# บทที่ 2

# ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

ในบทนี้จะกล่าวถึงทฤษฎีการวัดขนาดซอฟต์แวร์และงานวิจัยที่เกี่ยวข้อง โดยเริ่มจากการวัดเชิงขนาด (Size-Oriented Metrics) ได้แก่ จำนวนบรรทัดของโค้ด (Lines of Code: LOC) ซึ่งเป็นหน่วยวัดขนาดซอฟต์แวร์ที่ง่าย และใช้อย่างแพร่หลาย จากนั้นจะกล่าวถึงทฤษฎีและแนวคิดของการวัดเชิงหน้าที่ (Function-Oriented Metrics) โดยกล่าวถึงทฤษฎีฟังก์ชันพอยต์ ซึ่งเป็นการวัดซอฟต์แวร์เชิงหน้าที่ ซึ่งได้รับการพัฒนาขึ้นมาในช่วงที่แนวคิดการออกแบบและพัฒนาซอฟต์แวร์แบบโครงสร้าง (Structural Design) ยังเป็นที่แพร่หลายอยู่ ต่อมาเมื่อการวิเคราะห์และออกแบบซอฟต์แวร์เชิงวัตถุได้รับการพัฒนา ทฤษฎีการวัดแบบฟังก์ชันพอยต์ได้ถูกพัฒนาขึ้นเพื่อประยุกต์ให้สอดคล้องเข้ากับแนวทางการวิเคราะห์และออกแบบซอฟต์แวร์เชิงวัตถุ ผลที่ได้คือการวัดซอฟต์แวร์เชิงหน้าที่ใหม่ที่เรียกว่า ฟังก์ชันพอยต์เชิงวัตถุ

การวัด (Measurements) แบ่งได้เป็น 2 ลักษณะคือ การวัดในเชิงปริมาณ (Quantitative) และการวัดในเชิงคุณภาพ (Qualitative) การวัดเชิงปริมาณสามารถวัดได้โดยตรงจากสิ่งที่มองเห็นหรือสัมผัสได้ เช่น ความยาว ความหนา ปริมาตรสุทธิ เป็นต้น ซึ่งตรงกันข้ามกับการวัดเชิงคุณภาพที่ไม่สามารถวัดได้โดยตรง เช่น ประสิทธิภาพ ความทนทาน ความน่าเชื่อถือ เป็นต้น การวัดในเชิงปริมาณของกระบวนการทางวิศวกรรมซอฟต์แวร์นั้นพิจารณาถึงสิ่งต่างๆ อาทิเช่น จำนวนบรรทัดของโค้ด ความเร็วในการประมวลผล ขนาดหน่วยความจำที่ซอฟต์แวร์ต้องการใช้ ค่าใช้จ่าย และปริมาณกำลังคนที่ต้องใช้ สิ่งเหล่านี้ถูกพิจารณาว่าเป็นการวัดในเชิงปริมาณของซอฟต์แวร์ทั้งสิ้น ส่วนการวัดในเชิงคุณภาพทางวิศวกรรมซอฟต์แวร์นั้นจะพิจารณาถึงความสามารถต่างๆที่ซอฟต์แวร์พึงกระทำได้ (Functionality) คุณภาพการทำงาน (Quality) ความซับซ้อนของซอฟต์แวร์ (Complexity) ประสิทธิภาพ (Performance) ความสามารถ (Efficiency) ความน่าเชื่อถือ (Reliability) และความง่ายในการบำรุงรักษา (Maintainability) เป็นต้น

## 2.1 ทฤษฎีที่เกี่ยวข้อง

หัวข้อนี้จะอธิบายทฤษฎีที่เกี่ยวข้องกับงานวิจัย ได้แก่ การวิเคราะห์และออกแบบเชิงวัตถุ การวัดเชิงขนาด และการวัดเชิงหน้าที่ โดยในหัวข้อการวัดเชิงหน้าที่จะอธิบายในเรื่องของฟังก์ชันพอยต์และฟังก์ชันพอยต์เชิงวัตถุ

