครงการที่มีการปรับปรุงเพิ่มเติมจากระบบเดิม นั่นคือ นับส่วนที่จะเพิ่ม เปลี่ยนแปลง หรือลบทิ้งจากฟังก์ชันการทำงานเก่า
3. การนับฟังก์ชันพอยต์ในซอฟต์แวร์ที่พัฒนาเสร็จสมบูรณ์แล้ว (Application function point count) การนับฟังก์ชันพอยต์แบบนี้ จะนับเมื่อซอฟต์แวร์นั้นเสร็จสิ้นแล้ว เพื่อใช้ในประมาณขนาดของซอฟต์แวร์ที่เสร็จสมบูรณ์และส่งมอบให้กับลูกค้า ว่าขนาดของซอฟต์แวร์ในฉบับปัจจุบันเป็นอย่างไร เพื่อตรวจสอบการแก้ไขในแต่ละช่วงเวลา (Baseline) โดยการนับชนิดนี้จะนับเมื่อซอฟต์แวร์เสร็จสมบูรณ์หรือทุกครั้งหลังจากถูกปรับปรุง เปลี่ยนแปลง

ขั้นตอนที่ 2 กำหนดขอบเขตระบบของซอฟต์แวร์ที่จะนับ (Identify Counting Scope and Application Boundary)

ในขั้นตอนนี้จะเป็นการกำหนดขอบเขต (Boundary) เพื่อระบุคุณสมบัติของระบบที่ต้องการนับด้วยฟังก์ชันพอยต์ โดยการนับฟังก์ชันพอยต์ แบ่งออกเป็น 2 ชนิดคือ ดาต้าฟังก์ชัน (Data Functions) และ ทรานแซคชันฟังก์ชัน (Transaction Functions) ดังรูปที่ 2.2



รูปที่ 2.2 แสดงโมเดลของการวิธีฟังก์ชันพอยต์

ที่มา: http://www.codeproject.com สืบค้นเมื่อ 10 สิงหาคม 2554

ขั้นตอนที่ 3 นับจำนวนของดาต้าฟังก์ชัน (Data Functions)

หลังจากที่ได้ระบุขอบเขตแล้วจะสามารถบ่งบอกดาต้าฟังก์ชันของระบบได้ ซึ่งดาต้าฟังก์ชันจะถูกแบ่งออกเป็นสองประเภท ดังนี้

- อินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) คือ ไฟล์ข้อมูลหลักที่อยู่ในระบบ หรือไฟล์ข้อมูลควบคุมที่ใช้ภายในซอฟต์แวร์ โดยที่สามารถสร้าง เรียกใช้ หรือถูกเปลี่ยนแปลงได้โดยผู้ใช้งาน
- เอ็กเทอร์นอลอินเตอร์เฟสไฟล์ (External Interface Files: EIFs) คือ ไฟล์ข้อมูลที่ไม่ได้อยู่ภายในซอฟต์แวร์ ไม่สามารถเปลี่ยนแปลงได้โดยผู้ใช้งาน ต้องเปลี่ยนแปลงจากภายนอกหรือระบบอื่น หรือคือ ไฟล์ข้อมูลภายในของระบบอื่น

*ไฟล์ข้อมูล คือ ชุดของข้อมูลที่ใกล้เคียงกันที่ถูกเก็บเป็นกลุ่มเดียวกัน

นำดาต้าฟังก์ชันทั้งสองประเภทมาประเมินค่าความซับซ้อน จากการนับองค์ประกอบของข้อมูล (Data element types: DET) เรคคอร์ดข้อมูล (Record element types: RET) โดยมีรายละเอียดดังต่อไปนี้

องค์ประกอบของข้อมูล (Data element types: DET) คือ ฟิลด์ (Field) ข้อมูลที่สนใจในแต่ละฟิลด์ โดยมีหลักการในการนับ ดังนี้

1. สำหรับแต่ละฟิลด์ข้อมูลจะนับเป็น 1 DET
2. หากระบบงานตั้งแต่ 2 ระบบขึ้นไปมีการเก็บข้อมูล หรืออ้างอิงไปยัง อินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) หรือ เอ็กเทอร์นอลอินเตอร์เฟสไฟล์ (External Interface Files: EIFs) เดียวกันแต่แยก DET ให้นับ DET เฉพาะส่วนที่มีการเรียกใช้งาน
3. นับเป็น 1 DET สำหรับแต่ละกลุ่มของข้อมูลที่ถูกกำหนดโดยผู้ใช้ เพื่อสร้างความสัมพันธ์กับ อินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) หรือ เอ็กเทอร์นอลอินเตอร์เฟสไฟล์ (External Interface Files: EIFs) อื่น

เรคคอร์ดข้อมูล (Record element types: RET) คือ ประเภทของเรคคอร์ดข้อมูลที่เกี่ยวข้องสัมพันธ์กับฟังก์ชันที่สนใจหรือส่วนที่เป็นเอนทิตีย่อย (Sub Entity) ภายในเอนทิตี (Entity) โดยมีหลักการในการนับ ดังนี้

1. นับ 1 RET สำหรับแต่ละกลุ่มย่อยของ อินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) หรือ เอ็กเทอร์นอลอินเตอร์เฟสไฟล์ (External Interface Files: EIFs)
2. ถ้าไม่มีกลุ่มย่อยให้นับเป็น 1 RET

และนำมาประเมินระดับความซับซ้อนของ อินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) และ เอ็กเทอร์นอลอินเตอร์เฟสไฟล์ (External Interface Files: EIFs) จากจำนวน DET และ RET ที่ได้ตามตารางที่ 2.1

ตารางที่ 2.1 แสดงตารางเพื่อหาระดับความซับซ้อนของ ILFs และ EIFs (IFPUG, 1999)

	1 to 19 DET	**20 to 50 DET**	**51 or more DET**
1 RET	Low	Low	Average
2 to 5 RET	Low	Average	High
6 or more RET	Average	High	High

ขั้นตอนที่ 4 นับจำนวนของทรานแซคชันฟังก์ชัน (Transactional Functions)

หลังจากที่ได้ระบุขอบเขตแล้วจะสามารถบ่งบอกทรานแซคชันฟังก์ชัน (Transactional Functions) ของระบบได้ ซึ่งทรานแซคชันฟังก์ชันของระบบจะถูกแบ่งออกเป็นสามประเภท ดังนี้

- อินเทอร์นอลอินพุท (External Input: EI) คือ ข้อมูลหรือคำสั่งควบคุมที่นำมาจากภายนอกระบบของซอฟต์แวร์ โดยข้อมูลที่ผ่านเข้ามานั้นจะใช้สำหรับปรับปรุงหรือเปลี่ยนแปลง อินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) เช่น ข้อความจากระบบงานอื่นที่ส่งมาเพื่อให้ระบบทำงาน ไฟล์ข้อมูลงานจากระบบอื่น ข้อมูลที่ส่งมาจากภายนอก รายการข้อมูลที่ใช้ในการรักษา อินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs)
- เอ็กซ์เทอร์นอลเอาท์พุท (External Output: EO) คือ ข้อมูลหรือคำสั่งควบคุมที่ถูกส่งออกจากระบบของซอฟต์แวร์ไปยังนอกระบบของงาน โดยกระบวนการส่งข้อมูลออกไปนั้นมักจะเกี่ยวข้องกับการคำนวณทางคณิตศาสตร์หรือเกี่ยวข้องกับสูตรในการสร้างคำนวณหรือเปลี่ยนแปลงข้อมูลภายใน อินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) ในระบบ เช่น รายงานที่ต้องมีการคำนวณ เช่น ผลกำไรรายสัปดาห์ ข้อมูลที่ได้จากการคำนวณ ภาพกราฟต่างๆ
- เอ็กซ์เทอร์นอลอินไควรี (External Inquiry: EQ) คือ ข้อมูลที่ดึงมาจากนอกระบบหรือถูกส่งออกนอกระบบจากทั้ง อินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) หรือ เอ็กเทอร์นอลอินเตอร์เฟสไฟล์ (External Interface Files: EIFs) โดยไม่ผ่านการแก้ไขใดๆ หรือการคำนวณใดๆ เช่น ข้อมูลรายการที่ดึงมาจาก อินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) และ เอ็กซ์เทอร์นอลอินเตอร์เฟสไฟล์ (External Interface Files: EIFs) เพื่อเรียกดูหรือพิมพ์ รายงานที่ไม่ต้องมีการคำนวณใดๆ

นำทรานแซคชันฟังก์ชัน (Transactional Functions) ของระบบทั้งสามประเภทมาประเมินค่าความซับซ้อนของแต่ละทรานแซคชันฟังก์ชัน โดยแบ่งวิธีการประเมินระดับความซับซ้อนออกเป็นสองชนิด

การประเมินระดับความซับซ้อนของเอ็กเทอร์นอลอินพุท (External Input: EI)

การประเมินของแต่ละเอ็กเทอร์นอลอินพุท (External Input: EI) จะประเมินจากการนับจำนวนประเภทของไฟล์ข้อมูลอ้างอิง (File Type Referenced: FTR) และการนับจำนวนองค์ประกอบของข้อมูล (Data Element Type: DET) ซึ่งมีรายละเอียดในการนับแสดงได้ดังนี้

วิธีการนับองค์ประกอบของข้อมูล (Data element types: DET) ของ เอ็กเทอร์นอลอินพุท (External Input: EI) มีดังนี้

นับเป็น 1 DET สำหรับแต่ละฟิลด์หรือแอททริบิวต์ของข้อมูลที่ได้รับการกำหนดไว้ในลักษณะผู้ใช้งานจดจำได้ คือ มีการกำหนดมาจากความต้องการของซอฟต์แวร์ และไม่มีซ้ำ รวมทั้งนับในส่วนของฟอเรนคีย์ (Foreign key) ที่เกิดขึ้นระหว่างระบบงาน เพื่อให้กระบวนการย่อยสุดของระบบงานได้ทำงานอย่างสมบูรณ์

ไม่นับรวม DET สำหรับฟิลด์ข้อมูลที่ผู้ใช้ไม่ได้กรอกข้อมูล เช่น ฟิลด์วันที่ที่ระบบขึ้นให้อัตโนมัติ

นับเป็น 1 DET สำหรับแต่ละความสามารถในการส่งข้อความตอบรับ ไปยังภายนอกของระบบงานเพื่อแจ้งเตือนความผิดพลาดหรือยืนยันว่ากระบวนการทำงานนั้นเสร็จสมบูรณ์

นับเป็น 1 DET สำหรับแต่ความสามารถในการกำหนดการทำงานที่จัดการโดย EI เช่น ในส่วนของ Command line นั้น หากมีหลายๆ Command line ทำงานร่วมกันเพื่อทำกิจกรรมอย่างใดอย่างหนึ่ง จะนับเพียง 1 DET ไม่นับตามจำนวน Command line

วิธีการนับไฟล์ข้อมูลอ้างอิง (File Type Referenced: FTR) ของเอ็กเทอร์นอลอินพุท (External Input: EI) มีดังนี้

นับเป็น 1 FTR สำหรับแต่ละประเภทของอินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) ที่ได้รับการรักษาไว้ภายใต้กระบวนการของอินเทอร์นอลอินพุท (External Input: EI)

นับเป็น 1 FRT สำหรับแต่ละประเภทของ อินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) หรือ เอ็กเทอร์นอลอินเตอร์เฟสไฟล์ (External Interface Files: EIFs) ที่ได้รับการอ้างอิงจากกระบวนการของอินเทอร์นอลอินพุท (External Input: EI)

นับเพียง 1 FTR สำหรับแต่ละ อินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) ที่ได้รับการอ้างอิงหรือเรียกอ่านหรือเก็บรักษาไว้ภายใต้อินเทอร์นอลอินพุท (External Input: EI)

หลังจากที่ได้จำนวน DET และ FTR ของเอ็กเทอร์นอลอินพุท (External Input: EI) จะนำมาประเมินระดับความซับซ้อนของแต่ละข้อมูล External Input (EI) สามารถหาได้ตามตารางที่ 2.2

ตารางที่ 2.2 แสดงตารางเพื่อหาระดับความซับซ้อนของอินเทอร์นอลอินพุท (External Input: EI) (IFPUG, 1999)

	1 to 4 DET	**5 to 15 DET**	**16 or more DET**
0 to 1 FTR	Low	Low	Average
2 FTRs	Low	Average	High
3 or more FTRs	Average	High	High

การประเมินระดับความซับซ้อนเอ็กซ์เทอร์นอลเอาท์พุท (External Output: EO) และเอ็กซ์เทอร์นอลอินไควรี (External Inquiry: EQ)

การประเมินระดับความซับซ้อนของแต่ละเอ็กซ์เทอร์นอลเอาท์พุท (External Output: EO) และ เอ็กซ์เทอร์นอลอินไควรี (External Inquiry: EQ) จะนับจำนวนประเภทของไฟล์ข้อมูลอ้างอิง (File Type Referenced: FTR) และจำนวนองค์ประกอบของข้อมูล (Data Element Type: DET) ของ ซึ่งมีรายละเอียดในการนับแสดงได้ดังนี้

วิธีการนับองค์ประกอบของข้อมูล (Data element types: DET) ของ เอ็กซ์เทอร์นอลเอาท์พุท (External Output: EO) และ เอ็กซ์เทอร์นอลอินไควรี (External Inquiry: EQ) มีดังนี้

1. นับเป็น 1 DET สำหรับแต่ละฟิลด์หรือแอททริบิวต์ของข้อมูลที่ได้รับการกำหนดไว้ในลักษณะผู้ใช้งานจดจำได้ คือ มีการกำหนดมาจากความต้องการของซอฟต์แวร์ (Requirement) และไม่มีซ้ำ รวมทั้งนับในส่วนของฟอเรนคีย์ (Foreign Key) ที่เกิดขึ้นระหว่างระบบงาน เพื่อให้กระบวนการย่อยสุดของระบบงานได้ทำงานอย่างสมบูรณ์
2. นับเป็น 1 DET สำหรับแต่ละฟิลด์หรือแอททริบิวต์ที่อยู่ภายนอกระบบงาน
3. หาก DET นั้นๆ เกิดขึ้นทั้งภายนอกและเข้าสู่ระบบงานให้นับเพียงครั้งเดียว
4. นับเป็น 1 DET สำหรับแต่ละการส่งข้อความตอบรับหรือข้อความตอบกลับจากระบบ (System Response Message) ไปยังภายนอกของระบบงานเพื่อแจ้งเตือนความผิดพลาด (Error) หรือเพื่อทำการยืนยันว่ากระบวนการได้ทำงานเสร็จสมบูรณ์หรือทำการตรวจสอบกระบวนการ
5. นับเป็น 1 DET สำหรับแต่ละความสามารถในการกำหนดการทำงานที่จัดการโดยเอ็กซ์เทอร์นอลเอาท์พุท (External Output: EO) และ เอ็กซ์เทอร์นอลอินไควรี (External Inquiry: EQ) เช่น นับเป็น 1 DET สำหรับปุ่มตกลง ปุ่มฟังก์ชันคีย์ หรือการคลิกเมาส์ที่ให้ผลเท่ากับการตอบตกลง
6. นับเพียง 1 DET สำหรับฟิลด์ข้อมูลที่มีการจัดเก็บไว้หลายที่ แต่เป็นเพียงหนึ่งฟิลด์ในเชิงตรรกะ หรือแม้แต่ฟิลด์ที่ให้เพียงความหมายเดียวแต่จับเก็บแยกฟิลด์ เช่น ข้อมูลที่อยู่ที่สามารถแยกเป็นถนน เมือง และรหัสไปรษณีย์แต่แสดงอยู่ในป้ายที่อยู่เดียวกัน (Label)
7. นับเพียง 1 DET สำหรับข้อมูลที่ประกอบด้วยคำ ประโยค หรือย่อหน้า

วิธีการนับไฟล์ข้อมูลอ้างอิง (File Type Referenced: FTR) ของ เอ็กซ์เทอร์นอลเอาท์พุท (External Output: EO) และ เอ็กซ์เทอร์นอลอินไควรี (External Inquiry: EQ) มีดังนี้

1. นับเป็น 1 FTR สำหรับแต่ละประเภทของอินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) ที่ได้รับการเก็บรักษาไว้ภายใต้การบวนการของอีโอ (EO) หรือ อีคิว (EQ)
2. นับเป็น 1 FTR สำหรับแต่ละประเภทของ อินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) หรือ เอ็กเทอร์นอลอินเตอร์เฟสไฟล์ (External Interface Files: EIFs) ที่ได้รับการอ้างอิงจากกระบวนการของ อีโอ (EO) หรือ อีคิว (EQ)
3. นับเพียง 1 FTR สำหรับแต่ละ ILF ที่ทั้งได้รับการอ้างอิง หรือเรียกอ่าน หรือเก็บรักษาไว้ภายใต้ อีโอ (EO) หรือ อีคิว (EQ)

หลังจากที่ได้จำนวน DET และ FTR ของแต่ละเอ็กซ์เทอร์นอลเอาท์พุท (External Output: EO) และเอ็กซ์เทอร์นอลอินไควรี (External Inquiry: EQ) จะนำมาประเมินระดับความซับซ้อนของ ซึ่งสามารถหาได้ตามตารางที่ 2.3

ตารางที่ 2.3 แสดงตารางเพื่อหาระดับความซับซ้อนของเอ็กซ์เทอร์นอลเอาท์พุท (External output: EO) และเอ็กซ์เทอร์นอลอินไควรี (External Inquiry: EQ) (IFPUG, 1999)

	1 to 5 DET	**6 to 19 DET**	**20 or more DET**
0 to 1 FTR	Low	Low	Average
2 to 3 FTRs	Low	Average	High
4 or more FTRs	Average	High	High

ขั้นตอนที่ 5 คำนวณค่าฟังก์ชันพอยต์ที่ยังไม่ได้ปรับค่า (Determine Unadjusted Function Point Count)

ในขั้นตอนนี้จะนำค่าที่ได้จากการประเมินค่าความซับซ้อนจากขั้นตอนที่ 3 และ 4 จากการนำทุกชนิดของฟังก์ชันมาคูนกับค่าน้ำหนักความซับซ้อนตามระดับความซับซ้อนที่ประเมินได้ จะทำให้ได้ค่าฟังก์ชันพอยต์ที่ยังไม่ได้ปรับค่า (Unadjusted Function Point Count) ซึ่งสามารถแสดงรายละเอียดการคำนวณได้ ตามตารางที่ 2.4

ตารางที่ 2.4 แสดงตารางคำนวณค่าฟังก์ชันพอยต์ที่ยังไม่ได้ปรับค่า

Function Type	Functional Complexity		Complexity Totals		Function Type Totals
ILFs	____	Low	X 7 =	____	
	____	Average	X 10 =	____	
	____	High	X 15 =	____	_____
EIFs	____	Low	X 5 =	____	
	____	Average	X 7 =	____	
	____	High	X 10 =	____	_____
EIs	____	Low	X 3 =	____	
	____	Average	X 4 =	____	
	____	High	X 6 =	____	_____
EOs	____	Low	X 4 =	____	
	____	Average	X 5 =	____	
	____	High	X 7 =	____	_____
EQs	____	Low	X 3 =	____	
	____	Average	X 4 =	____	
	____	High	X 6 =	____	_____
Unadjusted Function Point Count					_____

ขั้นตอนที่ 6 กำหนดตัวแปรปรับค่า (Value Adjustment Factor)

สำหรับการคำนวณฟังก์ชันพอยต์นั้น จะมีการปรับตามปัจจัยความซับซ้อนทางเทคนิคของลักษณะซอฟต์แวร์ 14 ลักษณะ แต่ละลักษณะกำหนดค่าความมีอิทธิพลต่อซอฟต์แวร์ (Degree of influence หรือ DI) โดยมีค่าตั้งแต่ 0 – 5 คือ 0 ไม่เกี่ยวข้อง จนถึง 5 มีอิทธิพลมาก

โดยมีรายละเอียดในการกำหนดลักษณะของซอฟต์แวร์ทั้ง 14 ตัว ดังตารางที่ 2.5 และนำมากรอกค่ารวมในของลักษณะของซอฟต์แวร์ 14 ตัว ตามตารางที่ 2.6

ตารางที่ 2.5 แสดงรายละเอียดวิธีการกำหนดค่าปัจจัยที่มีผลกระทบประมาณค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์ด้วยของฟังก์ชันพอยต์ (IFPUG, 1999 แปลโดย วรัฏฐา เจริญสถาพงษ์)

1.Data Communication จะมองเรื่องของจำนวนของสิ่งที่ช่วยอำนวยความสะดวกในการถ่ายโอนข้อมูลหรือแลกเปลี่ยนข้อมูลระหว่างซอฟต์แวร์หรือระบบ	
0	ซอฟต์แวร์ทั้งหมดรันแบบ Batch หรือ standalone PC
1	ซอฟต์แวร์เป็นแบบ Batch แต่มีการใช้ลักษณะของ Remote Data Entry หรือ Remote Printing
2	ซอฟต์แวร์เป็นแบบ Batch แต่มีการใช้ลักษณะของ Remote Data Entry และ Remote Printing
3	ซอฟต์แวร์เป็นแบบ Online Data Collection หรือ Teleprocessing จากส่วน Front-end สู่ Batch
4	ซอฟต์แวร์มีลักษณะของ Teleprocessing ซึ่งมากกว่าในส่วนของ Front-end แต่สนับสนุนการติดต่อ Teleprocessing Protocol เพียงชนิดเดียว
5	ซอฟต์แวร์มีลักษณะของ Teleprocessing ซึ่งมากกว่าในส่วนของ Front-end แต่สนับสนุนการติดต่อ Teleprocessing Protocol ที่หลากหลายชนิดมากขึ้น

2. Distributed Data Processing จะมองในเรื่องการจัดการข้อมูลในเชิง Distributed และขั้นตอนหรือหน้าที่ในการจัดการ	
0	ซอฟต์แวร์สามารถช่วยในการถ่ายข้อมูลหรือช่วยในการดำเนินการระหว่างกันได้
1	ซอฟต์แวร์สามารถเตรียมข้อมูลสำหรับ End User เพื่อดำเนินขั้นตอนต่อไปในส่วนอื่นๆ เช่น PC Spreadsheets และ PC DBMS
2	ซอฟต์แวร์สามารถเตรียมข้อมูลเพื่อถ่ายโอนและถ่ายโอนไปยังขั้นตอนอื่นๆ ในระบบเพื่อประมวลผลได้ (ไม่เพียงแต่ถ่ายโอนสำหรับ End User)
3	ซอฟต์แวร์มีลักษณะของ Distributed processing และการถ่ายโอนข้อมูลเป็นแบบออนไลน์แต่เป็นในทิศทางเดียว
4	ซอฟต์แวร์มีลักษณะของ Distributed processing และการถ่ายโอนข้อมูลเป็นแบบออนไลน์แต่เป็นใน 2 ทิศทาง
5	ซอฟต์แวร์มีการจัดการการถ่ายโอนข้อมูลแบบไดนามิกขึ้นอยู่กับแต่ละส่วนของระบบ

ตารางที่ 2.5 (ต่อ) แสดงรายละเอียดวิธีการกำหนดค่าปัจจัยที่มีผลกระทบประมาณค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์ด้วยของฟังก์ชันพอยต์ (IFPUG, 1999 แปลโดย วรัฏฐา เจริญสถาพงษ์)

3. Performance จะมองในเรื่องของ Response Time ประสิทธิภาพของซอฟต์แวร์นั้นจะได้รับการประเมินโดย User ซึ่งเป็นผลสืบเนื่องจากการออกแบบพัฒนาการติดตั้งและรองรับซอฟต์แวร์นั้นๆ	
0	ไม่มีสภาพของประสิทธิภาพที่ User ต้องการเป็นพิเศษ
1	ประสิทธิภาพและการออกแบบตามความต้องการของ User ได้รับการทบทวนแต่ไม่มีความต้องการกระทำอะไรเป็นพิเศษ
2	Response Time และผลลัพธ์ของการทำงานเป็นเรื่องสำคัญและจำเป็นระหว่างช่วงเวลาที่มีการใช้งานสูงๆ การประมวลผลต้องให้เรียบร้อยก่อนวัดถัดไป
3	Response Time และผลลัพธ์ของการทำงานเป็นเรื่องที่สำคัญและจำเป็นระหว่างช่วงเวลาที่มีการใช้งานในธุรกิจ ไม่มีการออกแบบสำหรับ CPU ที่ต้องการใช้งานเฉพาะที่มีการประมวลผลที่มีความต้องการเฉพาะ รวมทั้งการออกแบบอินเตอร์เฟสมีการระบุเฉพาะ
4	เพิ่มในส่วนของความต้องการของ User ในเรื่องของประสิทธิภาพตั้งแต่ขั้นตอนของการวิเคราะห์ระบบงาน
5	เพิ่มในส่วนของการนำเครื่องมือในการวิเคราะห์ประสิทธิภาพเข้ามาใช้ในขั้นตอนการออกแบบพัฒนา และนำไปใช้เพื่อที่จะให้ตรงกับประสิทธิภาพที่ User มี

4. Heavily Used Configuration จะมองในเรื่องของการใช้ที่เกี่ยวกับฮาร์ดแวร์ และ Platform ซึ่งเป็นคุณลักษณะหนึ่งของซอฟต์แวร์ เช่น User ต้องการรันซอฟต์แวร์บนอุปกรณ์หรือเครื่องที่มีการใช้งานอย่างหนัก	
0	ไม่มีความชัดเจนในเรื่องของความเข้มงวดในความสามารถในการจัดการให้สามารถใช้งานได้
1	มีความเข้มงวดในการจัดการให้เพียบแค่ใช้งานได้แต่ความเข้มงวดก็ยังน้อยในซอฟต์แวร์ซึ่งเป็นตัวอย่าง
2	มีระบบรักษาความปลอดภัยหรือการคำนึงถึงข้อจำกัดด้านเวลารวมอยู่ด้วย
3	มีการเจาะจงถึง Processor (เช่น Centrino P4) สำหรับซอฟต์แวร์
4	มีการระบุถึงความเข้มงวดอย่างยิ่งในการประมวลผลซอฟต์แวร์ต่อหน่วยประมวลผลกลาง (Central Processor) ซึ่งมีลักษณะที่เฉพาะเจาะจง
5	มีข้อจำกัดพิเศษบนซอฟต์แวร์ในส่วนประกอบแบบ Distributed ของระบบ

ตารางที่ 2.5 (ต่อ) แสดงรายละเอียดวิธีการกำหนดค่าปัจจัยที่มีผลกระทบประมาณค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์ด้วยของฟังก์ชันพอยต์ (IFPUG, 1999 แปลโดย วรัฏฐา เจริญสถาพงษ์)

5. Transaction Rate จะมองในเรื่องความถี่ในการประมวลผล Transaction ว่าเป็นแบบรายวัน รายสัปดาห์ รายเดือน	
0	ไม่มีการคาดถึงช่วงที่มีระดับสูงสุดของ Transaction
1	ช่วงที่ระดับของจำนวน Transaction เพิ่มสูงสุดนั้นได้ถูกคาดคะเน (รายเดือน รายปักษ์ รายปี)
2	มีการคาดคะเนว่าจำนวน Transaction สูงสุดเป็นรายสัปดาห์
3	มีการคาดคะเนว่าจำนวน Transaction สูงสุดเป็นรายวัน
4	มีอัตราของ Transaction สูงมากซึ่งได้รับการบอกกล่าวจาก User ในซอฟต์แวร์นั้นๆ Requirement ต้องการมากเพียงพอเพื่อที่จะมีการวิเคราะห์ประสิทธิภาพของงานในขั้นตอนการออกแบบ
5	มีอัตราของ Transaction สูงมาก ซึ่งได้รับการบอกกล่าวจาก User ในซอฟต์แวร์นั้นๆ Requirement ต้องมากเพียงพอเพื่อที่จะมีการวิเคราะห์ประสิทธิภาพของงานในขั้นตอนการออกแบบ รวมไปถึงขั้นตอนของการพัฒนาและการติดตั้ง

6. On-Line Data Entry จะมองในเรื่องของเปอร์เซ็นต์ของข้อมูลที่ผ่านมาทางออนไลน์	
0	ทุกๆ Transaction จะเป็นรูปแบบของ Batch
1	1% ถึง 7% ของ Transaction เป็น Interactive Data Entry
2	8% ถึง 15% ของ Transaction เป็น Interactive Data Entry
3	16% ถึง 23% ของ Transaction เป็น Interactive Data Entry
4	24% ถึง 30% ของ Transaction เป็น Interactive Data Entry
5	> 30% ของ Transaction เป็น Interactive Data Entry

ตารางที่ 2.5 (ต่อ) แสดงรายละเอียดวิธีการกำหนดค่าปัจจัยที่มีผลกระทบประมาณค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์ด้วยของฟังก์ชันพอยต์ (IFPUG, 1999 แปลโดย วรัฏฐา เจริญสถาพงษ์)

7. End-User Efficiency จะมองในเรื่องของผู้ใช้งานระบบว่าสามารถใช้งานระบบได้อย่างมีประสิทธิภาพหรือไม่ ซึ่งลักษณะของปัจจัยที่ส่งผลต่อ User Friendly การให้ความสำคัญกับประสิทธิภาพประกอบด้วย Navigation Aids Menu Online Help and Document Automated Cursor Movement Scrolling Remote Printing (Via Online Transaction) Reassigned Function Keys Batched Job Submitted From Online Transactions Cursor Selection Of Screen Data Heavy Use Of Reverse Video Highlighting Color Underling And Other Indicators Hard Copy User Documentation Of Online Transactions Mouse Interface Pop Up Windows As Few Screen As Possible To Accomplish A Business Function Bilingual Support Multilingual Support	
0	ไม่มีการให้ความสำคัญกับประสิทธิภาพตามข้อกำหนดข้างต้น
1	มีการให้ความสำคัญกับประสิทธิภาพ 1-3 ข้อจากข้อกำหนดข้างต้น
2	มีการให้ความสำคัญกับประสิทธิภาพ 4-5 ข้อจากข้อกำหนดข้างต้น
3	มากกว่า 6 ข้อจากข้อกำหนดข้างต้นโดย User ไม่ได้มีความต้องการที่เฉพาะเจาะจงในเรื่องของ Efficiency
4	มากกว่า 6 ข้อจากข้อกำหนดข้างต้นโดย User มีการใช้ความสนใจกับเรื่องของ Efficiency โดยต้องมีการออกแบบซอฟต์แวร์อย่างเฉพาะเพื่อสนับสนุน Efficiency
5	มากกว่า 6 ข้อจากข้อกำหนดข้างต้นโดย User มีการใช้ความสนใจกับเรื่องของ Efficiency โดยต้องมีการใช้งานอุปกรณ์และขั้นตอนที่พิเศษซึ่งจะทำให้การทำงานนั้นบรรลุตามวัตถุประสงค์

ตารางที่ 2.5 (ต่อ) แสดงรายละเอียดวิธีการกำหนดค่าปัจจัยที่มีผลกระทบประมาณค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์ด้วยของฟังก์ชันพอยต์ (IFPUG, 1999 แปลโดย วรัฏฐา เจริญสถาพงษ์)

8. On-Line Update จะมองในเรื่องของ Internal Logical file ว่ามีจำนวนมากน้อยเพียงใด ที่ได้รับการอัพเดทผ่าน Online Transaction	
0	ไม่มีการอัพเดทออนไลน์
1	การอัพเดทเป็นการอัพเดท 1-3 ไฟล์โดยการอัพเดทนั้นเป็นไปได้ช้า
2	การอัพเดทเป็นการอัพเดท 4 ไฟล์ขึ้นไป โดยการอัพเดทนั้นเป็นไปได้ช้า
3	สามารถออนไลน์อัพเดทได้ในส่วนหลักๆของ Internal Logical files
4	เพิ่มเติมการป้องกันข้อมูลสูญหายในขณะทำ Online Update ซึ่งได้มีการออกแบบและสร้างอย่างพิเศษในซอฟต์แวร์
5	เพิ่มเติมการป้องกันข้อมูลสูญหายของข้อมูลรวมทั้งมีการจัดการระบบกู้ข้อมูลอัตโนมัติ

9. Complex Processing เป็นส่วนของคุณลักษณะของซอฟต์แวร์ โดยส่วนประกอบมีดังนี้ - มีส่วนขยายของ Logical Processing - มีส่วนขยายของ Mathematical Processing - มีการประมวลผลที่ซับซ้อนเพื่อจัดการกับ Input และ Output เช่นการมี Multimedia - ข้อผิดพลาดที่เกิดจากการประมวลผลก่อให้เกิด Transaction ที่ไม่สมบูรณ์นั้นต้องทำการประมวลผลอีกครั้ง - Sensitive Control เช่นความปลอดภัยในการประมวลผลข้อมูล	
0	ไม่มีคุณลักษณะของซอฟต์แวร์ตามลักษณะดังที่กล่าวได้ด้านบน
1	มีคุณลักษณะของซอฟต์แวร์ตามลักษณะดังที่กล่าวได้ด้านบน 1 ข้อ
2	มีคุณลักษณะของซอฟต์แวร์ตามลักษณะดังที่กล่าวได้ด้านบน 2 ข้อ
3	มีคุณลักษณะของซอฟต์แวร์ตามลักษณะดังที่กล่าวได้ด้านบน 3 ข้อ
4	มีคุณลักษณะของซอฟต์แวร์ตามลักษณะดังที่กล่าวได้ด้านบน 4 ข้อ
5	มีคุณลักษณะของซอฟต์แวร์ตามลักษณะดังที่กล่าวได้ด้านบนครบ 5 ข้อ

ตารางที่ 2.5 (ต่อ) แสดงรายละเอียดวิธีการกำหนดค่าปัจจัยที่มีผลกระทบประมาณค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์ด้วยของฟังก์ชันพอยต์ (IFPUG, 1999 แปลโดย วรัฏฐา เจริญสถาพงษ์)

10. Reusability จะมองในเรื่องของซอฟต์แวร์ที่ได้ทำการพัฒนานั้นสามารถนำไปปรับใช้หรือสามารถนำไปพัฒนาต่อไปในโครงการอื่นๆ ได้หรือไม่	
0	ไม่มีการ Reuse โค้ด
1	การ Reuse มีเพียงภายในซอฟต์แวร์
2	มีการนำโค้ดไป Reuse น้อยกว่า 10% ในหลายๆ งาน
3	มีการนำโค้ดไป Reuse มากกว่า 10% ในหลายๆ งาน
4	ทั้งซอฟต์แวร์และเอกสารประกอบการพัฒนาค่อนข้างเอื้อต่อการนำไป Reuse
5	ทั้งแอพพลิเคชันและเอกสารประกอบการพัฒนาเอื้อต่อการนำไป Reuse

11. Installation Ease จะมองในเรื่องของการติดตั้งซอฟต์แวร์	
0	ไม่มีลักษณะพิเศษในการติดตั้ง
1	มีลักษณะพิเศษในการติดตั้ง แต่ไม่มีความต้องการเพิ่มเติมจาก User
2	การติดตั้งมีการกำหนดจาก User มีคู่มือในการติดตั้งและปรับเปลี่ยนโดยคู่มือที่จัดทำนั้นต้องผ่านการตรวจสอบและจะไม่คำนึงถึงผลกระทบจากการติดตั้ง
3	การติดตั้งมีการกำหนดจาก User มีคู่มือในการติดตั้งและปรับเปลี่ยนโดยคู่มือที่จัดทำนั้นต้องผ่านการตรวจสอบและคำนึงถึงผลกระทบจากการติดตั้ง
4	เพิ่มจากระดับของปัจจัยที่ 2 โดยมีระบบติดตั้งและเปลี่ยนแปลงอัตโนมัติ
5	เพิ่มจากระดับของปัจจัยที่ 3 โดยมีระบบติดตั้งและเปลี่ยนแปลงอัตโนมัติ

12. Operation Ease จะมองในเรื่องของประสิทธิภาพหรือระบบอัตโนมัติในการ Start Up, Back Up และขั้นตอนการ Recovery และมีการทดสอบอย่างมีประสิทธิภาพ โดยต้องมีคู่มือประกอบซอฟต์แวร์ที่ได้จัดทำขึ้น (Manual)	
0	ไม่มีลักษณะพิเศษในการจัดทำ Start Up, Back Up และ Recovery
1	มีลักษณะพิเศษที่มีประสิทธิภาพในการจัดทำ Start Up, Back Up และ Recovery
2	
3	
4	

ตารางที่ 2.5 (ต่อ) แสดงรายละเอียดวิธีการกำหนดค่าปัจจัยที่มีผลกระทบประมาณค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์ด้วยของฟังก์ชันพอยต์ (IFPUG, 1999 แปลโดย วรัฏฐา เจริญสถาพงษ์)

12. Operation Ease จะมองในเรื่องของประสิทธิภาพหรือระบบอัตโนมัติในการ Start Up, Back Up และขั้นตอนการ Recovery และมีการทดสอบอย่างมีประสิทธิภาพ โดยต้องมีคู่มือประกอบซอฟต์แวร์ที่ได้จัดทำขึ้น (Manual)	
5	ไม่ต้องมีผู้จัดการซอฟต์แวร์ โดยซอฟต์แวร์นั้นสามารถจัดการเองได้ รวมทั้งสามารถย้อนเพื่อแก้ไขข้อผิดพลาด

13. Multiple Sites จะมองในเรื่องของซอฟต์แวร์ว่าสามารถพัฒนาและสนับสนุนการติดตั้ง ใช้งานได้หลายพื้นที่ หรือหลายองค์กร	
0	User ไม่มีความต้องการพิเศษในการทำ Multiple Site
1	ความจำเป็นที่จะต้องมี Multiple Site หรือไม่นั้นต้องมีการพิจารณา ซึ่งถ้าเป็นแบบ Multiple Site แล้วนั้น ซอฟต์แวร์จะถูกพัฒนาภายใต้ลักษณะของ ฮาร์ดแวร์และซอฟต์แวร์เดียวกัน
2	ความจำเป็นที่จะต้องมี Multiple Site หรือไม่นั้นต้องมีการพิจารณา ซึ่งถ้าเป็นแบบ Multiple Site แล้วนั้น ซอฟต์แวร์จะถูกพัฒนาภายใต้ลักษณะของ ฮาร์ดแวร์และซอฟต์แวร์ที่เหมือนกัน
3	ความจำเป็นที่จะต้องมี Multiple Site หรือไม่นั้นต้องมีการพิจารณา ซึ่งถ้าเป็นแบบ Multiple Site แล้วนั้น ซอฟต์แวร์จะถูกพัฒนาภายใต้ลักษณะของ ฮาร์ดแวร์และซอฟต์แวร์ที่แตกต่างกัน
4	เอกสารประกอบได้มีการจัดการทดสอบและสนับสนุนซอฟต์แวร์แบบ Multiple Sites ภายใต้ระดับของปัจจัยที่ 1 หรือ 2
5	เอกสารประกอบได้มีการจัดการทดสอบและสนับสนุนซอฟต์แวร์แบบ Multiple Sites ภายใต้ระดับของปัจจัยที่ 3

ตารางที่ 2.5 (ต่อ) แสดงรายละเอียดวิธีการกำหนดค่าปัจจัยที่มีผลกระทบประมาณค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์ด้วยของฟังก์ชันพอยต์ (IFPUG, 1999 แปลโดย วรัฏฐา เจริญสถาพงษ์)

14. Facilitate change จะมองในเรื่องของซอฟต์แวร์ว่าสามารถออกแบบและพัฒนาเพื่อสามารถเปลี่ยนซอฟต์แวร์ได้ตามความสะดวกหรือไม่ - การ Query ข้อมูลที่ยืดหยุ่นและสิ่งที่ช่วยออกแบบรายงานให้ง่ายขึ้น โดยสามารถดึงข้อมูลอย่างง่ายๆ ที่ต้องการได้ เช่น ตรรกะแบบ And/Or ซึ่งใช้งานกับ 1 Internal Logical file - การ Query ข้อมูลที่ยืดหยุ่นและสิ่งที่ช่วยออกแบบรายงานให้ง่ายขึ้น โดยสามารถดึงข้อมูลที่ซับซ้อนที่ต้องการได้ เช่น ตรรกะแบบ And/Or ซึ่งใช้งานมากกว่า 1 Internal Logical file - การ Query ข้อมูลที่ยืดหยุ่นและสิ่งที่ช่วยออกแบบรายงานให้ง่ายขึ้น โดยสามารถดึงข้อมูลที่ซับซ้อนได้ เช่น ตรรกะแบบ And/Or ซึ่งใช้งานได้มากกว่า 1 Internal Logical file - ข้อมูลที่ใช้ในการควบคุมทางธุรกิจนั้นจะถูกเก็บไว้ในตารางโดย User จะทำการ Maintain โดยการ Online Interactive Processes การเปลี่ยนแปลงใดๆ ที่เกิดขึ้นนั้นจะกระทบกับวันที่มาทำธุรกิจในวันถัดไปได้ - ข้อมูลที่ใช้ในการควบคุมทางธุรกิจนั้นจะถูกเก็บไว้ในตารางโดย User จะทำการ Maintain โดยการ Online Interactive Processes การเปลี่ยนแปลงใดๆ ที่เกิดขึ้นจะส่งผลทันทีทันใด	
0	ไม่มีคุณลักษณะของ Facilitate ตามที่กล่าวมาด้านบน
1	มีคุณลักษณะของ Facilitate ตามที่กล่าวมาด้านบน 1 ข้อ
2	มีคุณลักษณะของ Facilitate ตามที่กล่าวมาด้านบน 2 ข้อ
3	มีคุณลักษณะของ Facilitate ตามที่กล่าวมาด้านบน 3 ข้อ
4	มีคุณลักษณะของ Facilitate ตามที่กล่าวมาด้านบน 4 ข้อ
5	มีคุณลักษณะของ Facilitate ตามที่กล่าวมาด้านบน 5 ข้อ

ตารางที่ 2.6 แสดงตารางการคำนวณปัจจัยที่เกี่ยวข้องกับการประมาณค่าความพยายามโดยใช้ฟังก์ชันพอยต์

System Characteristic	Degree Influence	System Characteristic	Degree Influence
1. Data Communications	…..	8. Online Update	…..
2. Distributed Data Processing	…..	9. Complex Processing	…..
3. Performance	…..	10. Reusability	…..
4. Heavily Used Configuration	…..	11. Installation Ease	…..
5. Transaction Rate	…..	12. Operational Ease	…..
6. Online Data Entry	…..	13. Multiple Sites	…..
7. End-User Efficiency	…..	14. Facilitate Change	…..
TDI	Total degree of influence		…..

เมื่อได้ค่ารวมทั้งหมดของปัจจัยที่เกี่ยวข้องกับการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ (TDI) จากตารางที่ 2.6 แล้วจะนำมาแปลงเป็นค่าตัวแปรปรับค่า (VAF) ตามด้วยสูตร

$$VAF = (TDI * 0.01) + 0.65$$

ขั้นตอนที่ 7 คำนวณค่ารวมฟังก์ชันพอยต์ที่ได้ปรับค่าแล้ว (Calculate Adjusted Function Point Count)

หลังจากที่ได้ค่าฟังก์ชันพอยต์ที่ยังไม่ได้ปรับค่า (UFP) ในขั้นตอนที่ 5 และค่าตัวแปรปรับค่า (VAF) ในขั้นตอนที่ 6 นำมาคูณเพื่อหาฟังก์ชันพอยต์ที่ได้ปรับค่า

$$FP = VAF * UFP$$

จะสังเกตเห็นได้ว่าค่าฟังก์ชันพอยต์ก่อนปรับค่าจะสามารถปรับเปลี่ยนตามลักษณะของซอฟต์แวร์ได้มากถึง +/- 35%

วิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์ เป็นการประมาณขนาดซอฟต์แวร์จากการนับฟังก์ชันการทำงานของซอฟต์แวร์ โดยพบว่าการประมาณด้วยฟังก์ชันพอยต์เป็นวิธีการที่ผู้ใช้งานกล่าวว่าที่มีประโยชน์และสามารถนำมาใช้จริงได้มากกว่าวิธีการคำนวณขนาด

ด้วยวิธีการอื่นๆ (Peixoto et al., 2010) วิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังชันพอยต์ยังเป็นวิธีการได้รับการยอมรับจากผู้บริหารระดับสูง (Jorgensen & Ostvold, 2004)

ในงานวิจัยปี ค.ศ. 1997 Fetcke และคณะ ได้ทดลองประมาณขนาดซอฟต์แวร์เชิงวัตถุด้วยวิธีการฟังก์ชันพอยต์ พบว่าสามารถประมาณขนาดของซอฟต์แวร์เชิงวัตถุได้ใกล้เคียงกับความเป็นจริง ทำให้ Fetcke และคณะ (1997) สามารถสรุปว่า วิธีการประมาณขนาดด้วยฟังก์ชันพอยต์สามารถใช้วัดซอฟต์แวร์ได้โดยไม่ขึ้นกับเทคโนโลยีและสามารถวัดซอฟต์แวร์เชิงวัตถุได้เช่นเดียวกัน (Fetcke et al., 1997) อีกทั้งวิธีการวัดซอฟต์แวร์ด้วยฟังก์ชันพอยต์สามารถนำมาต่อยอดเพื่อคำนวณหาขนาดด้วยวิธีการอื่นๆ จึงถือว่าการประมาณขนาดด้วยฟังก์ชันพอยต์เป็นวิธีที่ถูกนำมาปรับปรุงต่อยอดมากที่สุดอีกด้วย

2.1.3 การประมาณขนาดซอฟต์แวร์ด้วยยูสเคสพอยต์ (Use case point) (Karner, 1993)

Karner ได้นำเสนอการประมาณขนาดจากแผนภาพยูสเคส ในปี ค.ศ. 1993 โดยใช้ชื่อว่ายูสเคสพอยต์ ซึ่งเป็นการคำนวณขนาดซอฟต์แวร์จากการนับจำนวนยูสเคส (use case), แอกเตอร์ (actor) รวมไปถึงดูรายละเอียดของยูสเคส (Use case description) เพื่อดูความสัมพันธ์ระหว่างวัตถุ (Object) ระหว่างการติดต่อ (Transaction) ที่ประกอบอยู่ในลำดับเหตุการณ์ สำหรับวิธีการประมาณขนาดด้วยยูสเคสพอยต์ มีขั้นตอนดังนี้

ขั้นตอนที่ 1 ระบุและกำหนดค่าน้ำหนักในแก่แอกเตอร์ (actor) โดยแบ่งออกได้ดังนี้

แอกเตอร์อย่างง่าย (Simple actor) หมายถึง ระบบงานอื่นๆที่เข้ามาติดต่อผ่าน API โดยจะกำหนดค่าถ่วงน้ำหนักเท่ากับ 1 (weight 1)

แอกเตอร์แบบปกติ (Average actor) หมายถึง ระบบงานอื่นๆที่เข้ามาติดต่อผ่านโปรโตคอล เช่น TCP/IP หรือ หมายถึง ผู้ใช้งานที่เข้ามาติดต่อผ่านเข้าด้วยเชิงข้อความ (Text-based) จะกำหนดค่าถ่วงน้ำหนักเท่ากับ 2 (weight 2)

แอกเตอร์อย่างซับซ้อน (Complex factor) หมายถึง ผู้ใช้งานที่ติดต่อผ่านเข้ามาด้วยส่วนประสานต่อผู้ใช้งานกราฟิก (GUI) จะกำหนดค่าถ่วงน้ำหนักเท่ากับ 3 (weight 3)

ขั้นตอนที่ 2 ระบุและกำหนดน้ำหนักให้กับยูสเคสซึ่งเป็นการกำหนดน้ำหนักตามจำนวนการติดต่อ (transaction) ที่ประกอบอยู่ในลำดับเหตุการณ์การทำงานในแต่ละยูสเคสโดยจะนับจำนวนการติดต่อ (transaction) ในแต่ละชนิดคูณด้วยค่าถ่วงน้ำหนักปัจจัยน้ำหนัก (Weight factor) ตามตารางที่ 2.7

ตารางที่ 2.7 แสดงค่าน้ำหนักของการคำนวณขนาดยูสเคส

Use Case type	Description	Weight factor
Sample	3 or fewer transactions	5
Average	4-7 transactions	10
Complex	Greater than 7 transactions	15

ขั้นตอนที่ 3 คำนวณค่ารวมขนาดยูสเคสพอยต์ UCP จากค่าที่คำนวณได้จากการกำหนดน้ำหนักปัจจัยต่างๆ ตามสูตร

UCP = Weighted actor + Weighted Use Case

ในการศึกษาวิธีการประมาณขนาดด้วยยูสเคสพอยต์ พบว่า ขนาดที่วัดได้จะขึ้นอยู่กับผู้เชี่ยวชาญในการออกแบบรายละเอียดของยูสเคสซึ่งอาจจะไม่เหมือนกัน ทำให้ได้ผลลัพธ์ที่แตกต่างกัน ซึ่งทดสอบเปรียบเทียบความแม่นยำระหว่างการประมาณขนาดด้วยยูสเคสพอยต์และการประมาณขนาดในช่วงการออกแบบระบบแล้ว ยูสเคสพอยต์ให้ความแม่นยำที่น้อยกว่าอย่างมีนัยสำคัญ (Zivkovic, 2005)

2.1.4 **การประมาณขนาดซอฟต์แวร์ด้วยอ็อบเจกต์พอยต์** (Object point) (Boehm et al., 1995)

อ็อบเจกต์พอยต์เป็นอีกหนึ่งวิธีการวัดที่ถูกนำเสนอออกในปี ค.ศ. 1995 จากการต่อยอดของวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์ แต่จะมุ่งเน้นไปที่การวัดในไปที่เอกสารการออกแบบหน้าจอ ซึ่งวิธีการดังกล่าวให้ผลลัพธ์ที่ใกล้เคียงไม่แตกต่างจากฟังก์ชันพอยต์

2.1.5 **การประมาณขนาดซอฟต์แวร์ด้วยมาตรวัดเชิงวัตถุ** (Object-oriented metric) (Henderson-Sellers, 1996)

ไซส์2 (Size2) เป็นวิธีที่ถูกกำหนดขึ้นเพื่อประมาณขนาดของคลาส ซึ่งสามารถหาได้ในขั้นตอนการออกแบบระบบด้วยสมการดังนี้

Size2 = NOA + NEM

NOA คือ จำนวนลักษณะประจำ (Attribute) ทั้งหมด

NEM คือ จำนวนเมท็อดสาธารณะ (Number of External Method: NEM),

ตัววัดของซีเค เป็นอีกหนึ่งวิธีที่เสนอเพื่อใช้หาขนาดของคลาสมาจากมาตรวัดเชิงวัตถุ จากตัววัดของซีเคจำนวนทั้งหมด 6 ตัว จะมี 3 ตัวที่สามารถวัดขนาดได้ ได้แก่ ดับเบิ้ลยูเอ็มซี (WMC: Weight method per class) ดีไอที (DIT: Average depth of class in hierarchy tree) เอ็นโอซี (NOC: Average number of children per class) (Chidamber & Kemerer, 1994)