### 2.1.1 การวิเคราะห์และออกแบบเชิงวัตถุ

แนวคิดและทฤษฎีที่ใช้ประกอบการออกแบบและพัฒนาเครื่องมือวัดในงานวิจัย ได้ผ่านกระบวนการวิเคราะห์และออกแบบเชิงวัตถุ โดยในหัวข้อนี้ได้แสดงรายละเอียดทฤษฎีทางด้านวิศวกรรมซอฟต์แวร์เชิงวัตถุ

กระบวนการกำหนดแนวคิดให้กับวัตถุ และการใช้งานแผนภาพของภาษายูเอ็มแอล (Unified Modeling Language: UML) ในงานวิจัย

2.1.1.1 วิศวกรรมซอฟต์แวร์เชิงวัตถุ (Object-Oriented Software Engineering, OOSE) [1]

การพัฒนาซอฟต์แวร์โดยหลักประกอบไปด้วยขั้นตอนสำคัญคือ การวิเคราะห์และออกแบบ การพัฒนาและการทดสอบ ซึ่งจะเรียกการพัฒนาซอฟต์แวร์ที่ใช้แนวทางดังกล่าวข้างต้นประกอบแนวคิดเชิงวัตถุ ว่า วิศวกรรมซอฟต์แวร์เชิงวัตถุ

วิศวกรรมซอฟต์แวร์เชิงวัตถุประกอบไปด้วย 4 ขั้นตอนหลักคือ

1) Object-Oriented Analysis (OOA)

ขั้นตอนการวิเคราะห์เพื่อให้ทราบว่าขอบเขตของปัญหา (Problem Domain) คืออะไร และเพื่อทำความเข้าใจในรายละเอียดของปัญหาเหล่านั้น เป็นการหาคำตอบให้กับคำถามที่ว่า "What is the problem to be solved?

2) Object-Oriented Design (OOD)

ขั้นตอนการออกแบบหรือจำลอง (Model) วิธีการเพื่อแก้ปัญหาในขอบเขตของปัญหาที่กำหนด (Problem Domain) ซึ่งเป็นการหาคำตอบให้กับคำถามที่ว่า "How to solve the problem?"

3) Object-Oriented Programming (OOP)

ขั้นตอนการสร้างหนทางแก้ปัญหาในรายละเอียดให้เกิดขึ้นและใช้งานได้จริง เป็นการตอบปัญหาที่ว่า "How to implement the solution?"

4) Object-Oriented Testing (OOT)

ขั้นตอนการทดสอบสิ่งที่ได้พัฒนาขึ้นมา ซึ่งใช้เป็นหนทางการแก้ปัญหา เป็นการตอบปัญหาที่ว่า "How to test the solution?"

ซึ่งทั้งหมดสามารถสรุปเป็นประโยคสัญลักษณ์ได้ว่า OOSE = OOA + OOD + OOP + OOT

ก่อนที่จะเริ่มการวิเคราะห์และออกแบบซอฟต์แวร์โดยใช้แนวคิดเชิงวัตถุเป็นหลักในการปฏิบัตินั้น มีหลายสิ่งที่ผู้วิจัยต้องทำความเข้าใจก่อน นั่นคือเนื้อหาของความเป็นเชิงวัตถุ โดยปกติในชีวิตประจำวัน เมื่อพิจารณารอบๆตัว จะพบว่ามีวัตถุ (Object) ต่างๆมากมาย ไม่ว่าจะเป็นวัตถุที่เราสามารถมองเห็นได้และจับต้องได้ เช่น โต๊ะ รถยนต์ คอมพิวเตอร์ หรือแม้แต่ตัวเรา เป็นต้น และวัตถุที่มีอยู่จริงแต่ไม่สามารถจับต้องได้ เช่น กฎหมาย เวลา หรือความรู้ต่างๆ เป็นต้น และสิ่งที่เกิดจากวัตถุต่างๆเหล่านี้ก็คือ กิจกรรม (Activities) ความเคลื่อนไหว (Movement) หรือการกระทำ (Actions) หากพิจารณาโดยละเอียดแล้วจะพบว่ากิจกรรมต่างๆที่เกิดขึ้นล้วนแล้วแต่เกิดจากการมีความสัมพันธ์ (Relationship) การมีปฏิสัมพันธ์ (Interaction) ระหว่างวัตถุสองตัวขึ้นไป ซึ่งสามารถสรุปแยกแยะระหว่างความสัมพันธ์และปฏิสัมพันธ์ ได้คือ

**ความสัมพันธ์ (Relationship)** คือ ความเกี่ยวข้องกันหรือความสัมพันธ์ระหว่างวัตถุ 2 ตัวขึ้นไป เช่นความเป็นแม่-ลูก ความเป็นเจ้าของ เป็นต้น

**ปฏิสัมพันธ์ (Interaction)** คือการกระทำใดๆที่เกิดขึ้นระหว่างวัตถุ 2 ตัวขึ้นไป เช่น การสร้าง การเปลี่ยนแปลง การเล่น การกระตุ้นเป็นต้น ซึ่งปฏิสัมพันธ์นี้ทำให้เกิดกิจกรรมต่างๆขึ้น

ต่อมาเมื่อทราบสิ่งที่เป็นวัตถุ ความสัมพันธ์และปฏิสัมพันธ์ของวัตถุต่างๆซึ่งอยู่ในโลกของความเป็นจริงแล้ว ผู้ทำการวิเคราะห์ต้องพิจารณากำหนดกรอบเพื่อครอบสิ่งที่กล่าวข้างต้นซึ่งต้องการพิจารณาหรือสนใจ

เรียกว่าขอบเขตการพิจารณา ในขอบเขตการพิจารณาหนึ่งๆนั้นสามารถมีวัตถุได้ตั้งแต่ 2 ตัวขึ้นไป จนถึงจำนวนนับไม่ถ้วน ในขณะเดียวกันวัตถุตัวเดียวกันก็สามารถอยู่ในหลายๆขอบเขตการพิจารณาได้เช่นเดียวกัน ดังนั้นถ้าพิจารณาความหมายของแนวคิดเชิงวัตถุแล้ว จะหมายถึง การใช้วัตถุเป็นตัวหลักเพื่อพิจารณาความเป็นจริงต่างๆที่เกิดขึ้นในโลก และเมื่อพิจารณาความหมายของแนวคิด (Concept) จะหมายถึงความคิดรวบยอดที่มีให้กับวัตถุนั้นๆ ซึ่งเป็นแนวคิดในแง่ของข้อเท็จจริง ไม่รวมถึงความรู้สึกที่มีต่อวัตถุนั้นๆ ภายใต้กรอบหรือขอบเขตที่กำหนด ผลจากการให้แนวคิดกับวัตถุนั้นทำให้เกิดการจัดกลุ่มของวัตถุขึ้น ซึ่งกลุ่มของวัตถุที่ได้จากกระบวนการนี้เรียกว่าแอบสแต็กอ็อบเจค (Abstract Object) หรือเรียกอีกอย่างหนึ่งว่าคลาส ข้อเท็จจริงประการหนึ่งในแนวคิดเชิงวัตถุคือคลาสถือเป็นนามธรรม ซึ่งไม่สามารถทำให้คลาสดำเนินกิจกรรมใดๆได้

2.1.1.2 กระบวนการกำหนดแนวคิดให้กับวัตถุ (Abstraction) [1]

กระบวนการกำหนดแนวคิดให้กับวัตถุเพื่อสร้างคลาส แบ่งออกเป็น 4 กระบวนการย่อยคือ

- คลาสซิฟิเคชันแอบสแต็กชัน (Classification Abstraction)