การประมาณขนาดด้วยมาตรวัดเชิงวัตถุนั้น จะสามารถประมาณขนาดได้ในระดับคลาส แต่ยากที่จะนำมาประมาณค่าความพยายามของระบบใหญ่หรือโครงการซอฟต์แวร์ ซึ่งตัววัดได้ถูกนำมาปรับปรุงเพื่อใช้ในวิธีการของคลาสพอยต์ และ ทำนายค่าอ็อบเจกต์พอยต์ (POP)

2.1.6 การประมาณขนาดซอฟต์แวร์ด้วยวิธีการทำนายค่าอ็อบเจกต์พอยต์ (Predictive object point: POP) (Minkiewicz, 1997)

ในปี ค.ศ. 1997 Minkiewicz ได้นำเสนอวิธีการทำนายค่าอ็อบเจกต์พอยต์เพื่อใช้ในการประมาณขนาดซอฟต์แวร์ ซึ่งเป็นวิธีการที่ใช้ตัวแปรหลักในการคำนวณจากมาตรวัดเชิงวัตถุ คือ ค่าเฉลี่ยน้ำหนักของเมท็อดต่อคลาส (Weight method per class: WMC) ซึ่งเป็นตัวหนึ่งในมาตรวัดของซีเค และปรับนำมาค่าด้วย ค่าเฉลี่ยความลึกของคลาสในลำดับชั้น (Average depth of class in hierarchy tree: DIT) ค่าเฉลี่ยจำนวนคลาสลูกต่อคลาส (Average number of children per class: NOC) และจำนวนคลาสลำดับชั้นบน (Number of top level class: TLC)

วิธีการทำนายค่าอ็อบเจกต์พอยต์จะให้ความสำคัญกับเมท็อด ซึ่งถูกแบ่งออกเป็น 5 ประเภท คือ

ตัวสร้าง (Constructors) โดยจะทำหน้าที่สร้างอ๊อบเจกต์

ตัวลบล้าง (Destructors) โดยจะทำหน้าที่ทำลายอ๊อบเจกต์

ตัวดัดแปลง (Modifiers) โดยจะทำหน้าที่เปลี่ยนแปลงสถานะอ๊อบเจกต์

ตัวเลือก (Selectors) โดยจะทำหน้าที่ในการแสดงข้อมูลของอ๊อบเจกต์ประกาศไว้เป็นส่วนตัว (Private)

ตัวทำซ้ำ (Iterators) โดยจะทำหน้าที่เป็นตัวซ้ำ ใช้เพื่อทำพฤติกรรมเดียวกันซ้ำในสมาชิกทุกตัวของอ๊อบเจกต์ที่เก็บไว้

โดยการให้ค่าความซับซ้อนของแต่ละเมท็อดโดยอ้างอิงจากจำนวนข้อความตอบสนองและจำนวนลักษณะประจำที่มีผลกระทบ ซึ่งจำให้วิธีการดังกล่าวต้องลงรายละเอียดให้ระดับลึกและไม่สามารถหาไม่ได้จริงในการออกแบบซอฟต์แวร์ในธุรกิจ

2.1.7 การประมาณขนาดซอฟต์แวร์ด้วยวิธีเมท็อดเชิงวัตถุฟังก์ชันพอยต์ (Object-oriented method function point: OOmFP) (Abrahao et al., 2004)

การประมาณขนาดซอฟต์แวร์เชิงวัตถุด้วยวิธีเมท็อดเชิงวัตถุฟังก์ชันพอยต์ เป็นอีกหนึ่งวิธีการที่อ้างอิงมาจากฟังก์ชันพอยต์ โดยอาศัยการคำนวณจาก 4 มุมมอง คือ

ข้อมูล (Data) ด้วยการวัดจาก อ็อบเจกต์โมเดล (Object model) เพื่อวัดตัวที่แสดงคลาสและความสัมพันธ์

กระบวนการ (Process) ด้วยการวัดจาก ฟังก์ชันโมเดล (Function model) ซึ่งเป็นการแสดงสถานการณ์เปลี่ยนแปลงของวัตถุ

พฤติกรรม (Behavior) ด้วยการวัดจาก ไดนามิกโมเดล (Dynamic model) ซึ่งเป็นการแสดงการติดต่อระหว่างวัตถุและวัตถุในระบบ

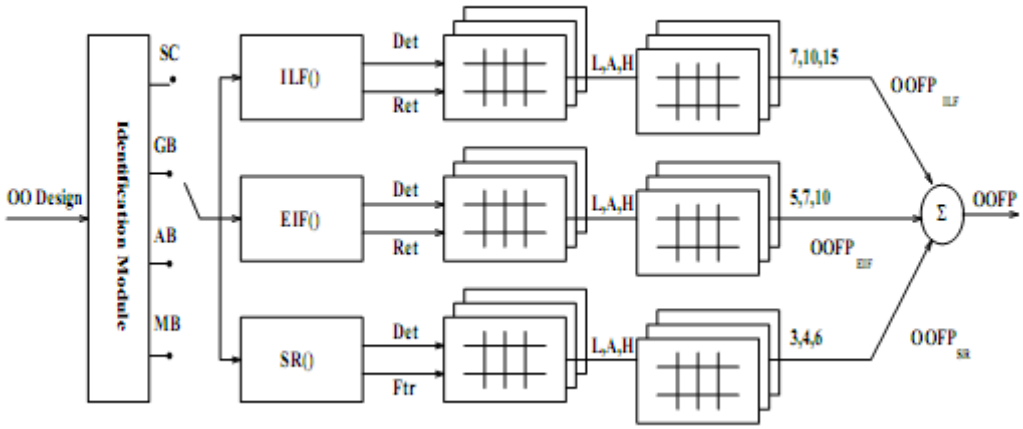
การแสดงผล (Presentation) ด้วยการวัดจาก พรีเซนเทชันโมเดล (Presentation model) ซึ่งเป็นการโต้ตอบระหว่างผู้ใช้งานกับระบบ

วิธีการประมาณซอฟต์แวร์เชิงวัตถุด้วยเมท็อดนั้น ได้ทดสอบแล้วว่าให้ผลที่ดีกว่าฟังก์ชันพอยต์ แต่การประมาณขนาดซอฟต์แวร์เมท็อดเชิงวัตถุนั้น เป็นไปได้ยาก เนื่องจากโมเดลในการคำนวณจำนวนถึงสี่โมเดล และไม่สามารถขาดโมเดลใดโมเดลหนึ่งได้ จึงยังไม่เป็นที่นิยมที่จะนำมาใช้จริง

2.1.8 การประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุ (Object-oriented function point: OOFP) (Antoniol et al., 1998)

Antoniol และคณะ (1998) ได้นำเสนอวิธีประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุ ซึ่งเป็นวิธีการที่ปรับเปลี่ยนจากการนับในมุมมองของผู้ใช้งานในช่วงการวิเคราะห์ระบบ คือ ฟังก์ชันพอยต์ มาเป็นการนับในมุมมองของผู้ออกแบบระบบ เพื่อให้สามารถนับได้กับการออกแบบเชิงวัตถุ (Object-oriented paradigm) ซึ่งสามารถปรับเปลี่ยนค่าได้เรื่อยๆเมื่อมีรายละเอียดของระบบมากขึ้น โดยมีขั้นตอนดังรูปที่ 2.3

หลักการของฟังก์ชันพอยต์เชิงวัตถุจะเป็นการจับคู่วิธีการของฟังก์ชันพอยต์เข้ากับแนวคิดของการพัฒนาเชิงวัตถุ กล่าวคือ จับคู่ ลอจิคอลไฟล์ (Logical file) เข้ากับคลาส (class) และ ทรานแซคชัน (transaction) กับ เมท็อด (method) กล่าวคือ กำหนดขอบเขตของระบบ ระบุ อินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) เอ็กเทอร์นอลอินเตอร์เฟสไฟล์ (External Interface Files: EIFs) และ เซอร์วิสรีเควส (Services request: SR) โดยกำหนดดังนี้



รูปที่ 2.3 แสดงขั้นตอนของวิธีการประมาณขนาดด้วย OOFP (Antoniol et al., 1998)

- อินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) คือ คลาส หรือ กลุ่มของคลาสที่อยู่ภายใต้ระบบ
- เอ็กเทอร์นอลอินเตอร์เฟสไฟล์ (External Interface Files: EIFs) คือ คลาส หรือกลุ่มของคลาสที่อยู่ภายนอกระบบหรือไลบรารี่ (Library) ที่พัฒนาขึ้นมาและระบบเรียกใช้

เซอร์วิสรีเควส (Services request: SR) คือ เมท็อด (Method) ที่อยู่ภายในระบบโดยมีขั้นตอนโดยละเอียดได้ ดังนี้

ขั้นตอนที่ 1 ระบุลอจิคอลไฟล์ (Logical file)

ขั้นตอนแรกของการนับด้วยฟังก์ชันพอยต์เชิงวัตถุ คือการระบุลอจิคอลไฟล์ในของระบบที่ต้องการจะนับ โดยสามารถกำหนดลอจิคอลไฟล์ (Logical file) ได้ดังนี้

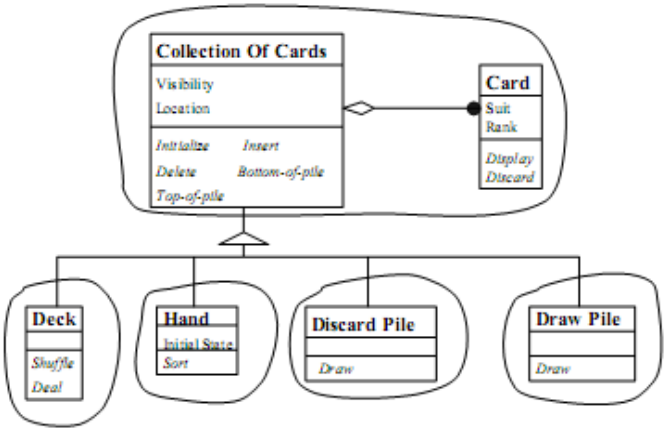
การนับแยกคลาสเดี่ยว (Single class) เป็นการนับลอจิคอลไฟล์ (Logical file) แยกระหว่างคลาสโดยทุกคลาสนับแยกกันไม่ขึ้นแก่กัน

การนับรวมคลาสการรวมกลุ่ม (Aggregation) นับกลุ่มของคลาสที่มีการรวมกลุ่มกัน (Aggregation) รวมเป็น 1 ลอจิคอลไฟล์ (Logical file) ดังรูปที่ 2.4

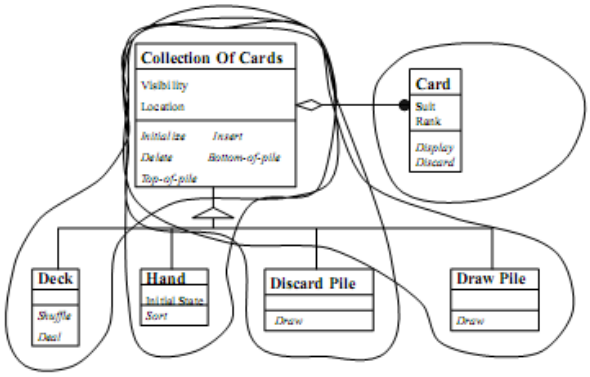
การนับแบบรวมคลาสที่มีการสืบทอดและการถ่ายทอด (Generalization/Specialization) รวมกลุ่มของคลาสที่มีการรับทอด (inheritance) โดยรวมระหว่างคลาสแม่กับคลาสลูก รวมเป็น 1 ลอจิคอลไฟล์ (Logical file) โดยจะมีส่วนทับซ้อนกัน ดังรูปที่ 2.5

การนับแบบผสม (Mixed) เป็นการนับแบบรวมทั้ง 2 วิธี คือ การนับรวมคลาสการรวมกลุ่ม (Aggregation) และการนับรวมคลาสที่มีการสืบทอดและถ่ายทอดไว้ด้วยกัน รวมเป็น 1 ลอจิคอลไฟล์ (Logical file) โดยจะมีส่วนทับซ้อนกันมาก ดังรูปที่ 2.6

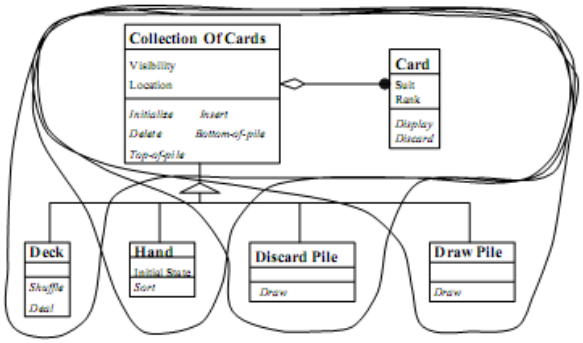
ทั้งนี้ในการเลือกวิธีการกำหนดลอจิคอลไฟล์ (Logical file) ขึ้นอยู่กับมุมมองและความสะดวกของผู้ออกแบบระบบ



รูปที่ 2.4 แสดงการรวมกลุ่มการคำนวณ LIF ด้วยวิธี aggregation (Antoniol et al., 1998)



รูปที่ 2.5 แสดงการรวมกลุ่มการคำนวณ LIF ด้วยวิธี Generalization/Specialization (Antoniol et al., 1998)



รูปที่ 2.6 แสดงการรวมกลุ่มการคำนวณ LIF ด้วยวิธี Mixed (Antoniol et al., 1998)

ขั้นตอนที่ 2 ประเมินระดับความซับซ้อนของลอจิคอลไฟล์ (Logical file)

จากวิธีการของฟังก์ชันพอยต์จะสามารถประเมินระดับความซับซ้อนของระบบได้จากจำนวนองค์ประกอบของข้อมูล (Data element types: DET) และ เรคคอร์ดข้อมูล (Record element types: RET) เช่นเดียวกันกับฟังก์ชันพอยต์เชิงวัตถุ โดยจะพิจารณาจำนวน DET และจำนวน RET จากลักษณะประจำ (Attribute) และความเชื่อมโยง (Association) ของคลาส โดยสามารถแสดงรายละเอียดได้ดังนี้

ลักษณะประจำที่ง่าย (Simple attribute) คือ ลักษณะประจำ (attribute) ที่มีชนิดง่ายๆ ที่ไม่ซับซ้อน เช่น ตัวเลข (Integer) สายอักขระ (String) ให้นับเป็น 1 DET

ลักษณะประจำที่ซับซ้อน (Complex attribute) คือ ลักษณะประจำ (attribute) ที่มีความซับซ้อน กล่าวคือ ประกอบด้วยกลุ่มของข้อมูล หรือ ประกอบด้วยวัตถุที่สามารถอ้างอิงต่อไปยังอีกหนึ่งคลาสได้ ให้นับเป็น 1 RET และพิจารณาจากความเชื่อมโยง (Association) ของแต่ละคลาส

ความเชื่อมโยงเพียง 1 (single-valued association) หมายถึง ความเชื่อมโยงกับคลาสอื่นซึ่งสามารถเกิดขึ้นได้เพียงแค่ 1 (Multiplicity =1) ให้นับเป็น 1 DET

ความเชื่อมโยงที่มากกว่า 1 (multiple-valued association) หมายถึง ความเชื่อมโยงกับคลาสอื่นที่สามารถเกิดขึ้นมากกว่า 1 (Multiplicity > 1) ให้นับเป็น 1 RET

*ยกเว้นการรวมกลุ่ม (Aggregation) ทุกตัวที่เป็นกลุ่มย่อยที่ถูกการรวมกลุ่ม (Aggregation) อยู่จะต้องบวกจำนวน RET อีก 1 ส่วนตัวแม่ของ Aggregate จะถูกเพิ่ม RET ตามประเภทของคลาสลูก

ประเมินค่าความซับซ้อนของแต่ละอินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) และ เอ็กเทอร์นอลอินเตอร์เฟสไฟล์ (External Interface Files: EIFs) จากจำนวน DET และ RET ที่นับได้ด้วยวิธีการเดียวกับวิธีของฟังก์ชันพอยต์ ซึ่งประเมินได้ตามตารางที่ 2.8

ตารางที่ 2.8 แสดงตารางเพื่อใช้คำนวณความซับซ้อนของลอจิคอลไฟล์ตามวิธี OOFP (IFPUG, 1999)

	1 to 19 DET	**20 to 50 DET**	**51 or more DET**
1 RET	Low	Low	Average
2 to 5 RET	Low	Average	High
6 or more RET	Average	High	High

ขั้นตอนที่ 3 ประเมินระดับความซับซ้อนเซอร์วิสรีเควส (Services request: SR)

ในขั้นตอนนี้การประมาณขนาดด้วยฟังก์ชันพอยต์จะต้องนับจำนวนทรานแซคชัน (Transaction) นั้นคือ อินเทอร์นอลอินพุท (External Input: EI) เอ็กซ์เทอร์นอลเอาท์พุท (External Output: EO) และเอ็กซ์เทอร์นอลอินไควรี (External Inquiry: EQ) แต่สำหรับซอฟต์แวร์เชิงวัตถุแล้วพิจารณาเพียงแค่เมท็อด (method) ว่าเป็น 1 อินเทอร์นอลอินพุท (External Input: EI) โดยที่เมท็อดที่อยู่ในแต่ละคลาสของระบบทั้งหมด ไม่รวมแอบสแตรกเมท็อด (abstract method) ส่วนของเมท็อด (method) ที่มีการรับทอด (Inheritance) จะนับเฉพาะส่วนที่ต้องมีการเขียนโปรแกรมเท่านั้น

หมายเหตุ เซอร์วิสรีเควส (Services request: SR) นี้ จะแตกต่างจากวิธีการประมาณขนาดซอฟต์แวร์ด้วยคลาสพอยต์ โดยในส่วนนี้ คือ บริการที่มีอยู่ในคลาส แต่ เซอร์วิสรีเควส (Services request) ในคลาสพอยต์ จะหมายถึง บริการที่คลาสนั้นร้องขอจากคลาสอื่น ซึ่งจะดูจำนวนองค์ประกอบของข้อมูล (Data element types: DET) ประเภทของไฟล์ข้อมูลอ้างอิง (File Type Referenced: FTR) จาก อาร์กิวเมนต์ (argument) ที่ประกอบอยู่ในเมท็อด ดังนี้

ไอเท็มที่ง่าย (Simple item) คือ อาร์กิวเมนต์ (argument) ที่มีชนิดไม่ซับซ้อน เช่น ตัวเลข (Integer) สายอักขระ (String) ที่ประกอบอยู่ในเมท็อด ให้นับเป็น 1 DET

ไอเท็มซับซ้อน (Complex item) คือ อาร์กิวเมนต์ (argument) ที่ซับซ้อน หรือชนิดที่ผู้ใช้งานระบุขึ้นเอง (user-define type) หรือการส่งวัตถุ (Object) ให้นับเป็น 1 FTR

*ในกรณีที่ยังไม่สามารถทราบ อาร์กิวเมนต์ (argument) ในเมท็อด Antoniol และคณะ (1998) ระบุว่าจะสามารถได้จะสามารถให้ระดับความซับซ้อนเป็นปานกลาง (Average) และ

นำมาหาค่าความซับซ้อนจากวิธีของฟังก์ชันพอยต์ โดยให้กำหนดค่าความซับซ้อนจากอินเทอร์นอลอินพุท (External Input: EI) ของฟังก์ชันพอยต์ตามตารางที่ 2.9

ตารางที่ 2.9 แสดงตารางเพื่อใช้คำนวณความซับซ้อนของเซอร์วิสรีเควส (Services request: SR) ในวิธีของฟังก์ชันพอยต์เชิงวัตถุ (IFPUG: 1999)

	1 to 4 DET	**5 to 15 DET**	**16 or more DET**
0 to 1 FTR	Low	Low	Average
2 FTRs	Low	Average	High
3 or more FTRs	Average	High	High

ขั้นตอนที่ 4 คำนวณขนาดรวมของทั้งระบบ

หลังจากที่ได้ระดับความซับซ้อนของแต่ละ อินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) เอ็กเทอร์นอลอินเตอร์เฟสไฟล์ (External Interface Files: EIFs) และ เซอร์วิสรีเควส (Services request: SR) ของทั้งระบบแล้ว นำมารวมค่าด้วยค่าน้ำหนักเดียวกับวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์ ดังตารางที่ 2.10

ตารางที่ 2.10 แสดงตารางเพื่อใช้คำนวณขนาดรวมของฟังก์ชันพอยต์เชิงวัตถุ

Function Type		Functional Complexity		Complexity Totals	Function Type Totals
ILFs	____	Low	X 7 =	____	
	____	Average	X 10 =	____	
	____	High	X 15 =	____	
					_____
EIFs	____	Low	X 5 =	____	
	____	Average	X 7 =	____	
	____	High	X 10 =	____	
					_____
SR	____	Low	X 3 =	____	
	____	Average	X 4 =	____	
	____	High	X 6 =	____	
					_____
Total OOFP					_____

ตัวอย่างการนับฟังก์ชันพอยต์เชิงวัตถุ (OOFP)

ขั้นตอนที่ 1 เลือกวิธีการระบุลอจิคอลไฟล์ (Logical file) โดยจะนับแบบแยกกันหมดได้ดัง รูปที่ 2.7

ขั้นตอนที่ 2 คำนวณจำนวน DET RET ดังรูปที่ 2.8 จากแต่ละลองจิคอลไฟล์ ซึ่งใช้วิธีการคำนวณโดยแยกคลาส สามารถแจกแจงได้ดังนี้

จากคลาส Collection of card

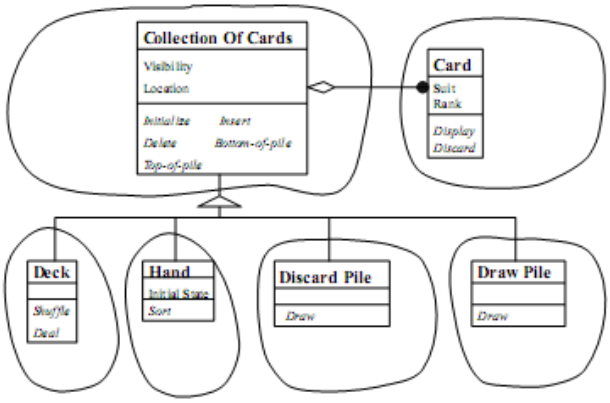
- DET 2 มาจากลักษณะประจำ 2 ตัว คือ Visibility และ Location
- RET 2 มาจากกลุ่มของกลุ่มย่อย (subgroup) และอีกหนึ่ง RET มาจากการรวมกลุ่ม (Aggregation) ซึ่งมีจำนวน card 1 ตัว

จากคลาส Deck

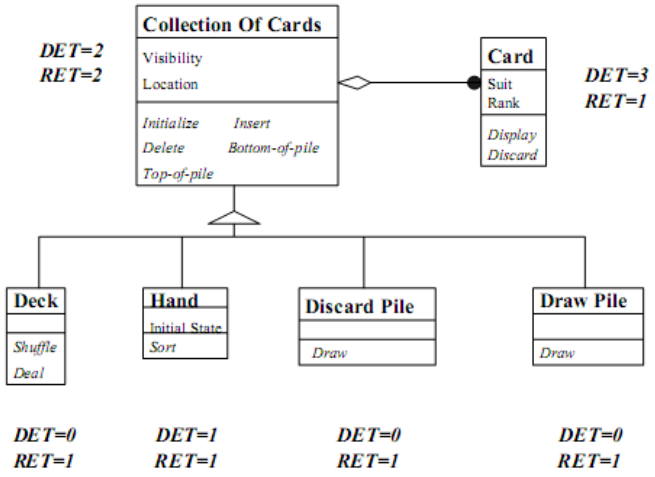
- DET 0 เนื่องจาก ไม่มีลักษณะประจำ attribute
- RET 1 มาจากความเชื่อมโยง (association)

จากคลาส Card

- DET 3 มาจากลักษณะประจำจำนวน 2 DET และ ความเชื่อมโยง (association) many-1 อีก 1 DET
- RET 1 มาจากการเก็บกลุ่มของการรวมกลุ่ม (Aggregation)



รูปที่ 2.7 แสดงตัวอย่างวิธีการนับ OOFP (Antoniol et al., 1998)



รูปที่ 2.8 แสดงจำนวนการนับด้วย OOFP (Antoniol et al., 1998)

ขั้นตอนที่ 3 ประเมินระดับความซับซ้อนของทรานแซคชัน (Transaction)

ฟังก์ชันพอยต์เชิงวัตถุ คือ ทรานแซคชัน หรือ เมท็อด (Method) ในระบบตัวอย่างจะมีทั้งหมดมี 12 เมท็อด (method) ซึ่งไม่สามารถรายละเอียด Antoniol และคณะจึงเสนอให้ใช้ระดับความซับซ้อนเป็นปกติ (average)

ขั้นตอนที่ 4 รวมขนาดรวมฟังก์ชันพอยต์เชิงวัตถุ (OOFP)

หลังจากที่ได้ระดับความซับซ้อนของแต่ละลอจิคอลไฟล์แล้ว จะรวมขนาดของฟังก์ชันพอยต์เชิงวัตถุได้ดังตารางที่ 2.11 ซึ่งพบว่าขนาดของระบบดังตัวอย่างมีขนาดฟังก์ชันพอยต์เชิงวัตถุเท่ากับ 90

ตารางที่ 2.11 แสดงการคำนวณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุ

Function Type		Functional Complexity		Complexity Totals	Function Type Totals
ILFs	__6_	Low	X 7 =	_42_	
	____	Average	X 10 =	____	
	____	High	X 15 =	____	__42___
EIFs	____	Low	X 5 =	____	
	____	Average	X 7 =	____	
	____	High	X 10 =	____	_____
SR	____	Low	X 3 =	____	
	__12_	Average	X 4 =	_48__	
	____	High	X 6 =	____	___48__
Total OOFP					___90__

Antoniol และคณะ (2003) ได้ทดสอบฟังก์ชันพอยต์เชิงวัตถุเพื่อขนาดของซอฟต์แวร์ด้วยจำนวนบรรทัดของโปรแกรม ในปี ค.ศ. 2003 พบว่า ขนาดที่ได้จากวิธีการประมาณฟังก์ชันพอยต์เชิงวัตถุมีความสัมพันธ์กับจำนวนบรรทัดของโปรแกรมและสามารถนำมาทำนายค่าความพยายามได้ (Antoniol et al., 2003)

2.1.9 การประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุสอง (OOFP2) (Zivkovic et al., 2005)

ในปี ค.ศ. 2005 Zivkovic และคณะ นำวิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุ (OOFP) ของ Antoniol และคณะในปี ค.ศ. 1998 มาปรับปรุง เนื่องจากพบว่าการวัดความซับซ้อนการนับ ทรานแซคชัน (Transaction) สำหรับซอฟต์แวร์เชิงวัตถุ จำนวน

พารามิเตอร์ต่อเมท็อดมักจะต่ำกว่าการวัดทั่วไป จึงได้กำหนดเปลี่ยนแปลงไปจากมาตรฐานเดิม คือ ฟังก์ชันพอยต์ ซึ่งได้ปรับเปลี่ยนไป ดังตารางที่ 2.12

ตารางที่ 2.12 แสดงการคำนวณความซับซ้อนของ Transactional function ด้วยวิธีการ OOFP2 (Zivkovic et al., 2005)

	0–4 (1–5) DET	5–10 (6–19) DET	More than 10 (20 or more) DET
0 or 1 FTR	Low (low)	Average (low)	High (average)
2 (2–3) FTR	Average (low)	Average	High
3 or more (4 or more) FTR	Average	High	High

วิธีการประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุสอง (OOFP2) ยังทดสอบแล้วว่า ให้ความแม่นยำที่ดีกว่าวิธีการประมาณค่าความพยายามด้วยยูสเคสพอยต์อีกด้วย (Zivkovic et al., 2005) แต่ยังไม่มีงานพิสูจน์เพื่อเทียบกับวิธีเก่าฟังก์ชันพอยต์เชิงวัตถุ ว่าวิธีการใดให้ความแม่นยำมากกว่า

2.1.10 การประมาณขนาดซอฟต์แวร์ด้วยคลาสพอยต์ (Class point: CP) (Costagliola et al., 2005)

ในปี ค.ศ. 2005 Costagliola และคณะได้นำเสนอวิธีการวัดซอฟต์แวร์ด้วยแผนภาพคลาส (Class diagram) เพื่อใช้วัดขนาดของระบบในช่วงการออกแบบระบบ เช่นเดียวกับฟังก์ชันพอยต์เชิงวัตถุ วิธีการประมาณขนาดด้วยคลาสพอยต์จะใช้แนวคิดของฟังก์ชันพอยต์และมาตรวัดเชิงวัตถุ (Object-oriented metric) โดยพิจารณาที่คลาสซึ่งประกอบด้วย เอนทีตี (Entities) เมท็อด (Method) และลักษณะประจำ (Attribute) ในการประเมินระดับความซับซ้อนของแต่ละคลาสและนำมาคำนวณเพื่อหาขนาดของซอฟต์แวร์

วิธีการประมาณขนาดด้วยคลาสพอยต์นำเสนอวิธีการประมาณซอฟต์แวร์ด้วยสองวิธีคือ CP1 และ CP2 วิธี CP1 สามารถใช้ได้ในขั้นตอนช่วงแรกๆของการพัฒนา (analysis เสร็จสิ้นและเริ่ม design) วิธี CP2 สามารถใช้ได้ช่วงที่มีรายละเอียดเพิ่มขึ้นของช่วงออกแบบ (design phase) ประกอบด้วย 4 ขั้นตอนหลักๆ โดยมีขั้นตอนดังนี้

ขั้นตอนที่ 1 ระบุและจำแนกประเภทของคลาส (Identification and classification of user classes)

จากการออกแบบแผนภาพของระบบในซอฟต์แวร์เชิงวัตถุ Coad และ Nicola ได้จำแนกประเภทของคลาสจากหน้าที่ของคลาส ว่าคลาสทุกคลาสจะสามารถจำแนกได้ออกเป็น 4 ประเภท (Coad & Nicola, 1993) ดังนี้

1. ประเภทขอบเขตของปัญหา (Problem domain type: PDT) คลาสนี้จะทำหน้าที่เป็นส่วนที่แสดงสิ่งที่มีตัวตนอยู่จริง เช่น Incident, FieldOfficer และ EvergencyReport
2. ประเภทโต้ตอบกับผู้ใช้งาน (Human Interaction type: HIT) คลาสนี้จะทำหน้าที่แสดงหรือโต้ตอบกับผู้ใช้งาน เช่น EmergencyReportForm และ ReportEmergencyButton
3. ประเภทการจัดการข้อมูล (Data Management type: DMT) คลาสนี้จะทำหน้าที่เรียกใช้และจัดการกับฐานข้อมูล เช่น IncidentManagement
4. ประเภทการจัดการงาน (Task Management type: TMT) คลาสนี้จะทำหน้าที่เป็นตัวควบคุม (Controller) ติดต่อระหว่างคลาสแต่ละคลาสในระบบ เช่น EmergencyControl และ ReportEmergencyControl

ขั้นตอนที่ 2 ประเมินระดับความซับซ้อนของคลาส (Evaluation of a class complexity level)

หลังจากที่ได้ระบุประเภทของคลาสแล้ว จะนำคลาสเหล่านั้นมาหาประเมินระดับความซับซ้อน Costagilia และคณะได้ได้วัดระดับความซับซ้อนจาก

1. จำนวนเมท็อดสาธารณะ (Number of External Method: NEM) คือ เมท็อดที่ถูกประกาศไว้ในคลาสชนิดพับบลิค (Public) กล่าวคือ มีชุดคำสั่งอยู่ภายในเมท็อดและเมท็อดนั้นสามารถถูกเรียกใช้จากคลาสอื่นได้
2. จำนวนการเรียกใช้บริการจากภายนอก (Number of Services Requested: NSR) คือ จำนวนคำขอร้องที่เรียกใช้บริการจากคลาสอื่น
3. จำนวนลักษณะประจำ (Number of Attribute: NOA) คือ จำนวนลักษณะประจำ (Attribute) รวมทั้งหมดของแต่ละคลาสไม่แยกประเภท

การประมาณขนาดของ CP1 จะวัดจาก จำนวนเมท็อดสาธารณะ (Number of External Method: NEM) และ จำนวนการเรียกใช้บริการจากภายนอก (Number of Services

Requested: NSR) จะหาได้จากเอกสารการออกแบบระบบ (design document) ในช่วงวิเคราะห์ระบบเสร็จ ความซับซ้อนของคลาสใน CP1 จะสามารถหาได้ตามตารางที่ 2.13

การประมาณขนาดของ CP2 จะวัดจาก NEM NSR และ NOA เมื่อโครงการมีรายละเอียดเพิ่มมากขึ้น ความซับซ้อนของคลาสใน CP2 จะสามารถหาได้ตามตารางที่ 2.14

ตารางที่ 2.13 แสดงวิธีการประเมินระดับความซับซ้อนของ CP1 (Costagliola et al., 2005)

	0 - 4 *NEM*	**5 - 8 *NEM***	**≥ 9 *NEM***
0 - 1 *NSR*	Low	Low	Average
2 - 3 *NSR*	Low	Average	High
≥ 4 *NSR*	Average	High	High

ตารางที่ 2.14 แสดงวิธีการประเมินระดับความซับซ้อนของ CP2 (Costagliola et al., 2005)

0 - 2 *NSR*	**0 - 5 *NOA***	**6 - 9 *NOA***	**≥ 10 *NOA***
0 - 4 *NEM*	Low	Low	Average
5 - 8 *NEM*	Low	Average	High
≥ 9 *NEM*	Average	High	High

(a)

3 - 4 *NSR*	**0 - 4 *NOA***	**5 - 8 *NOA***	**≥ 9 *NOA***
0 - 3 *NEM*	**Low**	**Low**	**Average**
4 - 7 *NEM*	**Low**	**Average**	**High**
≥ 8 *NEM*	**Average**	**High**	**High**

(b)

≥ 5 *NSR*	**0 - 3 *NOA***	**4 - 7 *NOA***	**≥ 8 *NOA***
0 - 2 *NEM*	**Low**	**Low**	**Average**
3 - 6 *NEM*	**Low**	**Average**	**High**
≥ 7 *NEM*	**Average**	**High**	**High**

ขั้นตอนที่ 3 คำนวณค่าคลาสพอยต์ที่ยังไม่ได้ปรับค่า (Estimating the total unadjusted class point) **นำค่าทั้งหมดมารวมเพิ่มคำนวณค่า** TUCP

หลังจากที่ได้ระดับความซับซ้อนของแต่ละคลาสและประเภทของคลาสแล้ว จะรวมค่าทั้งหมด ด้วยการคูณค่าน้ำหนักตามระดับความซับซ้อนของคลาส ซึ่ง Costagliola และคณะได้กำหนดขึ้นมาใหม่ ดังตารางที่ 2.15 จะได้ค่ารวมขนาดของคลาสพอยต์แต่ยังเป็นค่าที่ยังไม่ได้ถูกปรับด้วยปัจจัยความซับซ้อนทางเทคนิค

ตารางที่ 2.15 แสดงตารางการคำนวณค่าคลาสพอยต์

Class Type		Class Complexity		Class Complexity Totals	Class Totals
PDT	____	Low	X 3 =	____	
	____	Average	X 6 =	____	
	____	High	X 10 =	____	_____
HIT	____	Low	X 4 =	____	
	____	Average	X 7 =	____	
	____	High	X 12 =	____	_____
DMT	____	Low	X 5 =	____	
	____	Average	X 8 =	____	
	____	High	X 13 =	____	_____
TMT	____	Low	X 4 =	____	
	____	Average	X 6 =	____	
	____	High	X 9 =	____	_____
Total Unadjusted Class Point (TUCP)					_____

ขั้นตอนที่ 4 คำนวณค่าปัจจัยความซับซ้อนทางเทคนิค

ค่าปัจจัยความซับซ้อนทางเทคนิค (Technical complexity factor: TCF) ของคลาสพอยต์จะนำมาจากวิธีฟังก์ชันพอยต์โดยการเพิ่มปัจจัยขึ้น 4 ปัจจัย รวมเป็น 18 ปัจจัย ดังตารางที่ 2.16 ซึ่งเป็นปัจจัยที่มีผลกระทบกับซอฟต์แวร์เชิงวัตถุ โดยมีค่าตั้งแต่ 0-5 ตามระดับความมีอิทธิพล โดยนำมากรอกตามตารางที่ 2.17

ตารางที่ 2.16 แสดงค่าปัจจัยสี่ตัวที่เพิ่มเติมจากฟังก์ชันพอยต์เพื่อใช้วัดซอฟต์แวร์เชิงวัตถุ (Costagliola et al., 2005)

1. Adaptivity	
0	ไม่มีการปรับเปลี่ยนใดๆ
1	ระบบสามารถเปลี่ยนรูปแบบของข้อมูลออก (Output) เมื่อเปลี่ยนข้อมูลนำเข้า (Input) แต่สามารถทำได้ในพฤติกรรมที่จำกัดเพียงบางรูปแบบ
2	ระบบจำเป็นจะต้องแสดงข้อความบันทึก ซึ่งยอมให้โต้ตอบเป็นลำดับในหลายๆ input
3	ระบบจำเป็นต้องมีการบันทึกข้อความ ซึ่งปรับเปลี่ยนไปถึงข้อมูลในอดีต
4	ระบบสามารถจัดการการเปลี่ยนแปลงผลกระทบจากการเปลี่ยนและประเมิน output จากการทดลองและข้อผิดพลาดที่เกิดขึ้นจากการใส่ input นั้นๆ
5	ระบบจะต้องสามารถอนุมารกลไกลที่จะโต้ตอบได้ จากการแปลความหมายของลักษณะการใช้งานของผู้ใช้งาน

2. Rapid prototyping	
0	ไม่มีการจัดทำ prototype ใดๆ
1	มีการจัดทำ prototype ในรูปแบบกระดาษ
2	มีการจัดทำ prototype ในรูปแบบการออกแบบหน้าจอซึ่งสามารถแสดงการออกแบบในบางเหตุการณ์ แต่ไม่มีส่วนของ functionality ใดๆ
3	มีการจัดทำ prototype ในรูปแบบการออกแบบหน้าจอ ซึ่งสามารถแสดงการออกแบบในเกือบทั้งหมดของเหตุการณ์ แต่ไม่มีการแสดงส่วน functionality หรือการทดสอบระบบ
4	มีการออกแบบ prototype การทำงานของฟังก์ชัน functionality และเพิ่มการโต้ตอบเมื่อพิมพ์
5	มีการออกแบบ prototype การทำงานของฟังก์ชัน functional มีการโต้ตอบระบบจากผู้ใช้งาน รวมการเซ็นยอมรับของผู้ใช้งานในแต่ละเวอร์ชันของ prototyping

ตารางที่ 2.16 (ต่อ) แสดงค่าปัจจัยสี่ตัวที่เพิ่มเติมจากฟังก์ชันพอยต์เพื่อใช้วัดซอฟต์แวร์เชิงวัตถุ (Costagliola et al., 2005)

3. Multiple interfaces	
0-5	การให้คะแนนอ้างอิงจาก มีการปรับเปลี่ยนได้ในตัวอักษร ในกรณีที่ผู้ใช้งานเข้ามาต่างกันจะแสดงชื่อที่แตกต่าง แต่โครงสร้างส่วนใหญ่ไม่เปลี่ยนแปลง มีความแตกต่างกันระหว่างผู้ใช้งานที่มีประสบการณ์กับผู้เริ่มต้นใช้งาน มีความแตกต่างกันระหว่างประเภทของผู้ใช้งาน การศึกษาและพื้นฐานการทำงาน การปรับเปลี่ยนหน้าจอขึ้นกับความสามารถในการใช้งาน มีความแตกต่างระหว่างกานโต้ตอบกับ platform ขึ้นอยู่กับการพิจารณา

4. Multiuser interactivity	
0-5	การให้คะแนนอ้างอิงจาก ระบบต้องติดตั้งในหลายๆสถานที่การทำงาน ต้องมีการพิจารณา hardware และ environment ระบบต้องสามารถควบคุมแบบง่ายโดยการรับคำสั่งระหว่างผู้ใช้งาน ระบบต้องสามารถควบคุมการโต้ตอบของแต่ละผู้ใช้งานในเวลาเดียวกันได้ ระบบต้องสามารถจัดการได้อย่างฉลาด sophisticated synchronization และสามารถควบคุมคำสั่งและจัดการการโต้ตอบได้ในเวลาเดียวกันระหว่างหลายๆผู้ใช้งาน

ตารางที่ 2.17 แสดงปัจจัยที่มีอิทธิพลกับขนาดของซอฟต์แวร์จากวิธีการประมาณขนาดด้วยคลาสพอยต์ (Costagliola et al., 2005)

ID	System Characteristic	DI	ID	System Characteristic	DI
C1	Data Communication	...	C10	Reusability	...
C2	Distributed Functions	...	C11	Installation ease	...
C3	Performance	...	C12	Operational ease	...
C4	Heavily used configuration	...	C13	Multiple sites	...
C5	Transaction rate	...	C14	Facilitation of change	...
C6	Online data entry	...	C15	User Adaptivity	...
C7	End-user efficiency	...	C16	Rapid Prototyping	...
C8	Online update	...	C17	Multiuser Interactivity	...
C9	Complex processing	...	C18	Multiple Interfaces	...
TDI				Total Degree of Influence	...

รวมค่าปัจจัยที่มีอิทธิพลกับขนาดของซอฟต์แวร์จากวิธีการประมาณขนาดด้วยคลาสพอยต์ (Technical degree of influence: TDI) แล้วมาคำนวณหาค่าตัวแปรปรับค่าความซับซ้อนทางเทคนิค (Technical complexity factor: TCF) ด้วยสูตร

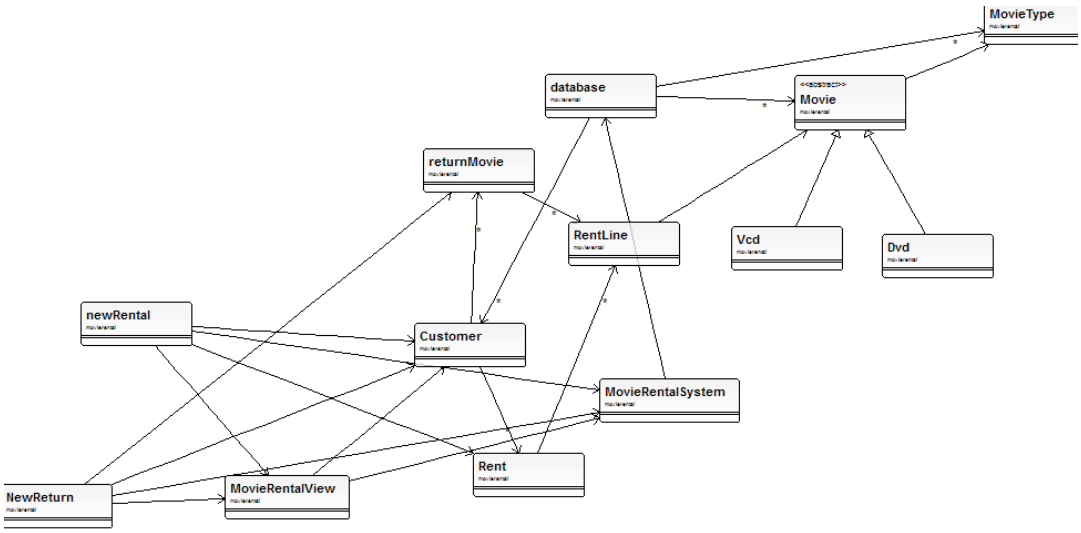
TCF = 0.55+ (0.01*TDI)

คำนวณค่ารวมทั้งหมดเพื่อหาค่าขนาดของคลาสพอยต์โดยการนำขนาดที่ยังไม่ได้ปรับค่าคูณกับตัวแปรปรับค่า

CP = TUCP * TCF

ตัวแปรปรับค่าของเทคนิคของคลาสพอยต์จะสามารถปรับเปลี่ยนขนาดได้ +/-45%

ตัวอย่างการนับค่า Class point



รูปที่ 2.9 แสดงตัวอย่างของแผนภาพคลาส (Class diagram) ของระบบเช่าภาพยนตร์ที่ได้จากการวิศวกรรมย้อนกลับจากซอร์สโค้ด ด้วย JavaDoc As UML

ขั้นตอนที่ 1 ระบุประเภทของคลาส

จากตัวอย่างดังรูปที่ 2.9 ของระบบเช่าภาพยนตร์สามารถแบ่งประเภทของคลาส ได้ดังตารางที่ 2.18

ตารางที่ 2.18 แสดงการแบ่งประเภทของคลาสจากตัวอย่างระบบเช่าภาพยนตร์

ประเภท	Class
Problem domain type (PDT)	NewReturn newRental Customer returnMovie RentLine Vcd Dvd Movie
Human interaction type (HIT)	MovieRentalView
Data management type (DMT)	database
Task management type (TMT)	MovieRentalSystem

ขั้นตอนที่ 2 ประเมินค่าความซับซ้อนของคลาส

นับจำนวนเมท็อดสาธารณะ (Number of External Method: NEM) จำนวนลักษณะประจำ (Number Of Attribute: NOA) และ จำนวนการเรียกใช้บริการจากภายนอก (Number of Services Requested: NSR)

รูปที่ 2.10 แสดงรายละเอียดของคลาสภาพยนตร์

จากตัวอย่างระบบเช่าภาพยนตร์ สามารถแบ่งค่าความซับซ้อนของคลาสได้ดังตารางที่ 2.19 เช่น คลาสหนัง ดังรูปที่ 2.10 นับจำนวนเมท็อดสาธารณะ 18 เมท็อด จำนวนลักษณะประจำ (Attribute) 9 ลักษณะประจำ และจำนวนบริการที่ร้องขอ (Services request) คือ 1 บริการร้องขอ ดังนั้นคลาสภาพยนตร์จะมีความซับซ้อนในการคำนวณ CP1 และ CP2 จะเท่ากับสูง

ตารางที่ 2.19 แสดงตารางการแบ่งค่าความซับซ้อนของคลาส

Type	Class	NSR	NOA	NEM	Complexity	
					CP1	CP2
Problem domain type (PDT)	Customer	2	11	36	High	High
	returnMovie	1	4	8	Average	Low
	RentLine	0	4	14	Average	Average
	Vcd	0	0	6	Average	Low
	Dvd	0	0	6	Average	Low
	Movie	1	9	18	High	High
	MovieType	0	3	7	Average	Low
Human interaction type (HIT)	MovieRentalView	2	2	12	High	Average
	NewReturn	4	7	4	Average	Average
	newRental	4	7	4	Average	Average
Data management type (DMT)	database	3	8	23	High	High
Task management type (TMT)	MovieRentalSystem	1	1	15	Average	Average

ขั้นตอนที่ 3 คำนวณค่าคลาสพอยต์ที่ยังไม่ได้ปรับค่า

หลังจากที่ได้ระดับความซับซ้อนของแต่ละคลาสของระบบเช่าภาพยนตร์แล้ว จะหาขนาดของระบบ โดย CP1 จะสามารถคำนวณได้ดังตารางที่ 2.20 และ CP2 จะสามารถคำนวณตามตารางที่ 2.21 แต่ยังเป็นค่าที่ยังไม่อ้างอิงกับปัจจัยอื่นๆ

ตารางที่ 2.20 แสดงตารางการคำนวณค่าคลาสพอยต์ CP1

Class Type		Class Complexity		Class Complexity Totals	Class Totals
PDT	____	Low	X 3 =	____	
	_5__	Average	X 6 =	_30_	
	_2__	High	X 10 =	_20_	__50___
HIT	____	Low	X 4 =	____	
	_2__	Average	X 7 =	_14_	
	_1__	High	X 12 =	_12_	__26___
DMT	____	Low	X 5 =	____	
	____	Average	X 8 =	____	
	_1__	High	X 13 =	____	__13___
TMT	____	Low	X 4 =	____	
	_1__	Average	X 6 =	____	
	____	High	X 9 =	____	__6____
Total Unadjusted Class Point (TUCP)					__78___

ตารางที่ 2.21 แสดงตารางการคำนวณค่าคลาสพอยต์ CP2

Class Type		Class Complexity		Class Complexity Totals	Class Totals
PDT	_4__	Low	X 3 =	__12__	
	_1__	Average	X 6 =	__6__	
			X 10		
	_2__	High	=	_20_	__38___
	____	Low	X 4 =	____	
HIT	_3__	Average	X 7 =	_21_	
			X 12		
	____	High	=	____	__21___
DMT	____	Low	X 5 =	____	
	____	Average	X 8 =	____	
			X 13		
	_1__	High	=	____	__13___
TMT	____	Low	X 4 =	____	
	_1__	Average	X 6 =	____	
	____	High	X 9 =	____	__6____
Total Unadjusted Class Point (TUCP)					__78___

ดังนั้นสำหรับระบบเช่าหนังจะมีขนาดคลาสพอยต์ด้วยวิธีคำนวณ CP1 = 95 และด้วยวิธีคำนวณ CP2 = 78

วิธีการประมาณขนาดซอฟต์แวร์ด้วยคลาสพอยต์มีงานวิจัยในการพิสูจน์ความแม่นยำใน Costagliola และคณะ ได้ทดลองกับในโครงการซอฟต์แวร์ของนักศึกษาในมหาวิทยาลัย จำนวน 40 โครงการ พบว่า การทำนายค่าความพยายาม ของ CP1 ให้คลาดเคลื่อนโดยเฉลี่ย 19 เปอร์เซ็นต์ ซึ่งอยู่ในขอบเขตที่สามารถยอมรับได้ คือ ค่าความคลาดเคลื่อนไม่เกิน 25% จำนวน

75 % และ CP2 คลาดเคลื่อนโดยเฉลี่ยประมาณ 18 เปอร์เซ็นต์ ซึ่งอยู่ในขอบเขตที่สามารถยอมรับได้ คือ ค่าความคลาดเคลื่อนไม่เกิน 25% จำนวน 83 %

วิธีการประมาณขนาดด้วยคลาสพอยต์นั้น ยังไม่พบงานวิจัยทดลองด้วยหน่วยตัวอย่างจริงเมื่อเทียบกับวิธีการอื่นๆ ดังนั้นในงานวิจัยนี้ ผู้วิจัยจึงเลือกวิธีการประมาณขนาดด้วยคลาสพอยต์โดยเลือก CP2 ซึ่งพบว่าให้ความแม่นยำที่ดีกว่า CP1 โดยเลือกวิธีการพื้นฐานในปี ค.ศ. 2005 ของ Costagliola และคณะ

2.1.11 การประมาณขนาดซอฟต์แวร์ด้วยวิธีแพทเทิร์นพอยต์ (Pattern point) (Adekile et al., 2010)