เป็นกระบวนการที่ใช้เพื่อแยกประเภทวัตถุต่างๆที่อยู่ในกรอบความสนใจ และกำหนดแนวคิดกับวัตถุต่างๆเหล่านั้น เพื่อให้ได้คลาสพื้นฐาน (Fundamental Classes) ที่ต้องการ

- แอกกรีเกชันแอบสแต็กชัน (Aggregation Abstraction)

เป็นกระบวนการที่นำเอาคลาสพื้นฐานมารวมกันหรือประกอบกัน (Aggregate) เพื่อให้เกิดเป็นคลาสที่ใหญ่ขึ้นหรือซับซ้อนขึ้น โดยคลาสพื้นฐานดังกล่าวคือคลาสที่สร้างขึ้นในขั้นตอนคลาสซิฟิเคชันแอบสแต็กชันนั่นเอง

- เจนเนอร์รัลไลเซชันแอบสแต็กชัน (Generalization Abstraction)

เป็นกระบวนการนำคลาสที่มีลักษณะเหมือนหรือคล้ายคลึงกันหรือมีคุณสมบัติอย่างใดอย่างหนึ่งร่วมกัน (General) มาจัดหมวดหมู่ไว้เป็นคลาสเดียวกัน และกระบวนการย้อนกลับของเจนเนอร์รัลไลเซชันแอบสแต็กชันเรียกว่าสเปเชียลไลเซชัน (Specialization) ซึ่งคือการพิจารณาว่าในคลาสหนึ่งๆนั้นสามารถจำแนกเป็นคลาสเป็นคลาสใดได้บ้าง

- แอสโซซิเอชันแอบสแต็กชัน (Association Abstraction)

เป็นกระบวนการในการสร้างความสัมพันธ์ระหว่างคลาสต่างๆในกรอบแนวคิดหรือขอบเขตของปัญหาที่สนใจ ความสัมพันธ์ดังกล่าวคือความสัมพันธ์ที่ไม่สามารถอธิบายได้ด้วยแอกกรีเกชัน หรือเจนเนอร์รัลไลเซชัน แต่แอสโซซิเอชันเป็นการอธิบายความสัมพันธ์ของคลาสในเชิงกิจกรรม

อย่างไรก็ตามกระบวนการกำหนดแนวคิดให้กับวัตถุทั้ง 4 ที่กล่าวมาเป็นเพียงแนวคิดในการสร้างและ จัดหมวดหมู่คลาสต่างๆเท่านั้น ซึ่งต้องอาศัยวิธีการนำเสนอแนวคิดเหล่านี้ เพื่อให้เห็นภาพที่ชัดเจนและเข้าใจได้ง่ายขึ้น วิธีการนำเสนอซึ่งเป็นที่ยอมรับและนิยมในปัจจุบันคือการนำเสนอด้วยภาษายูเอ็มแอล ซึ่งเป็นภาษาที่ประกอบไปด้วยแผนภาพ (Diagram) ประเภทต่างๆที่ใช้ในการแสดงผลลัพธ์ที่ได้จากกระบวนการกำหนดแนวคิดให้กับวัตถุต่างๆ

2.1.1.3 แผนภาพของภาษายูเอ็มแอลที่ใช้ในงานวิจัย

แผนภาพของภาษายูเอ็มแอล แบ่งออกเป็น 2 ลักษณะ คือแผนภาพเชิงสถิตย์ (Static Diagram) และแผนภาพเชิงกิจกรรม (Dynamic Diagram)

2.1.1.3.1 แผนภาพเชิงสถิตย์ (Static Diagram)

เป็นแผนภาพแสดงขอบเขตของปัญหา โดยแสดงการมีอยู่ของคลาสต่างๆและความสัมพันธ์ของคลาสเหล่านั้นในระบบ แต่ไม่แสดงกิจกรรมที่เกิดขึ้นกับคลาสต่างๆแต่อย่างใด แผนภาพเชิงสถิตย์ที่ใช้ในงานวิจัยนี้ได้แก่