วิธีการนี้ถูกนำเสนอเพื่อประมาณขนาดของซอฟต์แวร์เชิงวัตถุในช่วงท้ายของการวิเคราะห์ระบบ วิธีการดังกล่าวใช้วิธีการเดียวกับวิธีการประมาณขนาดด้วยคลาสพอยต์ ในมุมมองของผู้ออกแบบว่าจะใช้แพทเทิร์นใดบ้างในทั้งหมดจำนวน 23 ตัว โดยดูความสัมพันธ์ระหว่างวัตถุ (Object) ใน แผนภาพซีเควนซ์ (sequence diagram) มี PP1 และ PP2 โดยวิธีการแบ่งออกเป็น 3 ขั้นตอน

ตารางที่ 2.22 แสดงขนาดของความซับซ้อนของแพทเทิร์นในแต่ละแพทเทิร์น (Adekile et al., 2010)

Type	Pattern	Objs.	Mesgs.	DD	Classes	Assoc.	SC	Group	Complexity
Creational	Abstract Factory	2	1	3	3	4	7	PDT	High
	Builder	3	4	7	3	4	7	PDT	High
	Factory Method	1	1	2	2	2	4	PDT	Low
	Prototype	2	1	3	2	1	3	PDT	Low
	Singleton	1	1	2	2	1	3	PDT	Low
Structural	Adapter	2	1	3	4	3	7	PDT	Average
	Bridge	2	2	4	5	5	10	PDT	High
	Composite	2	1	3	4	5	9	PDT	High
	Decorator	2	1	3	4	3	7	HIT	Average
	Façade	1	0	1	1	1	2	PDT	Low
	Flyweight	2	1	3	5	5	10	PDT	High
	Proxy	2	1	3	1	1	2	PDT	Low
Behavioral	Chain of Responsibility	2	1	3	3	3	6	TMT	Average
	Command	4	3	7	5	3	8	TMT	High
	Interpreter	3	1	4	4	4	8	TMT	High
	Iterator	1	1	2	3	2	5	DMT	Low
	Mediator	3	3	6	4	4	8	TMT	High
	Memento	3	4	7	3	2	5	DMT	High
	Observer	2	5	7	4	3	7	TMT	High
	State	2	3	5	3	2	5	TMT	Average
	Strategy	2	3	5	3	2	5	PDT	Average
	Template Method	1	2	3	2	1	3	PDT	Low
	Visitor	3	3	6	5	5	10	TMT	High
	Filter	2	1	3			2	PDT	Low
	Interface	1	0	1	2	1	3	PDT	Low

ขั้นตอนที่ 1 จำแนกประเภทของวัตถุ

การประมาณขนาดด้วยแพทเทิร์นพอยต์จะจำแนกประเภทของวัตถุเช่นเดียวด้วยวิธีการเดียวกับคลาสพอยต์ โดยแบ่งประเภทวัตถุออกเป็น 4 กลุ่ม คือ ประเภทขอบเขตของปัญหา (Problem domain type: PDT) ประเภทโต้ตอบกับผู้ใช้งาน (Human Interaction type: HIT) ประเภทการจัดการข้อมูล (Data Management type: DMT) และ ประเภทการจัดการงาน (Task Management type: TMT) ซึ่งถ้าเป็น 23 แพทเทิร์นจะสามารถจำแนกได้ดังตารางที่ 2.22

ขั้นตอนที่ 2 หาค่าความซับซ้อนของวัตถุ

ถ้าเป็น 23 แพทเทิร์นจะสามารถหาได้ตามตารางที่ 2.22 แต่ถ้าไม่มีดังตารางให้ผู้ออกแบบประเมินด้วยความเชี่ยวชาญจากระดับความยาก (Degree of difficult) และโครงสร้างที่ซับซ้อน (structural complexity) ตามตารางที่ 2.23 จะสังเกตว่าการนับจะไม่นับ concrete class ซึ่ง concrete class จะหาได้ในภายหลังของ analysis จนถึงต้น design ซึ่งจะใช้ concrete class มาคำนวณ PP2 ดังสมการ

PP2 = PP1+ pattern ของ concrete class

ตารางที่ 2.23 แสดงการแบ่งความซับซ้อนของแต่ละกลุ่มซึ่งอยู่นอกเหนือจากแพทเทิร์น (Adekile et al., 2010)

	0 – 4 SC	5 – 8 SC	>= 9 SC
0 – 2 DD	Low	Low	Average
3 – 5 DD	Low	Average	High
>= 6 DD	Average	High	High

ขั้นตอนที่ 3 คำนวณค่ารวมทั้งระบบ

รวมขนาดของทั้งระบบจากแพทเทิร์นที่วัดออกมาได้ และ วัตถุอื่นๆ นอกเหนือจากแพทเทิร์น โดยรวมค่าดังตารางที่ 2.24

ตารางที่ 2.24 แสดงการคำนวณค่ารวมของแพทเทิร์นพอยต์ (Adekile et al., 2010)

System Component Type	Description	Complexity
		LowAverageHighTotal
PDT	Problem Domain	...*3=.....*6=.....*10=.....
HIT	Human Interaction	...*4=.....*7=.....*12=.....
DMT	Data Management	...*5=.....*8=.....*13=.....
TMT	Task Management	...*4=.....*6=.....*9=.....
TUPP		Total Unadjusted Pattern Point

สำหรับวิธีการประมาณขนาดด้วยแพทเทิร์นพอยต์จะต้องมองรายละเอียดของแต่ละแพทเทิร์นให้ได้ ซึ่งการนับขึ้นอยู่กับความคิดเห็นของผู้ออกแบบระบบ และต้องมีความสามารถในการใช้ดีไซน์แพทเทิร์น

2.1.12 การเปรียบเทียบวิธีการประมาณขนาดซอฟต์แวร์

สำหรับวิธีการประมาณขนาดซอฟต์แวร์เชิงวัตถุทั้งหมดสามารถสรุปได้ดัง ตารางที่ 2.25 และสรุปแยกตามช่วงเวลาการพัฒนา (Development Stage)ที่สามารถใช้ในการประมาณได้ดัง รูปที่ 2.11 (Carleton, et al.,1992; Karner, 1993; Henderson-Sellers, 1996; Minkiewicz, 1997; Antoniol et al., 1998; IFPUG, 1999; Abrahao et al., 2004; Costagliola et al., 2005; Zivkovic et al., 2005; Adekile et al., 2010) จากที่กล่าวมาในข้างต้น ทำให้ผู้วิจัยเลือกการประมาณขนาดในช่วงการออกแบบระบบ เนื่องจากพบว่าการออกแบบระบบจะให้ผลที่ดีที่สุดสำหรับการประมาณขนาดซอฟต์แวร์ (Zivkovic et al., 2005)

ตารางที่ 2.25 ตารางเปรียบเทียบข้อดีข้อเสียของวิธีการประมาณขนาดซอฟต์แวร์เชิงวัตถุ

วิธีการประมาณขนาดซอฟต์แวร์ / ลักษณะที่นำเสนอ	Source line of code	Function point	Use case point	OO Metric	Object point	Predictive object point	OOFP	OOmFP	OOFP2	Class point	Pattern point
		1979	1993	1994	1995	1997	1999	2004	2005	2005	2010
เหมาะสำหรับซอฟต์แวร์เชิงวัตถุ	×	-	√	√	×	√	√	√	√	√	√
สามารถใช้ทำนายค่าความพยายามในการพัฒนาซอฟต์แวร์ได้ (Effort)	×	√	√	-	√	√	√	√	√	√	√
ไม่อาศัยความเชี่ยวชาญของผู้ออกแบบระบบ	√	√	×	-	√	-	√	-	√	√	×
ใช้ข้อมูลที่หาง่าย ไม่ต้องลงรายละเอียดระดับลึก	√	√	√	-	√	×	√	×	√	√	√

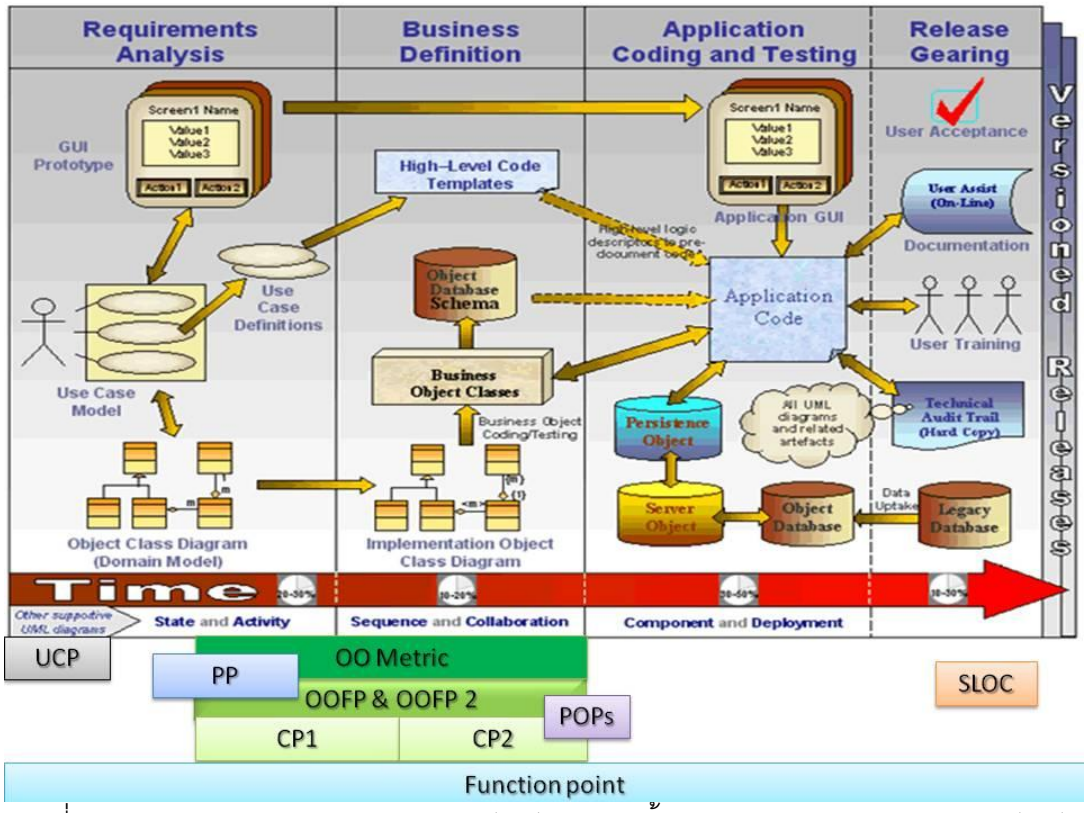
คำอธิบายเพิ่มเติม √ คือ สามารถทำได้

× คือ ไม่สามารถทำได้

- ยังไม่สามารถสรุปได้แน่ชัด

ตามที่กล่าวมาข้างต้นนั้น ผู้วิจัยจึงไม่เลือกวิธีการประมาณขนาดซอฟต์แวร์ด้วยยูสเคสมาทดสอบ เนื่องจากพบการทดสอบการประมาณขนาดซอฟต์แวร์ด้วยยูสเคสเทียบกับฟังก์ชันพอยต์เชิงวัตถุสอง (OOFP2) โดยพบว่าฟังก์ชันพอยต์เชิงวัตถุสองสามารถประมาณค่าความพยายามได้ใกล้เคียงมากกว่ายูสเคสพอยต์ (Zivkovic et al., 2005) เช่นเดียวกับมาตรวัดเชิงวัตถุพบว่า วิธีการประมาณขนาดด้วยคลาสพอยต์สามารถให้ผลที่ดีกว่า (Costagliola et al., 2005)

จึงทำให้ผู้วิจัยเลือกวิธีการที่เหมาะสมสำหรับการประมาณค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์ในช่วงการเวลาเดียวกันและใช้เอกสารที่ใกล้เคียงกัน โดยเลือกช่วงเวลาก่อนการพัฒนาซอฟต์แวร์ (Coding) ด้วยเอกสารการออกแบบระบบ คือ แผนภาพคลาส โดยผู้วิจัยจึงเลือกทดสอบความแม่นยำด้วยวิธีการประมาณขนาดซอฟต์แวร์ด้วย ฟังก์ชันพอยต์เชิงวัตถุ (Antoniol et al., 1998) ฟังก์ชันพอยต์เชิงวัตถุสอง (Zivkovic et al., 2005) และ คลาสพอยต์ (Costagliola et al., 2005) มาทดสอบกับวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์ (IFPUG, 1999) ซึ่งเป็นวิธีการประมาณขนาดสำหรับซอฟต์แวร์ด้วยวิธีพื้นฐานที่ถูกนำมาใช้มากที่สุด (Yang et al., 2008; Peixoto et al., 2010)



รูปที่ 2.11 แสดงการประมาณขนาดซอฟต์แวร์แยกตามขั้นตอนกระบวนการพัฒนาซอฟต์แวร์

ปรับแต่งรูปเพิ่มเติมจาก ที่มา : http://www.jot.fm สืบค้นเมื่อ 28 สิงหาคม 2554

ในงานวิจัยนี้ผู้วิจัยจึงเลือกสี่วิธีการดังกล่าวเพื่อนำมาเปรียบเทียบความแม่นยำ คือ ฟังก์ชันพอยต์ ฟังก์ชันพอยต์เชิงวัตถุ และคลาสพอยต์ ซึ่งสามารถแจกแจงรายละเอียดความแตกต่างระหว่างสี่วิธีการประมาณขนาดซอฟต์แวร์ได้ตามตารางที่ 2.26

ตารางที่ 2.26 แสดงการเปรียบเทียบระหว่างวิธีการประมาณขนาดซอฟต์แวร์ทั้งสี่วิธีประมาณขนาดซอฟต์แวร์

Method	Entity	ตัววัดค่าความซับซ้อนของ Entity	ความซับซ้อน	น้ำหนักคูณค่าความซับซ้อน	ตัวแปรปรับค่า	ผลของตัวแปรปรับค่า
Function point	Data ILF - Internal Logical files EIF - External Interface Files	Data element types: DET Record element types: RET	3 levels (Low/ Average/High) for each type of entity.	ILF - Internal Logical files Low x 7 Average x 10 High x 15 EIF - External Interface Files Low x 5 Average x 7 High x 10	14 General System Characteristics (GSC)	Weight (0-5) for each one of the 14 GSCs, with a ±35% variability on the value of unadjusted FPs
	Transactions EI - External Input EO - External Output EQ - External Inquiry	File Type Referenced: FTR Data Element Type: DET		EI & EQ Low x 3 Average x 4 High x 6		

Method	Entity	ตัววัดค่าความซับซ้อนของ Entity	ความซับซ้อน	น้ำหนักคูณค่าความซับซ้อน	ตัวแปรปรับค่า	ผลของตัวแปรปรับค่า
				EO Low x 4 Average x 5 High x 7		
Object-oriented Function point & Object-oriented Function point 2	Data ILF – internal class EIF – External class	Association Attribute	3 levels (Low/ Average/High) for each type of entity.	ILF - Internal Logical files Low x 7 Average x 10 High x 15 EIF - External Interface Files Low x 5 Average x 7 High x 10		
	Transaction SR - Method	Argument		Low x 3 Average x 4		

Method	Entity	ตัววัดค่าความซับซ้อนของ Entity	ความซับซ้อน	น้ำหนักคูณค่าความซับซ้อน	ตัวแปรปรับค่า	ผลของตัวแปรปรับค่า
				High x 6		
Class point	Problem Domain Class	Number of Attribute Number of External method Number of service request	3 levels (Low/ Average/High) for each type of entity.	Low x 3 Average x 6 High x 10	18 General System Characteristics (GSC)	Weight (0-5) for each one of the 18 GSCs, with a ±45% variability on the value of unadjusted CPs
	Human Interaction Class			Low x 4 Average x 7 High x 12		
	Data Management Class			Low x 5 Average x 8 High x 13		
	Task Management Class			Low x 4 Average x 6 High x 9		

จากตารางที่ 2.26 จะสามารถสรุปได้ดังนี้

สำหรับวิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์ ฟังก์ชันพอยต์เชิงวัตถุ ฟังก์ชันพอยต์เชิงวัตถุสองและคลาสพอยต์จะมีการกำหนดระดับความซับซ้อนของแต่ละเอนทิตี (Entity) ด้วยค่าความซับซ้อน 3 ประเภท คือ ต่ำ (Low) ปานกลาง (Average) และ สูง (High) เช่นเดียวกัน

วิธีการฟังก์ชันพอยต์เชิงวัตถุและฟังก์ชันพอยต์จะมีความใกล้เคียงกันมากที่สุด กล่าวคือ ฟังก์ชันพอยต์เชิงวัตถุจะเป็นการนับฟังก์ชันพอยต์ในรูปแบบของการออกแบบเชิงวัตถุ สำหรับฟังก์ชันพอยต์จะนับจากข้อมูล (Data) และทรานแซคชัน (Transaction) แต่สำหรับฟังก์ชันพอยต์เชิงวัตถุจะเปลี่ยนการนับ คือ

- คลาส (Class) นับเป็น ข้อมูล (Data)
- เมท็อด (Method) นับเป็น ทรานแซคชัน (transaction)

การหาระดับความซับซ้อนของข้อมูล (Data) สำหรับฟังก์ชันพอยต์จะหาได้จาก องค์ประกอบของข้อมูล (Data element types: DET) และ เรคคอร์ดข้อมูล (Record element types: RET) สำหรับฟังก์ชันพอยต์หาได้จากลักษณะประจำ (Attribute) และความเชื่อมโยง (Association) โดย

- ลักษณะประจำที่ง่าย (Simple attribute) นับเป็น องค์ประกอบของข้อมูล (Data element types: DET)
- ลักษณะประจำที่ซับซ้อน (Complex attribute) นับเป็น เรคคอร์ดข้อมูล (Record element types: RET)
- ความเชื่อมโยงจำนวน 1 (Single-value association) นับเป็น องค์ประกอบของข้อมูล (Data element types: DET)
- ความเชื่อมโยงจำนวนมากกว่า 1 (Multiple-value association) นับเป็น เรคคอร์ดข้อมูล (Record element types: RET)

โดยค่าน้ำหนักของความซับซ้อนในแต่ละข้อมูล (Data) จะเท่ากับฟังก์ชันพอยต์

การหาระดับความซับซ้อนของทรานแซคชัน สำหรับฟังก์ชันพอยต์จะหาได้จาก จำนวนองค์ประกอบของข้อมูล (Data element types: DET) ประเภทของไฟล์ข้อมูลอ้างอิง (File Type Referenced: FTR) สำหรับฟังก์ชันพอยต์เชิงวัตถุหาได้จากอาร์กิวเมนต์ (Argument) ในแต่ละเมท็อด โดย

- อาร์กิวเมนต์ที่ง่าย (Simple item) นับเป็น องค์ประกอบของข้อมูล (Data element types: DET)
- อาร์กิวเมนต์ที่ซับซ้อน (Complex item) นับเป็น ไฟล์ข้อมูลอ้างอิง (File Type Referenced: FTR)

โดยค่าน้ำหนักของความซับซ้อนของทรานแซคชัน (Transaction) จะใช้เท่ากับอินเทอร์นอลอินพุท (External Input: EI) ของฟังก์ชันพอยต์

สำหรับวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุสองเป็นมีกระบวนการและค่าน้ำหนักของความซับซ้อนแต่ละเอนทิตีเช่นเดียวกับฟังก์ชันพอยต์ แตกต่างกันที่การกำหนดตารางเพื่อหาระดับความซับซ้อนของทรานแซคชัน (Transaction)

แต่สำหรับวิธีการประมาณขนาดด้วยคลาสพอยต์จะแตกต่างออกไปจากวิธีการอื่น เนื่องจาก Costagliola และคณะ ได้กำหนดวิธีการขึ้นมาใหม่ ซึ่งเป็นการนับที่คลาส และหาระดับความซับซ้อนจากคุณสมบัติของคลาส คือ จำนวนเมท็อดสาธารณะ (Number of External Method: NEM) จำนวนการเรียกใช้บริการจากภายนอก (Number of Services Requested: NSR) และจำนวนลักษณะประจำ (Number Of Attribute: NOA) โดยค่าน้ำหนักความซับซ้อนถูกกำหนดขึ้นมาใหม่

ในแง่ของตัวแปรปรับค่า สามารถแจกแจงรายละเอียดได้ดังนี้

- วิธีประมาณขนาดด้วยฟังก์ชันพอยต์ กำหนดตัวแปรปรับค่าจำนวน 14 ตัว จากลักษณะของระบบที่พัฒนาขึ้น และสามารถปรับเปลี่ยนค่าได้ +/- 35% (IFPUG, 1999)
- วิธีประมาณขนาดด้วยคลาสพอยต์ กำหนดตัวแปรปรับค่าโดยใช้ตัวเดียวกับฟังก์ชันพอยต์ และเพิ่มตัวแปรปรับค่ามาอีก 4 ตัว ซึ่งเป็นคุณลักษณะของระบบซอฟต์แวร์เชิงวัตถุ ทำให้ตัวแปรปรับค่าของคลาสพอยต์มีทั้งหมดจำนวน 18 ตัว และสามารถปรับเปลี่ยนค่าได้ +/- 45% (Costagliola et al., 2005)
- วิธีประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุและฟังก์ชันพอยต์เชิงวัตถุสอง ไม่กำหนดให้มีตัวแปรปรับค่า (Antoniol & et al., 1998; Zivkovic et al., 2005)
- จากที่กล่าวมาข้างต้นนั้น ขนาดที่ได้จากการคำนวณด้วย ฟังก์ชันพอยต์ ฟังก์ชันพอยต์เชิงวัตถุ และคลาสพอยต์ จะแตกต่างกัน

2.2 การนำวิธีการประมาณขนาดไปใช้ในการประมาณค่าความพยายาม (Devlopment effort)

สำหรับขนาดของระบบที่ได้ในแต่ละวิธีการประมาณขนาดซอฟต์แวร์ ผู้วิจัยจะต้องแปลงขนาดของระบบเป็นค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์เพื่อทดสอบความถูกต้อง ในงานวิจัยของ Albrecht และ Gaffney (1983) ได้นำเสนอให้ใช้ขนาดของระบบแปลงเป็นค่าความพยายามด้วยการถดถอยเชิงเส้น (Linear regression) โดยให้ค่าความพยายามเป็นตัวแปรอิสระ (Dependent variable) และขนาดของระบบเป็นตัวแปรต้น (Independent variable) และพบว่าวิธีการประมาณขนาดของซอฟต์แวร์สามารถใช้ในการประมาณค่าความพยายามใกล้เคียงความเป็นจริง (Albrecht & Gaffney, 1983; Behrens, 1983; Conte et al.,1986, Costagliola et al., 2005)

สำหรับค่าความพยายามที่ใช้ต่อหนึ่งฟังก์ชันพอยต์ พบว่า ในข้อมูลของปี ค.ศ. 1980 มีค่าเฉลี่ยอยู่ที่ 18.3 ชั่วโมงต่อฟังก์ชันพอยต์ ด้วยข้อมูลของปี ค.ศ. 1981 มีค่าเฉลี่ยอยู่ที่ 9.4 ชั่วโมง สาเหตุที่ทำให้ข้อมูลของปี ค.ศ. 1981 และ ปี ค.ศ. 1980 ไม่ใกล้เคียงกัน เนื่องจากค่าเฉลี่ยชั่วโมงต่อฟังก์ชันพอยต์จะขึ้นอยู่กับภาษาและขนาดของระบบ (Albrecht & Gaffney, 1983; Behrens, 1983) ทำให้ในงานวิจัยนี้ผู้วิจัยจึงพยายามค้นหาอัตราค่าความพยายามที่ใช้ต่อขนาดซอฟต์แวร์และมีขนาดของหน่วยตัวอย่างที่ใกล้เคียงกับในงานวิจัย

วิธีการประมาณขนาดด้วยฟังก์ชันพอยต์ ในงานวิจัยปี ค.ศ. 1996 เพื่อทดสอบหาจำนวนวันต่อฟังก์ชันพอยต์บนระบบงานเชิงวัตถุ ด้วยระบบงานที่พัฒนาใหม่ ไม่มีการนำกลับมาใช้ใหม่ (Reuse) พบว่า 1.2 ฟังก์ชันพอยต์ต่อหนึ่งคน-วัน หรือเท่ากับ 6.667 ชั่วโมงต่อฟังก์ชันพอยต์ (Moser & Nierstrasz, 1996) ซึ่งสามารถนำมาสร้างสมการได้ดังนี้

$$\text{Effort}(\text{Man} - \text{hours}) = 6.667 * \text{FP}$$

ในปี ค.ศ. 2000 Lokan ได้นำข้อมูลจาก ISBSG (International Software Benchmarking Standards Group) จำนวน 235 โครงการซอฟต์แวร์ วิเคราะห์ความถูกต้องจากวิธีการประมาณการด้วยฟังก์ชันพอยต์ที่ใช้ตัวแปรปรับค่า จากข้อมูลจำนวน 235 โครงการซอฟต์แวร์ ทำให้สมการได้ดังนี้

$$\text{Effort}(\text{Man} - \text{hours}) = 21.0 * \text{FP}^{0.826}$$

สมการดังกล่าวของ Lokan ให้ค่าความคลาดเคลื่อนเฉลี่ยอยู่ที่ (MMRE) 1.25 หรือให้ค่าผลการทำนายเกินกับความเป็นจริงอยู่ถึง 125%

ในปี ค.ศ. 2000 เช่นเดียวกัน งานวิจัยของ Maxwell และ Froselius หาค่าความสามารถในการทำงานของหนึ่งคนในหนึ่งชั่วโมงจะได้จำนวนฟังก์ชันเท่าใด (Productivity FP/Hour)

นักวิจัยได้เก็บข้อมูลจาก 206 โครงการซอฟต์แวร์ในประเทศฟินแลนด์ โดยพบว่าค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์จะขึ้นอยู่กับประเภทของธุรกิจ โดยสามารถแยกออกไป 5 กลุ่มธุรกิจ ดังนี้ กลุ่มธนาคารมีค่าความสามารถในการทำงาน 0.116 ฟังก์ชันพอยต์ต่อหนึ่งคนต่อหนึ่งชั่วโมง กลุ่มการประกันมีค่าความสามารถในการทำงาน 0.116 ฟังก์ชันพอยต์ต่อหนึ่งคนต่อหนึ่งชั่วโมง กลุ่มอุตสาหกรรมการผลิตมีค่าความสามารถในการทำงาน 0.337 ฟังก์ชันพอยต์ต่อหนึ่งคนต่อหนึ่งชั่วโมง กลุ่มการค้าส่งและค้าปลีกมีค่าความสามารถในการทำงาน 0.253 ฟังก์ชันพอยต์ต่อหนึ่งคนต่อหนึ่งชั่วโมง และกลุ่มงานราชการมีค่าความสามารถในการทำงาน 0.232 ฟังก์ชันพอยต์ต่อหนึ่งคนต่อหนึ่งชั่วโมง

Jeffer และคณะ (2001) ได้ทดสอบความแตกต่างระหว่างการพัฒนาซอฟต์แวร์ด้วยบริษัทเดียวและพัฒนาด้วยหลากหลายบริษัท พบว่า การพัฒนาซอฟต์แวร์ด้วยบริษัทเดียวจะให้ค่าความสามารถในการทำงานของคนหนึ่งคนจะใช้เวลา 2.2 ชั่วโมงต่อหนึ่งฟังก์ชันพอยต์ ซึ่งแตกต่างซอฟต์แวร์ที่พัฒนาด้วยหลากหลายบริษัทจะมีค่าความสามารถของคนหนึ่งคนอยู่ที่ 9.5 ชั่วโมงต่อหนึ่งฟังก์ชันพอยต์ (Jeffer et al., 2001)

สำหรับวิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุ Hericko และ Zivkovic (2007) ได้คำนวณจำนวนค่าความพยายามที่ใช้ต่อฟังก์ชันพอยต์เชิงวัตถุ ด้วยหน่วยตัวอย่างภาษาจาวา (Java) ด้วยค่าความพยายามที่ใช้ตั้งแต่ 166-205 คน-ชั่วโมง หรือเท่ากับ 20.75- 34.17 คน-วัน โดยพบว่า 1 ฟังก์ชันพอยต์เชิงวัตถุจะใช้ 0.7 ชั่วโมง (Hericko & Zivkovic, 2007) ซึ่งสามารถกำหนดได้ดังสมการ

$$\text{Effort}(\text{Man} - \text{hours}) = 0.7 * \text{OOFP}$$

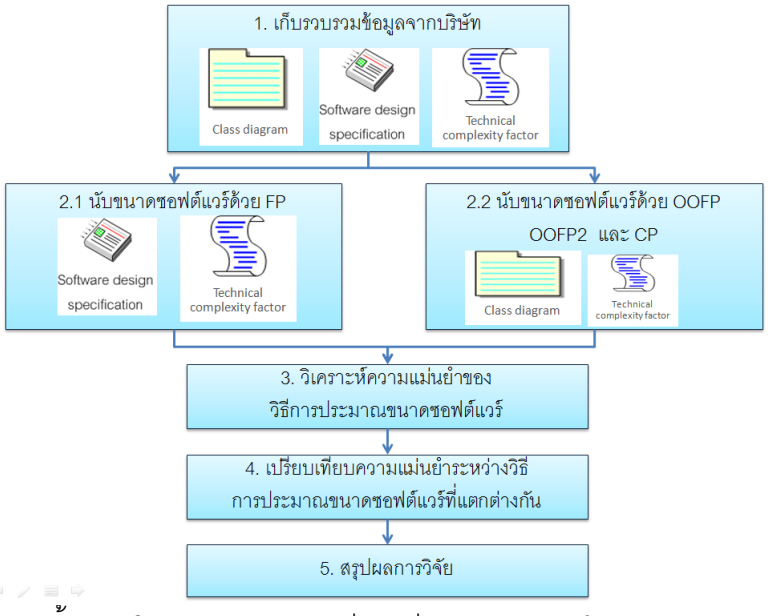
สำหรับวิธีการประมาณขนาดซอฟต์แวร์ด้วยคลาสพอยต์ Costagliola และคณะ ได้ทดสอบวิธีการประมาณขนาดด้วยคลาสพอยต์ด้วยหน่วยตัวอย่างภาษาจาวา (Java) จำนวน 40 ระบบด้วยค่าความพยายามในการพัฒนาซอฟต์แวร์ตั้งแต่ 180-1803 คน-ชั่วโมง หรือเท่ากับ 22.5-225.4 คน-วัน ซึ่งขนาดคลาสพอยต์ 64.61 – 1719.25 คลาสพอยต์ สามารถคำนวณค่าความพยายามในการพัฒนาซอฟต์แวร์ต่อ 1 คลาสพอยต์ ได้ 1.2232 คน-ชั่วโมง (Costagliola et al., 2005) ซึ่งสามารถกำหนดได้ดังสมการ

$$\text{Effort}(\text{Man} - \text{hours}) = 1.2232 * \text{CP}$$

บทที่ 3
วิธีดำเนินการวิจัย

3.1 แนวทางการวิจัย

งานวิจัยนี้มีจุดประสงค์เพื่อเปรียบเทียบความแม่นยำระหว่างการประมาณค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์ด้วยวิธีประมาณขนาดซอฟต์แวร์เชิงที่แตกต่างกัน ซึ่งจะทดสอบด้วยวิธีการประมาณขนาดซอฟต์แวร์ด้วย ฟังก์ชันพอยต์ (IFPUG, 1999) ฟังก์ชันพอยต์เชิงวัตถุ (Antoniol et al., 1998) ฟังก์ชันพอยต์เชิงวัตถุสอง (Zivkovic et al., 2005) คลาสพอยต์ (Costagliola et al., 2005) ด้วยระบบที่พัฒนาขึ้นในธุรกิจ โดยเก็บข้อมูลกับระบบที่พัฒนาเสร็จสมบูรณ์ เพื่อทดสอบข้อมูลเชิงเปรียบเทียบทั้งสี่วิธีการประมาณขนาดของซอฟต์แวร์ ดังรูปที่ 3.1 โดยมีขั้นตอนโดยสรุปดังนี้



รูปที่ 3.1 แสดงขั้นตอนโดยสรุปของการเปรียบเทียบความแม่นยำของการประมาณค่าความพยายามด้วยวิธีการประมาณขนาดที่แตกต่างกัน

ขั้นตอนที่ 1 เก็บรวบรวมข้อมูลจากบริษัท

งานวิจัยนี้ต้องการประมาณขนาดซอฟต์แวร์เชิงวัตถุจากข้อมูลในธุรกิจอุตสาหกรรมซอฟต์แวร์ดังนั้น ผู้พัฒนาจึงเก็บข้อมูลจากการออกแบบระบบที่ถูกพัฒนาขึ้นที่เสร็จสิ้นแล้ว โดย

เลือกภาษาจาวา เพราะเป็นภาษาในโปรแกรมซอฟต์แวร์เชิงวัตถุ โดยจะเก็บข้อมูลเพื่อให้ได้เอกสารการออกแบบของระบบที่ประกอบไปด้วย เอกสารการออกแบบระบบ (System design specification) แผนภาพคลาส (Class diagram) หรือในกรณีที่ไม่มีการออกแบบด้วยแผนภาพคลาสจะใช้ซอร์สโค้ด (Source code) เพื่อใช้ในการทำวิศวกรรมย้อนกลับ (Reverse engineering) ให้ได้แผนภาพคลาส รวมถึงค่าความพยายามในการพัฒนาซอฟต์แวร์ (Development effort) ของแต่ละระบบย่อย เพื่อใช้เปรียบเทียบความแม่นยำในการประมาณขนาดระบบในแต่ละวิธีการ และข้อมูลความซับซ้อนของโครงการซอฟต์แวร์ตามเอกสารดังภาคผนวก จากหนึ่งในผู้พัฒนาโครงการซอฟต์แวร์ เพื่อนำมาคำนวณตัวแปรปรับค่าซอฟต์แวร์ด้วยวิธีฟังก์ชันพอยต์ และคลาสพอยต์

ขั้นตอนที่ 2 นับขนาดซอฟต์แวร์ด้วยวิธีการประมาณขนาดซอฟต์แวร์ทั้งสี่วิธี

หลังจากที่ได้เก็บรวบรวมข้อมูลจากบริษัทแล้ว จะนำเอกสารที่ได้มาเหล่านั้นในการคำนวณด้วยวิธีการประมาณขนาดซอฟต์แวร์สี่วิธีการ โดยแบ่งรายละเอียดได้ดังนี้

ขั้นตอนที่ 2.1 นับขนาดด้วยวิธีการประมาณฟังก์ชันพอยต์ (Function point)

จากวิธีการประมาณขนาดระบบแบบฟังก์ชันพอยต์ตามคู่มือของไอเอฟพียูจี (International Function Point Users Group: IFPUG) การคำนวณขนาดฟังก์ชันพอยต์สามารถวัดได้จากเอกสารการออกแบบระบบ (System design specification) ในงานวิจัยนี้ผู้วิจัยใช้เอกสารดังรูปที่ 3.2 ในการคำนวณขนาดของฟังก์ชันพอยต์ โดยมีขั้นตอนดังนี้

1. ระบุขอบเขตของแต่ละระบบ เพื่อหา อินเทอร์นอลอินพุท (External Input: EI) เอ็กซ์เทอร์นอลเอาท์พุท (External Output: EO) เอ็กซ์เทอร์นอลอินไควรี (External Inquiry: EQ) ไฟล์ข้อมูลภายใน (Internal Logical files: ILFs) และไฟล์ข้อมูลภายนอก (External Interface Files: EIFs) ทั้งหมดของระบบ

2. ระบุรายละเอียดของดาต้าฟังก์ชัน (Data Functions) ประกอบด้วยไฟล์ข้อมูลภายใน (Internal Logical files: ILFs) และ ไฟล์ข้อมูลภายนอก (External Interface Files: EIFs) ในตารางที่ 1 ของเอกสารในรูปที่ 3.2 โดยระบุชื่อไฟล์ข้อมูล ประเภทของไฟล์ข้อมูล จำนวนองค์ประกอบของข้อมูล (Data Element Type :DET) จำนวนเรคคอร์ดข้อมูล (Record Element Type :RET) และประเมินระดับความซับซ้อนตามตารางที่ 3.1

ตารางที่ 3.1 แสดงตารางเพื่อหาระดับความซับซ้อนของ ILFs และ EIFs (IFPUG, 1999)

	1 to 19 DET	20 to 50 DET	51 or more DET
1 RET	Low	Low	Average
2 to 5 RET	Low	Average	High
6 or more RET	Average	High	High

3. ระบุรายละเอียดของทรานแซคชันฟังก์ชัน (Transactional Functions) ประกอบด้วย อินเทอร์นอลอินพุท (External Input: EI) เอ็กซ์เทอร์นอลเอาท์พุท (External Output: EO) และ เอ็กซ์เทอร์นอลอินไควรี (External Inquiry: EQ) ในตารางที่ 2 ของเอกสารรูปที่ 3.2 โดยระบุชื่อหน้าจอ จำนวนองค์ประกอบของข้อมูล (Data Element Type :DET) และจำนวนประเภทของไฟล์ข้อมูลอ้างอิง (File Type of Reference :FTR) ของแต่ละอินเทอร์นอลอินพุท (External Input: EI) เอ็กซ์เทอร์นอลเอาท์พุท (External Output: EO) และ เอ็กซ์เทอร์นอลอินไควรี (External Inquiry: EQ) และประเมินระดับความซับซ้อนตามตารางที่ 3.2 และ 3.3

ตารางที่ 3.2 แสดงตารางเพื่อหาระดับความซับซ้อนของอินเทอร์นอลอินพุท (External Input: EI) (IFPUG, 1999)

	1 to 4 DET	5 to 15 DET	16 or more DET
0 to 1 FTR	Low	Low	Average
2 FTRs	Low	Average	High
3 or more FTRs	Average	High	High

ตารางที่ 3.3 แสดงตารางเพื่อหาระดับความซับซ้อนของ External output (EO) และ External Inquiry (EQ) (IFPUG, 1999)

	1 to 5 DET	6 to 19 DET	20 or more DET
0 to 1 FTR	Low	Low	Average
2 to 3 FTRs	Low	Average	High
4 or more FTRs	Average	High	High

4. คำนวณค่ารวมขนาดของฟังก์ชันพอยต์ที่ยังไม่ถูกปรับค่า ในตารางล่างสุดของเอกสารรูปที่ 3.2 โดยนับจำนวนอินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) เอ็กซ์เทอร์นอลอินเตอร์เฟสไฟล์ (External Interface Files: EIFs) อินเทอร์นอลอินพุท (External Input: EI)

เอ็กซ์เทอร์นอลเอาท์พุท (External Output: EO) เอ็กซ์เทอร์นอลอินไควรี (External Inquiry: EQ) ในแต่ละระดับความซับซ้อน นำมากรอกลงด้านล่างของเอกสารรูปที่ 3.2 และคำนวณค่ารวมขนาดของฟังก์ชันพอยต์โดยยังไม่ถูกปรับค่าทั้งหมด

5. คำนวณตัวแปรปรับค่าที่ใช้ในการประมาณขนาดด้วยฟังก์ชันพอยต์ ซึ่งได้จากค่าผลรวมของค่าระดับความมีอิทธิพลของปัจจัยที่เกี่ยวข้องกับการประมาณขนาดซอฟต์แวร์ในเอกสารภาคผนวก ตั้งแต่ลักษณะที่ 1-14 นำมาคำนวณในสูตร VAF = (TDI * 0.01) + 0.65 ค่า VAF ที่ได้ คือ ตัวแปรปรับค่าของฟังก์ชันพอยต์

6. คำนวณค่ารวมขนาดของฟังก์ชันพอยต์ นำค่ารวมขนาดของฟังก์ชันพอยต์ที่ยังไม่ถูกปรับค่ามาคูณกับตัวแปรปรับค่าของฟังก์ชันพอยต์

ระบบที่

เวลาที่ใช้

	ชื่อ	ประเภท	จำนวน DET	จำนวน RET	C*
1		ILF EIF			L A H
2		ILF EIF			L A H
3		ILF EIF			L A H
4		ILF EIF			L A H
5		ILF EIF			L A H
6		ILF EIF			L A H
7		ILF EIF			L A H

	หน้าจอ	EI			EQ			EO		
		DET	FTR	C*	DET	FTR	C*	DET	FTR	C*
1				L A H			L A H			L A H
2				L A H			L A H			L A H
3				L A H			L A H			L A H
4				L A H			L A H			L A H
5				L A H			L A H			L A H
6				L A H			L A H			L A H
7				L A H			L A H			L A H
8				L A H			L A H			L A H

ชนิด	LOW	AVERAGE	HIGH	SUM
ILF	7	10	15	
ELF	5	7	10	
EI	3	4	6	
EQ	3	4	6	
EO	4	5	7	
			รวม	FP

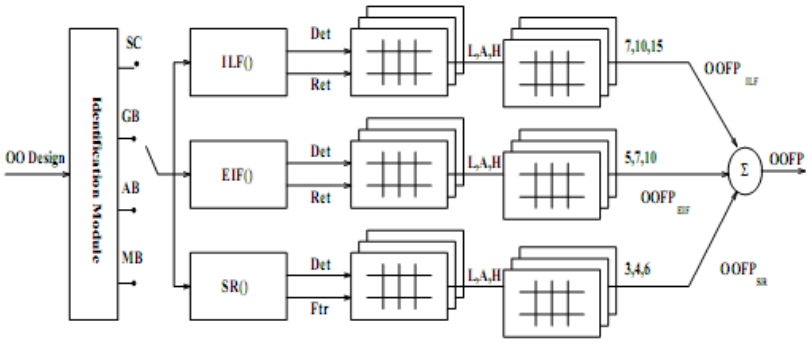
รูปที่ 3.2 แสดงเอกสารเพื่อใช้สำหรับนับขนาดระบบด้วยฟังก์ชันพอยต์

ขั้นตอนที่ 2.2 นับขนาดซอฟต์แวร์ด้วยวิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุ ฟังก์ชันพอยต์เชิงวัตถุสองและคลาสพอยต์

เนื่องจากการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุและฟังก์ชันพอยต์เชิงวัตถุสอง (OOFP, OOFP2) และคลาสพอยต์ (Class point) โดยมีวิธีการนับจากข้อมูลการออกแบบด้วยแผนภาพคลาส (Class diagram) ดังนี้

ขั้นตอนที่ 2.2.1 ประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุ (OOFP)

จากวิธีการประมาณขนาดระบบด้วยวิธีฟังก์ชันพอยต์เชิงวัตถุ (Antoniol et al., 1998) จะใช้เอกสารการออกแบบระบบส่วนแผนภาพคลาส (Class diagram) ที่ได้โดยมีขั้นตอนการคำนวณรูปที่ 3.3 ผู้วิจัยจึงพัฒนาเอกสารไฟล์ไมโครซอฟต์เอกซ์เซล (Microsoft Excel File) เพื่อใช้ในการคำนวณ เอกสารดังกล่าวแสดงตัวอย่างได้ดังรูปที่ 3.4 โดยมีรายละเอียดในการคำนวณดังต่อไปนี้



รูปที่ 3.3 แสดงขั้นตอนคำนวณขนาดจากฟังก์ชันพอยต์เชิงวัตถุ (OOFP) (Antoniol et al., 1998)

1. กำหนดขอบเขตและเลือกวิธีการนับลอจิคอลไฟล์ (Logical file) กำหนดขอบเขตของระบบย่อยว่าประกอบด้วยคลาสใดบ้าง ซึ่งแต่ละระบบย่อยจะถูกแยกด้วยแพคเกจที่คลาสนั้นๆ อยู่ หลังจากนั้นจึงทำวิศวกรรมย้อนกลับ (Reverse engineering) เป็นแผนภาพคลาส หลังจากนั้นเลือกวิธีการนับโดยมีวิธีการนับทั้งสิ้น 4 วิธี คือ วิธีการนับแยกคลาสเดี่ยว การนับรวมคลาสที่มีการสืบทอด การนับรวมคลาสการรวมกลุ่มและการนับแบบผสม ในงานวิจัยนี้ผู้วิจัยเลือกใช้วิธีการนับแยกคลาส เนื่องจากในงานวิจัยปี ค.ศ. 2005 พบว่า ได้ทดลองการนับแยกตามคลาสซึ่งให้ผลความแม่นยำอยู่ในเกณฑ์ที่สามารถใช้งานได้ หรือมีความผิดพลาดน้อยกว่า 25% (Zivkovic et al., 2005)

2. ประเมินระดับความซับซ้อนของอินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) และเอ็กเทอร์นอลอินเตอร์เฟสไฟล์ (External Interface Files: EIFs) ซึ่งพิจารณาจากจำนวนองค์ประกอบของข้อมูล (Data element types: DET) และเรคคอร์ดข้อมูล (Record

element types: RET) เมื่อได้ทราบคลาสแต่ละคลาสของระบบย่อยแล้ว นำแผนภาพคลาสมานับจำนวนลักษณะประจำที่ง่าย (Simple attribute) และจำนวนลักษณะประจำที่ซับซ้อน (Complex attribute) โดยลักษณะประจำที่ง่ายจะเป็นองค์ประกอบของข้อมูล (Data element types: DET) ส่วน 1 ลักษณะประจำที่ซับซ้อนจะเป็นเรคคอร์ดข้อมูล (Record element types: RET) หลังจากนั้นนับจำนวนความเชื่อมโยงเพียง 1 (single-valued association) และจำนวนความเชื่อมโยงที่มากกว่า 1 (multiple-valued association) โดยความเชื่อมโยงเพียง 1 จะเป็นองค์ประกอบของข้อมูล (Data element types: DET) ส่วนความเชื่อมโยงที่มากกว่าจะเป็นเรคคอร์ดข้อมูล (Record element types: RET) หลังจากนั้นนำมาประเมินระดับความซับซ้อนของคลาสตามตารางที่ 3.4 และนำจำนวนอินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) และ เอ็กเทอร์นอลอินเตอร์เฟสไฟล์ (External Interface Files: EIFs) ในแต่ละระดับความซับซ้อนมากรอกข้อมูลลงเอกสารในรูปที่ 3.4 ในช่องสีเทา

ตารางที่ 3.4 แสดงตารางเพื่อหาระดับความซับซ้อนของ ILFs และ EIFs (IFPUG, 1999)

	1 to 19 DET	**20 to 50 DET**	**51 or more DET**
1 RET	Low	Low	Average
2 to 5 RET	Low	Average	High
6 or more RET	Average	High	High

3. ประเมินระดับความซับซ้อนของเซอร์วิสรีเควส (Services request: SR) ซึ่งพิจารณาจากจำนวนองค์ประกอบของข้อมูล (Data element types: DET) และประเภทของไฟล์ข้อมูลอ้างอิง (File Type Referenced: FTR) เมื่อทราบคลาสในแต่ละระบบย่อยแล้ว จะดูที่เมท็อด (Method) ที่ประกอบอยู่ในแผนภาพคลาสนั้น โดยนับจำนวนอาร์กิวเมนต์ (argument) ที่ประกอบอยู่ในเมท็อด ถ้าเป็นอาร์กิวเมนต์ (argument) ที่มีชนิดไม่ซับซ้อน เช่น ตัวเลข (Integer) สายอักขระ (String) ที่ประกอบอยู่ในเมท็อด ให้นับเป็น 1 จำนวนองค์ประกอบของข้อมูล (Data element types: DET) แต่ถ้าเป็นอาร์กิวเมนต์ (argument) ที่ซับซ้อน ให้นับเป็น 1 ไฟล์ข้อมูลอ้างอิง (File Type Referenced: FTR) และนำมาประเมินระดับความซับซ้อนดังตารางที่ 3.5 และนำจำนวนเซอร์วิสรีเควส (Services request: SR) ในแต่ละระดับความซับซ้อนมากรอกข้อมูลลงเอกสารในรูปที่ 3.4 ในช่องสีเทา

ตารางที่ 3.5 แสดงตารางเพื่อหาระดับความซับซ้อนของเซอร์วิสรีเควส (Services request: SR) (IFPUG: 1999)

	1 to 4 DET	5 to 15 DET	16 or more DET
0 to 1 FTR	Low	Low	Average
2 FTRs	Low	Average	High
3 or more FTRs	Average	High	High

4. คำนวณค่ารวมฟังก์ชันพอยต์เชิงวัตถุซึ่งได้จากการรวมค่าในเอกสารรูปที่ 3.4

ขั้นตอนที่ 2.2.2 ประมาณขนาดซอฟต์แวร์ด้วยวิธีฟังก์ชันพอยต์เชิงวัตถุสอง (OOFP2)

จากวิธีการประมาณขนาดระบบด้วยวิธีฟังก์ชันพอยต์เชิงวัตถุสอง (Zivkovic et al., 2005) ซึ่งได้ปรับปรุงจากวิธีการประมาณขนาดฟังก์ชันพอยต์เชิงวัตถุในปี ค.ศ. 1998 ของ Antoniol และคณะ ซึ่งใช้วิธีการเดียวกับฟังก์ชันพอยต์เชิงวัตถุ (Antoniol et al., 1998) แต่เปลี่ยนตารางการให้ค่าความซับซ้อนของเซอร์วิสรีเควส (Services request: SR) จากตารางที่ 3.5 เป็นตารางที่ 3.6

ตารางที่ 3.6 แสดงการคำนวณความซับซ้อนของเซอร์วิสรีเควส (Services request: SR) ด้วยวิธีการ OOFP2 (Zivkovic et al., 2005)

	0–4 (1–5) DET	5–10 (6–19) DET	More than 10 (20 or more) DET
0 or 1 FTR	Low (low)	Average (low)	High (average)
2 (2–3) FTR	Average (low)	Average	High
3 or more (4 or more) FTR	Average	High	High

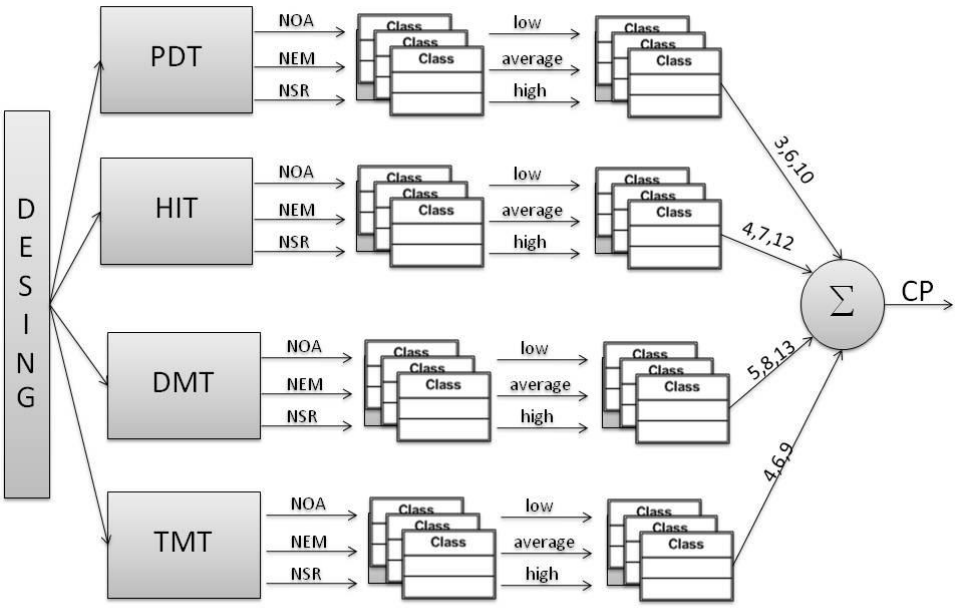
สำหรับการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุสองจะใช้เอกสารที่ใช้ในการคำนวณฟังก์ชันพอยต์เชิงวัตถุสองใช้เอกสารเดียวกับฟังก์ชันพอยต์ คือ เอกสารรูปที่ 3.4