- แผนภาพการใช้งาน (Use Case Diagram)

เป็นแผนภาพแสดงถึงส่วนประกอบต่างๆของขอบเขตของปัญหา และความสัมพันธ์ของส่วนประกอบต่างๆเหล่านั้น ซึ่งจะเรียกส่วนประกอบเหล่านั้นว่ายูสเคส (Use Case) ซึ่งเปรียบเสมือนเป็นคลาสหนึ่งคลาส

- แผนภาพคลาส (Class Diagram)

หลังจากการวิเคราะห์ขอบเขตของปัญหาจนกระทั่งได้โมเดลการใช้งาน ซึ่งแสดงให้เห็นถึงระบบย่อยต่างๆที่เรียกว่ายูสเคส และความสัมพันธ์ระหว่างยูสเคสเหล่านั้น ในการวิเคราะห์ขอบเขตของปัญหาหนึ่งๆ นอกจากยูสเคสแล้ว สิ่งที่ได้จากกระบวนการกำหนดแนวคิดให้กับวัตถุ คือคลาสต่างๆซึ่งได้จากกระบวนการคลาสซิฟิเคชันแอบสแต็กชัน หลังจากนั้นเมื่อใช้กระบวนการอื่นๆ ได้แก่ แอกกรีเกชัน เจนเนอร์รัลไลเซชัน และแอสโซซิเอชัน แล้วผลลัพธ์ที่ได้คือความสัมพันธ์ของคลาสในรูปแบบต่างๆ ได้แก่

**ความสัมพันธ์แบบแอกกรีเกชัน** เป็นความสัมพันธ์ในลักษณะประกอบกันของคลาส หรือมีความสัมพันธ์ในเชิง Is part of หรือ Has-a ซึ่งอาจจะเกิดกรณีที่คลาสหลักประกอบไปด้วยคลาสย่อยชนิดที่หนึ่งเพียง 1 ชิ้น ประกอบด้วยคลาสย่อยชนิดที่สองจำนวน 4 ชิ้นขึ้นไป และประกอบด้วยคลาสย่อยชนิดที่สาม จำนวนไม่จำกัด เป็นต้น

**ความสัมพันธ์แบบเจนเนอร์รัลไลเซชันหรือสเปเชียลไลเซชัน** เป็นความสัมพันธ์ในลักษณะสืบทอดจากคลาสแม่สู่คลาสลูก โดยการตัดคุณสมบัติพิเศษบางอย่างออกไป เพื่อทำให้คลาสมีลักษณะเป็นสามัญ (Generalization) หรือโดยการพิจารณาเพิ่มเติมคุณสมบัติใหม่ๆให้กับคลาสนั้นมีลักษณะพิเศษเพิ่มขึ้นกว่าเดิม (Specialization)

**ความสัมพันธ์แบบแอสโซซิเอชัน** เป็นความสัมพันธ์ในลักษณะการใช้งาน หรือการเป็นเจ้าของ ซึ่งความสัมพันธ์ดังกล่าวนี้ เป็นความสัมพันธ์แบบเกี่ยวพันกัน (is related to) ไม่ใช่ส่วนหนึ่งของอีกคลาสหนึ่งหรือเป็นชนิดหนึ่งของอีกคลาสหนึ่ง

ซึ่งแผนภาพที่ใช้แสดงสิ่งเหล่านี้ได้แก่แผนภาพคลาส ซึ่งเป็นแผนภาพที่ใช้แสดงคลาสและความสัมพันธ์ในแง่ต่างๆระหว่างคลาส ซึ่งความสัมพันธ์ที่กล่าวถึงในแผนภาพคลาสนี้ถือเป็นความสัมพันธ์เชิงสถิตย์ ซึ่งหมายถึงความสัมพันธ์ที่มีอยู่แล้วเป็นปกติระหว่างคลาสต่างๆไม่ใช่ความสัมพันธ์ที่เกิดขึ้นเนื่องจากกิจกรรมต่างๆในการที่จะจำลองคลาสและความสัมพันธ์ ของคลาสต่างๆ แผนภาพคลาสในการวิเคราะห์เชิงวัตถุ นั้น สิ่งที่ต้องคำนึงถึงก็คือคลาสทั้งหมดที่ต้องอยู่ในระบบหรือโลกแห่งความเป็นจริง โดยที่ยังไม่ต้องคำนึงถึงคลาสย่อยๆที่แสดงรายละเอียดต่างๆที่อาจเพิ่มขึ้นมาในระหว่างการพัฒนาโปรแกรม

2.1.1.3.2 แผนภาพเชิงกิจกรรม (Dynamic Diagram)

เป็นแผนภาพแสดงกิจกรรม ที่เกิดขึ้นในขอบเขตของปัญหา นั่นคือการแสดงถึงสิ่งที่เกิดขึ้นจากกิจกรรมของคลาสต่างๆที่มีในระบบ โดยแผนภาพเชิงกิจกรรมที่ใช้ในงานวิจัยนี้ได้แก่

- แผนภาพแสดงลำดับการทำงาน (Sequence Diagram)

เป็นแผนภาพที่แสดงถึงกิจกรรมรวมของระบบ โดยกิจกรรมดังกล่าวนั้นเกิดจากการเรียกใช้งานฟังก์ชัน ที่มีอยู่ในคลาสต่างๆนั่นเอง

- แผนภาพแสดงกิจกรรม (Activity Diagram)

แผนภาพแสดงกิจกรรม เป็นแผนภาพที่แสดงถึงกิจกรรมในภาพที่เจาะจงลงไปในฟังก์ชันต่างๆของคลาสแต่ละตัว

2.1.2 การวัดเชิงขนาด (Size-Oriented Metrics)

เป็นวิธีการวัดซอฟต์แวร์ในลักษณะที่ใช้ขนาดของซอฟต์แวร์เป็นตัวบ่งบอกถึงระดับคุณภาพ และผลิตผลของการทำงาน พิจารณาตารางตัวอย่างด้านล่างต่อไปนี้

ตารางที่ 2.1 แสดงตัวอย่างโครงงานพัฒนาซอฟต์แวร์พร้อมกับข้อมูลการวัดเชิงขนาด [1]

| Project | LOC | Effort | $(000) | pp. doc. | Errors | Defects | People |
|---|---|---|---|---|---|---|---|
| Alpha | 12,100 | 24 | 168 | 365 | 134 | 29 | 3 |
| Beta | 27,200 | 62 | 440 | 1224 | 321 | 86 | 5 |
| Gamma | 20,200 | 43 | 314 | 1050 | 256 | 64 | 6 |