Funtion Type	Function complexity	Complexity totals		Function type totals	
ILFs		Low	7	0	OOFP
		Average	10	0	OOFP
		High	15	0	OOFP
EIFs		Low	5	0	OOFP
		Average	7	0	OOFP
		High	10	0	OOFP
SR		Low	3	0	OOFP
		Average	4	0	OOFP
		High	6	0	OOFP
Total OOFP				**0**	OOFP
	Effort			**0**	**Man-hour**

รูปที่ 3.4 แสดงเอกสารเพื่อใช้สำหรับนับขนาดระบบด้วยฟังก์ชันพอยต์เชิงวัตถุและฟังก์ชันพอยต์เชิงวัตถุสอง

ขั้นตอนที่ 2.2.3 หาขนาดระบบด้วยวิธีการคำนวณขนาดจากคลาสพอยต์ (Class point)

จากวิธีการประมาณขนาดระบบด้วยวิธีคลาสพอยต์ (Costagliola et al., 2005) ซึ่งมีวิธีการประมาณขนาด 2 วิธีการ คือ CP1 และ CP2 สำหรับการประมาณขนาดด้วย CP1 เป็นการประมาณขนาดในช่วงแรกของการออกแบบซึ่งให้ความแม่นยำที่น้อยกว่าการประมาณขนาดด้วย CP2 ซึ่งเป็นการประมาณขนาดในช่วงที่มีการออกแบบแผนภาพคลาสสมบูรณ์แล้ว ผู้วิจัยจึงเลือกใช้วิธีการประมาณขนาดด้วย CP2 โดยมีขั้นตอนการคำนวณรูปที่ 3.5 ผู้วิจัยจึงพัฒนาเอกสารไฟล์ไมโครซอฟต์เอกซ์เซล (Microsoft Excel File) ดังรูปที่ 3.5 เพื่อใช้ในการคำนวณค่ารวมคลาสพอยต์ โดยมีรายละเอียดในการคำนวณดังต่อไปนี้



รูปที่ 3.5 แสดงขั้นตอนของการประมาณขนาดซอฟต์แวร์ด้วยคลาสพอยต์

1. ระบุคลาสทั้งหมด โดยการกรอกชื่อลงบนแถวแรกของเอกสาร

2. นับจำนวนเมท็อดสาธารณะ (Number of External Method: NEM) จำนวนการเรียกใช้บริการจากภายนอก (Number of Services Requested: NSR) จำนวนลักษณะประจำ (Number Of Attribute: NOA) ในแต่ละคลาสในคอลัมน์ที่ 2-4

3. จำแนกประเภทของคลาส โดยแบ่งเป็นประเภทขอบเขตของปัญหา (Problem domain) ประเภทโต้ตอบกับผู้ใช้งาน (Human Interaction) ประเภทการจัดการข้อมูล (Data Management) ประเภทการจัดการงาน (Task Management) ตามลักษณะการทำงานของคลาส

4. ประเมินระดับความซับซ้อนในแต่ละคลาสจากจำนวนเมท็อดสาธารณะ (Number of External Method: NEM) จำนวนการเรียกใช้บริการจากภายนอก (Number of Services Requested: NSR) จำนวนลักษณะประจำ (Number Of Attribute: NOA) ตามตารางที่ 3.7 ซึ่งจะถูกคำนวณในไฟล์ไมโครซอฟต์เอกซ์เซล

ตารางที่ 3.7 แสดงวิธีการคำนวณระดับความซับซ้อนของคลาสพอยต์ (Costagliola et al., 2005)

0 - 2 *NSR*	**0 - 5 *NOA***	**6 - 9 *NOA***	**≥ 10 *NOA***
0 - 4 *NEM*	Low	Low	Average
5 - 8 *NEM*	Low	Average	High
≥ 9 *NEM*	Average	High	High

(a)

3 - 4 *NSR*	**0 - 4 *NOA***	**5 - 8 *NOA***	**≥ 9 *NOA***
0 - 3 *NEM*	**Low**	**Low**	**Average**
4 - 7 *NEM*	**Low**	**Average**	**High**
≥ 8 *NEM*	**Average**	**High**	**High**

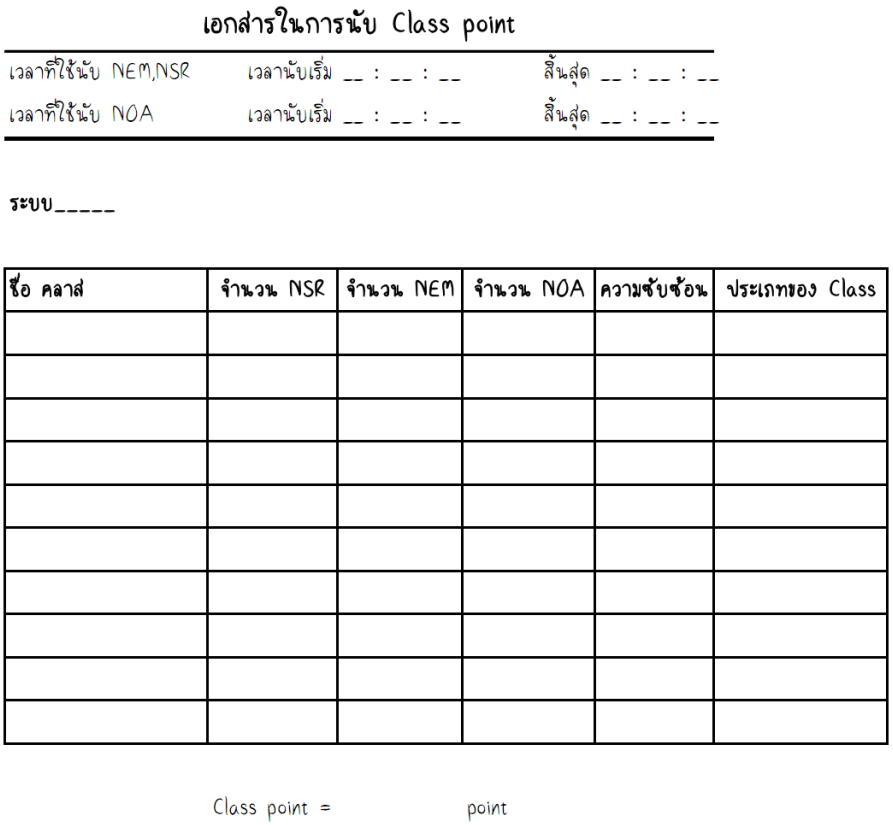
(b)

≥ 5 *NSR*	**0 - 3 *NOA***	**4 - 7 *NOA***	**≥ 8 *NOA***
0 - 2 *NEM*	**Low**	**Low**	**Average**
3 - 6 *NEM*	**Low**	**Average**	**High**
≥ 7 *NEM*	**Average**	**High**	**High**

5. คำนวณค่ารวมขนาดของคลาสพอยต์ที่ยังไม่ถูกปรับค่า โดยรวมค่าของคลาสทุกคลาสตามระดับความซับซ้อนและประเภทของคลาส ในเอกสารรูปที่ 3.5 จะแสดงอยู่ด้านล่างของเอกสารซึ่งค่าดังกล่าวเป็นค่าที่ยังไม่ถูกปรับค่าตามปัจจัยที่มีอิทธิพลกับขนาดของซอฟต์แวร์จากวิธีการประมาณขนาดด้วยคลาสพอยต์ (Degree of influence)

6. คำนวณตัวแปรปรับค่าที่ใช้ในการประมาณขนาดด้วยคลาสพอยต์ ซึ่งได้จากค่าผลรวมของค่าระดับความมีอิทธิพลของปัจจัยที่เกี่ยวข้องกับการประมาณขนาดซอฟต์แวร์ในเอกสารภาคผนวก ตั้งแต่ลักษณะที่ 1-18 นำมาคำนวณในสูตร TCF = 0.55 + (0.01*TDI) ค่า TCF ที่ได้คือ ตัวแปรปรับค่าของคลาสพอยต์

7. คำนวณค่ารวมขนาดของคลาสพอยต์ นำค่ารวมขนาดของคลาสพอยต์ที่ยังไม่ถูกปรับค่ามาคูณกับตัวแปรปรับค่าของคลาสพอยต์



เอกสารในการนับ Class point

เวลาที่ใช้นับ NEM,NSR เวลานับเริ่ม __ : __ : __ สิ้นสุด __ : __ : __

เวลาที่ใช้นับ NOA เวลานับเริ่ม __ : __ : __ สิ้นสุด __ : __ : __

ระบบ_____

ชื่อ คลาส	จำนวน NSR	จำนวน NEM	จำนวน NOA	ความซับซ้อน	ประเภทของ Class

Class point = point

รูปที่ 3.5 แสดงเอกสารที่ใช้ในการนับขนาดระบบด้วยคลาสพอยต์

ขั้นตอนที่ 2.3 วิเคราะห์ความแม่นยำของวิธีการประมาณขนาดซอฟต์แวร์

เมื่อได้ขนาดซอฟต์แวร์ทั้งสี่วิธีการประมาณ จึงนำมาวิเคราะห์ความแม่นยำกับค่าความพยายามที่ใช้ในการพัฒนาจริง ทำให้ผู้วิจัยจึงจำเป็นต้องแปลงขนาดของซอฟต์แวร์จากวิธีการประมาณขนาดซอฟต์แวร์ทั้งสี่วิธีการด้วยสมการที่ได้จากงานวิจัยในอดีต ซึ่งมีรายละเอียดดังนี้

การแปลงขนาดของฟังก์ชันพอยต์จะใช้สมการด้วยสมการจำนวน 4 สมการดังนี้

สมการที่ 1 จากงานวิจัยของ Moser และ Nierstrasz (1996)

$$\text{Effort}(\text{Man} - \text{hours}) = 6.667 * \text{FP}$$

สมการที่ 2 จากงานวิจัยของ Lokan (2000)

$$\text{Effort}(\text{Man} - \text{hours}) = 21.0 * \text{FP}^{0.826}$$

สมการที่ 3 จากงานวิจัยของ Maxwell และ Froselius (2006) เลือกสมการตามลักษณะของหน่วยตัวอย่าง ดังนี้

สำหรับหน่วยตัวอย่างในกลุ่มธนาคาร $\text{Effort}(\text{Man} - \text{hours}) = \frac{\text{FP}}{0.116}$

สำหรับหน่วยตัวอย่างกลุ่มการประกัน $\text{Effort}(\text{Man} - \text{hours}) = \frac{\text{FP}}{0.116}$

สำหรับหน่วยตัวอย่างกลุ่มอุตสาหกรรมการผลิต $\text{Effort}(\text{Man} - \text{hours}) = \frac{\text{FP}}{0.337}$

สำหรับหน่วยตัวอย่างกลุ่มการค้าส่งและค้าปลีก $\text{Effort}(\text{Man} - \text{hours}) = \frac{\text{FP}}{0.253}$

สำหรับหน่วยตัวอย่างกลุ่มงานราชการ $\text{Effort}(\text{Man} - \text{hours}) = \frac{\text{FP}}{0.232}$

สมการที่ 4 จากงานวิจัยของ Jeffer และคณะ (2001)

$$\text{Effort}(\text{Man} - \text{hours}) = 2.2 * \text{FP}$$

และคัดเลือก 1 สมการที่ให้ความแม่นยำที่ใกล้เคียงที่สุดมาเป็นตัวแทนของการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์

การแปลงขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุจะใช้สมการจากงานวิจัยของ Hericko และ Zivkovic ได้คำนวณจำนวนค่าความพยายามที่ใช้ต่อฟังก์ชันพอยต์เชิงวัตถุ ด้วยหน่วยตัวอย่างภาษาจาวา (Java) โดยพบว่า 1 ฟังก์ชันพอยต์เชิงวัตถุจะใช้ 0.7 คน-ชั่วโมง (Hericko & Zivkovic, 2007) ซึ่งสามารถกำหนดได้ดังสมการ

$$\text{Effort}(\text{Man} - \text{hours}) = 0.7 * \text{OOFP}$$

การแปลงขนาดซอฟต์แวร์ด้วยคลาสพอยต์จะใช้สมการจากงานวิจัยของ Costagliola และคณะ ซึ่งได้จากหน่วยตัวอย่างภาษาจาวา (Java) ซึ่งขนาดคลาสพอยต์และค่าความพยายามที่ใช้ในการพัฒนามีความสัมพันธ์เชิงเส้นตรง ดังสมการ

$$\text{Effort}(\text{Man} - \text{hours}) = 1.2232 * \text{CP}$$

ขั้นตอนที่ 2.4 เปรียบเทียบความแม่นยำระหว่างวิธีการประมาณขนาดที่แตกต่างกัน

หลังจากที่ได้ค่าความพยายามที่ได้จากวิธีการประมาณขนาดสี่วิธี คือ ฟังก์ชันพอยต์ ฟังก์ชันพอยต์เชิงวัตถุ ฟังก์ชันพอยต์เชิงวัตถุสอง และคลาสพอยต์ จะเปรียบเทียบความแม่นยำด้วยสมการของ Conte และคณะ เพื่อหาค่าความคลาดเคลื่อนจากค่าความพยายามในการพัฒนาซอฟต์แวร์ที่ประมาณได้กับค่าความพยายามในการพัฒนาซอฟต์แวร์ที่ใช้จริง ด้วยสมการหาค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ (Mean Magnitude of Relative Error: MMRE) (Conte et al., 1986)

ขั้นตอนที่ 2.5 สรุปผลการวิจัย

สรุปผลความแม่นยำของสี่วิธีการ โดยนำมาเปรียบเทียบความแม่นยำในแต่ละวิธีและค้นหาข้อสรุปของการวิจัย

3.2 ตัวแปรสำคัญที่ศึกษา

งานวิจัยนี้มีตัวแปรที่สนใจศึกษา แสดงรายละเอียดได้ดังนี้

3.2.1 ตัวแปรต้น

วัตถุประสงค์ของงานวิจัย คือ การเปรียบเทียบความแม่นยำของวิธีการประมาณขนาดซอฟต์แวร์เชิงวัตถุในการทำนายค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์เชิงวัตถุ ดังนั้นตัวแปรต้นของการศึกษาครั้งนี้ คือ วิธีการประมาณขนาดซอฟต์แวร์ ทั้งหมด 4 วิธีการ ดังนี้

1. วิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์ (Function point) (IFPUG, 1999)
2. วิธีการประมาณขนาดซอฟต์แวร์ด้วยวิธีฟังก์ชันพอยต์เชิงวัตถุ (Object-oriented Function point) (Antoniol et al., 1998)
3. วิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุสอง (Object-oriented Function point 2) (Zivkovic et al.,2005)
4. วิธีการประมาณขนาดซอฟต์แวร์ด้วยคลาสพอยต์ (Class point) (Costagliola et al., 2005)

3.2.2 ตัวแปรตาม

ในงานวิจัยนี้สนใจความแม่นยำของวิธีการประมาณขนาดซอฟต์แวร์เชิงวัตถุ โดยดูความแม่นยำจากค่าความพยายามในการพัฒนาซอฟต์แวร์ที่หาได้จากขนาดของระบบ ดังนั้นตัวแปรตามในงานวิจัยนี้ คือ ความแม่นยำของการประมาณค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์ นั่นคือ ค่าความพยายามจริงลบด้วยค่าความพยายามที่ประมาณได้ แต่เนื่องจากการคำนวณดังกล่าวจะทำให้เกิดค่าความคลาดเคลื่อนมากในกรณีที่ระบบที่ขนาดใหญ่ และเกิดความคลาดเคลื่อนน้อยในกรณีที่มีขนาดเล็ก จึงทำให้ Conte และคณะ ได้นำเสนอวิธีการหารเฉลี่ยขนาดของระบบจริง เพื่อลดความผิดพลาดในส่วนนี้ ซึ่งสามารถหาได้ด้วยวิธี คือ

ค่าเฉลี่ยของความคลาดเคลื่อนสัมพัทธ์ (Mean Magnitude of Relative Error: MMRE) (Conte et al., 1986)

$$MMRE = \frac{1}{n} \times \sum_{i=1}^{n} MRE$$

$$MRE = \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

y_i คือ ค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์ที่เกิดขึ้นจริง

$\hat{y}_i$ คือ ค่าความพยายามที่ได้จากวิธีการประมาณขนาดซอฟต์แวร์

ตัวแปรตามอีกหนึ่งตัวแปรที่ใช้วัดความแม่นยำของวิธีการประมาณขนาดซอฟต์แวร์ คือ อัตราร้อยละของระบบที่ประมาณค่าความพยายามได้ถูกต้อง (Percentage of predictions: PRED) คือ ร้อยละของระบบทั้งหมดที่มีค่าความคลาดเคลื่อนจากการประมาณค่าความพยายามได้ต่ำกว่าค่าขีดแบ่ง (Threshold) ซึ่งส่วนใหญ่จะใช้ 0.25 หรือให้ความคลาดเคลื่อนไม่เกิน 25% (Conte et al., 1986)

อัตราร้อยละของระบบที่ประมาณค่าความพยายามได้ถูกต้อง (Percentage of predictions: PRED) (Conte et al., 1986)

$$PRED(q) = \frac{k}{n}$$

q คือ ค่า MRE ที่ตั้งไว้ ค่าขีดแบ่ง (threshold) ในงานวิจัยส่วนใหญ่จะยอมรับที่ค่า 0.25

k คือ จำนวนระบบที่ MRE มีค่าน้อยกว่า หรือเท่ากับ q

n คือ จำนวนระบบทั้งหมด

ผู้วิจัยจึงเลือก การวิเคราะห์ข้อมูลใช้วิธีการ PRED(0.25) นั่นหมายความว่า วัดอัตราร้อยละของระบบที่ประมาณค่าความพยายามได้คลาดเคลื่อนน้อยกว่า 25%

3.2.3 ตัวแปรควบคุม

จากวัตถุประสงค์งานวิจัยนี้ คือ ความแม่นยำการประมาณค่าความพยายามที่ได้จากวิธีการประมาณขนาดซอฟต์แวร์เชิงวัตถุ ดังนั้นตัวแปรตามในงานวิจัยนี้คือ

1. ซอฟต์แวร์ที่ใช้เป็นหน่วยตัวอย่างเป็นชุดเดียวกัน ทำ 4 วิธีการประมาณขนาดซอฟต์แวร์ ทุกหน่วยตัวอย่างใช้ภาษาจาวา (Java) ในการพัฒนา
2. ค่าความพยายามที่ใช้พัฒนาซอฟต์แวร์จริงในแต่ละระบบ (Actual development effort) จะอยู่ในหน่วยของ คน-ชั่วโมง (Man-hours) เพื่อลดปัญหาช่องว่างระหว่างเวลาที่ใช้พัฒนาในแต่บริษัท
3. เครื่องมือที่ใช้ในการแปลงซอร์สโค้ด (Source code) เป็นแผนภาพคลาส (Class diagram) ใช้เครื่องมือเดียวกัน

3.3 สมมติฐานการวิจัย

เพื่อตอบวัตถุประสงค์ของงานวิจัย คือ การเปรียบเทียบความแม่นยำของวิธีการประมาณขนาดซอฟต์แวร์เชิงวัตถุที่แตกต่างกัน เนื่องจากขนาดซอฟต์แวร์สามารถวัดได้จากค่าความพยายามในการพัฒนาซอฟต์แวร์ ผู้วิจัยจึงเปรียบเทียบความแม่นยำจากค่าความพยายามที่ได้จากการประมาณขนาดซอฟต์แวร์กับค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์จริง หรือ

ความคลาดเคลื่อนสัมพัทธ์ (MRE) ของวิธีการประมาณขนาดซอฟต์แวร์ทั้งสี่วิธี คือ ฟังก์ชันพอยต์ ฟังก์ชันพอยต์เชิงวัตถุ ฟังก์ชันพอยต์เชิงวัตถุสอง และคลาสพอยต์ จึงทำให้ ผู้วิจัยได้ตั้งสมมติฐาน คือ

วิเคราะห์ความแม่นยำในการประมาณค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์เชิงวัตถุด้วยวิธีการประมาณขนาดซอฟต์แวร์ที่แตกต่างกัน

กำหนดให้ μ_1 คือ ค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ในการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์ (MRE)

μ_2 คือ ค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ในการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุ (MRE)

μ_3 คือ ค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ในการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุสอง (MRE)

μ_4 คือ ค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ในการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยคลาสพอยต์ (MRE)

จากวิธีการประมาณขนาดซอฟต์แวร์ที่แตกต่างกันซึ่งถูกนำเสนอออกมาเพื่อใช้ประมาณขนาดซอฟต์แวร์ คือ ฟังก์ชันพอยต์ (IFPUG, 1999) ฟังก์ชันพอยต์เชิงวัตถุ (Antoniol et al., 1998) ฟังก์ชันพอยต์เชิงวัตถุสอง (Zivkovic et al., 2005) และคลาสพอยต์ (Costagliola et al., 2005) ซึ่งในแต่ละวิธีการที่นำเสนอนั้นมีวิธีการคิดและคำนวณด้วยวิธีการที่แตกต่างกัน จึงทำให้ผู้วิจัยคาดว่า วิธีการประมาณขนาดซอฟต์แวร์ที่แตกต่างกันจะให้ผลความแม่นยำที่ต่างกัน

$$H_0: \mu_1 = \mu_2 = \mu_3 = \mu_4$$
$$H_1: \mu_i \neq \mu_j \quad \text{เมื่อ } i \neq j$$

การยอมรับ H_0 หมายถึง การประมาณขนาดซอฟต์แวร์ด้วยวิธีที่แตกต่างกันไม่มีผลต่อความแม่นยำของประมาณค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์

การปฏิเสธ H_0 หมายถึง การประมาณขนาดซอฟต์แวร์ด้วยวิธีที่แตกต่างกันมีผลต่อความแม่นยำของประมาณค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์

3.4 หน่วยตัวอย่าง

งานวิจัยนี้เป็นการทดสอบความแม่นยำของการประมาณขนาดซอฟต์แวร์ด้วยระบบซอฟต์แวร์เชิงวัตถุ เพื่อให้ได้หน่วยตัวอย่างที่มีคุณสมบัติตรงตามงานวิจัย ผู้วิจัยจึงเลือกใช้หน่วย

ตัวอย่าง คือ ระบบซอฟต์แวร์ที่พัฒนาด้วยภาษาจาวา (Java) ซึ่งเป็นภาษาโปรแกรมเชิงวัตถุ โดยเลือกระบบที่พัฒนาเสร็จสิ้นแล้ว เนื่องจากต้องการทดสอบความแม่นยำของวิธีการประมาณขนาดโดยพิจารณาจากค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์จริง ดังนั้น ระบบที่ได้จะต้องมีการเก็บข้อมูลค่าความพยายามที่ใช้ในการพัฒนาจริง

3.5 **เครื่องมือที่ใช้ในการวิจัย**

งานวิจัยนี้มีวัตถุประสงค์เพื่อวิเคราะห์เปรียบเทียบความแม่นยำในการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยวิธีการประมาณขนาดซอฟต์แวร์ที่แตกต่างกัน โดยเลือกใช้เครื่องมือดังต่อไปนี้

3.5.1 **เอกสารเพื่อเก็บข้อมูลค่าความซับซ้อนของระบบ**

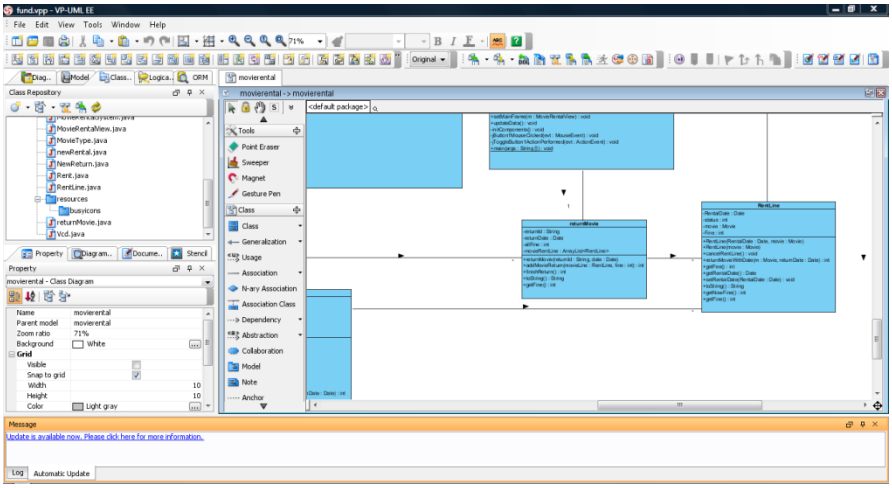
การประมาณขนาดด้วยฟังก์ชันพอยต์และคลาสพอยต์ต้องใช้ความซับซ้อนของลักษณะระบบหน่วยตัวอย่าง เป็นตัวแปรปรับค่าสำหรับการปรับเปลี่ยนขนาด ดังนั้นในงานวิจัยนี้ ผู้วิจัยจึงได้พัฒนาเอกสารซึ่งสามารถแสดงได้ในภาคผนวก ก. เพื่อใช้เก็บข้อมูลค่าความซับซ้อนของระบบที่จะใช้ในเป็นหน่วยตัวอย่างในงานวิจัย โดยเก็บข้อมูลค่าความซับซ้อนของระบบจากทีมผู้พัฒนาซอฟต์แวร์ ด้วยข้อถามจำนวน 18 ข้อถาม เป็นการกำหนดค่าคะแนนของช่วงน้ำหนักตามระดับความมีอิทธิพล 5 ระดับ และแนวทางในวิธีการกำหนดค่าความมีอิทธิพลซึ่งได้จากการอ้างอิงวิธีการคิดจากฟังก์ชันพอยต์ (IFPUG, 1999) และคลาสพอยต์ (Costaliola et al., 2005)

เอกสารที่ได้จากการกรอกจะนำไปใช้เพื่อคิดตัวแปรปรับค่าของฟังก์ชันพอยต์และคลาสพอยต์ โดยฟังก์ชันพอยต์จะมีตัวแปรปรับค่าจำนวน 14 ตัว ซึ่งแสดงในข้อถามของภาคผนวกข้อที่ 1-14 และคลาสพอยต์จะมีตัวแปรปรับค่าจำนวน 18 ตัว ซึ่งแสดงในข้อถามของภาคผนวกข้อที่ 1-18

3.5.2 **วิชวล พาราไดม์ สำหรับยูเอ็มแอล** (Visual Paradigm for UML)

เนื่องจากข้อมูลที่ต้องใช้ในการทดสอบในงานวิจัยนี้คือ แผนภาพคลาส (Class diagram) แต่เนื่องจากเก็บข้อมูลในธุรกิจไม่สามารถหาแผนภาพคลาสได้ ในงานวิจัยนี้ผู้วิจัยจึงเลือกใช้เครื่องมือเพื่อใช้ในการหาแผนภาพคลาสของซอร์สโค้ด โดยเลือกเครื่องมือ คือ วิชวล พาราไดม์ สำหรับยูเอ็มแอล (Visual Paradigm for UML) เนื่องจากสนับสนุนการออกแบบในระบบขนาดใหญ่ อีกทั้งสามารถส่งออก (Export) เป็นไฟล์ข้อมูลรูปแบบอื่นๆ หรือการสร้างเอกสารได้ เช่น ไฟล์เอกซ์เซล (EXL) ไฟล์พีดีแอฟ (PDF) ไฟล์รูปภาพ (image) สนับสนุนแผนภาพยูเอ็มแอลทุกรูปแบบ (UML Diagram) และสามารถทำวิศวกรรมย้อนกลับ (Reverse

engineering) จากภาษาจาวาเป็นแผนภาพคลาสได้แม่นยำและแสดงข้อมูลได้ครบถ้วนอีกด้วย (กานดา รุณนะพงศา, 2556)



รูปที่ 3.6 แสดงหน้าจอโปรแกรมวิชวล พาราไดม์ สำหรับยูเอ็มแอล (Visual Paradigm for UML)

3.5.3 ไมโครซอฟต์เอกซ์เซล (Mircosoft excel 2007)

การคำนวณขนาดที่ได้จากวิธีการประมาณขนาดซอฟต์แวร์แต่ละวิธีการ ผู้วิจัยเลือกใช้ไมโครซอฟต์เอกซ์เซลเพื่อคำนวณค่า เนื่องจากไมโครซอฟต์เอกซ์เซลสสามารถใส่สูตรคำนวณเพื่อรวมค่าขนาดซอฟต์แวร์ได้ถูกต้องแม่นยำและสะดวกสำหรับข้อมูลจำนวนมากๆ อีกทั้งยังสามารถวิเคราะห์ระดับความซับซ้อนด้วยสูตรการคำนวณตามเงื่อนไข (If-Else) ได้อีกด้วย

3.6 การเก็บรวบรวมข้อมูล

วัตถุประสงค์ของงานวิจัย คือ การเปรียบเทียบความแม่นยำของการประมาณค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์เชิงวัตถุด้วยวิธีการประมาณขนาดซอฟต์แวร์เชิงวัตถุที่แตกต่างกัน เพื่อให้สามารถช่วยเพิ่มความถูกต้องภายนอก (External Validity) และขยายผลได้จริงในธุรกิจการพัฒนาซอฟต์แวร์ ผู้วิจัยจึงเลือกเก็บรวบรวมจากข้อมูลในบริษัทที่มีการพัฒนาซอฟต์แวร์ โดยเลือกเก็บรวบรวมข้อมูลจากบริษัทที่พัฒนาระบบด้วยภาษาจาวา (Java) และมีการลงบันทึกค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์ซึ่งสามารถตรวจสอบได้ และพร้อมที่จะให้ข้อมูลได้ โดยเก็บข้อมูลรายละเอียดของระบบ รวมไปถึงค่าความซับซ้อนของระบบ ซึ่งผู้วิจัยได้พัฒนาเอกสาร ภาคผนวก ก. เพื่อใช้เก็บเอกสารค่าความซับซ้อนของระบบ โดยมีขั้นตอนการเก็บรวบรวมข้อมูลดังนี้

1. ค้นหาบริษัทที่มีการพัฒนาซอฟต์แวร์เชิงวัตถุด้วยภาษาจาวาและมีการเก็บค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์จริง

2. ชี้แจงรายละเอียดของงานวิจัยกับหน่วยตัวอย่าง โดยกล่าวถึงวัตถุประสงค์ของงานวิจัย และข้อมูลที่ต้องการในงานวิจัย
3. เก็บข้อมูลระบบย่อยของบริษัท ดังนี้
    1) เก็บข้อมูลเอกสารการออกแบบระบบ (System design specification) สำหรับใช้ในการคำนวณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์
    2) แผนภาพคลาส (Class diagram) หรือในกรณีที่ไม่มีการออกแบบด้วยแผนภาพคลาสจะใช้ซอร์สโค้ด (Source code) เพื่อใช้ในการทำวิศวกรรมย้อนกลับ (Reverse engineering) เป็นแผนภาพคลาส (Class diagram) สำหรับใช้คำนวณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุ ฟังก์ชันพอยต์เชิงวัตถุสอง และคลาสพอยต์
    3) ข้อมูลความซับซ้อนของระบบซึ่งได้จากการกรอกของทีมผู้พัฒนาดังเอกสารภาคผนวก เพื่อนำมาคำนวณตัวแปรปรับค่าที่จะใช้ในการปรับค่าขนาดซอฟต์แวร์ด้วยวิธีฟังก์ชันพอยต์ และคลาสพอยต์
    4) ค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์จริง (Actual development effort) ในกรณีที่ค่าความพยายามที่ใช้จริงมีหน่วยไม่ตรงกับคน-ชั่วโมง (Man-hour) จะสอบถามถึงชั่วโมงการทำงานในหนึ่งวันของบริษัทนั้น เพื่อแปลงหน่วยเป็นคน-ชั่วโมง

3.7 กรอบการวิเคราะห์ข้อมูล

การวิเคราะห์ข้อมูลเพื่อทดสอบสมมติฐานของการวิจัยนี้คือ การเปรียบเทียบความแม่นยำของวิธีการประมาณขนาดซอฟต์แวร์ด้วยวิธีที่แตกต่างกัน 4 วิธีการ คือ ฟังก์ชันพอยต์ ฟังก์ชันพอยต์เชิงวัตถุ ฟังก์ชันพอยต์เชิงวัตถุสอง และคลาสพอยต์

เนื่องจากงานวิจัยนี้มีตัวแปรเพียงตัวแปรเดียว คือ ค่าความคลาดเคลื่อนสัมพัทธ์ในการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยวิธีการประมาณขนาดจำนวน 4 วิธีการ คือ ฟังก์ชันพอยต์ ฟังก์ชันพอยต์เชิงวัตถุ ฟังก์ชันพอยต์เชิงวัตถุสองและคลาสพอยต์ จึงทำให้ผู้วิจัยจึงเลือกใช้การวิเคราะห์ความแปรปรวน (Analysis of Variance: ANOVA) ด้วยการวิเคราะห์ความแปรปรวนแบบทางเดียว (One-Way ANOVA) เมื่อทราบว่าความแปรปรวนที่ได้ไม่แตกต่างกัน ผู้วิจัยจะทดสอบในรายคู่ระหว่างแต่ละวิธีการด้วยค่า Scheffe แต่หาพบว่าความแปรปรวนที่ได้ต่างกัน ผู้วิจัยจะเลือกทดสอบในรายคู่ด้วยค่า Tamhane ซึ่งในงานวิจัยนี้ใช้ค่า Sig. (significant) 0.05 (กัลยา วาณิชย์บัญชา, 2553)

3.8 ความถูกต้องและความน่าเชื่อถือของข้อมูล

การตอบวัตถุประสงค์ของข้อมูลงานวิจัยให้ถูกต้อง (Validity) และเชื่อถือได้ (Reliability) จำเป็นต้องควบคุมปัจจัยที่เกี่ยวข้องอันได้แก่ การเลือกโครงการพัฒนาซอฟต์แวร์ การเก็บข้อมูลของโครงการ โดยในการทดลองมีปัจจัยที่ต้องควบคุมดังนี้

1. ข้อมูลที่ใช้ในงานวิจัย เป็นโครงการซอฟต์แวร์ที่เก็บรวบรวมมาจากการทำงานจริงในธุรกิจ ซึ่งเป็นบริษัทที่มีรายละเอียดการทำงานเก็บไว้อย่างครบถ้วน รวมไปถึงข้อมูลค่าความพยายามที่ใช้ในการพัฒนามีครบ ถูกต้องทุกระบบ ทุกฟังก์ชันและเป็นหน่วย คน-ชั่วโมง (Man-Hours) ซึ่งถือว่าละเอียดเพียงพอ
2. สำหรับวิธีการประมาณขนาดซอฟต์แวร์ด้วยวิธีของฟังก์ชันพอยต์ นักวิจัยได้ปฏิบัติตามคู่มือเอกสารวิธีการนับซอฟต์แวร์ที่เผยแพร่ของไอเอฟพียูจี (IFPUG) ซึ่งขั้นตอนดังกล่าวระบุละเอียดถึงวิธีการนับค่าของแต่ละวิธีการ ดังนั้นขนาดซอฟต์แวร์ที่วัดออกมาได้ จะถูกต้องและตรงกับมาตรฐาน
3. สำหรับวิธีการประมาณขนาดด้วยแผนภาพคลาส นักวิจัยได้วิศวกรรมย้อนกลับ (reverse engineering) มาจาก ซอร์สโค้ด (Source code) จึงมั่นใจได้ว่ามาจากระบบนั้นจริง และนักวิจัยได้ใช้วิชวล พาราไดม์ สำหรับ ยูเอ็มแอล (Visual Paradigm for UML) ซึ่งเป็นเครื่องมือที่วิศวกรรมย้อนกลับ (reverse engineering) กับอิคลิพซ์โดยสามารถทำได้บางส่วนกับ ซอร์สโค้ด (Source code) หรือทั้งหมด ซึ่งมีการสุ่มตรวจความถูกต้องของการนับจำนวนต่างๆ กับ ซอร์สโค้ด (Source code) จริง

บทที่ 4
ผลการวิเคราะห์ข้อมูล

4.1 ผลการเก็บรวบรวมข้อมูลจากบริษัท

หลังจากผู้วิจัยได้ดำเนินเก็บข้อมูลจากการทำงานจริงจาก 2 บริษัท (แหล่งข้อมูล) แหล่งละ 1 โครงการพัฒนาซอฟต์แวร์ รวมจำนวนทั้งสิ้น 23 ระบบย่อย ทุกระบบย่อยพัฒนาด้วยภาษาจาวา โดยมีส่วนประสานต่อผู้ใช้งานเป็นเว็บไซต์พัฒนาด้วยภาษาเจเอสพี (JSP) ระบบย่อยทุกระบบเป็นระบบที่พัฒนาขึ้นมาใหม่ทั้งหมด

ระบบย่อยจากแหล่งข้อมูลที่ 1 จำนวน 11 ระบบย่อย มาจากโครงการวางแผนทรัพยากรของธุรกิจการขายทองแดง ซึ่งประกอบไปด้วยระบบย่อย อันได้แก่ การบริหารจัดการคลังสินค้าทองแดง จำนวน 4 ระบบย่อย การบริหารจัดการสต๊อคทองแดง จำนวน 2 ระบบย่อย และการจัดส่งสินค้า จำนวน 5 ระบบย่อย มีค่าความพยายามที่ใช้ในการพัฒนา (Development effort) แต่ละระบบย่อยอยู่ระหว่าง 40-133.6 คน-ชั่วโมง และค่าเฉลี่ยอยู่ที่ 80.12 คนชั่วโมง รายละเอียดระบบย่อยของแหล่งข้อมูลที่ 1 แสดงดังตารางที่ 4.1 ในระบบที่ 1-1 ถึง 1-11

ระบบย่อยจากแหล่งข้อมูลที่ 2 จำนวน 12 ระบบย่อย มาจากโครงการบริหารกองทุน ซึ่งประกอบไปด้วยระบบย่อยของการจัดการข้อมูลภายในองค์กร ซึ่งได้แก่ การจัดการบริษัทกองทุน การจัดการธนาคาร การจัดการสาขาธนาคาร การจัดการอาชีพของลูกค้า การจัดการที่อยู่ไปรณีย์ การจัดการผู้สอบบัญชี การจัดการผู้ดูแลผลประโยชน์ การจัดการนโยบายกองทุน การจัดการการยกเลิกหน่วยลงทุน การจัดการโอนเงิน การจัดการเช็ค และการจัดการเงื่อนไขกองทุน โดยมีค่าความพยายามที่ใช้ในการพัฒนา (Development effort) แต่ละระบบย่อยอยู่ระหว่างอยู่ที่ 28-96 คน-ชั่วโมง และค่าเฉลี่ยอยู่ที่ 41.8 คนชั่วโมง รายละเอียดระบบย่อยของแหล่งข้อมูลที่ 2 แสดงดังตารางที่ 4.1 ในระบบที่ 2-1 ถึง 2-12

เมื่อรวมข้อมูลจากทั้ง 2 แหล่งข้อมูลค่าความพยายามที่ใช้ในการพัฒนา (Development effort) แต่ละระบบย่อยอยู่ที่ 28-133.6 คน-ชั่วโมง โดยมีค่าเฉลี่ยอยู่ที่ 63.55 คน-ชั่วโมง

ตารางที่ 4.1 แสดงข้อมูลรายละเอียดระบบที่ใช้เป็นหน่วยตัวอย่าง

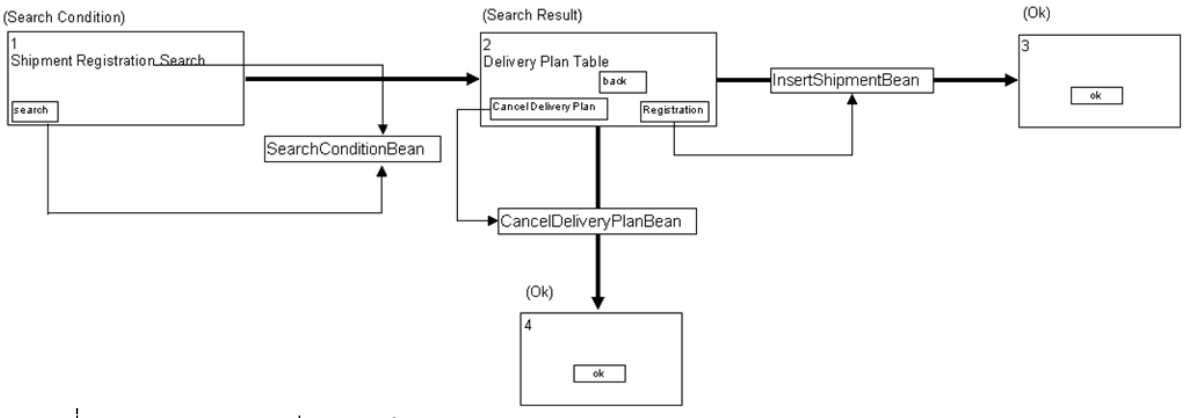
	EFH_{actual}	Number of Class	Number of Attribute	Number of Method
1-1	132.8	9	920	81
1-2	88.8	11	559	69
1-3	133.6	13	1410	99
1-4	96	9	730	53
1-5	105.6	5	527	33
1-6	105.6	6	386	25
1-7	64	10	437	60
1-8	65.6	15	537	87
1-9	40	4	98	11
1-10	64	6	367	31
1-11	64	10	297	41
2-1	36	6	51	30
2-2	34	6	15	19
2-3	40	6	28	22
2-4	48	6	34	35
2-5	38	6	13	13
2-6	96	6	79	42
2-7	36	6	15	17
2-8	33.6	6	14	18
2-9	28	6	16	18
2-10	40	6	18	21
2-11	28	6	12	17
2-12	44	6	20	20

4.1.1 เอกสารที่จัดเก็บจากแหล่งข้อมูลที่ 2

1. **เอกสารการออกแบบส่วนประสานผู้ใช้งาน** (UI Document)

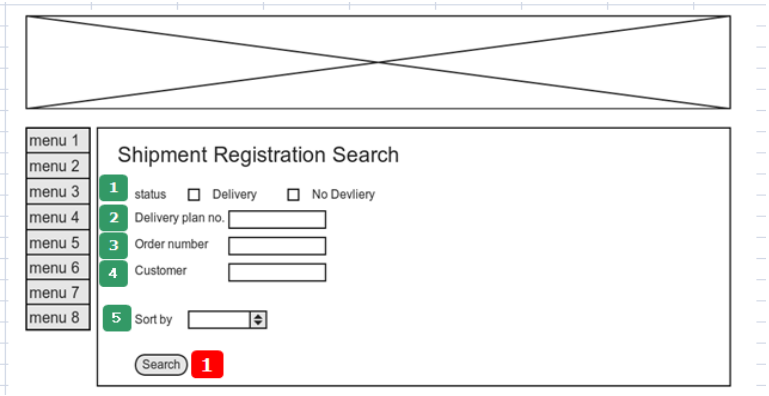
เอกสารแสดงรายละเอียดการออกแบบระบบจัดทำด้วยไฟล์ไมโครซอฟต์เอกซ์เซล (Mircosoft excel) ของแต่ละระบบย่อย ระบบย่อยละ 1 ไฟล์

ใน 1 ไฟล์ จะมีรายละเอียดของทำงานของระบบ จำนวน 4 แผ่นงาน (Work Sheet) แผ่นงานที่ 1 แสดงแผนผังการทำงานของระบบ การทำงานของหน้าจอว่าเชื่อมต่อกับส่วนใดและจะแสดงหน้าจอส่วนใดต่อ เอกสารดังกล่าวแสดงตัวอย่างดังรูปที่ 4.1



รูปที่ 4.1 แสดงรายละเอียดการทำงานของระบบจากเอกสารการออกแบบส่วนประสานต่อผู้ใช้งานในแหล่งข้อมูลที่ 1

แผ่นงานที่ 2 แสดงรายละเอียดหน้าจอ รายละเอียดที่แสดงจะประกอบด้วยส่วนประกอบในหน้าจอนั้นๆ โดยมีการกำหนดเลขของข้อมูลและปุ่ม (Button) ซึ่งจะสามารถแสดงรายละเอียดในแผ่นงานที่ 3 และ แผ่นงานที่ 4 เอกสารดังกล่าวแสดงตัวอย่างดังรูปที่ 4.2



รูปที่ 4.2 แสดงหน้าจอของระบบจากเอกสารการออกแบบส่วนประสานต่อผู้ใช้งานในแหล่งข้อมูลที่ 1

แผ่นงานที่ 3 แสดงรายละเอียดของข้อมูลที่แสดงในหน้าจอ โดยเรียงตามเลขที่แสดงไว้ในแต่ละหน้าจอ เอกสารดังกล่าวแสดงตัวอย่างดังรูปที่ 4.3

1 Shipment Registration Search

			Default value				Validate						
No	Name	Attribute	Value	Master	itemValue	itemLabel	Mandatory	character	MAX Value	Min Value	# of decimal point digit	type	Format
1	Status	CheckBox		Program	1.checked 2.checked	1.Delivery 2.No Delivery	×	half-size	-	-	-	string	-
2	Delivery Plan No	TextBox	null	-	-	-	×	half-size	12	12	-	string	
3	Order No	ComboBox	null	CustomerBean	customer_code	customer_code	×	half-size	5	-	-	string	
4	Customer	TextBox	null	-	-	-	×	half-size	-	-	-	string	-
5	Sort by	RadioButton	Ascending	Program	ASC OR DESC	Ascending or Descending	×	half-size	-	-	-	string	

รูปที่ 4.3 แสดงรายละเอียดของข้อมูลที่แสดงในหน้าจอของระบบจากเอกสารการออกแบบส่วนประสานต่อผู้ใช้งานในแหล่งข้อมูลที่ 1

แผ่นงานที่ 4 แสดงรายละเอียดการทำงานของปุ่ม (Button) โดยเรียงตามเลขที่แสดงไว้ในแต่ละหน้าจอ เอกสารดังกล่าวแสดงตัวอย่างดังรูปที่ 4.4

1 Shipment Registration Search

Button No	Button Name (English)	method
1	Search	Search condition Status, Delivery Plan No, Order No, Customer, from Order table

รูปที่ 4.4 แสดงรายละเอียดของปุ่มที่แสดงในหน้าจอของระบบจากเอกสารการออกแบบส่วนประสานต่อผู้ใช้งานในแหล่งข้อมูลที่ 1

2. **เอกสารการออกแบบฐานข้อมูล** (Data Document)

เอกสารแสดงรายละเอียดของฐานข้อมูล (Database) เป็นไมโครซอฟต์เอกซ์เซล (Mircosoft excel) ซึ่งประกอบด้วยแผนงานจำนวน 2 ส่วน

ส่วนแรกแสดงตารางฐานข้อมูลทั้งหมด เอกสารดังกล่าวแสดงตัวอย่างดังรูปที่ 4.5

No.	Table	Name
1	M_Order	Order
2	M_Customer	Customer
3	M_Product	Product
4	M_Delivery	Delivery
5	M_User	User

รูปที่ 4.5 แสดงตารางฐานข้อมูลจากเอกสารการออกแบบฐานข้อมูลในแหล่งข้อมูลที่ 1

ส่วนที่สองแสดงรายละเอียดในแต่ละตารางฐานข้อมูล (Table) ว่ามีฟิลด์ใดบ้าง พร้อมข้อมูลตัวอย่าง เอกสารดังกล่าวแสดงตัวอย่างดังรูปที่ 4.6

Order						
Order_id	Order_no	Customer_id	Product_code	Amount	Insert_by	Delivery_no
1	A0001	1	J22	100	2	1
2	A0002	2	J22	100	1	1
3	A0003	3	J22	100	1	1
4	A0004	1	J23	100	1	1
5	A0005	2	J24	300	1	1
6	A0006	2	J25	100	2	1
7	A0007	4	J26	100	1	1
8	A0008	2	J27	100	1	1
9	A0009	2	J28	100	1	1
10	A0010	3	J29	10	3	1
11	A0011	1	J30	400	4	1

รูปที่ 4.6 แสดงรายละเอียดในแต่ละตารางฐานข้อมูลจากเอกสารการออกแบบฐานข้อมูลในแหล่งข้อมูลที่ 1

3. **แผนภาพคลาส** (Class diagram)

เนื่องจากบริษัทนี้ไม่ได้ใช้แผนภาพคลาสในการออกแบบ ผู้วิจัยจึงขอซอร์สโค้ด (Source code) เพื่อนำมาแปลงกลับเป็นแผนภาพคลาส (Class diagram) ด้วยวิชวล พาราไดม์ สำหรับยูเอ็มแอล (Visual Paradigm for UML) โดยซอร์สโค้ด (Source code) ที่เก็บมาได้ จะแยกแพคเกจ (Package) ตามระบบย่อย ซึ่งแต่ละระบบย่อยสามารถทำงานได้ด้วยตนเอง ไม่มีการเรียกใช้ส่วนอื่นๆ ของแพคเกจ (Package)

4. **ข้อมูลค่าความพยายามที่ใช้ในการพัฒนาแต่ละระบบย่อย**

สำหรับค่าความพยายามที่ใช้ในระบบย่อยที่มีการจัดเก็บจากแหล่งข้อมูลที่ 1 เป็นค่าความพยายามในหน่วยวันต่อคน (Man-days) แต่เนื่องจาก งานวิจัยนี้กำหนดหน่วยของค่าความพยายามเป็นชั่วโมงต่อคน (Man-hours) จึงเปลี่ยนค่าความพยายามที่ได้จากวันต่อคน (Man-days) เป็นชั่วโมงต่อคน (Man-hours) โดยพบว่า บริษัทดังกล่าวมีการทำงาน 8 ชั่วโมงต่อหนึ่งวัน โดยผู้พัฒนา 11 ระบบย่อยนี้ มีจำนวน 3 คน โดยผู้พัฒนาแต่ละคนรับผิดชอบแต่ละระบบย่อย