ตารางที่ 2.1 แสดงข้อมูลรายละเอียดของโครงงานพัฒนาซอฟต์แวร์ที่ได้จากการวัดและเก็บบันทึก เช่น ข้อมูลเกี่ยวกับจำนวนบรรทัดของโค้ดที่พัฒนาขึ้น กำลังคนที่ใช้ในการโครงงาน ค่าใช้จ่ายที่เกิดขึ้นในระหว่างการพัฒนา จำนวนหน้าของเอกสารที่ได้จัดทำขึ้น เป็นต้น เมื่อพิจารณาโครงงาน Alpha โครงงานนี้มีโค้ดต้นฉบับของโปรแกรมทั้งสิ้น 12,100 บรรทัด ใช้ระยะเวลาในการพัฒนา 24 เดือน มีค่าใช้จ่ายเกิดขึ้นทั้งสิ้น 168,000 ดอลล่าห์ ได้จัดทำเอกสารจำนวน 365 หน้า พบข้อผิดพลาดของซอฟต์แวร์จำนวน 134 รายการ ก่อนซอฟต์แวร์จะถูกนำไปใช้ และยังพบข้อบกพร่องอีกจำนวน 29 รายการภายหลังจากซอฟต์แวร์ถูกนำไปใช้แล้ว โครงงานนี้ใช้ทีมผู้พัฒนาทั้งสิ้น 3 คนจะเห็นได้ว่าข้อมูลต่างๆที่ได้กล่าวข้างต้นเป็นข้อมูลการวัดในเชิงขนาดทั้งสิ้น ซึ่งแล้วแต่ผู้บริหารโครงงานจะนำข้อมูลใดมาใช้ในการพิจารณาเพื่อวิเคราะห์คุณภาพ และผลิตผลของการทำงาน ที่เกิดขึ้นในโครงงาน หนึ่งในข้อมูลเชิงขนาดที่นิยมใช้กันในปัจจุบันคือ จำนวนบรรทัดของโค้ด ซึ่งสามารถนำมาประยุกต์เพื่อวิเคราะห์การวัดได้เช่น ค่าใช้จ่ายต่อบรรทัด และจำนวนบรรทัดต่อชั่วโมง การวัดจำนวนบรรทัดของโค้ดสามารถทำได้ง่าย และสะดวก ซึ่งการวัดวิธีนี้ใช้มาตั้งแต่แนวคิดการพัฒนาซอฟต์แวร์แบบโครงสร้าง ยังเป็นที่แพร่หลายกันอยู่ ถึงแม้ในปัจจุบันการวิเคราะห์และออกแบบซอฟต์แวร์ได้พัฒนาไปมาก แต่การใช้จำนวนบรรทัดของโค้ดยังคงเป็นการวัดที่นิยมกันอยู่ สิ่งที่สำคัญคือเมื่อพิจารณาถึงค่าใช้จ่ายและเวลาที่ใช้ต่อบรรทัดของ

โค้ดแล้ว วิธีวัดนี้จะขาดความเป็นมาตรฐาน เพราะการคำนวณจำนวนบรรทัดมีหลายวิธี เช่น อาจได้จากผลรวมของการนับจำนวนสเตทเมนท์ จำนวนบรรทัดว่าง จำนวนบรรทัดของคำอธิบาย วงเล็บปีกกา ซึ่งขึ้นอยู่กับข้อกำหนดที่ทำขึ้น โดยเฉพาะอย่างยิ่งในปัจจุบันเครื่องมือที่ใช้ในการพัฒนาซอฟต์แวร์ได้พัฒนาไปมากมีประสิทธิภาพสูง ทำให้ช่วยลดจำนวนบรรทัดของโค้ดลงไปได้มาก สิ่งนี้ทำให้การคิดค่าใช้จ่ายต่อบรรทัดของโค้ดมีค่าสูงขึ้นอย่างผิดปกติ

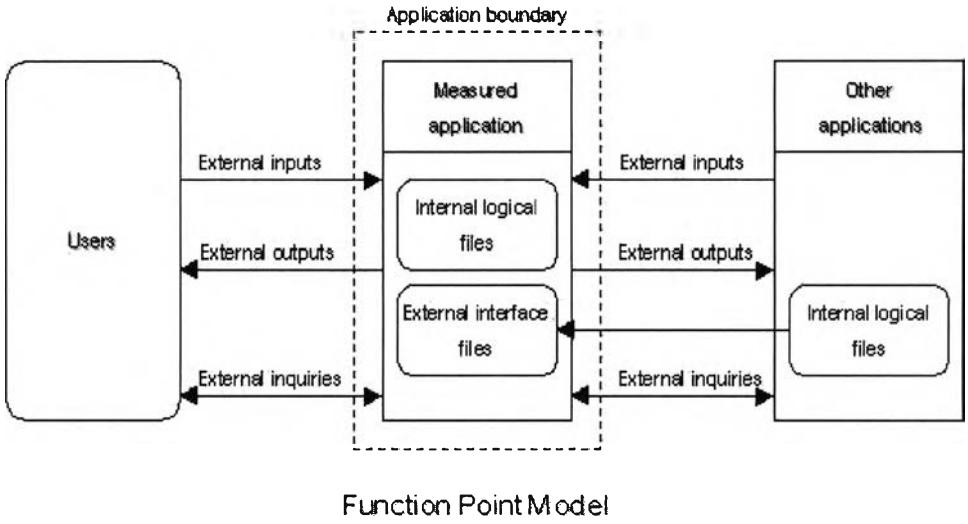
2.1.3 การวัดเชิงหน้าที่ (Function-Oriented Metrics)