5. **ค่าความซับซ้อนของโครงการซอฟต์แวร์**

เนื่องจากวิธีการประมาณด้วยฟังก์ชันพอยต์ (Funtion point) และคลาสพอยต์ (Class point) ต้องมีค่าความซับซ้อน ซึ่งนักวิจัยจึงให้ผู้พัฒนาในทีมพัฒนาซอฟต์แวร์ประเมินความซับซ้อนของโครงการซอฟต์แวร์ ตามลักษณะของซอฟต์แวร์ที่ระบุไว้ในเอกสารภาคผนวก ค่าที่ได้จากการจัดเก็บแสดงได้ดังตารางที่ 4.2

ตารางที่ 4.2 แสดงรายละเอียดค่าความซับซ้อนของโครงการซอฟต์แวร์จากแหล่งข้อมูลที่ 1

ลักษณะของระบบ	ระดับความมีอิทธิพล					
	0	1	2	3	4	5
	ไม่มี	เล็กน้อย	น้อย	ปานกลาง	มาก	มากที่สุด
1. Data Communications					√	
2. Distributed Data Processing				√		
3. Performance		√				
4. Heavily Used Configuration		√				
5. Transaction Rate	√					
6. Online Data Entry			√			
7. End-User Efficiency			√			
8. Online Update			√			
9. Complex Processing				√		
10. Reusability		√				
11. Installation Ease	√					
12. Operational Ease			√			
13. Multiple Sites			√			
14. Facilitate Change				√		
15. User Adaptivity		√				
16. Rapid Prototyping			√			
17. Multiple Interfaces		√				
18. Multiuser Interactivity			√			

เมื่อนำมาคำนวณสำหรับตัวแปรปรับค่าที่ใช้ในการประมาณขนาดด้วยฟังก์ชันพอยต์ จะได้ค่าผลรวมของค่าระดับความมีอิทธิพลของปัจจัยที่เกี่ยวข้องกับการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ (TDI) ตั้งแต่ลักษณะที่ 1-14 โดยมีค่ารวมเท่ากับ 26

เมื่อนำมาคำนวณในสูตรที่ใช้เป็นตัวแปรปรับค่าของฟังก์ชันพอยต์ คือ

VAF = (TDI * 0.01) + 0.65

แทนค่าได้ ดังนี้ VAF = (26*0.01) + 0.65 = 0.91

ดังนั้นค่าสำหรับตัวแปรปรับค่าที่ใช้ในการประมาณขนาดฟังก์ชันพอยต์คือ 0.91

เมื่อนำมาคำนวณสำหรับตัวแปรปรับค่าที่ใช้ในการประมาณขนาดด้วยคลาสพอยต์ จะได้ค่าผลรวมของค่าระดับความมีอิทธิพลของปัจจัยที่เกี่ยวข้องกับการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ (TDI) ตั้งแต่ลักษณะที่ 1-18 โดยมีค่ารวมเท่ากับ 32

เมื่อนำมาคำนวณในสูตรที่ใช้เป็นตัวแปรปรับค่าของคลาสพอยต์ คือ

TCF = 0.55 + (0.01*TDI)

แทนค่าได้ ดังนี้ TCF = 0.55 + (0.01*32) = 0.87

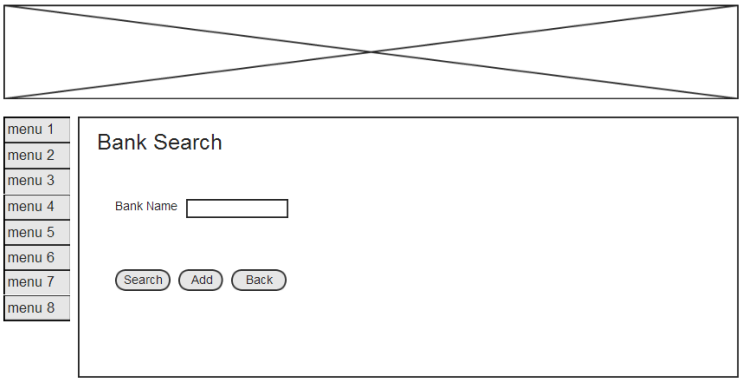
ดังนั้นค่าสำหรับตัวแปรปรับค่าที่ใช้ในการประมาณขนาดคลาสพอยต์คือ 0.87

4.1.2 เอกสารที่จัดเก็บจากแหล่งข้อมูลที่ 2

1. เอกสารการออกแบบระบบ

เอกสารดังกล่าวที่เก็บได้อยู่ในรูปแบบไมโครซอฟต์เวิร์ด (Mircrosoft word) โดยมีรายละเอียดที่แสดงดังนี้

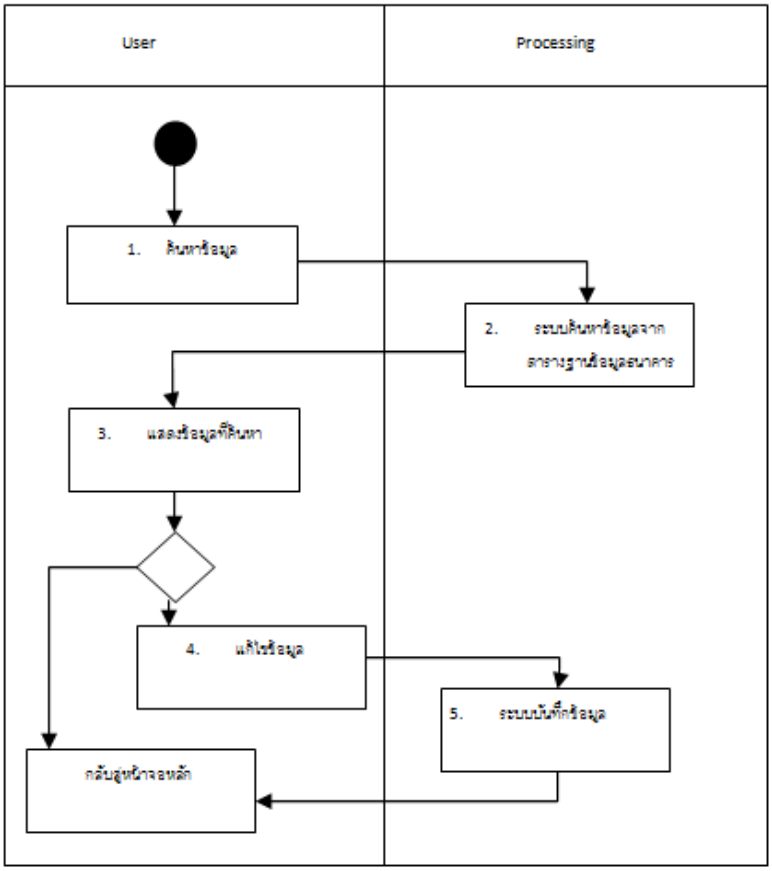
ส่วนที่ 1 แสดงรายละเอียหน้าจอการออกแบบระบบ เอกสารส่วนนี้จะมีหน้าจอแต่ละหน้าจอของแต่ละระบบย่อยแยกออกส่วน แสดงตัวอย่างดังรูปที่ 4.7



รูปที่ 4.7 แสดงหน้าจอของระบบจากเอกสารการออกแบบส่วนประสานต่อผู้ใช้งานในแหล่งข้อมูลที่ 2

ส่วนที่ 2 แผนภาพกิจกรรม (Activity diagram) แสดงการทำงานของระบบย่อยในรูปแบบกิจกรรมการทำงานในแต่ละระบบย่อย รวมไปถึงมีรายละเอียดของแต่ละกิจกรรม (Activity description) แสดงตัวอย่างดังรูปที่ 4.8 และรูปที่ 4.9

Activity Diagram



รูปที่ 4.8 แสดงแผนภาพกิจกรรมของระบบจากเอกสารการออกแบบในแหล่งข้อมูลที่ 2

Description

Activity	Description
1. ค้นหาข้อมูล	รับข้อมูลการค้นหา รหัสธนาคาร ชื่อธนาคาร และผู้ใช้งานกดปุ่มค้นหา
2. ระบบค้นหาข้อมูล	ระบบนำข้อมูลที่ผู้ใช้งานป้อนเข้ามาค้นหากับตารางฐานข้อมูลธนาคาร
3. แสดงข้อมูลที่ค้นหา	แสดงข้อมูลธนาคารที่ค้นหาได้ โดยแสดงตามลำดับรหัสธนาคาร
4. แก้ไขข้อมูล	เมื่อผู้ใช้งานเลือกธนาคารที่ต้องการค้นหา จะแสดงข้อมูลที่มีอยู่และสามารถพิมพ์ข้อมูลเพื่อเปลี่ยนแปลงได้
5. ระบบบันทึกข้อมูล	ระบบบันทึกข้อมูลตามรหัสธนาคารที่ถูกเลือก
6. กลับสู่หน้าจอหลัก	กลับสู่หน้าจอเมนูหน้าแรก "index"

รูปที่ 4.9 แสดงรายละเอียดของแผนภาพกิจกรรมของระบบจากเอกสารการออกแบบในแหล่งข้อมูลที่ 2

ส่วนที่ 3 ฐานข้อมูล (Database) จะแสดงฐานข้อมูลโดยมีรายละเอียดของตารางและฟิลด์ในตารางที่เกี่ยวข้องในแต่ละระบบย่อย แสดงตัวอย่างดังรูปที่ 4.10

ตารางแสดงฐานข้อมูลที่แสดงในหน้าจอการค้นหา

ลำดับที่	ชื่อตาราง	ชื่อฟิลด์	ชื่อข้อมูล
1	Bank	Bank_Code	รหัสธนาคาร
2	Bank	Bank_NameT	ชื่อธนาคารภาษาไทย
3	Bank	Bank_NameE	ชื่อธนาคารภาษาอังกฤษ

ตารางแสดงฐานข้อมูลที่แสดงในหน้าจอการเพิ่มหรือแก้ไขธนาคาร

ลำดับที่	ชื่อตาราง	ชื่อฟิลด์	ชื่อข้อมูล	Mandatory
1	Bank	Bank_Code	รหัสธนาคาร	Y
2	Bank	Bank_NameT	ชื่อธนาคารภาษาไทย	Y
3	Bank	Bank_NameE	ชื่อธนาคารภาษาอังกฤษ	Y
4	Bank	Bank_Type	ชนิดธนาคาร	Y

รูปที่ 4.10 แสดงรายละเอียดฐานข้อมูลของระบบจากเอกสารการออกแบบในแหล่งข้อมูลที่ 2

2. **แผนภาพคลาส** (Class diagram)

เนื่องจากแหล่งข้อมูลนี้ไม่ได้ใช้แผนภาพคลาสในการออกแบบ ผู้วิจัยจึงขอซอร์สโค้ด (Source code) เพื่อนำมาแปลงกลับเป็นแผนภาพคลาส (Class diagram) ด้วยวิชวล พาราไดม์ สำหรับ ยูเอ็มแอล (Visual Paradigm for UML) โดยซอร์สโค้ด (Source code) ที่เก็บมาได้ จะแยกผู้วิจัยตั้งใจจะเก็บแผนภาพคลาส (Class diagram) เพื่อใช้นับวิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุ ฟังก์ชันพอยต์เชิงวัตถุสองและคลาสพอยต์

3. **ค่าความพยายามที่ใช้ในการพัฒนาแต่ละระบบย่อย**

สำหรับค่าความพยายามที่ใช้ในการพัฒนาระบบย่อยที่มีการจัดเก็บในแหล่งข้อมูลที่ 2 เป็นค่าความพยายามในหน่วยชั่วโมงต่อคน (Man-Hours) โดยในแต่ละระบบย่อยย่อย มีผู้พัฒนาเพียงคนเดียว นั่นหมายถึงแต่ละระบบย่อยจะถูกพัฒนาเพียง 1 คน

4. **ค่าความซับซ้อนของโครงการซอฟต์แวร์**

เนื่องจากวิธีการประมาณด้วยฟังก์ชันพอยต์ (Function point) และคลาสพอยต์ (Class point) จะต้องมีค่าความซับซ้อนของลักษณะโครงการซอฟต์แวร์ ซึ่งนักวิจัยให้ผู้พัฒนาประเมินความซับซ้อนของโครงการซอฟต์แวร์นี้ ตามเอกสารในภาคผนวก ค่าที่ได้จากการจัดเก็บแสดงได้ดังตารางที่ 4.3

ตารางที่ 4.3 แสดงรายละเอียดค่าความซับซ้อนของโครงการซอฟต์แวร์จากแหล่งข้อมูลที่ 2

ลักษณะของระบบ	ระดับความมีอิทธิพล					
	0	1	2	3	4	5
	ไม่มี	เล็กน้อย	น้อย	ปานกลาง	มาก	มากที่สุด
1. Data Communications				√		
2. Distributed Data Processing	√					
3. Performance		√				
4. Heavily Used Configuration			√			
5. Transaction Rate	√					
6. Online Data Entry		√				
7. End-User Efficiency		√				
8. Online Update		√				
9. Complex Processing		√				
10. Reusability	√					
11. Installation Ease	√					
12. Operational Ease		√				
13. Multiple Sites			√			
14. Facilitate Change	√					
15. User Adaptivity			√			
16. Rapid Prototyping		√				
17. Multiple Interfaces		√				
18. Multiuser Interactivity			√			

เมื่อนำมาคำนวณสำหรับตัวแปรปรับค่าที่ใช้ในการประมาณขนาดด้วยฟังก์ชันพอยต์ จะได้ค่าผลรวมของค่าระดับความมีอิทธิพลของปัจจัยที่เกี่ยวข้องกับการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ (TDI) ตั้งแต่ลักษณะที่ 1-14 โดยมีค่ารวมเท่ากับ 13

เมื่อนำมาคำนวณในสูตรที่ใช้เป็นตัวแปรปรับค่าของฟังก์ชันพอยต์ (IFPUG, 1999) คือ

$$VAF = (TDI * 0.01) + 0.65$$

แทนค่าได้ ดังนี้ $VAF = (13*0.01) + 0.65 = 0.78$

ดังนั้นค่าสำหรับตัวแปรปรับค่าที่ใช้ในการประมาณขนาดฟังก์ชันพอยต์คือ 0.78

เมื่อนำมาคำนวณสำหรับตัวแปรปรับค่าที่ใช้ในการประมาณขนาดด้วยคลาสพอยต์ จะได้ค่าผลรวมของค่าระดับความมีอิทธิพลของปัจจัยที่เกี่ยวข้องกับการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ (TDI) ตั้งแต่ลักษณะที่ 1-18 โดยมีค่ารวมเท่ากับ 19

เมื่อนำมาคำนวณในสูตรที่ใช้เป็นตัวแปรปรับค่าของคลาสพอยต์ (Costagliola et al., 2005) คือ

$$TCF = 0.55 + (0.01*TDI)$$

แทนค่าได้ ดังนี้ $TCF = 0.55 + (0.01*19) = 0.74$

ดังนั้นค่าสำหรับตัวแปรปรับค่าที่ใช้ในการประมาณขนาดคลาสพอยต์คือ 0.74

4.2 นับขนาดซอฟต์แวร์ด้วยสี่วิธีการประมาณ

หลังจากผู้วิจัยเก็บข้อมูลจากแหล่งข้อมูลทั้งสองแหล่งแล้ว จึงนำมานับขนาดของระบบย่อยที่ได้ในแต่ละระบบย่อย โดยแบ่งเป็น 2 วิธีการแยกตามประเภทของข้อมูลที่ใช้ คือ นับขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์ซึ่งใช้เอกสารการออกแบบระบบ และนับขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุ ฟังก์ชันพอยต์เชิงวัตถุสอง และคลาสพอยต์ ซึ่งใช้แผนภาพคลาส

4.2.1 นับขนาดจากวิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์

ผู้วิจัยพัฒนาเอกสารขึ้นเพื่อใช้บันทึกรายละเอียดสำหรับการคำนวณฟังก์ชันพอยต์ในแต่ละระบบย่อย ในส่วนแรกใช้นับลอจิคอลไฟล์ (Logical file) ด้วยการกรอกชื่อ และ วงประเภทของไฟล์ข้อมูลนั้น ว่าเป็นอินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) หรือเอ็กซ์เทอร์นอลอินเตอร์เฟสไฟล์ (External Interface Files: EIFs) และกรอกจำนวนองค์ประกอบของข้อมูล (Data element types: DET) และเรคคอร์ดข้อมูล (Record element types: RET) ส่วนช่องสุดท้ายคือ ค่าความซับซ้อนที่ประเมินมาได้ โดยจะวงค่าความซับซ้อนนั้น L(Low)

A(Average) H(High) โดยจัดพิมพ์เอกสารออกมาในรูปแบบกระดาษเพื่อง่ายต่อการจด เอกสารดังกล่าว สามารถแสดงได้ดังรูปที่ 4.11

ระบบย่อยที่

เวลาที่ใช้

	ชื่อ	ประเภท	จำนวน DET	จำนวน RET	C*
1		ILF EIF			L A H
2		ILF EIF			L A H
3		ILF EIF			L A H
4		ILF EIF			L A H
5		ILF EIF			L A H
6		ILF EIF			L A H
7		ILF EIF			L A H

	หน้าจอ	EI			EQ			EO		
		DET	FTR	C*	DET	FTR	C*	DET	FTR	C*
1				L A H			L A H			L A H
2				L A H			L A H			L A H
3				L A H			L A H			L A H
4				L A H			L A H			L A H
5				L A H			L A H			L A H
6				L A H			L A H			L A H
7				L A H			L A H			L A H
8				L A H			L A H			L A H

ชนิด	LOW	AVERAGE	HIGH	SUM
ILF	7	10	15	
ELF	5	7	10	
EI	3	4	6	
EQ	3	4	6	
EO	4	5	7	
			รวม	FP

รูปที่ 4.11 แสดงเอกสารที่ใช้ในการนับขนาดด้วยฟังก์ชันพอยต์

ตัวอย่างการนับจำนวนฟังก์ชันพอยต์

เนื่องจากข้อมูลจริงที่เก็บได้จากบริษัทไม่สามารถเปิดเผยข้อมูลได้ ผู้วิจัยจึงพัฒนาเอกสารที่ใกล้เคียงมาแสดงตัวอย่างในการคำนวณ โดยมีรายละเอียดดังนี้

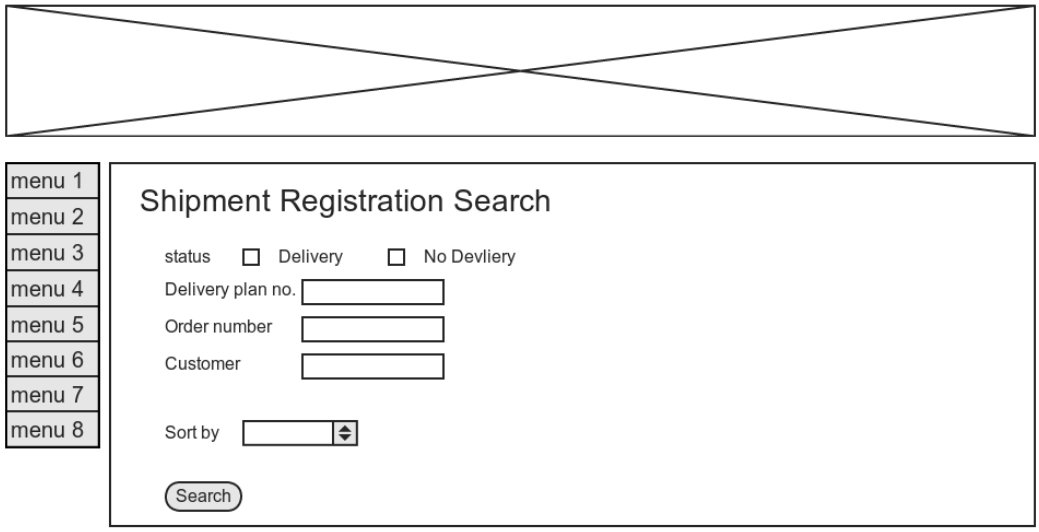
ระบบเพื่อลงทะเบียนการจัดส่งสินค้า มีจำนวนทั้งสิ้น 3 หน้าจอ ดังนี้

หน้าจอที่ 1 รับข้อมูลเพื่อค้นหารายการเพื่อจัดส่ง โดยสามารถเลือกกรอกข้อมูลเพื่อค้นหาได้ 4 รายละเอียด คือ

- เช็คบ็อก (Check-box) เพื่อเลือกสถานะรายการสั่งซื้อ (Order) ว่าถูกจัดส่งหรือยัง ซึ่งสามารถเลือกอย่างใดอย่างหนึ่งหรือเลือกทั้งสองก็ได้
- หมายเลขแผนการจัดส่ง
- หมายเลขคำสั่งซื้อ
- ชื่อลูกค้า

และดรอปดาวน์ลิส (Drop down list) สำหรับตั้งเงื่อนไขการแสดงผลได้ว่าเรียงลำดับตามข้อมูล จาก Delivery plan no. หรือ Order number หรือ Customer

หน้าจอที่ 1 แสดงดังรูปที่ 4.12



รูปที่ 4.12 แสดงตัวอย่างหน้าจอที่ 1

หน้าจอที่ 2 แสดงผลที่ได้จากการค้นหา โดยแสดงเป็นตารางที่เรียงตามลำดับการค้นหาประกอบด้วยข้อมูลดังนี้

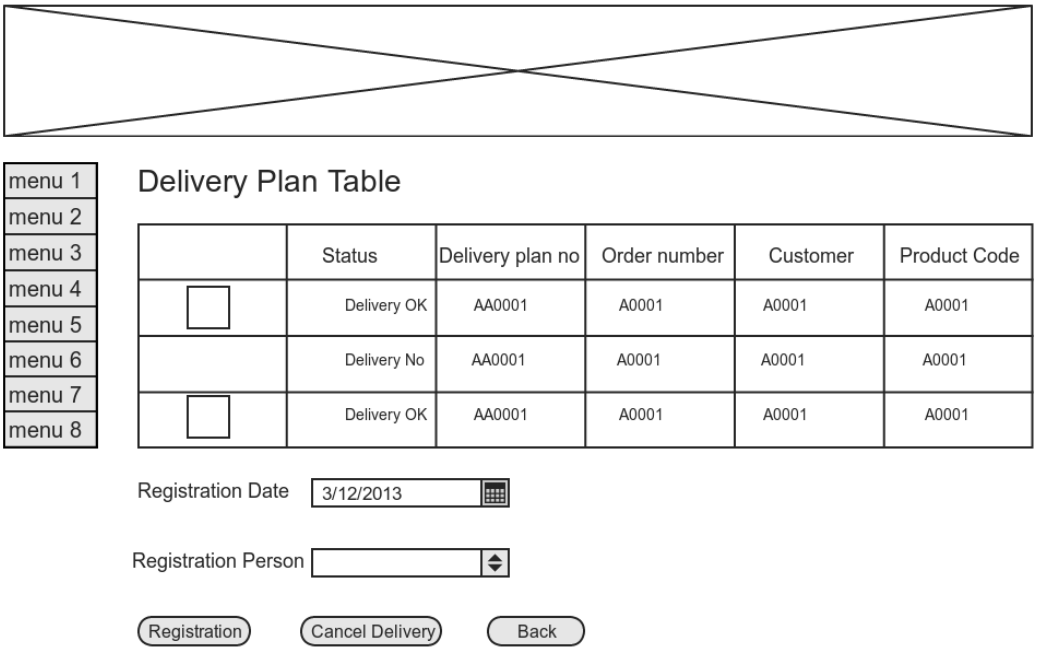
- เช็คบ็อก (Check-box) เพื่อเลือกรายการคำสั่งซื้อนี้หรือไม่
- สถานะสินค้าว่าครบสามารถจัดส่งได้หรือไม่

- หมายเลขแผนการจัดส่ง
- หมายเลขคำสั่งซื้อ
- ชื่อลูกค้า
- โค้ดสินค้าที่สั่งซื้อ

ข้อมูลด้านล่าง คือ วันที่ลงทะเบียนการจัดส่ง และ ผู้ลงทะเบียน ซึ่งเป็นดรอปดาว์นลิส (Drop down list) ซึ่งจะดึงข้อมูลมาจากข้อมูลผู้ใช้งานที่อยู่ในฐานข้อมูล

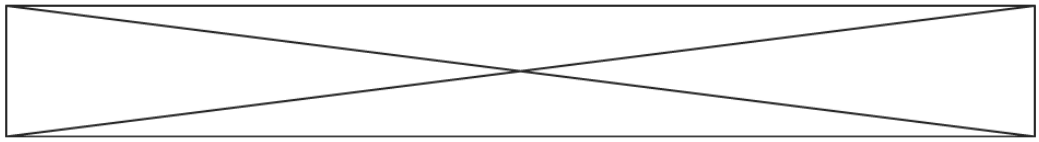
ปุ่มเพื่อลงทะเบียน และปุ่มเพื่อยกเลิกการลงทะเบียนการ โดยจะบันทึกข้อมูลในรายการคำสั่งซื้อที่เลือกไว้ด้านบนในเช็คบ็อก (Check-box) ช่องแรก

หน้าจอที่ 2 แสดงดังรูปที่ 4.13



รูปที่ 4.13 แสดงตัวอย่างหน้าจอที่ 2

หน้าจอที่ 3 แสดงข้อความเมื่อระบบบันทึกข้อมูลเรียบร้อยแล้วและปุ่ม OK เพื่อกลับสู่เมนูหลัก แสดงดังรูปที่ 4.14



menu 1
menu 2
menu 3
menu 4
menu 5
menu 6
menu 7
menu 8

Shipment Registration Finish

OK

รูปที่ 4.14 แสดงตัวอย่างหน้าจอที่ 3

โดยมีฐานข้อมูลดังรูปที่ 4.15

Order

Order_id	Order_no	Customer_id	Product_code	Amount	Insert_by	Delivery_no
1	A0001	1	J22	100	1	1
2	A0002	2	J22	100		
3	A0003	3	J22	100		

Customer

Customer_id	Customer_name	Customer_address
1	AA	xxxxxxxx
2	BB	xxxxxxxx
3	CC	xxxxxxxx

Product

Product_id	Product_code	Product_name	Amount_instock	Amount_order
1	J22	xxxx	1200	220
2	J23	xxxx	100	10

Delivery

Delivery_no	Delivery_date	Update_delivery_by
1	12/03/2013	1

User

User_id	User_name	User_level_permission
1	AAA	1
2	BBB	2

รูปที่ 4.15 แสดงตัวอย่างฐานข้อมูล

เมื่อนำมาคำนวณด้วยวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์ สามารถคำนวณได้ดังนี้

ขั้นตอนที่ 1 ประเมินค่าความซับซ้อนของลอจิคอลไฟล์ (Logical file)

สำหรับระบบตัวอย่างมี ไฟล์ข้อมูลที่เกี่ยวข้องจำนวน 5 ไฟล์ข้อมูล โดยแบ่งตามชนิดได้ดังนี้

ตารางฐานข้อมูล Order เป็น 1 อินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) เนื่องจากระบบดังกล่าวมีการเพิ่มหรือเปลี่ยนแปลงแก้ไขฐานข้อมูลนี้ ซึ่งมี 6 DET ประกอบไปด้วย Order_no Customer_id Product_code Amount Insert_by และ Delivery_no

ตารางฐานข้อมูล Customer เป็น 1 เอ็กซ์เทอร์นอลอินเตอร์เฟสไฟล์ (External Interface Files: EIFs) เนื่องจากระบบดังกล่าวไม่มีการเพิ่มหรือเปลี่ยนแปลงแก้ไขฐานข้อมูลนี้ ซึ่งมี 2 DET ประกอบไปด้วย Customer_name และ Customer_address

ตารางฐานข้อมูล Product เป็น 1 อินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) เนื่องจากระบบดังกล่าวมีการเพิ่มหรือเปลี่ยนแปลงแก้ไขฐานข้อมูลนี้ ซึ่งมี 4 DET ประกอบไปด้วย Product_code Product_name Amount_instock และ Amount_order

ตารางฐานข้อมูล Delivery เป็น 1 อินเทอร์นอลลอจิคอลไฟล์ (Internal Logical files: ILFs) เนื่องจากระบบดังกล่าวมีการเพิ่มหรือเปลี่ยนแปลงแก้ไขฐานข้อมูลนี้ ซึ่งมี 2 DET ประกอบไปด้วย Delivery_date และ Update_delivery_by

ตารางฐานข้อมูล User เป็น 1 เอ็กซ์เทอร์นอลอินเตอร์เฟสไฟล์ (External Interface Files: EIFs) เนื่องจากระบบดังกล่าวไม่มีการเพิ่มหรือเปลี่ยนแปลงแก้ไขฐานข้อมูลนี้ ซึ่งมี 2 DET ประกอบไปด้วย User_name และ User_level_permission

เมื่อนำมากรอกลงเอกสารแสดงได้ดังรูปที่ 4.16

	ชื่อ	ประเภท		จำนวน DET	จำนวน RET	C*
1	Order	(ILF)	EIF	6	1	(L) A H
2	Customer	ILF	(EIF)	2	1	(L) A H
3	Product	(ILF)	EIF	4	1	(L) A H
4	Delivery	(ILF)	EIF	2	1	(L) A H
5	User	ILF	(EIF)	2	1	(L) A H
6		ILF	EIF			L A H
7		ILF	EIF			L A H

รูปที่ 4.16 แสดงตัวอย่างการกรอกข้อมูลลอจิคอลไฟล์

ขั้นตอนที่ 2 ประเมินค่าความซับซ้อนของทรานแซคชันฟังก์ชัน (Transactional Functions)

สำหรับระบบตัวอย่างมี 3 หน้าจอ ซึ่งแสดงการนับทรานแซคชันฟังก์ชัน (Transactional Functions) ได้ดังนี้

หน้าจอที่ 1 มีการรับข้อมูลเข้า (Input) ทางหน้าจอ จึงนับเป็น 1 EI ซึ่งมี จำนวน DET และ จำนวน FTR แสดงได้ดังนี้

- เช็คบ็อก (Check-box) จำนวน 2 Check-box เท่ากับ 2 DET
- หมายเลขแผนการจัดส่ง เท่ากับ 1 DET
- หมายเลขคำสั่งซื้อ เท่ากับ 1 DET
- ชื่อลูกค้า เท่ากับ 1 DET
- ดรอปดาว์นลิส (Drop down list) เท่ากับ 1 DET
- ปุ่ม Search เท่ากับ 1 DET

และ FTR จำนวน 2 FTR คือ Order และ Customer

หน้าจอที่ 1 ไม่มี EO หรือ EQ

หน้าจอที่ 2 มีการแสดงข้อมูลออก (Output) ทางหน้าจอ จึงนับเป็น 1 EQ ซึ่งมี จำนวน DET และ จำนวน FTR แสดงได้ดังนี้

- สถานะสินค้าว่าครบสามารถจัดส่งได้หรือไม่ เท่ากับ 1 DET
- หมายเลขแผนการจัดส่ง เท่ากับ 1 DET
- หมายเลขคำสั่งซื้อ เท่ากับ 1 DET
- ชื่อลูกค้า เท่ากับ 1 DET
- โค้ดสินค้าที่สั่งซื้อ เท่ากับ 1 DET
- ข้อมูลผู้ลงทะเบียนที่แสดงในดรอปดาว์นลิส (Drop down list) เท่ากับ 1 DET

และ FTR จำนวน 4 FTR คือ Order Customer Product และ User

สำหรับหน้าจอที่ 2 มีการรับข้อมูลเข้าทางหน้าจอ จึงนับเป็น 1 EI ซึ่งมี จำนวน DET และ จำนวน FTR แสดงได้ดังนี้

- เช็คบ็อก (Check-box) เพื่อเลือกรายการคำสั่งซื้อนี้หรือไม่ เท่ากับ 1 DET
- วันที่ลงทะเบียนการจัดส่งสินค้า เท่ากับ 1 DET
- ผู้ลงทะเบียน เท่ากับ 1 DET
- ปุ่มลงทะเบียน เท่ากับ 1 DET
- ปุ่มยกเลิกการลงทะเบียน เท่ากับ 1 DET

และ FTR จำนวน 3 FTR คือ Order Product และ Delivery

สำหรับหน้าจอที่ 3 มีการแสดงผลลัพธ์การทำงานของการบันทึกข้อมูล เท่ากับมี 1 EQ ซึ่งมี จำนวน DET และ จำนวน FTR แสดงได้ดังนี้

- ข้อความแสดงผลการบันทึกข้อมูล

และไม่มี FTR

เมื่อนำมากรอกลงเอกสารแสดงได้ดังรูปที่ 4.17

	หน้าจอ	EI			EQ			EO		
		DET	FTR	C*	DET	FTR	C*	DET	FTR	C*
1	หน้าจอ 1	7	2	L (A) H			L A H			L A H
2	หน้าจอ 2	5	3	L A (H)	6	4	L A (H)			L A H
3	หน้าจอ 3			L A H	1	0	(L) A H			L A H
4				L A H			L A H			L A H

รูปที่ 4.17 แสดงตัวอย่างการกรอกข้อมูลทรานแซคชันฟังก์ชัน

เมื่อรวมขนาดฟังก์ชันพอยต์รวม ในด้านล่างเอกสาร แสดงได้ดังรูปที่ 4.18

ชนิด	LOW	AVERAGE	HIGH	SUM
ILF	3 7	10	15	21
ELF	2 5	7	10	10
EI	3	1 4	1 6	10
EQ	1 3	4	1 6	9
EO	4	5	7	0
			รวม	50 FP

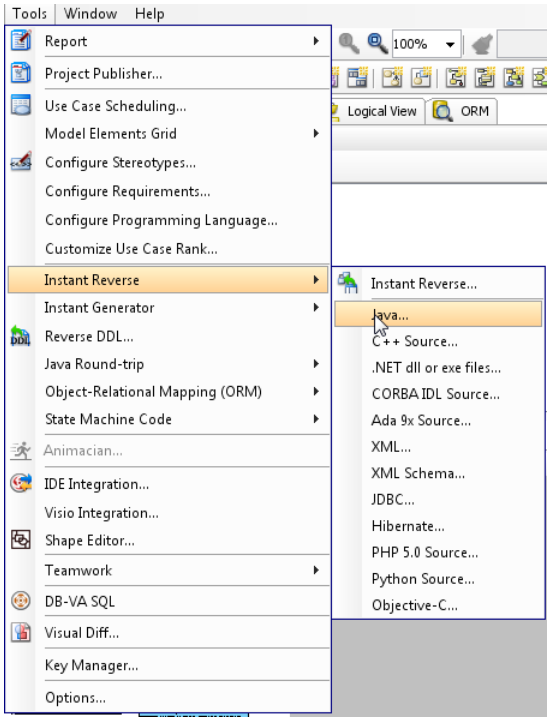
รูปที่ 4.18 แสดงตัวอย่างการรวมขนาดฟังก์ชันพอยต์

ระบบที่แสดงในตัวอย่างมีขนาดฟังก์ชันพอยต์รวมเท่ากับ 50 ฟังก์ชันพอยต์

4.2.2 นับขนาดจากวิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุ ฟังก์ชันพอยต์เชิงวัตถุสอง และคลาสพอยต์

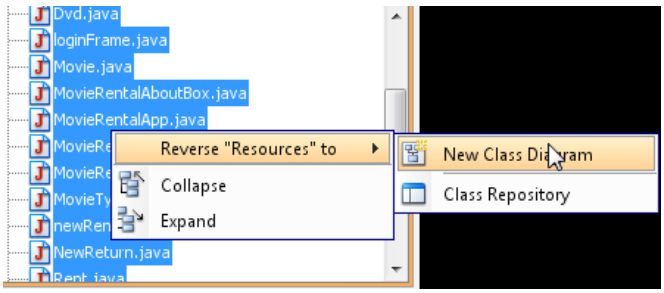
เนื่องจากวิธีการประมาณขนาดซอฟต์แวร์ด้วย ฟังก์ชันพอยต์เชิงวัตถุ ฟังก์ชันพอยต์เชิงวัตถุสอง และคลาสพอยต์จำเป็นต้องใช้แผนภาพคลาสเพื่อใช้นับขนาดในช่วงของการออกแบบระบบ แต่ทั้งสองแหล่งข้อมูลผู้วิจัยไม่สามารถจัดเก็บแผนภาพคลาสได้ ผู้วิจัยจึงจำเป็นต้องทำวิศวกรรมย้อนกลับ (Reverse engineering) จากซอร์สโค้ดเป็นแผนภาพคลาส โดยมีวิธีการแปลง ดังนี้

1. เลือกใช้เครื่องมือในโปรแกรมวิชวล พาราไดม์ สำหรับ ยูเอ็มแอล (Visual Paradigm for UML) โดยเข้าเมนู Tool เลือก Instant reverse จาก Java และเลือกโฟลเดอร์ (Floder) ของแพคเกจ (Package) ในระบบย่อย ดังรูปที่ 4.19

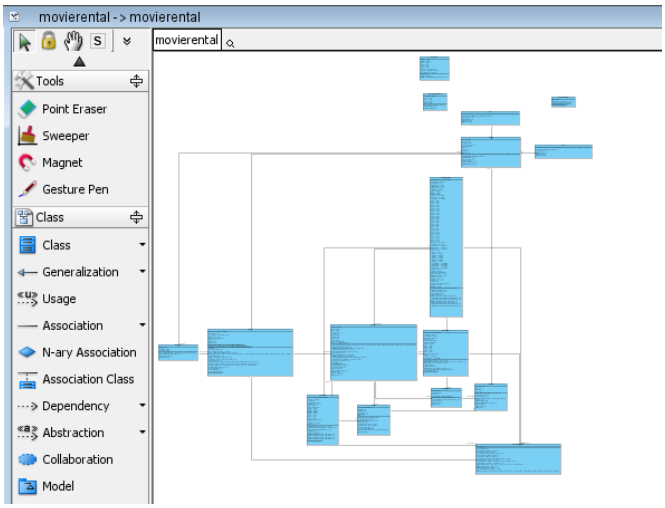


รูปที่ 4.19 แสดงขั้นตอนการเลือกซอร์สโค้ดให้เป็นแผนภาพคลาสด้วยโปรแกรมวิชวล พาราไดม์ สำหรับ ยูเอ็มแอล

2. โปรแกรมวิชวล พาราไดม์จะแสดงคลาส (Class) ทั้งหมดจะแปลงเป็นแผนภาพคลาส (Class diagram) ในแถบด้านซ้าย หลังจากนั้นเลือกให้แสดงเป็นแผนภาพคลาส ดังรูปที่ 4.20 จะได้แผนภาพคลาสดังรูปที่ 4.21

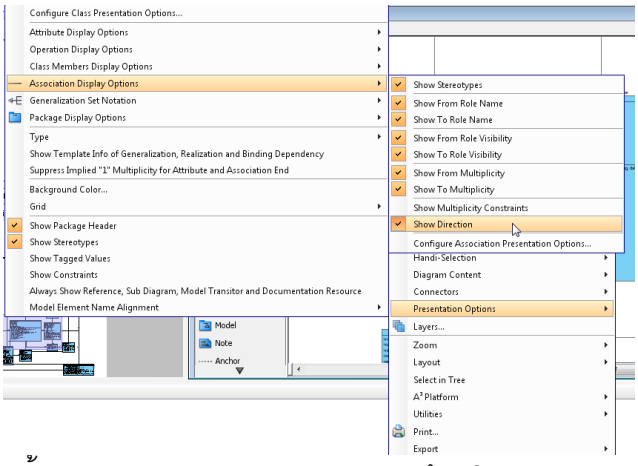


รูปที่ 4.20 แสดงขั้นตอนการแสดงแผนภาพคลาสด้วยโปรแกรมวิชวล พาราไดม์ สำหรับ ยูเอ็มแอล



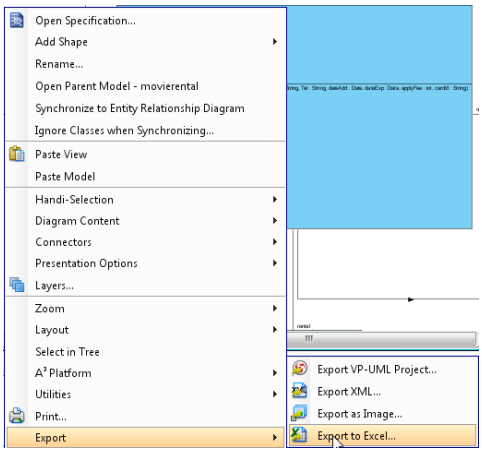
รูปที่ 4.21 แสดงแผนภาพคลาสด้วยโปรแกรมวิชวล พาราไดม์ สำหรับ ยูเอ็มแอล

3. หลังจากนั้นตั้งค่าการแสดงผลเพื่อให้แสดงรายละเอียดความสัมพันธ์ให้ครบทุกแบบดังรูปที่ 4.22



รูปที่ 4.22 แสดงการตั้งค่าการแสงผลของแผนภาพคลาสด้วยโปรแกรมวิชวล พาราไดม์ สำหรับ ยูเอ็มแอล

4. เพื่อความถูกต้องผู้วิจัยจึงใช้วิธีการส่งแฟ้มออก (Export) ข้อมูลของแผนภาพคลาส (Class diagram) ออกมาเป็นไฟล์ไมโครซอฟต์เอกซ์เซล (Mircosoft excel) เพื่อความสะดวกในการนับ ดังรูปที่ 4.23



รูปที่ 4.23 แสดงการส่งออกไฟล์เอกซ์เซลของแผนภาพคลาสด้วยโปรแกรมวิชวล พาราไดม์ สำหรับ ยูเอ็มแอล

5. ไฟล์ไมโครซอฟต์เอกซ์เซล (Mircosoft excel) ที่ได้จะแสดงรายละเอียดทั้งหมดของแผนภาพคลาส (Class diagram) ตัวอย่างแสดงได้ดังรูปที่ 4.24

Diagram	ID	Name	Type	Documentation	Delete ?		
	1	movierental	ClassDiagram		No		
Class	ID	Name	Stereotypes	Visibility	Documentation	Abstract	Leaf
	2	Vcd		public	@author Dell	No	No
	Operation	ID	Name	Stereotypes	Return type	Type Modifier	Visibility
		3	Vcd				public
		Parameter	ID	Name	Stereotypes	Type	Type Modifier
			4	movieID		String	
			5	name		String	
			6	desction		String	
			7	movieType		movierental.MovieType	
			8	status		int	
			9	price		int	
	Operation	ID	Name	Stereotypes	Return type	Type Modifier	Visibility
		10	calculateFine		int		public
		Parameter	ID	Name	Stereotypes	Type	Type Modifier
			11	d		java.util.Date	
			12	reD		java.util.Date	
	Operation	ID	Name	Stereotypes	Return type	Type Modifier	Visibility
		13	getCredit_day		int		public
		14	getRentalFee		int		public
		15	toString		String		public
		16	getName		String		public
Class	ID	Name	Stereotypes	Visibility	Documentation	Abstract	Leaf
	17	returnMovie		public	@author Dell	No	No
	Attribute	ID	Name	Stereotypes	Initial value	Multiplicity	Visibility
		18	returnId			Unspecified	private
		19	returnDate			Unspecified	private

รูปที่ 4.24 แสดงไฟล์เอกซ์เซลของแผนภาพคลาสด้วยโปรแกรมวิชวล พาราไดม์ สำหรับ ยูเอ็มแอล

นับขนาดจากวิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุและฟังก์ชันพอยต์เชิงวัตถุสอง

ผู้วิจัยใช้ไมโครซอฟต์เอกซ์เซล (Mircosoft excel) เพื่อเป็นเครื่องมือในการบันทึกผลและนับจำนวนข้อมูลต่างๆและคำนวณเป็นจำนวนฟังก์ชันพอยต์เชิงวัตถุ ดังนี้

แผ่นงาน (Sheet) ที่ 1 ประกอบด้วยการนับลอจิคอลไฟล์ (Logical file) หรือคลาส (Class) ตัวอย่างแสดงดังรูปที่ 4.25

	A	B	C	D	E	F	G	H	I	J	K	L
1				นับ LIFs และ EIFs								
2	ชื่อ Class	จน. Attributeทั้งหมด	Attibute ซับซ้อน	Attribute ไม่ซับซ้อน	single-valued association	multiple-value association	DET	RET	OOFP		OOFP	
3	Class 1	291	14	277	1	1	278	15	3		จำนวน Low	4
4	Class 2	30	0	30	1	0	31	0	1		จำนวน Average	2
5	Class 3	260	2	258	3	0	261	2	3		จำนวน High	3
6	Class 4	284	3	281	1	1	282	4	3			
7	Class 5	6	3	3	1	1	4	4	1			
8	Class 6	2	1	1	0	1	1	2	1			
9	Class 7	19	8	11	3	1	14	9	2			
10	Class 8	7	4	3	3	0	6	4	1			
11	Class 9	21	10	11	3	1	14	11	2			

Class Method OOFP1 OOFP2

รูปที่ 4.25 แสดงตัวอย่างไฟล์เอกซ์เซลในส่วนลอจิคอลไฟล์ของการนับด้วยฟังก์ชันพอยต์เชิงวัตถุ

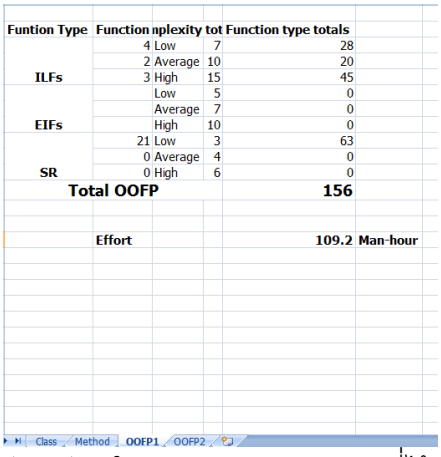
แผ่นงาน (Sheet) ที่ 2 ประกอบด้วยการนับเซอร์วิสรีเควส (Services request: SR) หรือเมท็อด (Method) ตัวอย่างแสดงดังรูปที่ 4.26

	A	B	C	D	E	F	G	H	I	J
1				นับ Service request					OOFP	
2	ชื่อ Class	ชื่อ Method	argument ปกติ	argumentซับซ้อน	ความซับซ้อน OOFP	ความซับซ้อน OOFP2			จำนวน Low	21
3	Class 1	Method 1()	0	0	1	1			จำนวน Ave	0
4		Method 2()	0	0	1	1			จำนวน High	0
5		Method 3(int)	1	0	1	1				
6	Class 2	Method 1(int,string)	2	0	1	1			OOFP2	
7		Method 2()	0	0	1	1			จำนวน Low	21
8		Method 3(int)	1	0	1	1			จำนวน Ave	0
9	Class 3	Method 1()	0	0	1	1			จำนวน High	0
10		Method 2()	0	0	1	1				
11		Method 3()	0	0	1	1				
12		Method 4(Class 1)	0	1	1	1				
13		Method 5(double)	1	0	1	1				
14		Method 6()	0	0	1	1				
15		Method 7()	0	0	1	1				
16		Method 8()	0	0	1	1				
17		Method 9()	0	0	1	1				
18		Method 10()	0	0	1	1				
19		Method 11()	0	0	1	1				
20		Method 12()	0	0	1	1				
21	Class 4	Method 1()	0	0	1	1				
22		Method 2()	0	0	1	1				
23		Method 3()	0	0	1	1				

Class Method OOFP1 OOFP2

รูปที่ 4.26 แสดงตัวอย่างไฟล์เอกซ์เซลในส่วนเซอร์วิสรีเควสของการนับด้วยฟังก์ชันพอยต์เชิงวัตถุ

แผ่นงาน (Sheet) ที่ 3 และ 4 นับขนาดรวมของฟังก์ชันพอยต์เชิงวัตถุ (OOFP) และฟังก์ชันพอยต์เชิงวัตถุสอง (OOFP2) ตัวอย่างแสดงดังรูปที่ 4.27

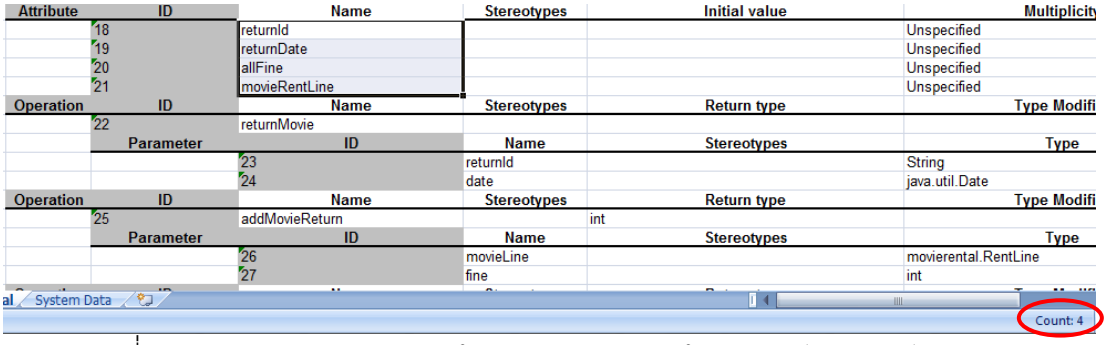


Funtion Type	Function	nplexity tot	Function type totals
	4	Low 7	28
	2	Average 10	20
ILFs	3	High 15	45
		Low 5	0
		Average 7	0
EIFs		High 10	0
	21	Low 3	63
	0	Average 4	0
SR	0	High 6	0
Total OOFP			156
Effort			109.2 Man-hour

รูปที่ 4.27 แสดงตัวอย่างไฟล์เอกซ์เซลในส่วนของผลรวมขนาดที่ได้จากการคำนวณด้วยฟังก์ชันพอยต์เชิงวัตถุ

เมื่อนำมาคำนวณด้วยวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุและฟังก์ชันพอยต์เชิงวัตถุสอง คำนวณได้ดังนี้

1. จากแฟ้มเอกซ์เซลที่ได้จากแผนภาพคลาสที่แปลงมาจากซอร์สโค้ด ผู้วิจัยนับจำนวนลักษณะประจำ (Number Of Attribute: NOA) ในคลาส (Class) โดยนำมากรอกลงข้อมูลที่ใช้ในการนับ โดยเริ่มจากการดูที่ลักษณะประจำ (Attribute) ทั้งหมดในคลาส (Class) และลากเซลล์ (Cell) ที่เกี่ยวข้องทั้งหมด ข้อมูลด้านล่าง ไมโครซอฟต์เอกซ์เซล (Mircosoft excel) แสดงจำนวนที่นับได้ ดังรูปที่ 4.28



Attribute	ID	Name	Stereotypes	Initial value	Multiplicity
	18	returnId			Unspecified
	19	returnDate			Unspecified
	20	allFine			Unspecified
	21	movieRentLine			Unspecified
Operation	**ID**	**Name**	**Stereotypes**	**Return type**	**Type Modifi**
	22	returnMovie			
	Parameter	**ID**	**Name**	**Stereotypes**	**Type**
		23	returnId		String
		24	date		java.util.Date
Operation	**ID**	**Name**	**Stereotypes**	**Return type**	**Type Modifi**
	25	addMovieReturn		int	
	Parameter	**ID**	**Name**	**Stereotypes**	**Type**
		26	movieLine		movierental.RentLine
		27	fine		int

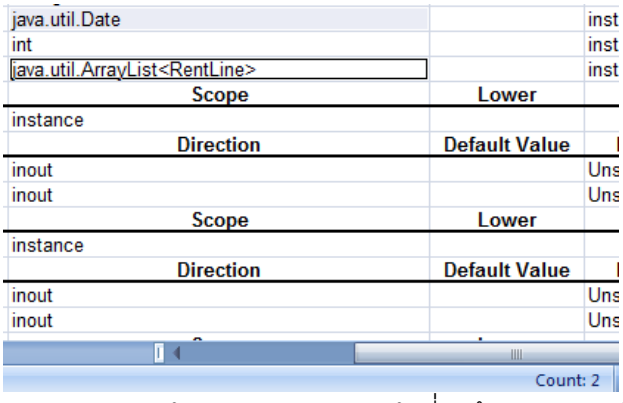
System Data

Count: 4

รูปที่ 4.28 แสดงตัวอย่างการนับจำนวนลักษณะประจำของฟังก์ชันพอยต์เชิงวัตถุ

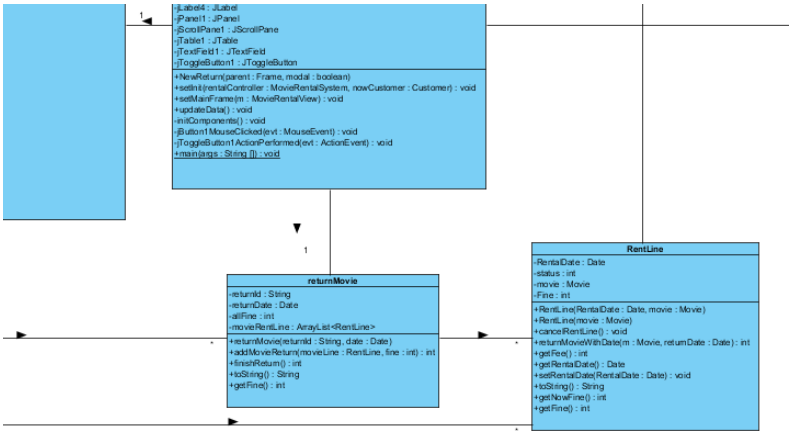
จากรูปที่ 4.28 พบว่า คลาสดังกล่าวมีทั้งสิ้น 4 ลักษณะประจำ (Attribute) นำมากรอกลงช่องลักษณะประจำ (Attribute) ทั้งหมดในเอกสารที่พัฒนาขึ้นมา หลังจากนั้นผู้วิจัยจะนับเฉพาะลักษณะประจำ (Attribute) ที่ซับซ้อนด้วยการเลือกเฉพาะชนิดลักษณะประจำ (Attribute) ซับซ้อน เช่น Object arrayList ข้อมูลด้านล่างของไมโครซอฟต์เอกซ์เซล (Mircosoft excel)

แสดงจำนวนที่เลือกไว้ ดังรูปที่ 4.29 นำมากรอกลงช่องลักษณะประจำ (Attribute) ที่ซับซ้อน การนับด้วยวิธีดังกล่าว ทำให้ผู้วิจัยไม่จำเป็นต้องนับลักษณะประจำ (Attribute) ที่ไม่ซับซ้อน เนื่องจากจะเท่ากับจำนวนลักษณะประจำ (Attribute) ทั้งหมด ลบ จำนวนลักษณะประจำ (Attribute) ที่ซับซ้อน



รูปที่ 4.29 แสดงตัวอย่างการนับจำนวนลักษณะประจำที่ซับซ้อนของฟังก์ชันพอยต์เชิงวัตถุ

2. นับจำนวนความเชื่อมโยง (Association) จากแผนภาพคลาสภายในโปรแกรมวิชวล พาราไดม์ สำหรับ ยูเอ็มแอล (Visual Paradigm for UML) ดังรูปที่ 4.30



รูปที่ 4.30 แสดงตัวอย่างการนับความเชื่อมโยงของฟังก์ชันพอยต์เชิงวัตถุ

ตัวอย่างรูปที่ 4.30 คลาส returnMovie มีจำนวนความเชื่อมโยง (Association) ทั้งหมด 3 โดยแบ่งเป็น Single Association จำนวน 2 และ Multiple Association จำนวน 1 หลังจากนั้นนำไปกรอกลงบนไฟล์ไมโครซอฟต์เอกซ์เซล (Mircosoft excel)

3. เมื่อได้รายละเอียดของแต่ละคลาส (Class) และนำมากรอกลงบนไฟล์ไมโครซอฟต์เอกซ์เซล (Mircosoft excel) ที่พัฒนาขึ้น

			นับ LIFs และ EIFs					
ชื่อ Class	จน. Attributeทั้งหมด	Attibute ซับซ้อน	Attribute ไม่ซับซ้อน	single-valued association	multiple-value association	DET	RET	OOFP
returnMo	10	3	7	1	1	8	4	1
RentLine	22	3	19	1	0	20	3	2
Rent	4	1	3	3	0	6	1	1
NewRetu	5	0	5	1	1	6	1	1
newRent	30	2	28	1	1	29	3	2
MovieTyp	2	2	0	0	1	0	3	1
Vcd	10	2	8	3	1	11	3	1
Movie	10	2	8	3	0	11	2	1
Dvd	20	2	18	3	1	21	3	2

รูปที่ 4.31 แสดงตัวอย่างการกรอกข้อมูลเพื่อใช้คำนวณลอจิคอลของฟังก์ชันพอยต์เชิงวัตถุ

ในแถว DET จะแสดงจำนวนองค์ประกอบของข้อมูล (Data element types: DET) ที่นับได้จากลักษณะประจำ (Attribute) ที่ไม่ซับซ้อน รวมกับความเชื่อมโยงเพียง 1 (single-valued association)

ในแถว RET จะแสดงจำนวนเรคคอร์ดข้อมูล (Record element types: RET) ที่นับได้จากลักษณะประจำ (Attribute) ที่ซับซ้อน รวมกับความเชื่อมโยงที่มากกว่า 1 (multiple-valued association)

หลังจากนั้นในแถวสุดท้ายจะประเมินระดับความซับซ้อนของลอจิคอลไฟล์ (Logical file) หรือคลาส (Class) นั้น เงื่อนไขการประเมินระดับความซับซ้อนอ้างอิงจากการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์ (IFPUG, 1999) ตามตารางที่ 4.4

ตารางที่ 4.4 แสดงตารางการประเมินระดับความซับซ้อนของลอจิคอลไฟล์

	1 to 19 DET	**20 to 50 DET**	**51 or more DET**
1 RET	Low	Low	Average
2 to 5 RET	Low	Average	High
6 or more RET	Average	High	High

จากตารางที่ 4.4 ผู้วิจัยจึงแปลงให้อยู่ในรูปการใช้สูตรเงื่อนไข (IF ELSE) ในไฟล์ไมโครซอฟต์เอกซ์เซล (Microsoft Excel File) ดังนี้

=IF(H3<=1,IF(G3<=50,1,2),IF(H3<=5,IF(G3<=19,1,IF(G3<=50,2,3)),IF(G3<=19,2,3)))

โดย 1 จะเท่ากับความซับซ้อนระดับ Low

2 จะเท่ากับความซับซ้อนระดับ Medium

3 จะเท่ากับความซับซ้อนระดับ High

4. นับจำนวนเซอร์วิสรีเควส (Services request: SR) จากเมท็อด (Method) ที่แสดงอยู่บนแผนภาพคลาส (Class diagram) โดยการคัดลอกเมท็อดลงในไฟล์ไมโครซอฟต์เอกซ์เซล (Mircosoft excel) และกรอกข้อมูลจำนวนอาร์กิวเมนต์ (Argument) ปกติและอาร์กิวเมนต์ (Argument) ซับซ้อน ตามเมท็อด (Method) ที่แสดงอยู่ ดังรูปที่ 4.32

		นับ Service request	
ชื่อ Class	ชื่อ Method	argument ปกติ	argumentซับซ้อน
MovieType	MovieType(typeID : String, TypeName : String, Description : String)	3	0
	setTypeName(TypeName : String) : void	1	0
	getTypeName() : String	0	0
returnMovie	returnMovie(returnId : String, date : java.util.Date)	1	1
	addMovieReturn(movieLine : movierental.RentLine, fine : int) : int	1	1
Movie	Movie(movieID : String, name : String, descption : String, MovieType	5	1
	calculateFine(d : java.util.Date, reD : java.util.Date) : int	0	2
	getCredit_day() : int	0	0
	getAveriableName() : String	0	0
	getStatusName() : String	0	0
Dvd	Dvd(movieID : String, name : String, descption : String, MovieType :	0	0

รูปที่ 4.32 แสดงตัวอย่างการกรอกข้อมูลเพื่อใช้คำนวณเซอร์วิสรีเควสของฟังก์ชันพอยต์เชิงวัตถุ

5. ไฟล์ไมโครซอฟต์เอกซ์เซล (Mircosoft excel) ที่ผู้พัฒนาสร้างขึ้น ในแถวสุดท้ายจะแสดงระดับความซับซ้อน ดังรูปที่ 4.33

	นับ Service request			
ชื่อ Method	argument ปกติ	argumentซับซ้อน	ความซับซ้อน OOFP	ความซับซ้อน OOFP2
MovieType(typeID : String, TypeName : String, Description : String)	3	0	1	1
setTypeName(TypeName : String) : void	1	0	1	1
getTypeName() : String	0	0	1	1
returnMovie(returnId : String, date : java.util.Date)	1	1	1	1
addMovieReturn(movieLine : movierental.RentLine, fine : int) : int	1	1	1	1
Movie(movieID : String, name : String, descption : String, MovieType	5	1	1	2
calculateFine(d : java.util.Date, reD : java.util.Date) : int	0	2	1	2
getCredit_day() : int	0	0	1	1
getAveriableName() : String	0	0	1	1
getStatusName() : String	0	0	1	1
Dvd(movieID : String, name : String, descption : String, MovieType :	0	0	1	1

รูปที่ 4.33 แสดงตัวอย่างระดับความซับซ้อนของเซอร์วิสรีเควสของฟังก์ชันพอยต์เชิงวัตถุ

เงื่อนไขการประเมินระดับความซับซ้อนของเซอร์วิสรีเควสของฟังก์ชันพอยต์เชิงวัตถุ (Antoniol et al., 1998) ให้ใช้ตามการประเมินระดับความซับซ้อนของอินเทอร์นอลอินพุทด้วยฟังก์ชันพอยต์ (IFPUG, 1999) ตามตารางที่ 4.5

ตารางที่ 4.5 แสดงตารางการประเมินระดับความซับซ้อนของเซอร์วิสรีเควสของฟังก์ชันพอยต์เชิงวัตถุ

	1 to 4 DET	5 to 15 DET	16 or more DET
0 to 1 FTR	Low	Low	Average
2 FTRs	Low	Average	High
3 or more FTRs	Average	High	High

ผู้วิจัยจึงแปลงให้อยู่ในรูปการใช้สูตรเงื่อนไข (IF ELSE) ในไฟล์ไมโครซอฟต์เอกซ์เซล (Microsoft Excel File) ดังนี้

=IF(D3<=1,IF(C3<=15,1,2),IF(D3<=2,IF(C3<=4,1,IF(C3<=15,2,3)),IF(C3<=4,2,3)))

โดย 1 จะเท่ากับความซับซ้อนระดับ Low

2 จะเท่ากับความซับซ้อนระดับ Medium

3 จะเท่ากับความซับซ้อนระดับ High

ส่วนการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุสอง (Zivkovic et al., 2005) ปรับปรุงตารางการคำนวณระดับความซับซ้อน ตามตารางที่ 4.6

ตารางที่ 4.6 แสดงตารางการประเมินระดับความซับซ้อนของเซอร์วิสรีเควสของฟังก์ชันพอยต์เชิงวัตถุสอง

	0–4 (1–5) DET	5–10 (6–19) DET	More than 10 (20 or more) DET
0 or 1 FTR	Low (low)	Average (low)	High (average)
2 (2–3) FTR	Average (low)	Average	High
3 or more (4 or more) FTR	Average	High	High

ผู้วิจัยจึงแปลงให้อยู่ในรูปการใช้สูตรเงื่อนไข (IF ELSE) ในไฟล์ไมโครซอฟต์เอกซ์เซล (Microsoft Excel File) ดังนี้

=IF(D3<=1,IF(C3<=4,1,IF(C3<=10,2,3)),IF(D3<=2,IF(C3<=10,2,3),IF(C3<=4,2,3)))

โดย 1 จะเท่ากับความซับซ้อนระดับ Low

2 จะเท่ากับความซับซ้อนระดับ Medium

3 จะเท่ากับความซับซ้อนระดับ High

6. ในส่วนของ 2 แผ่นงานสุดท้ายแสดงผลสรุป โดยการรวมค่าทั้งหมดที่ได้ในแต่ละความซับซ้อนของลอจิคอลไฟล์ (Logical file) และเซอร์วิสรีเควส (Services request) รวมทั้งคำนวณขนาดของฟังก์ชันพอยต์เชิงวัตถุและฟังก์ชันพอยต์เชิงวัตถุสอง รวมไปถึงค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์ (Development effort) ที่ได้จากสูตรการคำนวณ คือ Effort = OOFP*0.7 man-hours (Hericko & Zivkovic, 2007) แสดงได้ดังรูปที่ 4.34 และ รูปที่ 4.35

Funtion Type	Function complexity	Complexity totals		Function type totals	
	6	Low	7	42	
	3	Average	10	30	
ILFs	0	High	15	0	
	0	Low	5	0	
	0	Average	7	0	
EIFs	0	High	10	0	
	19	Low	3	57	
	0	Average	4	0	
SR	0	High	6	0	
Total OOFP				**129**	
Effort				**90.3**	**Man-hour**

รูปที่ 4.34 แสดงตัวอย่างผลรวมขนาดของฟังก์ชันพอยต์เชิงวัตถุ

Funtion Type	Function complexity	Complexity totals		Function type totals	
	6	Low	7	42	
	3	Average	10	30	
ILFs	0	High	15	0	
	0	Low	5	0	
	0	Average	7	0	
EIFs	0	High	10	0	
	17	Low	3	51	
	2	Average	4	8	
SR	0	High	6	0	
Total OOFP2				**131**	
Effort				**91.7**	**Man-hour**

รูปที่ 4.35 แสดงตัวอย่างผลรวมขนาดของฟังก์ชันพอยต์เชิงวัตถุสอง

จากรูปที่ 4.34 และ รูปที่ 4.35 ในตัวอย่าง ขนาดฟังก์ชันพอยต์เชิงวัตถุเท่ากับ 129 ฟังก์ชันพอยต์เชิงวัตถุ และฟังก์ชันพอยต์เชิงวัตถุสองเท่ากับ 131 ฟังก์ชันพอยต์เชิงวัตถุสอง

นับขนาดจากวิธีการประมาณขนาดซอฟต์แวร์ด้วยคลาสพอยต์

ผู้วิจัยพัฒนาเอกสารขึ้นเพื่อใช้ในการนับคลาสพอยต์เพื่อง่ายต่อการบันทึก ดังรูปที่ 4.36

เอกสารในการนับ Class point

เวลาที่ใช้นับ NEM,NSR	เวลานับเริ่ม __ : __ : __	สิ้นสุด __ : __ : __
เวลาที่ใช้นับ NOA	เวลานับเริ่ม __ : __ : __	สิ้นสุด __ : __ : __

ระบบ_____

ชื่อ คลาส	จำนวน NSR	จำนวน NEM	จำนวน NOA	ความซับซ้อน	ประเภทของ Class

Class point = point

รูปที่ 4.36 แสดงเอกสารที่ใช้ในการนับขนาดซอฟต์แวร์ด้วยคลาสพอยต์

1. นับจำนวนการเรียกใช้บริการจากภายนอก (Number of Services Requested: NSR) โดยดูจากแผนภาพคลาส (Class diagram) ว่าในแต่ละคลาส (Class) ว่ามีการเรียกใช้คลาส (Class) ใดบ้าง

2. นับจำนวนเมท็อด (Method) โดยดูจากแผนภาพคลาส (Class diagram) โดยนับเฉพาะคลาสที่ถูกประกาศเป็นสาธารณะ (Public method)

3. นับจำนวนลักษณะประจำ (Number Of Attribute: NOA) ซึ่งได้มาจากการนับของฟังก์ชันพอยต์เชิงวัตถุ (OOFP) ในเอกสารรูปที่ 4.31 ในแถวที่ 2

4. กรอกชนิดของคลาส (Class) โดยดูจากชื่อคลาส (Class) รวมไปถึงรายละเอียดของคลาส (Class) และตีความหมาย โดยแบ่งทั้งสิ้น 4 ชนิด คือ ประเภทขอบเขตของปัญหา (Problem domain type: PDT) ประเภทโต้ตอบกับผู้ใช้งาน (Human Interaction type: HIT) ประเภทการจัดการข้อมูล (Data Management type: DMT) และประเภทการจัดการงาน (Task Management type: TMT)

5. นำค่าทั้งหมดที่ได้เติมในกระดาษมากรอกลงไฟล์ไมโครซอฟต์เอกซ์เซล (Mircosoft excel) เพื่อประเมินหาความซับซ้อนของคลาส (Class) ซึ่งอ้างอิงจากตารางที่ 4.7 (Costagliola et al., 2005) ดังรูปที่ 4.37

ตารางที่ 4.7 แสดงตารางการประเมินระดับความซับซ้อนของคลาสด้วยวิธีคลาสพอยต์

0 - 2 *NSR*	**0 - 5 *NOA***	**6 - 9 *NOA***	**≥ 10 *NOA***
0 - 4 *NEM*	Low	Low	Average
5 - 8 *NEM*	Low	Average	High
≥ 9 *NEM*	Average	High	High

(a)

3 - 4 *NSR*	**0 - 4 *NOA***	**5 - 8 *NOA***	**≥ 9 *NOA***
0 - 3 *NEM*	**Low**	**Low**	**Average**
4 - 7 *NEM*	**Low**	**Average**	**High**
≥ 8 *NEM*	**Average**	**High**	**High**

(b)

≥ 5 *NSR*	**0 - 3 *NOA***	**4 - 7 *NOA***	**≥ 8 *NOA***
0 - 2 *NEM*	**Low**	**Low**	**Average**
3 - 6 *NEM*	**Low**	**Average**	**High**
≥ 7 *NEM*	**Average**	**High**	**High**

ผู้วิจัยจึงแปลงให้อยู่ในรูปการใช้สูตรเงื่อนไข (IF ELSE) ในไฟล์ไมโครซอฟต์เอกซ์เซล (Microsoft Excel File) ดังนี้

=IF(B8<=2,IF(D8<=5,IF(C8<9,1,2),IF(D8<=9,IF(C8<=4,1,IF(C8<=8,2,3)),IF(C8<=4,2,3))),IF(B8<=4,IF(D8<=4,IF(C8<=7,1,2),IF(D8<=8,IF(C8<=3,1,IF(C8<=7,2,3)),IF(C8<=3,2,3))),IF(D8<=2,IF(C8<7,1,2),IF(D8<=7,IF(C8<=2,1,IF(C8<=6,2,3)),IF(C8<=2,2,3)))))

โดย 1 จะเท่ากับความซับซ้อนระดับ Low

2 จะเท่ากับความซับซ้อนระดับ Medium

3 จะเท่ากับความซับซ้อนระดับ High

A	B	C	D	E	F	G
ระบบ_____						
ชื่อ คลาส์	จำนวน NSR	จำนวน NEM	จำนวน NOA	ความซับซ้อน	ประเภทของ Class	ขนาดรวม
Class 1	1	146	73	3	PDT	10
Class 2	1	404	202	3	PDT	10
Class 3	2	90	45	3	PDT	10
Class 4	2	118	59	3	PDT	10
Class 5	2	9	9	3	DMT	13
Class 6	0	39	16	3	HIT	12
Class 7	0	8	5	1	HIT	4
Class 8	0	45	21	3	HIT	12
Class 9	0	8	4	1	TMT	4
Class 10	0	6	3	1	TMT	4
	Class point =		89 point			

รูปที่ 4.37 แสดงตัวอย่างผลการนับขนาดซอฟต์แวร์ด้วยคลาสพอยต์

จากตัวอย่างในรูปที่ 4.37 ในแถว G คำนวณขนาดโดยดูจากระดับความซับซ้อนและชนิดของคลาส (Class) โดยค่ารวมขนาดทั้งหมดแสดงที่ด้านล่างของไฟล์ไมโครซอฟต์เอกซ์เซล (Mircosoft excel) จากตัวอย่างขนาดของคลาสพอยต์เท่ากับ 89 คลาสพอยต์

4.2.3 การเปรียบเทียบวิธีการนับขนาดด้วยฟังก์ชันพอยต์ ฟังก์ชันพอยต์เชิงวัตถุ และคลาสพอยต์

ในการนับจำนวนฟังก์ชันพอยต์ ฟังก์ชันพอยต์เชิงวัตถุ และคลาสพอยต์ ผู้วิจัยพบว่า ผู้นับจะต้องมีการเรียนรู้ระบบหรือซอฟต์แวร์แตกต่างกัน ดังนี้

วิธีการนับขนาดด้วยฟังก์ชันพอยต์ เป็นวิธีการที่ผู้นับจะต้องมีความรู้และความเข้าใจในระบบที่ต้องการคำนวณมากที่สุด เนื่องจากผู้นับจะต้องเข้าใจกระบวนการทำงานของระบบเป็นอย่างดีว่าข้อมูลส่งผ่านอย่างไร จัดเก็บที่ตารางฐานข้อมูลใด (Database table) รวมไปถึงการแสดงผลเป็นการแสดงผลแบบใด ผ่านการคำนวณหรือไม่ วิธีการดังกล่าวจึงเหมาะสมหรับผู้ที่มีความเข้าใจระบบเป็นอย่างดีจึงสามารถใช้วิธีการดังกล่าวได้

วิธีการนับขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุ เป็นวิธีการที่ผู้นับแทบจะไม่ต้องมีความรู้เกี่ยวกับระบบเลย เนื่องจากศึกษาที่คลาสไดอะแกรมและไม่มีการแบ่งประเภทใดๆของคลาส แต่ผู้นับจะต้องมีความรู้และเข้าใจเกี่ยวกับหลักการทำงานของคลาสและชนิดของอาร์กิวเมนต์ (Argument)

วิธีการนับขนาดด้วยคลาสพอยต์ เป็นวิธีการที่ผู้นับจะต้องเข้าใจหน้าที่การทำงานของแต่ละคลาส เนื่องจากวิธีการคำนวณของคลาสพอยต์ไม่ยาก เพียงแต่ผู้นับต้องสามารถแบ่งแยกการทำงานของคลาสนี้ให้ถูกต้อง ว่าเป็นชนิดใด มีหน้าที่เป็น ประเภทขอบเขตของปัญหา (Problem domain type: PDT) ประเภทโต้ตอบกับผู้ใช้งาน (Human Interaction type: HIT) ประเภทการจัดการข้อมูล (Data Management type: DMT) หรือ ประเภทการจัดการงาน (Task Management type: TMT)

4.3 การวิเคราะห์ความแม่นยำของวิธีการประมาณขนาดซอฟต์แวร์

หลังจากผู้วิจัยได้ขนาดที่ได้จากการนับในแต่ละวิธีการประมาณขนาดซอฟต์แวร์ ในขั้นตอนนี้นำมาวิเคราะห์ความแม่นยำที่ได้ โดยเปรียบเทียบกับค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์จริงที่เก็บได้จากแหล่งข้อมูลทั้งสองแหล่งข้อมูล

4.3.1 ความแม่นยำของวิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์ (Function point)

สำหรับวิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์นั้น ขนาดซอฟต์แวร์ที่ได้จากการนับอยู่ระหว่าง 14.44 - 68.25 ฟังก์ชันพอยต์ โดยมีค่าเฉลี่ยอยู่ที่ 36.03 ฟังก์ชันพอยต์ เมื่อนำมาเปลี่ยนเป็นค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์ตามสมการที่ได้กล่าวมาในบทที่ 3 จำนวน 4 สมการ ดังนี้

สมการที่ 1 มาจาก Moser และ Nierstrasz (1996) โดยมีสมการ ดังนี้

$\text{Effort}(\text{Man} - \text{hours}) = 6.667 * \text{FP}$ โดย FP คือ จำนวนฟังก์ชันพอยต์ของซอฟต์แวร์

สมการที่ 2 มาจาก Lokan (2000) โดยมีสมการ ดังนี้

$\text{Effort}(\text{Man} - \text{hours}) = 21.0 * \text{FP}^{0.826}$ โดย FP คือ จำนวนฟังก์ชันพอยต์ของซอฟต์แวร์

สมการที่ 3 มาจาก Maxwell และ Froselius (2000) ซึ่งคิดแปลงค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์ตามประเภทของธุรกิจ

ในงานวิจัยนี้ ข้อมูลจากแหล่งที่ 1 คือ ข้อมูลจากบริษัทธุรกิจทองแดง ซึ่งมาจากกลุ่มอุตสาหกรรมการผลิต โรงงาน จะใช้สมการ ดังนี้

$\text{Effort}(\text{Man} - \text{hours}) = \frac{\text{FP}}{0.337}$ โดย FP คือ จำนวนฟังก์ชันพอยต์ของซอฟต์แวร์

ส่วนแหล่งข้อมูลที่ 2 มาจากธุรกิจการเงิน ซึ่งมาจากกลุ่มธนาคาร จะใช้สมการ ดังนี้

$\text{Effort}(\text{Man} - \text{hours}) = \frac{\text{FP}}{0.116}$ โดย FP คือ จำนวนฟังก์ชันพอยต์ของซอฟต์แวร์

และสมการที่ 4 มาจาก Jeffer และคณะ (2001) จะใช้สมการ ดังนี้

$\text{Effort}(\text{Man} - \text{hours}) = 2.2 * \text{FP}$ โดย FP คือ จำนวนฟังก์ชันพอยต์ของซอฟต์แวร์

เมื่อนำแต่ละสมการมาค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ (Mean Magnitude of Relative Error: MMRE) และอัตราเปอร์เซ็นต์ที่ทำนายถูก (Percentage of predictions: PRED) โดยมาตรฐานความคลาดเคลื่อนไม่เกิน 25% ได้ผลลัพธ์แสดงได้ดังตารางที่ 4.8 ผู้วิจัยพบว่าสมการที่เหมาะสำหรับการคำนวณหาขนาดด้วยวิธีฟังก์ชันพอยต์สำหรับข้อมูลในงานวิจัยนี้ คือ สมการที่ 4 (Jeffery et al., 2001) ซึ่งค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ (MMRE) น้อยที่สุดอยู่ที่ 36% และมีสัดส่วนจำนวนระบบที่ประมาณขนาดคลาดเคลื่อนน้อยกว่า 25% อยู่ 61%

ตารางที่ 4.8 การเปรียบเทียบความแม่นยำจากแต่ละสมการของวิธีฟังก์ชันพอยต์

สมการ	MMRE	PRED(0.25)
(1) Effort = FP*6.667 (Moser & Nierstrasz, 1996)	2.85	0.00
(2) Effort = 21 x $FP^{0.826}$ (Lokan, C. J.,2000)	5.59	0.00
(3) Effort = $\frac{\text{FP}}{0.337}$ หรือ $\frac{\text{FP}}{0.116}$ (Maxwell & Froselius, 2000)	2.04	0.00
(4) Effort = FP*2.2 (Jeffery et al., 2001)	0.36	0.61

ผู้วิจัยจึงเลือกสมการ 4 เป็นตัวแทนในการวิเคราะห์ความแม่นยำการประมาณขนาดจากวิธีฟังก์ชันพอยต์ เนื่องจากเป็นสมการที่ให้ผลการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ความแม่นยำที่ใกล้ข้อมูลหน่วยตัวอย่างที่ได้มากที่สุด คือ มีค่าเฉลี่ยของความคลาดเคลื่อนสัมพัทธ์ (MMRE) อยู่ที่ 0.36 (หรือ 36%)

ตารางที่ 4.9 ผลการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ (Development Effort) จากการประมาณขนาดซอฟต์แวร์ด้วยวิธีฟังก์ชันพอยต์ (โดยสมการที่ 4)

	EFH_{actual}	UFP-value	$EFH_{pred\text{-}UFP}$	FP-value	$EFH_{pred\text{-}FP}$	MRE
1-1	132.8	72	158.4	65.52	144.144	0.09
1-2	88.8	73	160.6	66.43	146.146	0.65
1-3	133.6	67	147.4	60.97	134.134	0.00
1-4	96	53	116.6	48.23	106.106	0.11
1-5	105.6	54	118.8	49.14	108.108	0.02

ตารางที่ 4.9 (ต่อ) ผลการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ (Development Effort) จากการประมาณขนาดซอฟต์แวร์ด้วยวิธีฟังก์ชันพอยต์ (โดยสมการที่ 4)

	EFH_{actual}	UFP	$EFH_{pred-UFP}$	FP-value	$EFH_{pred-FP}$	MRE
1-6	105.6	54	118.8	49.14	108.108	0.02
1-7	64	69	151.8	62.79	138.138	1.16
1-8	65.6	75	165	68.25	150.15	1.29
1-9	40	47	103.4	42.77	94.094	1.35
1-10	64	61	134.2	55.51	122.122	0.91
1-11	64	56	123.2	50.96	112.112	0.75
2-1	36	19	41.8	14.44	31.768	0.12
2-2	34	22	48.4	16.72	36.784	0.08
2-3	40	19	41.8	14.44	31.768	0.21
2-4	48	31	68.2	23.56	51.832	0.08
2-5	38	22	48.4	16.72	36.784	0.03
2-6	96	26	57.2	19.76	43.472	0.55
2-7	36	25	55	19	41.8	0.16
2-8	33.6	19	41.8	14.44	31.768	0.05
2-9	28	22	48.4	16.72	36.784	0.31
2-10	40	22	48.4	16.72	36.784	0.08
2-11	28	22	48.4	16.72	36.784	0.31
2-12	44	26	57.2	19.76	43.472	0.01

ผลการประมาณขนาดซอฟต์แวร์ด้วยวิธีฟังก์ชันพอยต์แสดงดังตารางที่ 4.9 มีค่าเฉลี่ยของความคลาดเคลื่อนสัมพัทธ์ (MMRE) อยู่ที่ 0.36 และร้อยละของจำนวนระบบที่สามารถทำนายได้ถูกต้องหรือประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ได้อยู่ในช่วงความคลาดเคลื่อนไม่เกิน 25% (PRED(0.25))มีจำนวนทั้งสิ้น 14 ระบบจาก 23 ระบบ หรือ 61 เปอร์เซ็นต์

ตารางที่ 4.10 ความแม่นยำที่ได้จากการประมาณขนาดด้วยวิธีฟังก์ชันพอยต์แยกตามแหล่งที่มาของซอฟต์แวร์

	จำนวน	MMRE	PRED(0.25)
Company 1	11	0.58	0.45
Company 2	12	0.17	0.75
All	23	0.36	0.61

เมื่อนำมาพิจารณาแยกแหล่งที่มาของข้อมูล แสดงได้ดังตารางที่ 4.10 พบว่า แหล่งข้อมูลที่หนึ่ง (Company 1) มีค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ (MMRE) อยู่ที่ 0.58 และมีจำนวนร้อยละ 45 ของระบบย่อยทั้งหมดที่สามารถประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ได้ผิดพลาดไม่เกิน 25% (PRED(0.25)) แหล่งข้อมูลที่สอง มีค่าเฉลี่ยของความคลาดเคลื่อนสัมพัทธ์ (MMRE) อยู่ที่ 17% และมีร้อยละ 75 ของระบบย่อยทั้งหมดที่สามารถประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ (Development effort) ได้ผิดพลาดไม่เกิน 25% (PRED(0.25)) ซึ่งผลสรุปจำนวนระบบย่อยที่ประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ได้ใกล้เคียงมีมากกว่าหรือเท่ากับ 75%

ผู้วิจัยต้องการทดสอบความแม่นยำของการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์ในเชิงสถิติ พบว่า การประมาณขนาดซอฟต์แวร์ที่ยอมรับได้จะประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ได้ผิดพลาดไม่เกิน 25% หรือ มีค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ (MRE) เท่ากับ 0.25 (Conte et al., 1986)

จากการทดสอบดังกล่าว ผู้วิจัยจึงตรวจสอบการแจกแจงของข้อมูลความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์ โดยถ้าข้อมูลมีการแจกแจงปกติ ผู้วิจัยจะใช้การทดสอบสมมติฐานแบบอิงพารามิเตอร์ แต่ถ้าผลการทดสอบพบว่าข้อมูลไม่มีการแจกแจงปกติ จะใช้การทดสอบสมมติฐานด้วยวิธีการแบบไม่อิงพารามิเตอร์ (กัลยา วาณิชย์บัญชา, 2553) โดยมีสมมติฐานของการทดสอบ คือ

H_0: ความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์ มีการแจกแจงปกติ

H_1: ความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์ ไม่มีการแจกแจงปกติ

งานวิจัยนี้หน่วยตัวอย่างมีจำนวนน้อยกว่า 50 หน่วย ดังนั้น ผู้วิจัยจึงใช้เทคนิค Shapiro-Wilk โดยจะปฏิเสธ H_0 ถ้าค่า Sig. ของการทดสอบน้อยกว่าระดับนัยสำคัญที่กำหนด คือ 0.05 ผลสรุปการทดสอบแจกแจงปกติ ดังตารางที่ 4.11

ตารางที่ 4.11 ค่าสถิติการทดสอบการแจกแจงปกติของค่าความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์

	ค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์ (MRE)		
แหล่งที่มาของข้อมูล	ทั้งสองแหล่งข้อมูล	แหล่งข้อมูลที่ 1	แหล่งข้อมูลที่ 2
Sig.	0.000	0.42	0.032
การแจกแจง	ไม่มีการแจกแจงปกติ	มีการแจกแจงปกติ	ไม่มีการแจกแจงปกติ

จากการวิเคราะห์การแจกแจงข้อมูลพบว่าค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์ทั้งสองแหล่งข้อมูล และแหล่งข้อมูลที่ 2 ไม่มีแจกแจงแบบปกติ ผู้วิจัยจึงเลือกใช้สถิติทดสอบของวิลคอกซัน (Wilcoxon Signed Ranks Test) ส่วนแหล่งข้อมูลที่ 1 มีการแจกแจงแบบปกติจึงทดสอบด้วยสถิติที

สำหรับแหล่งข้อมูลที่ 1 ซึ่งมีการแจกแจงปกติ ผู้วิจัยทดสอบด้วยสถิติทีแบบทางเดียว (t-test one way) กำหนดสมมติฐานดังนี้

กำหนดให้

μ คือ ค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ (MMRE) จากการประมาณด้วยวิธีฟังก์ชันพอยต์

H_0: $\mu \leq 0.25$

H_1: $\mu > 0.25$

กำหนดให้ระดับนัยสำคัญเป็น 0.05

H_0: ความแม่นยำของการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์มีความคลาดเคลื่อนน้อยกว่า 25%

H_1: ความแม่นยำของการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์มีความคลาดเคลื่อนมากกว่า 25%

โดยจะปฏิเสธสมมติฐาน H_0 ได้ ต่อเมื่อ ค่า t มากกว่าค่าวิกฤตที่ได้จากการเปิดตาราง เมื่อ df=n-1 ระดับความเชื่อมั่น 95% และ ค่า sig น้อยกว่า 0.05 ครบทั้งสองเงื่อนไข ผลการทดสอบ t-test แบบทางเดียว สามารถแสดงค่าสถิติได้ดังตารางที่ 4.12

ตารางที่ 4.12 ค่าสถิติ t-test ของความแม่นยำจากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์

	ค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์ (MRE)
แหล่งที่มาของข้อมูล	แหล่งข้อมูลที่ 1
t	1.982
Sig. (2-tailed)	.076

สำหรับทั้งสองแหล่งข้อมูล และแหล่งข้อมูลที่ 2 ไม่มีการแจกแจงปกติ ผู้วิจัยจึงเลือกสถิติทดสอบค่ากลางของข้อมูลทดสอบของวิลคอกซัน (Wilcoxon Signed Ranks Test) กำหนดสมมติฐานดังนี้

กำหนดให้

M คือ ค่ากลางของความคลาดเคลื่อนสัมพัทธ์ (MRE) จากการประมาณด้วยวิธีฟังก์ชันพอยต์

H_0: $M \leq 0.25$

H_1: $M > 0.25$

กำหนดให้ระดับนัยสำคัญเป็น 0.05

H_0: ความแม่นยำของการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์มีความคลาดเคลื่อนน้อยกว่า 25%

H_1: ความแม่นยำของการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์มีความคลาดเคลื่อนมากกว่า 25%

โดยที่จะปฏิเสธสมมติฐาน H_0 ได้เมื่อถ้าค่าสถิติ T_- น้อยกว่าหรือเท่ากับ $T_{\text{ตาราง}}$

สำหรับทั้งสองแหล่งข้อมูล ค่าสถิติ T ที่ได้จากการเปิดตารางเมื่อจำนวนข้อมูลเท่ากับ 23 และระดับความเชื่อมั่น 95% คือ 83 เมื่อนำค่ามาเรียงตามสถิติวิลคอกซัน (Wilcoxon Signed Ranks Test) ได้ดังตารางที่ 4.13

ตารางที่ 4.13 ค่าสถิติวิลคอกซันของความแม่นยำจากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์จากข้อมูลทั้งสองแหล่ง

	MRE	|Di|= |MRE-0.25|	Rank	**เครื่องหมาย Di**
1-1	0.09	0.16	7	-
1-2	0.65	0.40	18	+
1-3	0.00	0.25	16	-
1-4	0.11	0.14	6	-
1-5	0.02	0.23	13.5	-
1-6	0.02	0.23	13.5	-
1-7	1.16	0.91	21	+
1-8	1.29	1.04	22	+
1-9	1.35	1.10	23	+
1-10	0.91	0.66	20	+
1-11	0.75	0.50	19	+
2-1	0.12	0.13	5	-
2-2	0.08	0.17	9	-
2-3	0.21	0.04	1	-
2-4	0.08	0.17	9	-
2-5	0.03	0.22	12	-
2-6	0.55	0.30	17	+
2-7	0.16	0.09	4	-
2-8	0.05	0.20	11	-
2-9	0.31	0.06	2.5	+
2-10	0.08	0.17	9	-
2-11	0.31	0.06	2.5	+
2-12	0.01	0.24	15	-

ผลรวมค่า T$_-$ จากตารางที่ 4.13 เท่ากับ 131

สำหรับทั้งแหล่งข้อมูลที่ 2 ค่าสถิติ T ที่ได้จากการเปิดตารางเมื่อจำนวนข้อมูลเท่ากับ 12 และระดับความเชื่อมั่น 95% คือ 17 เมื่อนำค่ามาเรียงตามสถิติวิลคอกซัน (Wilcoxon Signed Ranks Test) ได้ดังตารางที่ 4.14

ตารางที่ 4.14 ค่าสถิติวิลคอกซันของความแม่นยำจากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์จากแหล่งข้อมูลที่สอง

	MRE	|Di|= |MRE-0.25|	Rank	เครื่องหมาย Di
2-1	0.12	0.13	5	-
2-2	0.08	0.17	7	-
2-3	0.21	0.04	1	-
2-4	0.08	0.17	7	-
2-5	0.03	0.22	10	-
2-6	0.55	0.30	12	+
2-7	0.16	0.09	4	-
2-8	0.05	0.20	9	-
2-9	0.31	0.06	2.5	+
2-10	0.08	0.17	7	-
2-11	0.31	0.06	2.5	+
2-12	0.01	0.24	11	-

ผลรวมค่า T_- จากตารางที่ 4.14 เท่ากับ 61

จากตารางที่ 4.12 – 4.14 ได้ผลทดสอบดังนี้

1. จากข้อมูลทั้งสองแหล่งที่มา ได้สถิติทดสอบค่า T_- เท่ากับ 131 ซึ่งมากกว่า $T_{ตาราง}$ คือ 83 ทำให้ไม่สามารถปฏิเสธ H_0 ได้ หรือหมายความว่า ความแม่นยำจากวิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์มีความคลาดเคลื่อนน้อยกว่า 25% สำหรับข้อมูลทั้งสองแหล่ง
2. จากข้อมูลแหล่งที่ 1 ได้สถิติทดสอบค่า t เท่ากับ 1.982 ซึ่งมากกว่า $t_{ตาราง}$ เมื่อ df=10 ที่ระดับความเชื่อมั่น 95% คือ 1.812 และ ผลสรุปค่า Sig. (2-tailed) ที่คำนวณได้เท่ากับ 0.076 เนื่องจากตัวอย่างเป็นการทดสอบสมมติฐานแบบทางเดียวทำให้ค่า Sig ที่นำมาเปรียบเทียบจะต้องนำหารด้วย 2 ก่อน ค่า sig ที่ได้จากการทดสอบ คือ 0.076/2 = 0.038 ซึ่งมีค่าน้อยกว่า Sig ที่กำหนดคือ 0.05 ค่าสถิติที่ได้แสดงให้เห็นว่าสามารถปฏิเสธ H_0 ได้ หรือหมายความว่า ความแม่นยำของวิธีการประมาณด้วยฟังก์ชันพอยต์มีความคลาดเคลื่อนมากกว่า 25% ในข้อมูลแหล่งที่ 1
3. จากข้อมูลแหล่งที่ 2 ได้สถิติทดสอบค่า T_- เท่ากับ 61 ซึ่งมากกว่า $T_{ตาราง}$ คือ 17 ทำให้ไม่สามารถปฏิเสธ H_0 ได้ หรือหมายความว่า ความแม่นยำจากวิธีการประมาณ

ขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์มีความคลาดเคลื่อนน้อยกว่า 25% ในข้อมูลแหล่งที่ 2

ดังนั้น วิธีการประมาณการซอฟต์แวร์ด้วยฟังก์ชันพอยต์เหมาะกับข้อมูลที่ใช้ในงานวิจัยชิ้นนี้ ซึ่งสามารถประมาณขนาดซอฟต์แวร์โดยมีความคลาดเคลื่อนน้อยกว่า 25% สำหรับข้อมูลทั้งสองแหล่งและแหล่งข้อมูลที่สอง

4.3.2 **ความแม่นยำของวิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุ** (Object-oriented Function point)

สำหรับวิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุ ขนาดซอฟต์แวร์ที่ได้จากการนับอยู่ระหว่าง 52.5 – 312.9 ฟังก์ชันพอยต์เชิงวัตถุ โดยมีค่าเฉลี่ยอยู่ที่ 120.22 ฟังก์ชันพอยต์เชิงวัตถุ เมื่อนำมาเปลี่ยนเป็นค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์ตามสมการที่ได้กล่าวไว้ในบทที่ 3 คือ สมการของ Hericko และ Zivkovic (2007) ดังนี้

$\text{Effort}(\text{Man} - \text{hours}) = 0.7 * \text{OOFP}$ โดย OOFP คือ จำนวนฟังก์ชันพอยต์เชิงวัตถุของซอฟต์แวร์

สามารถแสดงผลลัพธ์ดังตารางที่ 4.15

ตารางที่ 4.15 ผลการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ (Development effort) ด้วยวิธีฟังก์ชันพอยต์เชิงวัตถุ

	EFH_{acutal}	OOFP-value	EFH_{pred}	MRE
1-1	132.8	318	222.6	0.68
1-2	88.8	317	221.9	1.50
1-3	133.6	447	312.9	1.34
1-4	96	257	179.9	0.87
1-5	105.6	151	105.7	0.00
1-6	105.6	133	93.1	0.12
1-7	64	232	162.4	1.54
1-8	65.6	316	221.2	2.37
1-9	40	75	52.5	0.31
1-10	64	135	94.5	0.48
1-11	64	216	151.2	1.36

ตารางที่ 4.15 (ต่อ) ผลการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ (Development effort) ด้วยวิธีฟังก์ชันพอยต์เชิงวัตถุ

	EFH_{actaul}	OOFP2-value	EFH_{pred}	MRE
2-1	36	137	95.9	1.66
2-2	34	99	69.3	1.04
2-3	40	108	75.6	0.89
2-4	48	150	105	1.19
2-5	38	81	56.7	0.49
2-6	96	184	128.8	0.34
2-7	36	93	65.1	0.81
2-8	33.6	96	67.2	1.00
2-9	28	96	67.2	1.40
2-10	40	105	73.5	0.84
2-11	28	99	69.3	1.48
2-12	44	105	73.5	0.67

จากตารางที่ 4.15 แสดงให้เห็นว่าวิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุให้ผลความคลาดเคลื่อนไปจากข้อมูลจริงค่อนข้างมาก โดยมีค่าความคลาดเคลื่อนจากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ (Development effort) แต่ละระบบย่อยตั้งแต่ 0.00-2.37 โดยมีค่าเฉลี่ยอยู่ที่ 0.97 นั่นหมายความว่า วิธีการขนาดที่ได้จากประมาณการด้วยซอฟต์แวร์เชิงวัตถุให้ค่าเบี่ยงเบนไปจากข้อมูลจริงถึง 97% โดยเฉลี่ย

ตารางที่ 4.16 ความแม่นยำที่ได้จากการประมาณขนาดด้วยวิธีฟังก์ชันพอยต์เชิงวัตถุแยกตามแหล่งที่มาของซอฟต์แวร์

	จำนวน	MMRE	PRED(0.25)
Company 1	11	0.96	0.18
Company 2	12	0.98	0.00
All	23	0.97	0.09

เมื่อนำมาพิจารณาแยกแหล่งที่มาของข้อมูลแสดงได้ดังตารางที่ 4.16 ผลจากทั้งสองแหล่งข้อมูลที่ได้ไม่แตกต่างกันทั้งสองแหล่งข้อมูล ซึ่งมีค่าเฉลี่ยความคลาดเคลื่อนจากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ที่ใกล้เคียงกัน คือ 0.96 และ 0.98 ตามลำดับ

ผู้วิจัยต้องการทดสอบความแม่นยำของการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์ในเชิงสถิติ พบว่า การประมาณขนาดซอฟต์แวร์ที่ยอมรับได้จะประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ได้ผิดพลาดไม่เกิน 25% หรือ มีค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ (MRE) เท่ากับ 0.25 (Conte et al., 1986)

จากการทดสอบดังกล่าว ผู้วิจัยจึงตรวจสอบการแจกแจงของข้อมูลความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุ โดยถ้าข้อมูลมีการแจกแจงปกติ ผู้วิจัยจะใช้การทดสอบสมมติฐานแบบอิงพารามิเตอร์ แต่ถ้าผลการทดสอบพบว่าข้อมูลไม่มีการแจกแจงปกติ จะใช้การทดสอบสมมติฐานด้วยวิธีการแบบไม่อิงพารามิเตอร์ (กัลยา วาณิชย์บัญชา, 2553) โดยมีสมมติฐานของการทดสอบ คือ

H_0: ความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุ มีการแจกแจงปกติ

H_1: ความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุ ไม่มีการแจกแจงปกติ

งานวิจัยนี้หน่วยตัวอย่างมีจำนวนน้อยกว่า 50 หน่วย ดังนั้น ผู้วิจัยจึงใช้เทคนิค Shapiro-Wilk โดยจะปฏิเสธ H_0 ถ้าค่า Sig. ของการทดสอบน้อยกว่าระดับนัยสำคัญที่กำหนด คือ 0.05 ผลสรุปการทดสอบแจกแจงปกติ ดังตารางที่ 4.17

ตารางที่ 4.17 ค่าสถิติการทดสอบการแจกแจงปกติของค่าความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์

	ค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุ (MRE)		
แหล่งที่มาของข้อมูล	ทั้งสองแหล่งข้อมูล	แหล่งข้อมูลที่ 1	แหล่งข้อมูลที่ 2
Sig.	0.803	0.622	0.982
การแจกแจง	มีการแจกแจงปกติ	มีการแจกแจงปกติ	มีการแจกแจงปกติ

จากการวิเคราะห์การแจกแจงข้อมูลพบว่าค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุ พบว่า ทุกแหล่งข้อมูลมีการแจกแจงแบบปกติ ผู้วิจัยจึงทดสอบด้วยสถิติทีแบบทางเดียว (t-test one way) จึงกำหนดสมมติฐานดังนี้

กำหนดให้

H_0: $\mu \leq 0.25$

H_1: $\mu > 0.25$

กำหนดให้ระดับนัยสำคัญเป็น 0.05

H_0: ความแม่นยำของการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุมีความคลาดเคลื่อนน้อยกว่า 25%

H_1: ความแม่นยำของการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุมีความคลาดเคลื่อนมากกว่า 25%

ทดสอบด้วยสถิติ t-test แบบทางเดียว โดยจะปฏิเสธสมมติฐาน H_0 ได้ ต่อเมื่อ ค่า t มากกว่าวิกฤตที่ได้จากการเปิดตาราง เมื่อ df=n-1 และ ค่า sig น้อยกว่า 0.05 ครบทั้งสองเงื่อนไข ผลการทดสอบ t-test แบบทางเดียว สามารถแสดงค่าสถิติได้ดังตารางที่ 4.18

ตารางที่ 4.18 ค่าสถิติ t-test ของความแม่นยำจากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุ

	ค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุ (MRE)		
แหล่งที่มาของข้อมูล	ทั้งสองแหล่งข้อมูล	แหล่งข้อมูลที่ 1	แหล่งข้อมูลที่ 2
t	6.135	3.243	6.396
Sig. (2-tailed)	.000	.009	.000

จากตารางที่ 4.18 ได้ผลทดสอบดังนี้

1. จากข้อมูลทั้งสองแหล่งที่มา ได้สถิติทดสอบค่า t เท่ากับ 6.135 ซึ่งมากกว่าค่าวิกฤตของ t เมื่อ df=22 คือ 1.717 และ ผลสรุปค่า Sig. (2-tailed) ที่คำนวณได้เท่ากับ 0.000 เนื่องจากตัวอย่างเป็นการทดสอบสมมติฐานแบบทางเดียวทำให้ค่า Sig ที่นำมาเปรียบเทียบจะต้องนำหารด้วย 2 ก่อน ค่า sig ที่ได้จากการทดสอบ คือ

0.000/2 = 0.000 ซึ่งมีค่าน้อยกว่า Sig ที่กำหนดคือ 0.05 ค่าสถิติที่ได้แสดงให้เห็นว่าสามารถปฏิเสธ H_0 ได้ นั่นหมายความว่า ความแม่นยำของวิธีการประมาณด้วยฟังก์ชันพอยต์เชิงวัตถุมีความคลาดเคลื่อนมากกว่า 25%

2. จากข้อมูลแหล่งที่ 1 ได้สถิติทดสอบค่า t เท่ากับ 3.243 ซึ่งมากกว่าค่าวิกฤตของ t เมื่อ df=10 คือ 1.812 และ ผลสรุปค่า Sig. (2-tailed) ที่คำนวณได้เท่ากับ 0.009 เนื่องจากตัวอย่างเป็นการทดสอบสมมติฐานแบบทางเดียวทำให้ค่า Sig ที่นำมาเปรียบเทียบจะต้องนำหารด้วย 2 ก่อน ค่า sig ที่ได้จากการทดสอบ คือ 0.009/2 = 0.0045 ซึ่งมีค่าน้อยกว่า Sig ที่กำหนดคือ 0.05 ค่าสถิติที่ได้แสดงให้เห็นว่าสามารถปฏิเสธ H_0 ได้ นั่นหมายความว่า ความแม่นยำของวิธีการประมาณด้วยฟังก์ชันพอยต์เชิงวัตถุมีความคลาดเคลื่อนมากกว่า 25% ได้ในข้อมูลแหล่งที่ 1
3. จากข้อมูลแหล่งที่ 2 ได้สถิติทดสอบค่า t เท่ากับ 6.396 ซึ่งมากกว่าค่าวิกฤตของ t เมื่อ df=11 คือ 1.796 และ ผลสรุปค่า Sig. (2-tailed) ที่คำนวณได้เท่ากับ 0.000 เนื่องจากตัวอย่างเป็นการทดสอบสมมติฐานแบบทางเดียวทำให้ค่า Sig ที่นำมาเปรียบเทียบจะต้องนำหารด้วย 2 ก่อน ค่า sig ที่ได้จากการทดสอบ คือ 0.000/2 = 0.000 ซึ่งมีค่าน้อยกว่า Sig ที่กำหนดคือ 0.05 ค่าสถิติที่ได้แสดงให้เห็นว่าสามารถปฏิเสธ H_0 ได้ นั่นหมายความว่า ความแม่นยำของวิธีการประมาณด้วยฟังก์ชันพอยต์เชิงวัตถุมีความคลาดเคลื่อนมากกว่า 25% ได้ในข้อมูลแหล่งที่ 2

ดังนั้น ผลที่ได้สามารถสรุปได้ชัดเจนว่า วิธีการประมาณการซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุไม่เหมาะสำหรับการประมาณขนาดซอฟต์แวร์ในงานวิจัยนี้ โดยมีความคลาดเคลื่อนมากกว่า 25% อย่างมีนัยสำคัญ

4.3.3 **ความแม่นยำของวิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุ** 2 (Object-oriented Function point 2)

สำหรับขนาดที่ได้จากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุ 2 ใกล้เคียงกับวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุที่ถูกนำเสนอขึ้นมาก่อนหน้า ขนาดที่ได้จากการนับอยู่ที่ 52.5 – 315 ฟังก์ชันพอยต์เชิงวัตถุ โดยมีค่าเฉลี่ยอยู่ที่ 122.02 ฟังก์ชันพอยต์เชิงวัตถุ เมื่อนำมาเปลี่ยนเป็นค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์ตามสมการที่ได้กล่าวไว้ในบทที่ 3 คือ สมการของ Hericko และ Zivkovic (2007)

$\text{Effort}(\text{Man} - \text{hours}) = 0.7 * \text{OOFP}$ โดย OOFP คือ จำนวนฟังก์ชันพอยต์เชิงวัตถุของซอฟต์แวร์

สามารถแสดงผลลัพธ์ดังตารางที่ 4.19

ตารางที่ 4.19 ผลการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ (Development effort) ด้วยวิธีฟังก์ชันพอยต์เชิงวัตถุสอง

	EFH_{actaul}	OOFP2-value	EFH_{pred}	MRE
1-1	132.8	324	226.8	0.71
1-2	88.8	319	223.3	1.51
1-3	133.6	450	315	1.36
1-4	96	261	182.7	0.90
1-5	105.6	151	105.7	0.00
1-6	105.6	131	91.7	0.13
1-7	64	232	162.4	1.54
1-8	65.6	316	221.2	2.37
1-9	40	75	52.5	0.31
1-10	64	135	94.5	0.48
1-11	64	216	151.2	1.36
2-1	36	142	99.4	1.76
2-2	34	101	70.7	1.08
2-3	40	110	77	0.93
2-4	48	156	109.2	1.28
2-5	38	85	59.5	0.57
2-6	96	199	139.4	0.45
2-7	36	95	66.5	0.85
2-8	33.6	98	68.6	1.04
2-9	28	98	68.6	1.45
2-10	40	107	74.9	0.87
2-11	28	101	70.7	1.53
2-12	44	107	74.9	0.70

จากตารางที่ 4.19 ผลลัพธ์ความแม่นยำใกล้เคียงกับฟังก์ชันพอยต์เชิงวัตถุที่ถูกนำเสนอก่อน คือ ให้ผลที่เกินกับข้อมูลจริงอยู่มาก โดยมีค่าความคลาดเคลื่อนสัมพัทธ์ (MRE) ระหว่าง 0.00-2.37 และมีค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ (MMRE) อยู่ที่ 1.01 หรือประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ผิดพลาดมากกว่าความเป็นจริงถึง 101%

ตารางที่ 4.20 ความแม่นยำที่ได้จากการประมาณขนาดด้วยวิธีฟังก์ชันพอยต์เชิงวัตถุสองแยกตามแหล่งที่มาของซอฟต์แวร์

	จำนวน	MMRE	PRED(0.25)
Company 1	11	0.97	0.18
Company 2	12	1.04	0.00
All	23	1.01	0.09

จากตารางที่ 4.20 แสดงให้เห็นว่า ความแม่นยำจากทั้งสองแหล่งข้อมูลไม่แตกต่าง กล่าวคือ ให้ค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ (MMRE) อยู่ที่ 0.97 และ 1.04 ตามลำดับ

ผู้วิจัยต้องการทดสอบความแม่นยำของการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์ในเชิงสถิติ พบว่า การประมาณขนาดซอฟต์แวร์ที่ยอมรับได้จะประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ได้ผิดพลาดไม่เกิน 25% หรือ มีค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ (MRE) เท่ากับ 0.25 (Conte et al., 1986)

จากการทดสอบดังกล่าว ผู้วิจัยจึงตรวจสอบการแจกแจงของข้อมูลความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุสอง โดยถ้าข้อมูลมีการแจกแจงปกติ ผู้วิจัยจะใช้การทดสอบสมมติฐานแบบอิงพารามิเตอร์ แต่ถ้าผลการทดสอบพบว่าข้อมูลไม่มีการแจกแจงปกติ จะใช้การทดสอบสมมติฐานด้วยวิธีการแบบไม่อิงพารามิเตอร์ (กัลยา วาณิชย์บัญชา, 2553) โดยมีสมมติฐานของการทดสอบ คือ

H_0: ความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุสอง มีการแจกแจงปกติ

H_1: ความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุสอง ไม่มีการแจกแจงปกติ

งานวิจัยนี้หน่วยตัวอย่างมีจำนวนน้อยกว่า 50 หน่วย ดังนั้น ผู้วิจัยจึงใช้เทคนิค Shapiro-Wilk โดยจะปฏิเสธ H_0 ถ้าค่า Sig. ของการทดสอบน้อยกว่าระดับนัยสำคัญที่กำหนด คือ 0.05 ผลสรุปการทดสอบแจกแจงปกติ ดังตารางที่ 4.21

ตารางที่ 4.21 ค่าสถิติการทดสอบการแจกแจงปกติของค่าความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุสอง

	ค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุสอง (MRE)		
แหล่งที่มาของข้อมูล	ทั้งสองแหล่งข้อมูล	แหล่งข้อมูลที่ 1	แหล่งข้อมูลที่ 2
Sig.	0.913	0.629	0.932
การแจกแจง	มีการแจกแจงปกติ	มีการแจกแจงปกติ	มีการแจกแจงปกติ

จากการวิเคราะห์การแจกแจงข้อมูลพบว่าค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุสอง พบว่า ทุกแหล่งข้อมูลมีการแจกแจงแบบปกติ ผู้วิจัยจึงทดสอบด้วยสถิติ t (t-test) จึงกำหนดสมมติฐานดังนี้

กำหนดให้

H_0: $\mu \leq 0.25$

H_1: $\mu > 0.25$

กำหนดให้ระดับนัยสำคัญเป็น 0.05

H_0: ความแม่นยำของการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุสองมีความคลาดเคลื่อนน้อยกว่า 25%

H_1: ความแม่นยำของการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุสองมีความคลาดเคลื่อนมากกว่า 25%

ทดสอบด้วยสถิติ t-test แบบทางเดียว โดยจะปฏิเสธสมมติฐาน H_0 ได้ ต่อเมื่อ ค่า t มากกว่าวิกฤตจากการเปิดตาราง เมื่อ df=n-1 และ ค่า sig น้อยกว่า 0.05 ครบทั้งสองเงื่อนไข ผลการทดสอบ t-test แบบทางเดียว สามารถแสดงค่าสถิติได้ดังตารางที่ 4.22

ตารางที่ 4.22 ค่าสถิติ t-test ของความแม่นยำจากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุสอง

	ค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุสอง (MRE)		
แหล่งที่มาของข้อมูล	ทั้งสองแหล่งข้อมูล	แหล่งข้อมูลที่ 1	แหล่งข้อมูลที่ 2
t	6.419	3.288	6.888
Sig. (2-tailed)	.000	0.008	0.000

จากตารางที่ 4.22 ได้ผลทดสอบดังนี้

1. จากข้อมูลทั้งสองแหล่งที่มา ได้สถิติทดสอบค่า t เท่ากับ 6.419 ซึ่งมากกว่าค่าวิกฤตของ t เมื่อ df=22 คือ 1.717 และ ผลสรุปค่า Sig. (2-tailed) ที่คำนวณได้เท่ากับ 0.000 เนื่องจากตัวอย่างเป็นการทดสอบสมมติฐานแบบทางเดียวทำให้ค่า Sig ที่นำมาเปรียบเทียบจะต้องนำหารด้วย 2 ก่อน ค่า sig ที่ได้จากการทดสอบ คือ 0.000/2 = 0.000 ซึ่งมีค่าน้อยกว่า Sig ที่กำหนดคือ 0.05 ค่าสถิติที่ได้แสดงให้เห็นว่าสามารถปฏิเสธ H_0 หรือหมายความว่า ความแม่นยำของวิธีการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุสองมีความคลาดเคลื่อนมากกว่า 25%
2. จากข้อมูลแหล่งที่ 1 ได้สถิติทดสอบค่า t เท่ากับ 3.288 ซึ่งมากกว่าค่าวิกฤตของ t เมื่อ df=10 คือ 1.812 และ ผลสรุปค่า Sig. (2-tailed) ที่คำนวณได้เท่ากับ 0.008 เนื่องจากตัวอย่างเป็นการทดสอบสมมติฐานแบบทางเดียวทำให้ค่า Sig ที่นำมาเปรียบเทียบจะต้องนำหารด้วย 2 ก่อน ค่า sig ที่ได้จากการทดสอบ คือ 0.008/2 = 0.004 ซึ่งมีค่าน้อยกว่า Sig ที่กำหนดคือ 0.05 ค่าสถิติที่ได้แสดงให้เห็นว่าสามารถปฏิเสธ H_0 หรือหมายความว่า ความแม่นยำของวิธีการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุสองมีความคลาดเคลื่อนมากกว่า 25% ในข้อมูลแหล่งที่ 1
3. จากข้อมูลแหล่งที่ 2 ได้สถิติทดสอบค่า t เท่ากับ 6.888 ซึ่งมากกว่าค่าวิกฤตของ t เมื่อ df=11 คือ 1.796 และ ผลสรุปค่า Sig. (2-tailed) ที่คำนวณได้เท่ากับ 0.000 เนื่องจากตัวอย่างเป็นการทดสอบสมมติฐานแบบทางเดียวทำให้ค่า Sig ที่นำมา

เปรียบเทียบจะต้องนำหารด้วย 2 ก่อน ค่า sig ที่ได้จากการทดสอบ คือ 0.000/2 = 0.000 ซึ่งมีค่าน้อยกว่า Sig ที่กำหนดคือ 0.05 ค่าสถิติที่ได้แสดงให้เห็นว่าสามารถปฏิเสธ H_0 หรือหมายความว่า ความแม่นยำของวิธีการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุมีความคลาดเคลื่อนมากกว่า 25% ในข้อมูลแหล่งที่ 2

ดังนั้น วิธีการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุสองไม่เหมาะสำหรับการประมาณขนาดซอฟต์แวร์ในงานวิจัย โดยมีความคลาดเคลื่อนมากกว่า 25% อย่างมีนัยสำคัญ

4.3.4 **ความแม่นยำของวิธีการประมาณขนาดซอฟต์แวร์ด้วยคลาสพอยต์** (Class point)

สำหรับวิธีการประมาณขนาดด้วยคลาสพอยต์ ขนาดที่ได้จากการนับก่อนการใช้ตัวแปรปรับค่า (Unadjusted class point) อยู่ที่ 31.80 – 171.25 คลาสพอยต์ และมีค่าเฉลี่ยคลาสพอยต์ก่อนการปรับค่าอยู่ที่ 66.74 คลาสพอยต์ ในขณะที่ขนาดหลังใช้ตัวแปรปรับค่าของคลาสพอยต์จำนวนทั้งสิ้น 18 ตัวที่ถูกนำเสนอของ Costagliola และคณะ (2005) อยู่ที่ 23.54-148.99 คลาสพอยต์ โดยมีค่าเฉลี่ยอยู่ที่ 55.54 คลาสพอยต์ เมื่อนำมาแปลงเป็นค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์ที่กล่าวไว้ในบทที่ 3 จำนวนหนึ่งสมการ (Costagliola et al., 2005)

$\mathrm{Effort(Man-hours)} = 1.2232 * \mathrm{CP}$ โดย CP คือ จำนวนคลาสพอยต์ของซอฟต์แวร์

ตารางที่ 4.23 ผลการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ (Development effort) ด้วยวิธีคลาสพอยต์

	EFH_{real}	UCP-value	EFH_{pred}	MRE	CP-value	EFH_{pred}	MRE
1-1	132.8	91	111.31	0.16	79.17	96.84	0.27
1-2	88.8	100	122.32	0.38	87.00	106.42	0.20
1-3	133.6	108	132.11	0.01	93.96	114.93	0.14
1-4	96	80	97.86	0.02	69.60	85.13	0.11
1-5	105.6	49	59.94	0.43	42.63	52.15	0.51
1-6	105.6	59	72.17	0.32	51.33	62.79	0.41
1-7	64	89	108.86	0.70	77.43	94.71	0.48

ตารางที่ 4.23 (ต่อ) ผลการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ (Development effort) ด้วยวิธีคลาสพอยต์

	EFH_{real}	UCP-value	EFH_{pred}	MRE	CP-value	EFH_{pred}	MRE
1-8	65.6	140	171.25	1.61	121.80	148.99	1.27
1-9	40	37	45.26	0.13	32.19	39.37	0.02
1-10	64	51	62.38	0.03	44.37	54.27	0.15
1-11	64	85	103.97	0.62	73.95	90.46	0.41
2-1	36	29	35.47	0.01	21.46	26.25	0.27
2-2	34	26	31.80	0.06	19.24	23.53	0.31
2-3	40	35	42.81	0.07	25.90	31.68	0.21
2-4	48	43	52.60	0.10	31.82	38.92	0.19
2-5	38	26	31.80	0.16	19.24	23.53	0.38
2-6	96	51	62.38	0.35	37.74	46.16	0.52
2-7	36	26	31.80	0.12	19.24	23.53	0.35
2-8	33.6	26	31.80	0.05	19.24	23.53	0.30
2-9	28	26	31.80	0.14	19.24	23.53	0.16
2-10	40	26	31.80	0.20	19.24	23.53	0.41
2-11	28	26	31.80	0.14	19.24	23.53	0.16
2-12	44	26	31.80	0.28	19.24	23.53	0.47

จากตารางที่ 4.23 ผลสรุปพบว่า ความแม่นยำก่อนใช้ตัวแปรปรับค่าดีกว่าความแม่นยำหลังใช้ตัวแปรปรับค่า โดยก่อนใช้ตัวแปรปรับค่ามีค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ (MMRE) อยู่ที่ 26% และมีจำนวนระบบย่อยที่ประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ได้ผิดพลาดไม่เกิน 25% (PRED(0.25)) จำนวน 65% ในขณะที่เมื่อใช้ตัวแปรปรับค่าแล้ว ค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ (MMRE) 33% และมีจำนวนระบบย่อยที่ประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ได้ผิดพลาดไม่เกิน 25% (PRED(0.25)) จำนวน 39%

ผลสรุปนี้ไม่ตรงกับงานวิจัยของผู้นำเสนอวิธีการประมาณขนาดด้วยคลาสพอยต์ (Costagliola et al., 2005) โดยในงานวิจัยดังกล่าว ขนาดที่ใช้ตัวแปรปรับค่าของคลาสพอยต์ให้ผลความแม่นยำมากกว่าขนาดที่ไม่ใช้ตัวแปรปรับค่าของคลาสพอยต์ แต่ยังไม่พบงานวิจัยอื่น

ที่วิเคราะห์เรื่องดังกล่าว ผู้วิจัยจึงนำขนาดก่อนใช้ตัวแปรปรับค่าและหลังใช้ตัวแปรปรับค่ามาวิเคราะห์ความแตกต่าง ด้วยการวิเคราะห์ความแปรปรวน (Analysis of Variance: ANOVA) ด้วยการวิเคราะห์ความแปรปรวนแบบทางเดียว (One-Way ANOVA) โดยมีสมมติฐานการทดสอบ คือ

กำหนดให้ μ_1 คือ ค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ในการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยคลาสพอยต์ที่ไม่ใช้ตัวแปรปรับค่า (MMRE)

μ_2 คือ ค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ในการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยคลาสพอยต์ที่ใช้ตัวแปรปรับค่า (MMRE)

$H_0: \mu_1 = \mu_2$ (ความคลาดเคลื่อนในการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ก่อนใช้ตัวแปรปรับค่าเท่ากับหลังใช้ตัวแปรปรับค่า)

$H_1: \mu_1 \neq \mu_2$ เมื่อ $i \neq j$ (ความคลาดเคลื่อนในการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ก่อนใช้ตัวแปรปรับค่าไม่เท่ากับหลังใช้ตัวแปรปรับค่า)

เนื่องจากเป็นการทดสอบระหว่างค่าเฉลี่ยความคลาดเคลื่อนระหว่างสองกลุ่มจึงเลือกใช้การวิเคราะห์ความแปรปรวน (Analysis of Variance: ANOVA) ด้วยการวิเคราะห์ความแปรปรวนแบบทางเดียว (One-Way ANOVA) โดยจะปฏิเสธ H_0 ถ้าค่า Sig. (Significance) จากการทดสอบมีค่าน้อยกว่าระดับนัยสำคัญที่กำหนด ในงานวิจัยนี้กำหนดระดับนัยสำคัญที่ 0.05

เมื่อนำข้อมูลมาทดสอบการวิเคราะห์ความแปรปรวนแบบทางเดียว (One-Way ANOVA) แสดงค่าได้ ดังตารางที่ 4.24

ตารางที่ 4.24 ค่าสถิติการทดสอบการวิเคราะห์ความแปรปรวนแบบทางเดียวของความคลาดเคลื่อนจากการประมาณขนาดซอฟต์แวร์ก่อนใช้ตัวแปรปรับค่าและหลังใช้ตัวแปรปรับค่า

Levene Statistic	df1	df2	Sig.
.851	1	44	.361

จากผลการทดสอบในตารางที่ 4.24 พบว่า ค่า Sig. (Significance) มีค่าเท่ากับ 0.361 ซึ่งมากกว่าค่านัยสำคัญที่งานวิจัยนี้กำหนด (0.05) ทำให้สามารถยอมรับ H_0 นั่นหมายความว่า ค่าเฉลี่ยความคลาดเคลื่อนในการประมาณค่าความพยายามด้วยคลาสพอยต์ที่ไม่ได้ใช้ตัวแปรปรับค่าและค่าเฉลี่ยความคลาดเคลื่อนในการประมาณค่าความพยายามด้วยคลาสพอยต์ที่ใช้ตัวแปรปรับค่าไม่แตกต่างกัน ผู้วิจัยจึงใช้เลือกใช้ขนาดหลังใช้ตัวแปรปรับค่า เนื่องจากในการคำนวณขนาดกับระบบในธุรกิจจริงขนาดที่ได้จากการคำนวณต้องเป็นขนาดที่ถูกปรับตัวแปรปรับค่าเรียบร้อยแล้ว

ข้อมูลสรุปความแม่นยำจากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยคลาสพอยต์ได้ ดังตารางที่ 4.25

ตารางที่ 4.25 ความแม่นยำที่ได้จากการประมาณขนาดด้วยวิธีคลาสพอยต์แยกตามแหล่งที่มาของซอฟต์แวร์

	จำนวน	MMRE	PRED(0.25)
Company 1	11	0.36	0.45
Company 2	12	0.31	0.33
All	23	0.33	0.39

ผลสรุปในตารางที่ 4.25 แสดงให้เห็นว่า ผลที่ได้จากวิธีการประมาณขนาดซอฟต์แวร์ด้วยคลาสพอยต์พบว่า ค่าเฉลี่ยความคลาดเคลื่อนอยู่ที่ 33% และมีจำนวนระบบย่อยที่ประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ได้ผิดพลาดไม่เกิน 25% (PRED(0.25)) จำนวน 39% ซึ่งเมื่อนำมาพิจาณาแยกแหล่งที่มาของข้อมูล พบว่า แหล่งข้อมูลที่หนึ่งให้ค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ (MMRE) อยู่ที่ 0.36 และมีจำนวนระบบย่อยที่ประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ได้ผิดพลาดไม่เกิน 25% (PRED(0.25)) จำนวน 45% แหล่งข้อมูลที่สองให้ค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ (MMRE) 0.31 และมีจำนวนระบบย่อยที่ประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ได้ผิดพลาดไม่เกิน 25% (PRED(0.25)) จำนวน 33%

ผู้วิจัยต้องการทดสอบความแม่นยำของการประมาณขนาดซอฟต์แวร์ด้วยคลาสพอยต์ในเชิงสถิติ พบว่า การประมาณขนาดซอฟต์แวร์ที่ยอมรับได้จะประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ได้ผิดพลาดไม่เกิน 25% หรือ มีค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ (MRE) เท่ากับ 0.25 (Conte et al., 1986)

จากการทดสอบดังกล่าว ผู้วิจัยจึงตรวจสอบการแจกแจงของข้อมูลความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยคลาสพอยต์ โดยถ้าข้อมูลมีการแจกแจงปกติ ผู้วิจัยจะใช้การทดสอบสมมติฐานแบบอิงพารามิเตอร์ แต่ถ้าผลการทดสอบพบว่าข้อมูลไม่มีการแจกแจงปกติ จะใช้การทดสอบสมมติฐานด้วยวิธีการแบบไม่อิงพารามิเตอร์ (กัลยา วาณิชย์บัญชา, 2553) โดยมีสมมติฐานของการทดสอบ คือ

H_0: ความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยคลาสพอยต์ มีการแจกแจงปกติ

H_1: ความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยคลาสพอยต์ ไม่มีการแจกแจงปกติ

งานวิจัยนี้หน่วยตัวอย่างมีจำนวนน้อยกว่า 50 หน่วย ดังนั้น ผู้วิจัยจึงใช้เทคนิค Shapiro-Wilk โดยจะปฏิเสธ H_0 ถ้าค่า Sig. ของการทดสอบน้อยกว่าระดับนัยสำคัญที่กำหนด คือ 0.05 ผลสรุปการทดสอบแจกแจงปกติ ดังตารางที่ 4.26

ตารางที่ 4.26 ค่าสถิติการทดสอบการแจกแจงปกติของค่าความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยคลาสพอยต์

	ค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยคลาสพอยต์ (MRE)		
แหล่งที่มาของข้อมูล	ทั้งสองแหล่งข้อมูล	แหล่งข้อมูลที่ 1	แหล่งข้อมูลที่ 2
Sig.	0.000	0.006	0.631
การแจกแจง	ไม่มีการแจกแจงปกติ	ไม่มีการแจกแจงปกติ	มีการแจกแจงปกติ

จากการวิเคราะห์การแจกแจงข้อมูลพบว่าค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยคลาสพอยต์ทั้งสองแหล่งข้อมูล และแหล่งข้อมูลที่ 1 ไม่มีแจกแจงแบบปกติ ผู้วิจัยจึงเลือกใช้สถิติทดสอบของวิลคอกซัน (Wilcoxon Signed Ranks Test) ส่วนแหล่งข้อมูลที่ 2 มีการแจกแจงแบบปกติ จึงใช้สถิติที

สำหรับแหล่งข้อมูลที่ 2 ซึ่งมีการแจกแจงปกติ ผู้วิจัยจึงทดสอบด้วยสถิติทีแบบทางเดียว (t-test one way) จึงกำหนดสมมติฐานดังนี้

กำหนดให้

H_0: $\mu \leq 0.25$

H_1: $\mu > 0.25$

กำหนดให้ระดับนัยสำคัญเป็น 0.05

H_0: ความแม่นยำของการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยคลาสพอยต์มีความคลาดเคลื่อนน้อยกว่า 25%

H_1: ความแม่นยำของการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยคลาสพอยต์มีความคลาดเคลื่อนมากกว่า 25%

สำหรับแหล่งข้อมูลที่ 2 ซึ่งมีการแจกแจงปกติ ผู้วิจัยจึงทดสอบด้วยสถิติทีแบบทางเดียว (t-test one way) โดยจะปฏิเสธสมมติฐาน H_0 ได้ ต่อเมื่อ ค่า t มากกว่าค่าวิกฤตซึ่งได้จากการเปิดตาราง เมื่อ df = n-1 ที่ระดับความเชื่อมั่น 95%และ ค่า sig น้อยกว่า 0.05 ครบทั้งสองเงื่อนไข ผลการทดสอบ t-test แบบทางเดียว สามารถแสดงค่าสถิติได้ดังตารางที่ 4.27

ตารางที่ 4.27 ค่าสถิติ t-test ของความแม่นยำจากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยคลาสพอยต์

	ค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์จากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยคลาสพอยต์ (MRE)
แหล่งที่มาของข้อมูล	แหล่งข้อมูลที่ 2
t	1.762
Sig. (2-tailed)	0.106

สำหรับทั้งสองแหล่งข้อมูล และแหล่งข้อมูลที่ 2 ไม่มีการแจกแจงปกติ ผู้วิจัยจึงเลือกสถิติทดสอบค่ากลางของข้อมูลทดสอบของวิลคอกซัน (Wilcoxon Signed Ranks Test) กำหนดสมมติฐานดังนี้

กำหนดให้

M คือ ค่ากลางของความคลาดเคลื่อนสัมพัทธ์ (MRE) จากการประมาณด้วยวิธีคลาสพอยต์

H_0: $M \leq 0.25$

H_1: $M > 0.25$

กำหนดให้ระดับนัยสำคัญเป็น 0.05

H_0: ความแม่นยำของการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยคลาสพอยต์มีความคลาดเคลื่อนน้อยกว่า 25%

H_1: ความแม่นยำของการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยคลาสพอยต์มีความคลาดเคลื่อนมากกว่า 25%

โดยที่จะปฏิเสธสมมติฐาน H_0 ได้เมื่อถ้าค่าสถิติ T_- น้อยกว่าหรือเท่ากับ $T_{ตาราง}$

สำหรับทั้งสองแหล่งข้อมูล ค่าสถิติ T ที่ได้จากการเปิดตารางเมื่อจำนวนข้อมูลเท่ากับ 23 และระดับความเชื่อมั่น 95% คือ 83 เมื่อนำค่ามาเรียงตามสถิติวิลคอกซัน (Wilcoxon Signed Ranks Test) ได้ดังตารางที่ 4.28

ตารางที่ 4.28 ค่าสถิติวิลคอกซันของความแม่นยำจากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยคลาสพอยต์จากข้อมูลทั้งสองแหล่ง

	MRE	|Di|= |MRE-0.25|	Rank	เครื่องหมาย Di
1-1	0.27	0.02	1.5	+
1-2	0.20	0.05	4.5	-
1-3	0.14	0.11	12	-
1-4	0.11	0.14	14	-
1-5	0.51	0.26	21	+
1-6	0.41	0.16	16	+
1-7	0.48	0.23	19.5	+
1-8	1.27	1.02	23	+
1-9	0.02	0.23	19.5	-
1-10	0.15	0.10	10.5	-
1-11	0.41	0.16	16	+
2-1	0.27	0.02	1.5	+
2-2	0.31	0.06	6	+
2-3	0.21	0.04	3	-
2-4	0.19	0.06	7	-
2-5	0.38	0.13	13	+
2-6	0.52	0.27	22	+

ตารางที่ 4.28 (ต่อ) ค่าสถิติวิลคอกซันของความแม่นยำจากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยคลาสพอยต์จากข้อมูลทั้งสองแหล่ง

	MRE	|Di|= |MRE-0.25|	Rank	**เครื่องหมาย Di**
2-7	0.35	0.10	10.5	+
2-8	0.30	0.05	4.5	+
2-9	0.16	0.09	8.5	-
2-10	0.41	0.16	16	+
2-11	0.16	0.09	8.5	-
2-12	0.47	0.22	18	+

ผลรวมค่า T- จากตารางที่ 4.28 เท่ากับ 87.5

สำหรับทั้งสองแหล่งข้อมูล ค่าสถิติ T ที่ได้จากการเปิดตารางเมื่อจำนวนข้อมูลเท่ากับ 11 และระดับความเชื่อมั่น 95% คือ 5 เมื่อนำค่ามาเรียงตามสถิติวิลคอกซัน (Wilcoxon Signed Ranks Test) ได้ดังตารางที่ 4.29

ตารางที่ 4.29 ค่าสถิติวิลคอกซันของความแม่นยำจากการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยคลาสพอยต์จากแหล่งข้อมูลที่หนึ่ง

	MRE	|Di|= |MRE-0.25|	Rank	**เครื่องหมาย Di**
1-1	0.27	0.02	1	+
1-2	0.20	0.05	2	-
1-3	0.14	0.11	4	-
1-4	0.11	0.14	5	-
1-5	0.51	0.26	10	+
1-6	0.41	0.16	6.5	+
1-7	0.48	0.23	8.5	+
1-8	1.27	1.02	11	+
1-9	0.02	0.23	8.5	-
1-10	0.15	0.10	3	-
1-11	0.41	0.16	6.5	+

ผลรวมค่า T- จากตารางที่ 4.29 เท่ากับ 22.5

จากตารางที่ 4.27 – 4.29 ได้ผลทดสอบดังนี้

1. จากข้อมูลทั้งสองแหล่งที่มา ได้สถิติทดสอบค่า T เท่ากับ 87.5 ซึ่งมากกว่า $T_{ตาราง}$ คือ 83 ทำให้ไม่สามารถปฏิเสธ H_0 ได้ หรือหมายความว่า ความแม่นยำจากวิธีการประมาณขนาดซอฟต์แวร์ด้วยคลาสพอยต์มีความคลาดเคลื่อนน้อยกว่า 25% สำหรับข้อมูลทั้งสองแหล่ง
2. จากข้อมูลแหล่งที่ 1 ได้สถิติทดสอบค่า T เท่ากับ 22.5 ซึ่งมากกว่า $T_{ตาราง}$ คือ 5 ทำให้ไม่สามารถปฏิเสธ H_0 ได้ หรือหมายความว่า ความแม่นยำจากวิธีการประมาณขนาดซอฟต์แวร์ด้วยคลาสพอยต์มีความคลาดเคลื่อนน้อยกว่า 25% สำหรับแหล่งข้อมูลที่หนึ่ง
3. จากข้อมูลแหล่งที่ 2 ได้สถิติทดสอบค่า t เท่ากับ 1.762 ซึ่งน้อยกว่าค่าวิกฤตของ t เมื่อ df=11 คือ 1.796 และ ผลสรุปค่า Sig. (2-tailed) ที่คำนวณได้เท่ากับ 0.003 เนื่องจากตัวอย่างเป็นการทดสอบสมมติฐานแบบทางเดียวทำให้ค่า Sig ที่นำมาเปรียบเทียบจะต้องนำหารด้วย 2 ก่อน ค่า sig ที่ได้จากการทดสอบ คือ 0.106/2 = 0.053 ซึ่งมีค่าน้อยกว่า Sig ที่กำหนดคือ 0.05 ค่าสถิติที่ได้แสดงให้เห็นว่าสามารถยอมรับ H_0 หรือหมายความว่า ความแม่นยำของวิธีการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยคลาสพอยต์มีความคลาดเคลื่อนน้อยกว่า 25% ในข้อมูลแหล่งที่ 2 ที่ระดับนัยสำคัญ 0.05

ดังนั้น วิธีการประมาณขนาดซอฟต์แวร์ด้วยคลาสพอยต์สามารถใช้ประมาณขนาดซอฟต์แวร์โดยมีความคลาดเคลื่อนน้อยกว่า 25% ในทุกแหล่งข้อมูล

4.4 การเปรียบเทียบความแม่นยำระหว่างวิธีการประมาณขนาดซอฟต์แวร์ที่ต่างกัน

ผู้วิจัยตรวจสอบความแตกต่างกันระหว่างความแม่นยำของวิธีการประมาณการ 4 วิธีการด้วย การวิเคราะห์ความแปรปรวน (Analysis of Variance: ANOVA) ด้วยการวิเคราะห์ความแปรปรวนแบบทางเดียว (One-Way ANOVA) โดยมีสมมติฐานการทดสอบ คือ

กำหนดให้ μ_1คือ ค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ในการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์ (MMRE)

μ_2คือ ค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ในการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุ (MMRE)

μ_3คือ ค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ในการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุสอง (MMRE)

μ_4คือ ค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ในการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ด้วยคลาสพอยต์ (MMRE)

$H_0: \mu_1 = \mu_2 = \mu_3 = \mu_4$ (ความคลาดเคลื่อนในการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ ทั้ง 4 วิธีการประมาณขนาดมีค่าเท่ากัน)

$H_1: \mu_i \neq \mu_j$เมื่อ $i \neq j$ (ความคลาดเคลื่อนในการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์ ทั้ง 4 วิธีการประมาณขนาดไม่เท่ากัน)

เนื่องจากเป็นการทดสอบระหว่างค่าเฉลี่ยความคลาดเคลื่อนระหว่างกลุ่มจึงเลือกใช้การวิเคราะห์ความแปรปรวน (Analysis of Variance: ANOVA) ด้วยการวิเคราะห์ความแปรปรวนแบบทางเดียว (One-Way ANOVA) โดยจะปฏิเสธ H_0 ถ้าค่า Sig. (Significance) จากการทดสอบมีค่าน้อยกว่าระดับนัยสำคัญที่กำหนด ในงานวิจัยนี้กำหนดระดับนัยสำคัญที่ 0.05

ในงานวิจัยนี้ผู้วิจัยแยกวิเคราะห์ข้อมูลเป็น 3 ชุด คือ 1) ข้อมูลจากทั้ง 2 แหล่งข้อมูล จำนวน 23 ชุดข้อมูล 2) ข้อมูลเฉพาะแหล่งข้อมูลที่หนึ่ง จำนวน 11 ชุดข้อมูล 3) ข้อมูลเฉพาะแหล่งข้อมูลที่สอง จำนวน 12 ชุดข้อมูล

ผลความแม่นยำของข้อมูลทั้งหมดจากสี่วิธีการประมาณขนาดซอฟต์แวร์ คือ การประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์ ฟังก์ชันพอยต์เชิงวัตถุ ฟังก์ชันพอยต์เชิงวัตถุสอง และคลาสพอยต์ จะพบว่า วิธีการประมาณการที่ดีที่สุด คือ การประมาณการด้วยคลาสพอยต์ซึ่งมีค่าเฉลี่ยความคลาดเคลื่อนอยู่ที่ 0.33 ผลลัพธ์แสดงดังตารางที่ 4.30

ตารางที่ 4.30 แสดงผลสถิติเชิงบรรยายของความคลาดเคลื่อนสัมพัทธ์จากข้อมูลทั้งหมด

	N	Mean	Std. Deviation	Std. Error	Minimum	Maximum
FP	23	.3626	.43920	.09158	.00	1.35
OOFP	23	.9730	.56525	.11786	.00	2.37
OOFP2	23	1.0078	.56621	.11806	.00	2.37
CP	23	.3348	.24735	.05158	.02	1.27
Total	*92*	*.6696*	*.56614*	*.05902*	*.00*	*2.37*

เมื่อนำข้อมูลทั้งหมดมาทดสอบการวิเคราะห์ความแปรปรวนแบบทางเดียว (One-Way ANOVA) เพื่อทดสอบความแตกต่างระหว่างแต่ละวิธีการ แสดงค่าสถิติได้ดังตารางที่ 4.31

ตารางที่ 4.31 ค่าสถิติการทดสอบการวิเคราะห์ความแปรปรวนแบบทางเดียวของความคลาดเคลื่อนจากการประมาณการทั้งสี่วิธีการจากข้อมูลทั้งหมด

Levene Statistic	df1	df2	Sig.
3.100	3	88	.002

จากผลการทดสอบค่า Sig. (Significance) มีค่าเท่ากับ 0.002 ซึ่งน้อยกว่าค่านัยสำคัญที่งานวิจัยนี้กำหนด (0.05) ทำให้สามารถปฏิเสธ H_0 นั่นหมายความว่า มีค่าเฉลี่ยความคลาดเคลื่อนในการประมาณค่าความพยายามอย่างน้อยหนึ่งคู่ไม่เท่ากัน กล่าวคือ วิธีการประมาณขนาดซอฟต์แวร์ที่แตกต่างกันให้ผลความแม่นยำที่แตกต่างกัน

เมื่อผู้วิจัยทราบว่า วิธีประมาณขนาดที่แตกต่างกันจะให้ผลความแม่นยำที่แตกต่างกัน จึงได้ทดสอบสถิติรายคู่ระหว่างแต่ละวิธีการประมาณขนาดซอฟต์แวร์ โดยเลือกใช้สถิติ Tamhane เนื่องจากเป็นการวิเคราะห์ความแปรปรวนในกรณีที่แตกต่างกัน (กัลยา วาณิชย์บัญชา, 2553) ผลการวิเคราะห์รายคู่แสดงได้ดังตารางที่ 4.32

ตารางที่ 4.32 ค่าสถิติการทดสอบการวิเคราะห์ความแปรปรวนจากการประมาณการแต่ละวิธีรายคู่จากข้อมูลทั้งหมด

(I) วิธี	(J) วิธี	Mean Difference (I-J)	Std. Error	Sig.
FP	OOFP	-.61043*	.14926	.001
	OOFP2	-.64522*	.14942	.001
	CP	.02783	.10510	1.000
OOFP	FP	.61043*	.14926	.001
	OOFP2	-.03478	.16683	1.000
	CP	.63826*	.12865	.000
OOFP2	FP	.64522*	.14942	.001
	OOFP	.03478	.16683	1.000
	CP	.67304*	.12884	.000
CP	FP	-.02783	.10510	1.000
	OOFP	-.70826*	.13846	.000
	OOFP2	-.74304*	.13863	.000

ผลการทดสอบความแตกต่างในรายคู่ของแต่ละวิธีการ สามารถสรุปได้ดังนี้

ความแม่นยำที่ได้จากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์ให้ผลความแม่นยำแตกต่างจากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุ ซึ่งให้ค่า Sig อยู่ที่ 0.001 ซึ่งน้อยกว่า 0.05

ความแม่นยำที่ได้จากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์ให้ผลความแม่นยำแตกต่างจากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุสอง ซึ่งให้ค่า Sig อยู่ที่ 0.001 ซึ่งน้อยกว่า 0.05

ความแม่นยำที่ได้จากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์ให้ผลความแม่นยำไม่แตกต่างจากวิธีการประมาณขนาดด้วยคลาสพอยต์ ซึ่งให้ค่า Sig อยู่ที่ 1.000 ซึ่งมากกว่า 0.05

ความแม่นยำที่ได้จากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุให้ผลความแม่นยำไม่แตกต่างจากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุสอง ซึ่งให้ค่า Sig อยู่ที่ 1.000 ซึ่งมากกว่า 0.05

ความแม่นยำที่ได้จากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุให้ผลความแม่นยำแตกต่างจากวิธีการประมาณขนาดด้วยคลาสพอยต์ ซึ่งให้ค่า Sig อยู่ที่ 0.000 ซึ่งน้อยกว่า 0.05

ความแม่นยำที่ได้จากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุสองให้ผลความแม่นยำแตกต่างจากวิธีการประมาณขนาดด้วยคลาสพอยต์ ซึ่งให้ค่า Sig อยู่ที่ 0.000 ซึ่งน้อยกว่า 0.05

จากผลการวิเคราะห์ความแม่นยำในการประมาณขนาดซอฟต์แวร์ในหัวข้อที่ 4.3 ทราบได้ว่า อาจมีความแตกต่างกันเมื่อใช้กับข้อมูลจากคนละแหล่ง ดังนั้นจึงมีการเปรียบเทียบระหว่าง 4 วิธีการในแต่ละแหล่งข้อมูล

4.4.1 ผลการวิเคราะห์ความแตกต่างหว่างวิธีการประมาณขนาดซอฟต์แวร์จากแหล่งข้อมูลที่หนึ่ง

หลังจากผู้วิจัยทราบว่า วิธีการประมาณการซอฟต์แวร์ที่แตกต่างกันให้ผลความแม่นยำที่ต่างกันจากข้อมูลทั้งสองแหล่งข้อมูล จึงนำข้อมูลทั้งหมดพิจารณาแยกแหล่งที่มาของข้อมูลเพื่อวิเคราะห์ จากข้อมูลความคลาดเคลื่อนด้วยวิธีการประมาณการทั้งสี่วิธีจากแหล่งข้อมูลที่หนึ่ง จำนวน 11 ชุด สามารถแสดงสถิติเชิงบรรยายได้ดังตารางที่ 4.33

ตารางที่ 4.33 แสดงผลสถิติเชิงบรรยายของความคลาดเคลื่อนสัมพัทธ์จากแหล่งข้อมูลที่หนึ่ง

	N	Mean	Std. Deviation	Std. Error	Minimum	Maximum
FP	11	.5773	.54778	.16516	.00	1.35
OOFP	11	.9609	.72712	.21924	.00	2.37
OOFP2	11	.9700	.72632	.21899	.00	2.37
CP	11	.3609	.34268	.10332	.02	1.27
Total	*44*	*.7173*	*.64168*	*.09674*	*.00*	*2.37*

ผลลัพธ์จากตารางที่ 4.33 แสดงให้เห็นว่า วิธีการประมาณขนาดซอฟต์แวร์ที่ดีที่สุดสำหรับแหล่งข้อมูลที่หนึ่ง คือ การประมาณขนาดด้วยคลาสพอยต์เช่นเดียวกัน โดยพบว่ามีค่าเฉลี่ยความคลาดเคลื่อนอยู่ที่ 0.36 หรือ ประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์คลาดเคลื่อนไป 36%

ผู้วิจัยจึงวิเคราะห์ความแปรปรวนทางเดียวเพื่อพิสูจน์สมมติฐาน คือ วิธีการทั้งสี่วิธีการให้ผลความแม่นยำที่แตกต่างกันหรือไม่สำหรับข้อมูลแหล่งที่หนึ่ง ผลการทดสอบแสดงค่าสถิติได้ดังตารางที่ 4.34

ตารางที่ 4.34 ค่าสถิติการทดสอบการวิเคราะห์ความแปรปรวนแบบทางเดียวของความคลาดเคลื่อนจากการประมาณการทั้งสี่วิธีการจากแหล่งข้อมูลที่หนึ่ง

Levene Statistic	df1	df2	Sig.
3.583	3	40	.022

จากผลการทดสอบค่า Sig. (Significance) จากการทดสอบมีค่า 0.022 ซึ่งน้อยกว่าค่านัยสำคัญที่งานวิจัยนี้กำหนด (0.05) หมายความว่า สำหรับแหล่งข้อมูลที่หนึ่งวิธีการประมาณขนาดซอฟต์แวร์ที่แตกต่างกันให้ผลความแม่นยำที่แตกต่างกัน

เมื่อผู้วิจัยทราบว่า วิธีประมาณขนาดที่แตกต่างกันจะให้ผลความแม่นยำที่แตกต่างกัน จึงได้ทดสอบสถิติรายคู่ระหว่างแต่ละวิธีการประมาณขนาดซอฟต์แวร์ โดยเลือกใช้สถิติ Tamhane เนื่องจากเป็นการวิเคราะห์ความแปรปรวนในกรณีที่แตกต่างกัน (กัลยา วาณิชย์บัญชา, 2553) แสดงผลได้ดังตารางที่ 4.35

ตารางที่ 4.35 ค่าสถิติการทดสอบการวิเคราะห์ความแปรปรวนจากการประมาณการแต่ละวิธีรายคู่จากแหล่งข้อมูลที่หนึ่ง

(I) วิธี	(J) วิธี	Mean Difference (I-J)	Std. Error	Sig.
FP	OOFP	-.38364	.27449	.693
	OOFP2	-.39273	.27429	.670
	CP	.21636	.19482	.863
OOFP	FP	.38364	.27449	.693
	OOFP2	-.00909	.30988	1.000
	CP	.60000	.24236	.149
OOFP2	FP	.39273	.27429	.670
	OOFP	.00909	.30988	1.000
	CP	.60909	.24214	.138

ตารางที่ 4.35 (ต่อ) ค่าสถิติการทดสอบการวิเคราะห์ความแปรปรวนจากการประมาณการแต่ละวิธีรายคู่จากแหล่งข้อมูลที่หนึ่ง

(I) วิธี	(J) วิธี	Mean Difference (I-J)	Std. Error	Sig.
CP	FP	-.21636	.19482	.863
	OOFP	-.60000	.24236	.149
	OOFP2	-.60909	.24214	.138

ผลการทดสอบความแตกต่างในรายคู่ของแต่ละวิธีการจากแหล่งข้อมูลที่หนึ่ง สามารถสรุปได้ดังนี้

ความแม่นยำที่ได้จากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์ให้ผลความแม่นยำไม่แตกต่างจากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุในแหล่งข้อมูลที่หนึ่ง ซึ่งให้ค่า Sig อยู่ที่ 0.693 ซึ่งมากกว่า 0.05

ความแม่นยำที่ได้จากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์ให้ผลความแม่นยำไม่แตกต่างจากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุสองในแหล่งข้อมูลที่หนึ่ง ซึ่งให้ค่า Sig อยู่ที่ 0.670 ซึ่งมากกว่า 0.05

ความแม่นยำที่ได้จากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์ให้ผลความแม่นยำไม่แตกต่างจากวิธีการประมาณขนาดด้วยคลาสพอยต์ในแหล่งข้อมูลที่หนึ่ง ซึ่งให้ค่า Sig อยู่ที่ 0.863 ซึ่งมากกว่า 0.05

ความแม่นยำที่ได้จากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุให้ผลความแม่นยำไม่แตกต่างจากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุสองในแหล่งข้อมูลที่หนึ่ง ซึ่งให้ค่า Sig อยู่ที่ 1.000 ซึ่งมากกว่า 0.05

ความแม่นยำที่ได้จากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุให้ผลความแม่นยำไม่แตกต่างจากวิธีการประมาณขนาดด้วยคลาสพอยต์ในแหล่งข้อมูลที่หนึ่ง ซึ่งให้ค่า Sig อยู่ที่ 0.149 ซึ่งมากกว่า 0.05

ความแม่นยำที่ได้จากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุสองให้ผลความไม่แม่นยำแตกต่างจากวิธีการประมาณขนาดด้วยคลาสพอยต์ในแหล่งข้อมูลที่หนึ่ง ซึ่งให้ค่า Sig อยู่ที่ 0.139 ซึ่งมากกว่า 0.05

4.4.2 ผลการวิเคราะห์ความแตกต่างหว่างวิธีการประมาณขนาดซอฟต์แวร์จากแหล่งข้อมูลที่สอง

หลังจากผู้วิจัยทราบว่า วิธีการประมาณการซอฟต์แวร์ที่แตกต่างกันให้ผลความแม่นยำไม่ต่างกันจากแหล่งข้อมูลที่หนึ่ง ผู้วิจัยจึงทดสอบวิเคราะห์ข้อมูลเฉพาะแหล่งข้อมูลที่สอง จากข้อมูลความคลาดเคลื่อนด้วยวิธีการประมาณการทั้งสี่วิธีจากแหล่งข้อมูลที่หนึ่งจำนวน 12 ชุด สามารถแสดงสถิติเชิงบรรยายได้ดังตารางที่ 4.36

ตารางที่ 4.36 แสดงผลสถิติเชิงบรรยายของความคลาดเคลื่อนสัมพัทธ์จากแหล่งข้อมูลที่สอง

	N	Mean	Std. Deviation	Std. Error	Minimum	Maximum
FP	12	.1658	.15710	.04535	.01	.55
OOFP	12	.9842	.39762	.11478	.34	1.66
OOFP2	12	1.0425	.39857	.11506	.45	1.76
CP	12	.3108	.11958	.03452	.16	.52
Total	*48*	*.6258*	*.48967*	*.07068*	*.01*	*1.76*

ผลลัพธ์จากตารางที่ 4.36 แสดงให้เห็นว่า วิธีการประมาณการที่ดีที่สุดสำหรับแหล่งข้อมูลที่สอง คือ การประมาณขนาดด้วยฟังก์ชันพอยต์ โดยพบว่ามีค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์อยู่ที่ 0.17 หรือ ประมาณขนาดได้ผิดพลาดไปจากความเป็นจริงเฉลี่ย 17%

ผู้วิจัยจึงวิเคราะห์ความแปรปรวนทางเดียวเพื่อพิสูจน์สมมติฐาน คือ วิธีการทั้งสี่วิธีการให้ผลความแม่นยำที่แตกต่างกันหรือไม่สำหรับข้อมูลแหล่งที่สอง ผลการทดสอบแสดงค่าสถิติได้ดังตารางที่ 4.37

ตารางที่ 4.37 ค่าสถิติการทดสอบการวิเคราะห์ความแปรปรวนแบบทางเดียวของความคลาดเคลื่อนจากการประมาณการทั้งสี่วิธีการจากแหล่งข้อมูลที่สอง

Levene Statistic	df1	df2	Sig.
5.787	3	44	.002

จากผลการทดสอบค่า Sig. (Significance) จากการทดสอบมีค่า 0.002 ซึ่งน้อยกว่าค่านัยสำคัญที่งานวิจัยนี้กำหนด (0.05) หมายความว่า แหล่งข้อมูลที่สองวิธีการประมาณการที่แตกต่างกันให้ผลความแม่นยำที่แตกต่างกัน

ผู้วิจัยจึงสนใจที่จะทดสอบความแตกต่างในรายคู่ระหว่างแต่ละวิธีการด้วยค่า Tamhane (กัลยา วาณิชย์บัญชา, 2553) ผลการทดสอบสามารถแสดงได้ดังตารางที่ 4.38

ตารางที่ 4.38 ค่าสถิติการทดสอบการวิเคราะห์ความแปรปรวนจากการประมาณการแต่ละวิธีรายคู่จากแหล่งข้อมูลที่สอง

(I) วิธี	(J) วิธี	Mean Difference (I-J)	Std. Error	Sig.
FP	OOFP	-.81833*	.12342	.000
	OOFP2	-.87667*	.12367	.000
	CP	-.14500	.05699	.109
OOFP	FP	.81833*	.12342	.000
	OOFP2	-.05833	.16252	1.000
	CP	.67333*	.11986	.001
OOFP2	FP	.87667*	.12367	.000
	OOFP	.05833	.16252	1.000
	CP	.73167*	.12012	.000
CP	FP	.14500	.05699	.109
	OOFP	-.67333*	.11986	.001
	OOFP2	-.73167*	.12012	.000

ผลการทดสอบความแตกต่างในรายคู่ของแต่ละวิธีการประมาณการจากแหล่งข้อมูลที่สอง พบว่า

ความแม่นยำที่ได้จากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์ให้ผลความแม่นยำแตกต่างจากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุในแหล่งข้อมูลที่สอง ซึ่งให้ค่า Sig อยู่ที่ 0.000 ซึ่งน้อยกว่า 0.05

ความแม่นยำที่ได้จากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์ให้ผลความแม่นยำแตกต่างจากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุสองในแหล่งข้อมูลที่สอง ซึ่งให้ค่า Sig อยู่ที่ 0.000 ซึ่งน้อยกว่า 0.05

ความแม่นยำที่ได้จากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์ให้ผลความแม่นยำไม่แตกต่างจากวิธีการประมาณขนาดด้วยคลาสพอยต์ในแหล่งข้อมูลที่สอง ซึ่งให้ค่า Sig อยู่ที่ 0.109 ซึ่งมากกว่า 0.05

ความแม่นยำที่ได้จากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุให้ผลความแม่นยำไม่แตกต่างจากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุสองในแหล่งข้อมูลที่สอง ซึ่งให้ค่า Sig อยู่ที่ 1.000 ซึ่งมากกว่า 0.05

ความแม่นยำที่ได้จากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุให้ผลความแม่นยำแตกต่างจากวิธีการประมาณขนาดด้วยคลาสพอยต์ในแหล่งข้อมูลที่สอง ซึ่งให้ค่า Sig อยู่ที่ 0.001 ซึ่งน้อยกว่า 0.05

ความแม่นยำที่ได้จากวิธีการประมาณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุสองให้ผลความแม่นยำแตกต่างจากวิธีการประมาณขนาดด้วยคลาสพอยต์ในแหล่งข้อมูลที่สอง ซึ่งให้ค่า Sig อยู่ที่ 0.000 ซึ่งน้อยกว่า 0.05

4.5 สรุปผลการเปรียบเทียบความแม่นยำของวิธีการประมาณขนาดซอฟต์แวร์ที่ต่างกัน

ผลสรุปจากข้อมูลสามชุดพบว่า แหล่งข้อมูลที่สอง วิธีการประมาณขนาดซอฟต์แวร์ที่แตกต่างกันให้ผลความแม่นยำที่แตกต่างกัน แต่สำหรับแหล่งข้อมูลที่หนึ่ง วิธีการประมาณขนาดซอฟต์แวร์ที่แตกต่างกันไม่มีผลต่อความแม่นยำ แต่จากสามชุดข้อมูลสามารถสรุปตรงกันได้ว่าวิธีการประมาณขนาดซอฟต์แวร์ด้วยคลาสพอยต์ให้ผลความแม่นยำที่ดีที่สุด

เมื่อผู้วิจัยวิเคราะห์ความแตกต่างความแม่นยำของแต่ละวิธีในรายคู่ ผลสรุปจากข้อมูลทั้งสามชุดไม่สามารถสรุปได้ตรงกัน ซึ่งสามารถแสดงได้ดังตารางที่ 4.39

ตารางที่ 4.39 ผลสรุปความแตกต่างระหว่างความแม่นยำของวิธีการประมาณการในรายคู่

	แหล่งข้อมูลที่หนึ่ง	**แหล่งข้อมูลที่สอง**	**รวมทั้งสองแหล่งข้อมูล**
FP vs OOFP	ไม่แตกต่างกัน	แตกต่างกัน	แตกต่างกัน
FP vs OOFP2	ไม่แตกต่างกัน	แตกต่างกัน	แตกต่างกัน
FP vs CP	ไม่แตกต่างกัน	ไม่แตกต่างกัน	ไม่แตกต่างกัน
OOFP vs OOFP2	ไม่แตกต่างกัน	ไม่แตกต่างกัน	ไม่แตกต่างกัน
OOFP vs CP	ไม่แตกต่างกัน	แตกต่างกัน	แตกต่างกัน
OOFP2 vs CP	ไม่แตกต่างกัน	แตกต่างกัน	แตกต่างกัน

ผลสรุปที่ตรงกันและสามารถสรุปได้ชัดเจน คือ วิธีการประมาณการซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุทั้งสองวิธีให้ผลความแม่นยำที่ไม่แตกต่างกัน ดังเช่น Zivkovic และคณะ (2005) กล่าวไว้ว่าจะให้ความแม่นยำที่แม่นยำกว่าวิธีการเดิมของ Antoniol และคณะ (1998)

อีกทั้งในงานวิจัยนี้ยังพบว่า วิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์และคลาสพอยต์ให้ผลความแม่นยำที่ไม่แตกต่างกัน โดยพบว่า การประมาณขนาดซอฟต์แวร์ทั้งสองวิธีให้ค่าเฉลี่ยความคลาดเคลื่อนน้อยกว่า 25%

4.6 ผลสรุปการวิเคราะห์ข้อมูล

ผลการวิเคราะห์ข้อมูลจากข้อมูลที่จัดเก็บได้ทั้งสอง พบว่า การประมาณขนาดซอฟต์แวร์ด้วยวิธีที่แตกต่างกัน 4 วิธีการ คือ การประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์ การประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุ การประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุสอง และการประมาณขนาดซอฟต์แวร์ด้วยคลาสพอยต์ ให้ความแม่นยำที่แตกต่างกัน โดยเมื่อวิเคราะห์ในรายคู่พบว่า วิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์กับการประมาณขนาดซอฟต์แวร์ด้วยคลาสพอยต์ให้ความแม่นยำที่ไม่แตกต่างกัน และ วิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุกับการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุสองให้ความแม่นยำที่ไม่แตกต่างกัน

ผลการวิเคราะห์ข้อมูลในแง่ความแม่นยำ พบว่า วิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์กับการประมาณขนาดซอฟต์แวร์ด้วยคลาสพอยต์มีค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์น้อยกว่า 25% แต่สำหรับวิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุกับการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุสองให้ความแม่นยำโดยมีค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์มากกว่า 25%

บทที่ 5
สรุปผลการวิจัยและข้อเสนอแนะ

5.1 สรุปผลการวิจัย

วัตถุประสงค์ของงานวิจัยนี้ คือ การเปรียบเทียบความแม่นยำของการประมาณขนาดซอฟต์แวร์จากสี่วิธีการ คือ การประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์ (Function point) (IFPUG, 1999) การประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุ (Object-oriented function point) (Antoniol et al., 1998) การประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุสอง (Object-oriented function point 2) (Zivkovic et al., 2005) และการประมาณขนาดซอฟต์แวร์ด้วยคลาสพอยต์ (Class point) (Costagliola et al., 2005) ในงานวิจัยนี้สามารถตอบวัตถุประสงค์ได้ว่า วิธีการประมาณขนาดซอฟต์แวร์ทั้งสี่วิธีให้ผลความแม่นยำที่แตกต่างกัน

ผลสรุปยังสามารถชี้ให้เห็นว่า วิธีการประมาณขนาดซอฟต์แวร์ด้วยคลาสพอยต์ให้ผลความแม่นยำที่ใกล้เคียงกับฟังก์ชันพอยต์อย่างมีนัยสำคัญ โดยพบว่า วิธีการประมาณขนาดซอฟต์แวร์สองวิธีดังกล่าวสามารถใช้ประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์โดยมีค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์มากกว่า 25%

ผลสรุปเกี่ยวกับวิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์ยังชี้ให้เห็นว่าสามารถใช้ประมาณขนาดซอฟต์แวร์เชิงวัตถุโดยให้ค่าความคลาดเคลื่อนน้อยกว่า 25% ซึ่งตรงกับงานวิจัยของ Fetcke และคณะ ในปี ค.ศ. 1998 แต่ขัดแย้งกับงานวิจัยของ Kermere และ Porter ในปี ค.ศ. 1992 ที่ไม่สนับสนุนให้นำฟังก์ชันพอยต์มาประมาณขนาดซอฟต์แวร์เชิงวัตถุ

วิธีการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์เชิงวัตถุและฟังก์ชันพอยต์เชิงวัตถุสองให้ผลความแม่นยำที่ใกล้เคียงกัน ซึ่งพบว่าการประมาณขนาดซอฟต์แวร์ทั้งสองวิธีการนี้ไม่มีความแม่นยำสำหรับข้อมูลที่ใช้ในงานวิจัยชิ้นนี้ โดยพบว่า ผลการประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์มีค่าเฉลี่ยความคลาดเคลื่อนสัมพัทธ์ประมาณหนึ่งเท่า

สำหรับความรู้เบื้องต้นที่ต้องใช้ในการนับ ผู้วิจัยเห็นว่าวิธีการที่สามารถเรียนรู้ได้ง่ายที่สุด คือ วิธีการนับด้วยฟังก์ชันพอยต์เชิงวัตถุ เนื่องจากผู้นับมีความรู้เพียงแต่หลักการทำงานของคลาส หรือหลักการของซอฟต์แวร์เชิงวัตถุก็สามารถดำเนินการนับได้ รองลงมา คือ คลาสพอยต์ และฟังก์ชันพอยต์ ทั้งนี้ขึ้นอยู่กับความรู้พื้นฐานของผู้นับอีกด้วย

5.2 **การนำงานวิจัยไปใช้** (Contribution)

งานวิจัยนี้สามารถนำไปใช้ทั้งในเชิงทฤษฎี และเชิงประยุกต์ได้ ดังต่อไปนี้

5.2.1 **การนำงานวิจัยไปใช้ในเชิงทฤษฎี** (Theoretical Contribution)

ข้อค้นพบในการศึกษานี้ ช่วยต่อยอดองค์ความรู้ในการประมาณขนาดซอฟต์แวร์ ดังรายละเอียดต่อไปนี้

1. งานวิจัยนี้เป็นการศึกษาความรู้เกี่ยวกับการประมาณขนาดซอฟต์แวร์ใหม่ โดยนำวิธีการประมาณขนาดซอฟต์แวร์ด้วยคลาสพอยต์มาเปรียบเทียบกับฟังก์ชันพอยต์ซึ่งยังไม่มีงานวิจัยชิ้นใดเคยทดสอบมาก่อน ผลการทดสอบพบว่า ผลการประมาณขนาดซอฟต์แวร์ทั้งสองวิธีการดังกล่าวไม่มีความแตกต่างกัน
2. งานวิจัยนี้เป็นต่อยอดองค์ความรู้เกี่ยวกับการใช้การประมาณขนาดด้วยฟังก์ชันพอยต์เพื่อใช้ประมาณค่าความพยายามในการพัฒนาซอฟต์แวร์เชิงวัตถุ ซึ่งพบว่าการประมาณขนาดซอฟต์แวร์ด้วยฟังก์ชันพอยต์สามารถใช้ประมาณขนาดซอฟต์แวร์เชิงวัตถุได้ ซึ่งผลสรุปในข้อนี้ตรงกับงานวิจัยของ Fetcke และคณะ (1998)

5.2.2 **การนำงานวิจัยไปใช้ในเชิงประยุกต์** (Practical Contribution)

ผู้บริหารโครงการซอฟต์แวร์ควรทำความเข้าใจเกี่ยวกับองค์กรของตนเองเพื่อเลือกวิธีการประมาณการให้เหมาะสม ทั้งนี้ขึ้นอยู่กับความเชี่ยวชาญของผู้ออกแบบระบบ ในกรณีที่สามารถออกแบบระบบด้วยแผนภาพคลาส (Class diagram) ได้ดี ควรเลือกใช้วิธีการประมาณการด้วยคลาสพอยต์ (Class point) ในกรณีที่ผู้ออกแบบสามารถเขียนความต้องการของระบบได้ละเอียดควรเลือกใช้ฟังก์ชันพอยต์ (Function point) ซึ่งทั้งสองวิธีการสามารถให้ผลความแม่นยำที่ใกล้เคียงกัน โดยมีความคลาดเคลื่อนน้อยกว่า 25%

5.3 **ข้อจำกัดของงานวิจัยและข้อเสนอแนะ**

1. การศึกษาในงานวิจัยนี้เป็นการเก็บข้อมูลเพียง 2 บริษัท โดยเก็บจากบริษัทที่ 1 จำนวน 11 ระบบย่อย และเก็บจากบริษัทที่ 2 จำนวน 12 ระบบย่อย แม้ผู้วิจัยจะพยายามเก็บข้อมูลเพื่อให้สามารถเป็นตัวแทนข้อมูลที่ดีในธุรกิจจริง แต่เนื่องมาจากการเก็บข้อมูลกับทางบริษัทจริงจำนวนมากทำได้ยาก อันเนื่องมาจากข้อมูลเป็น

ความลับอีกทั้งการหาการออกแบบที่ครบถ้วนสมบูรณ์ในระบบที่พัฒนาเสร็จสิ้นไปแล้วหาได้ยาก

2. การเก็บข้อมูลในงานวิจัยนี้มองว่า ประสิทธิภาพการทำงานของผู้พัฒนาระบบมีความสามารถเท่าเทียมกัน
3. วิธีการประมาณขนาดซอฟต์แวร์ในงานวิจัยนี้ ใช้คำนวณกับซอฟต์แวร์ที่พัฒนาขึ้นมาใหม่เท่านั้น
4. ขนาดของแต่ละวิธีการประมาณขนาดซอฟต์แวร์ คำนวณจากเอกสารการออกแบบจากระบบที่พัฒนาเสร็จสมบูรณ์แล้ว ทำให้ระบบที่ใช้คำนวณขนาดมีรายละเอียดและฟังก์ชันการทำงานของระบบครบถ้วนสมบูรณ์มากกว่าในการออกแบบระบบในช่วงแรกๆ
5. แผนภาพคลาสที่ใช้ในการคำนวณขนาดด้วยฟังก์ชันพอยต์เชิงวัตถุ ฟังก์ชันพอยต์เชิงวัตถุสอง และคลาสพอยต์ ได้จากการทำวิศวกรรมย้อนกลับจากซอร์สโค้ดที่พัฒนาเสร็จสมบูรณ์แล้วด้วยเครื่องมือวิชวล พาราไดม์ สำหรับ ยูเอ็มแอล เนื่องจากบริษัทที่เก็บข้อมูลทั้งสองบริษัทไม่มีการออกแบบด้วยแผนภาพคลาส ข้อมูลที่ได้จากแผนภาพคลาสจึงมีรายละเอียดที่ครบถ้วนสมบูรณ์มากกว่าการออกแบบในช่วงแรกๆ
6. สูตรที่ใช้ในการคำนวณค่าความพยายามที่ใช้ในการพัฒนาซอฟต์แวร์ ผู้วิจัยคัดเลือกสมการจากงานวิจัยในอดีตเพื่อนำมาใช้คำนวณ ซึ่งอาจเป็นไปได้ว่า สมการเหล่านั้นอาจจะไม่เหมาะสำหรับการคำนวณข้อมูลในชุดนี้ เนื่องจากเป็นสมการที่ได้มาจากงานวิจัยในต่างประเทศ
7. ผู้วิจัยไม่ได้ทดสอบเปรียบเทียบกับค่าความพยายามในการพัฒนาซอฟต์แวร์ที่แต่ละบริษัทประเมินขึ้นมา ซึ่งทั้งสองบริษัทใช้วิธีการประมาณด้วยผู้เชี่ยวชาญ ทั้งนี้การประมาณด้วยผู้เชี่ยวชาญของแต่บริษัทอาจมีความแม่นยำหรือไม่แม่นยำกว่าวิธีการประมาณด้วยวิธีการคำนวณ ซึ่งเป็นสิ่งที่น่าศึกษาและค้นคว้าต่อไป

รายการอ้างอิง

ภาษาไทย

กัลยา วานิชย์บัญชา. 2551. หลักสถิติ. พิมพ์ครั้งที่ 8. กรุงเทพฯ: โรงพิมพ์แห่งจุฬาลงกรณ์มหาวิทยาลัย.

วรัฏฐา เจริญสถาพงษ์. 2549. วิธีการประเมินต้นทุนสำหรับการพัฒนาเว็บแอพพลิเคชัน. วิทยานิพนธ์ปริญญามหาบัณฑิต จุฬาลงกรณ์มหาวิทยาลัย.

กานดา รุณนะพงศา. เครื่องมือที่ใช้ในการออกแบบ UML (UML Tool). [ออนไลน์]. 2556, แหล่งที่มา: http://www.gotoknow.org [10 เมษายน2556]

ภาษาอังกฤษ

Abran, J. -M. Desharnais, S. Oligny, D. St -Pierre, and C. Symons. 2001. COSMIC-FFP Measurement Manual, Version 2.1, The Common Software Measurement International Consortium.

Abrahao, S., Poels, G., Pastor, O. 2004. Functional Size Measurement Method for Object Oriented Conceptual Schemas: Design and Evaluation Issues. Working Paper Faculty of Economics and Business Administration, Ghent University

Adekile, O., Simmons, Dick B. and Lively, William M. 2010. Object-Oriented Software Development Effort Prediction Using Design Patterns from Object Interaction Analysis. 2010 Fifth International Conference on Systems, pp. 47-53.

Agarwal, R. and Sinha, A.P. 2003. Object-oriented modeling with uml: a study of developers' perceptions. Commun. ACM 46, 248-256

Albrecht, AJ. Gaffney, JE. 1983. Software function, source lines of code, and development effort prediction: a software science validation. IEEE Transactions on Software Engineering SE-9(6): 639–648.

Antoniol, G., Caldiera, G., Fiutem, R. and Lokan, C. 1998. A definition and experimental evaluation of function points for object-oriented systems. In Proc. of the Fifth International Symposium on Software Metrics - METRICS98. pp. 167-178, 2-5 Nov. 1998.

Antoniol, G., Fiutem, R. and Lokan, C. 2003. Object-Oriented Function Points: An Empirical Validation. Empirical Software Engineering, 8, 225–254.

Behrens, C. 1983. Measuring the Productivity of Computer Systems Development Activities with Function Points. IEEE Transactions on Software Engineering. June pp. 648-652

Bianco, D. V. and Lavazza, L. 2005. An Empirical Assessment of Function Point-Like Object-Oriented Metrics. METRICS 2005, 11th International Software Metrics Symposium, Como, 19-22 September 2005.

Boehm, B. 2000. COCOMO II Model Definition Manual. University of Southern California, Version 1.4.

Boehm, B. 2000. Software development cost estimation approaches – A survey. Annals of Software Engineering 10, 1-4: 177-205.

Boehm, B.W., Clark, B. and Hiriwitz, E. 1995. Cost Models for Future Life Cycle Processes: COCOMO 2.0. Ann. Software Eng 1, 1: 1-24.

Carleton, A, et al.. 1992 Software Measurement for DoD Systems: Recommendations for Initial Core Measures, Technical Report CMU/SEI-92-019, ESC-TR-92-019, Software Eng. Inst., Carnegie Mellon University, Pittsburgh.

Chidamber, S.R. and Kemerer, C.F. 1994. A Metrics Suite for Object-Oriented Design. IEEE Trans. Software Eng 20, 6: 476-493

Coad, P. and Nicola, J. 1993. Object-Oriented Programming. Prentice Hall.

Conte. S., Dunsmore, H.. and Shen, V. 1986. Software Engineering Metrics and Models. Menlo Park, Benjamin/Cummings.

Costagliola, G. and Tortora, G. 2005. Class points: an approach for the size estimation of object-oriented systems. IEEE Transactions on Software Engineering 31 : 52-74.

Fetcke. T. , Abran, A. and Nguyen, T. 1997. Mapping the OO-Jacobson Approach into Function Point Analysis. Proceedings of TOOLS-23'97, 28 July – 1 August 1997

Henderson-Sellers, B. 1996. Object-Oriented Metrics: Measures of Complexity. Prentice-Hall.

Hericko, M. and Zivkovic, A. 2008. The size and effort estimates in iterative development. Information and Software Technology 50, 7-8: 772–781.

IFPUG. 1999. Function Point Counting Practices Manual Release 4.1. International Function Point Users Group (IFPUG).

ISO/IEC 20926: 2003. Software Engineering IFPUG 4.1 Unadjusted Functional Size Measurement Method - Counting Practices Manual, International Organization for Standardization, Geneva, Switzerland.

Jeffery, R., Ruhe, M., and Wieczorek, I. 2001. Using Public Domain Metrics to Estimate Software Development Effort. Proc. Int'l Software Metrics Symp (Metrics '01), :16-27.

Jorgensen, M. and Ostvold, K.M. 2004. Reasons for Software Effort Estimation Error: Impact of Respondent Role, Information Collection Approach, and Data Analysis Method. IEEE Transactions on software engineering 30, Dec. 2004.

Kanmani, S., Kathiravan, J., Senthil Kumar, S. and Shanmugam, M. 2007. Neural Network Based Effort Estimation Using Class Points for OO Systems. International Conference on Computing: Theory and Applications (iccta'07), pp. 261-266

Karner, G. 1993. Resource Estimation for Objectory Projects. Objective Systems.

Kemerer, C.F. and Porter, B. 1992. Improving the reliability of function point measurement: an empirical study. IEEE Trans. Soft. Eng 18. 10: 1011-1024.

Lokan, C.J. 2000. An Empirical Analysis of Function Point Adjustment Factors. Information and Software Technology 42, 1: 649-659.

Linberg, K. R. 1999. Software developer perceptions about software project failure: A case study. Journal of Systems and Software 49.

Maxwell, K. and Froselius, P. 2000. Benchmarking Software Development Productivity. IEEE Software (January-February, 2000): 80-88.

Minkiewicz, A.F. 1997. Measuring Object Oriented Software with Predictive Object Points. Proc. Conf. Applications in Software Measurements (ASM'97).

Misic, V.B. and Tesic, D.N. 1999. Estimation of Effort and Complexity: an Object-Oriented Case Study. J. Systems and Software 41 :133-143.

Moser, S., Henderson-Sellers, B. and Misic V.B. 1997. Measuring Object-Oriented Business Models. Proc. TOOL Pacific'97 Conf.

Moser, S., Henderson-Sellers, B. and Misic V.B. 1999. Cost Estimation Based on Business Models. J. Systems and Software, Vol. 49, pp. 33-42.

Moser, S. and Nierstrasz, O. 1996.The Effect of Object-Oriented Frameworks on Developer Productivity, IEEE Computer, 29(9):45–51,

Nesi, P. and Querci, T. 1998. Effort Estimation and Prediction of Object-Oriented Systems. J. Systems and Software, Vol. 42, pp. 89-102.

Peixoto, C.E.L., Audy, J.L.N. and Prikladnicki, R. 2010. Effort Estimation in Global Software Development Projects: Preliminary Results from a Survey. 2010 5th IEEE International Conference on Global Software Engineering, pp. 123-127.

Peixoto, C.E.L., Audy, J.L.N. and Prikladnicki, R. 2010. The Importance of the Use of an Estimation Process. Proceedings of the 2010 ICSE Workshop on Software Development Governance, New York, NY, USA.

Poels, G. 2003. Definition and Validation of a COSMIC-FFP Functional Size Measure for Object-Oriented Systems. In 7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2003), Darmstadt, Germany.

Ramsin, R. and Paige, R.F. 2008. Process-centered review of object oriented software development methodologies. ACM Comput. Surv. 40, 1, Article 3 (February 2008), 89 pages.

Rubenstein, D. 2007. Standish group report: There's less development chaos today. SD Times, [Online] . Available from: http://www.sdtimes.com/article/story-20070301-01.html. [12 August, 2011]

Standish Group International. 1994. CHAOS: Project failure and success report. Dennis, MA.

Standish Group. 2004. 2004 Third Quarter Research Report: Standish Group International, Inc.

Tan, H. B. K., Zhao, Y. and Zhang, H. 2009. Conceptual data model-based software size estimation for information systems. ACM Trans. Soft. Eng. Methodol.

Zhou, W. and Liu, Q. 2010. Extended Class Point Approach of Size Estimation for OO Product. Computer Engineering and Technology (ICCET), 2010 2nd International Conference on, Apr. 2010.

Yang, D., Wang, Q., Li, M., Yang, Y., Ye, K. and Du, J. 2008. A survey on software cost estimation in the chinese software industry. ESEM 2008, 253-262.

Zivkovic, A., Rozman, I. and Hericko, M. 2005. Automated software size estimation based on function points using UML models. Information and Software Technology 47, 881–890.

ภาคผนวก

ภาคผนวก
เอกสารเพื่อเก็บข้อมูลค่าความซับซ้อนของระบบ

เนื่องจากข้อมูลที่จะต้องใช้ในการทดสอบการประมาณขนาดของระบบบจะต้องใช้ค่าความซับซ้อนของระบบ จึงขอความกรุณาช่วยกรอกเอกสารความซับซ้อนของระบบ เพื่อนำไปพิจารณาปรับค่าขนาดซอฟต์แวร์ในแต่ละระบบ

ระบบ _____

โปรดระบุระดับความมีอิทธิพลของระบบ โดยอัตรากำหนดไว้โดยที่ค่าแค่ละค่าจะถูกประเมินด้วยระดับการมีอิทธิพลต่อระบบ โดยมีระดับความมีอิทธิพลดังนี้

0 หมายถึง ไม่มีอิทธิพลต่อระบบ

1 หมายถึง มีอิทธิพลต่อระบบเพียงเล็กน้อย

2 หมายถึง มีอิทธิพลต่อระบบน้อย

3 หมายถึง มีอิทธิพลต่อระบบในปานกลาง

4 หมายถึง มีอิทธิพลต่อระบบค่อนข้างมากและเห็นได้เด่นชัด

5 หมายถึง มีอิทธิพลต่อระบบอย่างมาก

ลักษณะของระบบ	ระดับความมีอิทธิพล					
	0	1	2	3	4	5
1. Data Communications						
2. Distributed Data Processing						
3. Performance						
4. Heavily Used Configuration						
5. Transaction Rate						
6. Online Data Entry						
7. End-User Efficiency						
8. Online Update						
9. Complex Processing						
10. Reusability						
11. Installation Ease						
12. Operational Ease						
13. Multiple Sites						
14. Facilitate Change						
15. User Adaptivity						
16. Rapid Prototyping						
17. Multiple Interfaces						
18. Multiuser Interactivity						

โดยมีรายละเอียดในการให้ค่าความมีอิทธิพลดังนี้

1.Data Communication จะมองเรื่องของจำนวนของสิ่งที่ช่วยอำนวยความสะดวกในการถ่ายโอนข้อมูลหรือแลกเปลี่ยนข้อมูลระหว่างซอฟต์แวร์หรือระบบ	
0	ซอฟต์แวร์ทั้งหมดรันแบบ Batch หรือ standalone PC
1	ซอฟต์แวร์เป็นแบบ Batch แต่มีการใช้ลักษณะของ Remote Data Entry หรือ Remote Printing
2	ซอฟต์แวร์เป็นแบบ Batch แต่มีการใช้ลักษณะของ Remote Data Entry และ Remote Printing
3	ซอฟต์แวร์เป็นแบบ Online Data Collection หรือ Teleprocessing จากส่วน Front-end สู่ Batch
4	ซอฟต์แวร์มีลักษณะของ Teleprocessing ซึ่งมากกว่าในส่วนของ Front-end แต่สนับสนุนการติดต่อ Teleprocessing Protocol เพียงชนิดเดียว
5	ซอฟต์แวร์มีลักษณะของ Teleprocessing ซึ่งมากกว่าในส่วนของ Front-end แต่สนับสนุนการติดต่อ Teleprocessing Protocol ที่หลากหลายชนิดมากขึ้น

2. Distributed Data Processing จะมองในเรื่องการจัดการข้อมูลในเชิง Distributed และขั้นตอนหรือหน้าที่ในการจัดการ	
0	ซอฟต์แวร์สามารถช่วยในการถ่ายข้อมูลหรือช่วยในการดำเนินการระหว่างกันได้
1	ซอฟต์แวร์สามารถเตรียมข้อมูลสำหรับ End User เพื่อดำเนินขั้นตอนต่อไปในส่วนอื่นๆ เช่น PC Spreadsheets และ PC DBMS
2	ซอฟต์แวร์สามารถเตรียมข้อมูลเพื่อถ่ายโอนและถ่ายโอนไปยังขั้นตอนอื่นๆ ในระบบเพื่อประมวลผลได้ (ไม่เพียงแต่ถ่ายโอนสำหรับ End User)
3	ซอฟต์แวร์มีลักษณะของ Distributed processing และการถ่ายโอนข้อมูลเป็นแบบออนไลน์แต่เป็นในทิศทางเดียว
4	ซอฟต์แวร์มีลักษณะของ Distributed processing และการถ่ายโอนข้อมูลเป็นแบบออนไลน์แต่เป็นใน 2 ทิศทาง
5	ซอฟต์แวร์มีการจัดการการถ่ายโอนข้อมูลแบบไดนามิกขึ้นอยู่กับแต่ละส่วนของระบบ

3. Performance จะมองในเรื่องของ Response Time ประสิทธิภาพของซอฟต์แวร์นั้นจะได้รับการประเมินโดย User ซึ่งเป็นผลสืบเนื่องจากการออกแบบพัฒนาการติดตั้งและรองรับซอฟต์แวร์นั้นๆ	
0	ไม่มีสภาพของประสิทธิภาพที่ User ต้องการเป็นพิเศษ
1	ประสิทธิภาพและการออกแบบตามความต้องการของ User ได้รับการทบทวนแต่ไม่มีความต้องการกระทำอะไรเป็นพิเศษ
2	Response Time และผลลัพธ์ของการทำงานเป็นเรื่องสำคัญและจำเป็นระหว่างช่วงเวลาที่มีการใช้งานสูงๆ การประมวลผลต้องให้เรียบร้อยก่อนวัดถัดไป
3	Response Time และผลลัพธ์ของการทำงานเป็นเรื่องที่สำคัญและจำเป็นระหว่างช่วงเวลาที่มีการใช้งานในธุรกิจ ไม่มีการออกแบบสำหรับ CPU ที่ต้องการใช้งานเฉพาะที่มีการประมวลผลที่มีความต้องการเฉพาะ รวมทั้งการออกแบบอินเตอร์เฟสมีการระบุเฉพาะ
4	เพิ่มในส่วนของความต้องการของ User ในเรื่องของประสิทธิภาพตั้งแต่ขั้นตอนของการวิเคราะห์ระบบงาน
5	เพิ่มในส่วนของการนำเครื่องมือในการวิเคราะห์ประสิทธิภาพเข้ามาใช้ในขั้นตอนการออกแบบพัฒนา และนำไปใช้เพื่อที่จะให้ตรงกับประสิทธิภาพที่ User มี

4. Heavily Used Configuration จะมองในเรื่องของการใช้ที่เกี่ยวกับฮาร์ดแวร์ และ Platform ซึ่งเป็นคุณลักษณะหนึ่งของซอฟต์แวร์ เช่น User ต้องการรันซอฟต์แวร์บนอุปกรณ์หรือเครื่องที่มีการใช้งานอย่างหนัก	
0	ไม่มีความชัดเจนในเรื่องของความเข้มงวดในความสามารถในการจัดการให้สามารถใช้งานได้
1	มีความเข้มงวดในการจัดการให้เพียบแค่ใช้งานได้แต่ความเข้มงวดก็ยังน้อยในซอฟต์แวร์ซึ่งเป็นตัวอย่าง
2	มีระบบรักษาความปลอดภัยหรือการคำนึงถึงข้อจำกัดด้านเวลารวมอยู่ด้วย
3	มีการเจาะจงถึง Processor (เช่น Centrino P4) สำหรับซอฟต์แวร์
4	มีการระบุถึงความเข้มงวดอย่างยิ่งในการประมวลผลซอฟต์แวร์ต่อหน่วยประมวลผลกลาง (Central Processor) ซึ่งมีลักษณะที่เฉพาะเจาะจง

5	มีข้อจำกัดพิเศษบนซอฟต์แวร์ในส่วนประกอบแบบ Distributed ของระบบ

5. Transaction Rate จะมองในเรื่องความถี่ในการประมวลผล Transaction ว่าเป็นแบบรายวัน รายสัปดาห์ รายเดือน	
0	ไม่มีการคาดถึงช่วงที่มีระดับสูงสุดของ Transaction
1	ช่วงที่ระดับของจำนวน Transaction เพิ่มสูงสุดนั้นได้ถูกคาดคะเน (รายเดือน รายปักษ์ รายปี)
2	มีการคาดคะเนว่าจำนวน Transaction สูงสุดเป็นรายสัปดาห์
3	มีการคาดคะเนว่าจำนวน Transaction สูงสุดเป็นรายวัน
4	มีอัตราของ Transaction สูงมากซึ่งได้รับการบอกกล่าวจาก User ในซอฟต์แวร์นั้นๆ Requirement ต้องการมากเพียงพอเพื่อที่จะมีการวิเคราะห์ประสิทธิภาพของงานในขั้นตอนการออกแบบ
5	มีอัตราของ Transaction สูงมาก ซึ่งได้รับการบอกกล่าวจาก User ในซอฟต์แวร์นั้นๆ Requirement ต้องมากเพียงพอเพื่อที่จะมีการวิเคราะห์ประสิทธิภาพของงานในขั้นตอนการออกแบบ รวมไปถึงขั้นตอนของการพัฒนาและการติดตั้ง

6. On-Line Data Entry จะมองในเรื่องของเปอร์เซ็นต์ของข้อมูลที่ผ่านมาทางออนไลน์	
0	ทุกๆ Transaction จะเป็นรูปแบบของ Batch
1	1% ถึง 7% ของ Transaction เป็น Interactive Data Entry
2	8% ถึง 15% ของ Transaction เป็น Interactive Data Entry
3	16% ถึง 23% ของ Transaction เป็น Interactive Data Entry
4	24% ถึง 30% ของ Transaction เป็น Interactive Data Entry
5	> 30% ของ Transaction เป็น Interactive Data Entry

7. End-User Efficiency จะมองในเรื่องของผู้ใช้งานระบบว่าสามารถใช้งานระบบได้อย่างมีประสิทธิภาพหรือไม่ ซึ่งลักษณะของปัจจัยที่ส่งผลต่อ User Friendly การให้ความสำคัญกับประสิทธิภาพประกอบด้วย • Navigation Aids • Menu • Online Help and Document • Automated Cursor Movement • Scrolling • Remote Printing (Via Online Transaction) • Reassigned Function Keys • Batched Job Submitted From Online Transactions • Cursor Selection Of Screen Data • Heavy Use Of Reverse Video Highlighting Color Underling And Other Indicators • Hard Copy User Documentation Of Online Transactions • Mouse Interface • Pop Up Windows • As Few Screen As Possible To Accomplish A Business Function • Bilingual Support • Multilingual Support	
0	ไม่มีการให้ความสำคัญกับประสิทธิภาพตามข้อกำหนดข้างต้น
1	มีการให้ความสำคัญกับประสิทธิภาพ 1-3 ข้อจากข้อกำหนดข้างต้น
2	มีการให้ความสำคัญกับประสิทธิภาพ 4-5 ข้อจากข้อกำหนดข้างต้น
3	มากกว่า 6 ข้อจากข้อกำหนดข้างต้นโดย User ไม่ได้มีความต้องการที่เฉพาะเจาะจงในเรื่องของ Efficiency
4	มากกว่า 6 ข้อจากข้อกำหนดข้างต้นโดย User มีการใช้ความสนใจกับเรื่องของ Efficiency โดยต้องมีการออกแบบซอฟต์แวร์อย่างเฉพาะเพื่อสนับสนุน Efficiency
5	มากกว่า 6 ข้อจากข้อกำหนดข้างต้นโดย User มีการใช้ความสนใจกับเรื่องของ Efficiency โดย

	ต้องมีการใช้งานอุปกรณ์และขั้นตอนที่พิเศษซึ่งจะทำให้การทำงานนั้นบรรลุตามวัตถุประสงค์

8. **On-Line Update** จะมองในเรื่องของ Internal Logical file ว่ามีจำนวนมากน้อยเพียงใด ที่ได้รับการอัพเดทผ่าน Online Transaction	
0	ไม่มีการอัพเดทออนไลน์
1	การอัพเดทเป็นการอัพเดท 1-3 ไฟล์โดยการอัพเดทนั้นเป็นไปได้ช้า
2	การอัพเดทเป็นการอัพเดท 4 ไฟล์ขึ้นไป โดยการอัพเดทนั้นเป็นไปได้ช้า
3	สามารถออนไลน์อัพเดทได้ในส่วนหลักๆของ Internal Logical files
4	เพิ่มเติมการป้องกันข้อมูลสูญหายในขณะทำ Online Update ซึ่งได้มีการออกแบบและสร้างอย่างพิเศษในซอฟต์แวร์
5	เพิ่มเติมการป้องกันข้อมูลสูญหายของข้อมูลรวมทั้งมีการจัดการระบบกู้ข้อมูลอัตโนมัติ

9. **Complex Processing** เป็นส่วนของคุณลักษณะของซอฟต์แวร์ โดยส่วนประกอบมีดังนี้ - มีส่วนขยายของ Logical Processing - มีส่วนขยายของ Mathematical Processing - มีการประมวลผลที่ซับซ้อนเพื่อจัดการกับ Input และ Output เช่นการมี Multimedia - ข้อผิดพลาดที่เกิดจากการประมวลผลก่อให้เกิด Transaction ที่ไม่สมบูรณ์นั้นต้องทำการประมวลผลอีกครั้ง - Sensitive Control เช่นความปลอดภัยในการประมวลผลข้อมูล	
0	ไม่มีคุณลักษณะของซอฟต์แวร์ตามลักษณะดังที่กล่าวได้ด้านบน
1	มีคุณลักษณะของซอฟต์แวร์ตามลักษณะดังที่กล่าวได้ด้านบน 1 ข้อ
2	มีคุณลักษณะของซอฟต์แวร์ตามลักษณะดังที่กล่าวได้ด้านบน 2 ข้อ
3	มีคุณลักษณะของซอฟต์แวร์ตามลักษณะดังที่กล่าวได้ด้านบน 3 ข้อ
4	มีคุณลักษณะของซอฟต์แวร์ตามลักษณะดังที่กล่าวได้ด้านบน 4 ข้อ
5	มีคุณลักษณะของซอฟต์แวร์ตามลักษณะดังที่กล่าวได้ด้านบนครบ 5 ข้อ

10. Reusability จะมองในเรื่องของซอฟต์แวร์ที่ได้ทำการพัฒนานั้นสามารถนำไปปรับใช้หรือสามารถนำไปพัฒนาต่อไปในโครงการอื่นๆ ได้หรือไม่	
0	ไม่มีการ Reuse โค้ด
1	การ Reuse มีเพียงภายในซอฟต์แวร์
2	มีการนำโค้ดไป Reuse น้อยกว่า 10% ในหลายๆ งาน
3	มีการนำโค้ดไป Reuse มากกว่า 10% ในหลายๆ งาน
4	ทั้งซอฟต์แวร์และเอกสารประกอบการพัฒนาค่อนข้างเอื้อต่อการนำไป Reuse
5	ทั้งแอพพลิเคชันและเอกสารประกอบการพัฒนาเอื้อต่อการนำไป Reuse

11. Installation Ease จะมองในเรื่องของการติดตั้งซอฟต์แวร์	
0	ไม่มีลักษณะพิเศษในการติดตั้ง
1	มีลักษณะพิเศษในการติดตั้ง แต่ไม่มีความต้องการเพิ่มเติมจาก User
2	การติดตั้งมีการกำหนดจาก User มีคู่มือในการติดตั้งและปรับเปลี่ยนโดยคู่มือที่จัดทำนั้นต้องผ่านการตรวจสอบและจะไม่คำนึงถึงผลกระทบจากการติดตั้ง
3	การติดตั้งมีการกำหนดจาก User มีคู่มือในการติดตั้งและปรับเปลี่ยนโดยคู่มือที่จัดทำนั้นต้องผ่านการตรวจสอบและคำนึงถึงผลกระทบจากการติดตั้ง
4	เพิ่มจากระดับของปัจจัยที่ 2 โดยมีระบบติดตั้งและเปลี่ยนแปลงอัตโนมัติ
5	เพิ่มจากระดับของปัจจัยที่ 3 โดยมีระบบติดตั้งและเปลี่ยนแปลงอัตโนมัติ

12. Operation Ease จะมองในเรื่องของประสิทธิภาพหรือระบบอัตโนมัติในการ Start Up, Back Up และขั้นตอนการ Recovery และมีการทดสอบอย่างมีประสิทธิภาพ โดยต้องมีคู่มือประกอบซอฟต์แวร์ที่ได้จัดทำขึ้น (Manual)	
0	ไม่มีลักษณะพิเศษในการจัดทำ Start Up, Back Up และ Recovery
1	มีลักษณะพิเศษที่มีประสิทธิภาพในการจัดทำ Start Up, Back Up และ Recovery
2	
3	
4	
5	ไม่ต้องมีผู้จัดการซอฟต์แวร์ โดยซอฟต์แวร์นั้นสามารถจัดการเองได้ รวมทั้งสามารถย้อนเพื่อแก้ไขข้อผิดพลาด

13. Multiple Sites จะมองในเรื่องของซอฟต์แวร์ว่าสามารถพัฒนาและสนับสนุนการติดตั้ง ใช้งานได้หลายพื้นที่ หรือหลายองค์กร	
0	User ไม่มีความต้องการพิเศษในการทำ Multiple Site
1	ความจำเป็นที่จะต้องมี Multiple Site หรือไม่นั้นต้องมีการพิจารณา ซึ่งถ้าเป็นแบบ Multiple Site แล้วนั้น ซอฟต์แวร์จะถูกพัฒนาภายใต้ลักษณะของ ฮาร์ดแวร์และซอฟต์แวร์เดียวกัน
2	ความจำเป็นที่จะต้องมี Multiple Site หรือไม่นั้นต้องมีการพิจารณา ซึ่งถ้าเป็นแบบ Multiple Site แล้วนั้น ซอฟต์แวร์จะถูกพัฒนาภายใต้ลักษณะของ ฮาร์ดแวร์และซอฟต์แวร์ที่เหมือนกัน
3	ความจำเป็นที่จะต้องมี Multiple Site หรือไม่นั้นต้องมีการพิจารณา ซึ่งถ้าเป็นแบบ Multiple Site แล้วนั้น ซอฟต์แวร์จะถูกพัฒนาภายใต้ลักษณะของ ฮาร์ดแวร์และซอฟต์แวร์ที่แตกต่างกัน
4	เอกสารประกอบได้มีการจัดการทดสอบและสนับสนุนซอฟต์แวร์แบบ Multiple Sites ภายใต้ระดับของปัจจัยที่ 1 หรือ 2
5	เอกสารประกอบได้มีการจัดการทดสอบและสนับสนุนซอฟต์แวร์แบบ Multiple Sites ภายใต้ระดับของปัจจัยที่ 3

14. Facilitate change จะมองในเรื่องของซอฟต์แวร์ว่าสามารถออกแบบและพัฒนาเพื่อสามารถเปลี่ยนซอฟต์แวร์ได้ตามความสะดวกหรือไม่ - การ Query ข้อมูลที่ยืดหยุ่นและสิ่งที่ช่วยออกแบบรายงานให้ง่ายขึ้น โดยสามารถดึงข้อมูลอย่างง่ายๆ ที่ต้องการได้ เช่น ตรรกะแบบ And/Or ซึ่งใช้งานกับ 1 Internal Logical file - การ Query ข้อมูลที่ยืดหยุ่นและสิ่งที่ช่วยออกแบบรายงานให้ง่ายขึ้น โดยสามารถดึงข้อมูลที่ซับซ้อนที่ต้องการได้ เช่น ตรรกะแบบ And/Or ซึ่งใช้งานมากกว่า 1 Internal Logical file - การ Query ข้อมูลที่ยืดหยุ่นและสิ่งที่ช่วยออกแบบรายงานให้ง่ายขึ้น โดยสามารถดึงข้อมูลที่ซับซ้อนได้ เช่น ตรรกะแบบ And/Or ซึ่งใช้งานได้มากกว่า 1 Internal Logical file - ข้อมูลที่ใช้ในการควบคุมทางธุรกิจนั้นจะถูกเก็บไว้ในตารางโดย User จะทำการ Maintain โดยการ Online Interactive Processes การเปลี่ยนแปลงใดๆ ที่เกิดขึ้นนั้นจะกระทบกับวันที่มาทำธุรกิจในวันถัดไปได้ - ข้อมูลที่ใช้ในการควบคุมทางธุรกิจนั้นจะถูกเก็บไว้ในตารางโดย User จะทำการ Maintain โดยการ Online Interactive Processes การเปลี่ยนแปลงใดๆ ที่เกิดขึ้นจะส่งผลทันทีทันใด	
0	ไม่มีคุณลักษณะของ Facilitate ตามที่กล่าวมาด้านบน
1	มีคุณลักษณะของ Facilitate ตามที่กล่าวมาด้านบน 1 ข้อ
2	มีคุณลักษณะของ Facilitate ตามที่กล่าวมาด้านบน 2 ข้อ
3	มีคุณลักษณะของ Facilitate ตามที่กล่าวมาด้านบน 3 ข้อ
4	มีคุณลักษณะของ Facilitate ตามที่กล่าวมาด้านบน 4 ข้อ
5	มีคุณลักษณะของ Facilitate ตามที่กล่าวมาด้านบน 5 ข้อ

15. Adaptivity	
0	ไม่มีการปรับเปลี่ยนใดๆ
1	ระบบสามารถเปลี่ยนรูปแบบของการข้อมูลออก (Output) เมื่อเปลี่ยนข้อมูลนำเข้า (Input) แต่สามารถทำได้ในพฤติกรรมที่จำกัดเพียงบางรูปแบบ
2	ระบบจำเป็นจะต้องแสดงข้อความบันทึก ซึ่งยอมให้โต้ตอบเป็นลำดับในหลายๆ input มากกว่าแค่ input เดียว
3	ระบบจำเป็นต้องมีการบันทึกข้อความ ซึ่งปรับเปลี่ยนไปถึงข้อมูลในอดีต
4	ระบบสามารถจัดการการเปลี่ยนแปลงผลกระทบจากการเปลี่ยนและประเมิน output จากการทดลองและข้อผิดพลาดที่เกิดขึ้นจากการใส่ input นั้นๆ
5	ระบบจะต้องสามารถอนุมารกลไกลที่จะโต้ตอบได้ จากการแปลความหมายของลักษณะการใช้งานของผู้ใช้งาน

16. Rapid prototyping	
0	ไม่มีการจัดทำ prototype ใดๆ
1	มีการจัดทำ prototype ในรูปแบบกระดาษ
2	มีการจัดทำ prototype ในรูปแบบการออกแบบหน้าตอซึ่งสามารถแสดงการออกแบบในบาง use case แต่ไม่มีส่วนของ function ใดๆ
3	มีการจัดทำ prototype ในรูปแบบการออกแบบหน้าจอ ซึ่งสามารถแสดงการออกแบบในเกือบทั้งหมดของ use case แต่ไม่มีการแสดงส่วน functionality หรือการทดสอบระบบ
4	มีการออกแบบ prototype การทำงานของฟังก์ชัน functional และเพิ่มการโต้ตอบเมื่อพิมพ์
5	มีการออกแบบ prototype การทำงานของฟังก์ชัน functional มีการโต้ตอบระบบจากผู้ใช้งาน รวมการเซ็นยอมรับของผู้ใช้งานในแต่ละเวอร์ชันของ prototyping

17. Multiple interfaces	
0-5	การให้คะแนนอ้างอิงจาก

	- มีการปรับเปลี่ยนได้ในตัวอักษร ในกรณีที่ผู้ใช้งานเขข้ามาต่างกันจะแสดงชื่อที่แตกต่าง แต่โครงสร้างส่วนใหญ่ไม่เปลี่ยนแปลง - มีความแตกต่างกันระหว่างผู้ใช้งานที่มีประสบการณ์กับผู้เริ่มต้นใช้งาน - มีความแตกต่างกันระหว่างประเภทของผู้ใช้งาน การศึกษาและพื้นฐานการทำงาน - การปรับเปลี่ยนหน้าจอขึ้นกับความสามารถในการใช้งาน - มีความแตกต่างระหว่างกานโต้ตอบกับ platform ขึ้นอยู่กับการพิจารณา

18. Multiuser interactivity	
0-5	การให้คะแนนอ้างอิงจาก - ระบบต้องติดตั้งในหลายๆสถานที่การทำงาน - ต้องมีการพิจารณา hardware และ environment - ระบบต้องสามารถควบคุมแบบง่ายโดยการรับคำสั่งระหว่างผู้ใช้งาน - ระบบต้องสามารถควบคุมการโต้ตอบของแต่ละผู้ใช้งานในเวลาเดียวกันได้ - ระบบต้องสามารถจัดการได้อย่างฉลาดและสามารถควบคุมคำสั่งและจัดการการโต้ต้องได้ในเวลาเดียวกันระหว่างหลายๆผู้ใช้งาน

ประวัติผู้เขียนวิทยานิพนธ์

นางสาวชมพูนุช เผ่าประพัธน์ เกิดวันที่ 9 ธันวาคม พ.ศ.2529 สำเร็จการศึกษาวิทยาศาสตร์บัณฑิต สาขาวิทยาการคอมพิวเตอร์ ภาควิชาคณิตศาสตร์ คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปี พ.ศ.2551 จากนั้นได้เข้าศึกษาต่อในระดับปริญญาโทวิทยาศาสตรมหาบัณฑิต สาขาการพัฒนาซอฟต์แวร์ด้านธุรกิจ ภาควิชาสถิติ คณะพาณิชยศาสตร์และการบัญชี จุฬาลงกรณ์มหาวิทยาลัย