การวัดเชิงหน้าที่จะพิจารณาหน้าที่และความสามารถที่มีอยู่ของซอฟต์แวร์ (Functionality) ซึ่งผู้ใช้งาน (End User) ซอฟต์แวร์สามารถรับรู้และเข้าใจได้ ด้วยเหตุที่ความสามารถของซอฟต์แวร์ไม่สามารถถูกวัดได้โดยตรง ดังนั้นจึงมีความพยายามที่จะหาหน่วยวัดซึ่งทำหน้าที่บ่งบอกถึงจำนวนหรือปริมาณความสามารถของซอฟต์แวร์ ดังนั้นจึงเกิดแนวคิดของการวัดฟังก์ชันพอยต์ขึ้น

2.1.3.1 ฟังก์ชันพอยต์ (Function Point) [2]

ในปลายทศวรรษที่ 1970 A.J. Albrecht ขณะทำงานวิจัยอยู่ที่บริษัท IBM ได้เสนอแนวทางการวัดขนาดซอฟต์แวร์ที่สามารถใช้ได้กับทุกๆภาษาคอมพิวเตอร์ อีกทั้งวิธีที่ใช้วัด ผู้ใช้ซอฟต์แวร์ ยังสามารถมองเห็น รับรู้และทำความเข้าใจได้ไม่ยาก สิ่งที่ A.J. Albrecht ต้องการวัดคือความสามารถต่างๆที่มีของซอฟต์แวร์ (Functionality of software) A.J. Albrecht สังเกตว่า ซอฟต์แวร์ต่างๆมีองค์ประกอบพื้นฐานที่คล้ายคลึงกัน ซึ่งมีด้วยกัน 5 ส่วน ดังรูปที่ 2.1 โดยมีรายละเอียดดังนี้

- เอ็กซ์เทอร์นอลอินพุต (External Inputs: EI) เป็นข้อมูลจากภายนอกที่ซอฟต์แวร์ต้องการใช้เพื่อประมวลผล
- เอ็กซ์เทอร์นอลเอาพุต (External Outputs: EO) เป็นผลลัพธ์ของการทำงานของซอฟต์แวร์ที่ออกสู่ภายนอก
- เอ็กซ์เทอร์นอลอินไควรี (External Inquiries: EQ) การสอบถามข้อมูลจากภายนอก
- อินเทอร์นอลลอจิคอลไฟล์ (Internal Logical Files: ILF) เป็นไฟล์ข้อมูลที่ถูกแก้ไขเปลี่ยนแปลง หรือเพิ่มเติม โดยตัวของซอฟต์แวร์
- เอ็กซ์เทอร์นอลอินเตอร์เฟสไฟล์ (External Interface Files: EIF) เป็นไฟล์ข้อมูลที่มีจุดประสงค์เพียงแค่ใช้อ้างอิงในซอฟต์แวร์เท่านั้น ไม่มีการแก้ไขเปลี่ยนแปลง หรือเพิ่มเติมใดๆโดยซอฟต์แวร์ แฟ้มข้อมูลนี้จะถูกดูแลโดยซอฟต์แวร์อื่น ดังนั้นเอ็กซ์เทอร์นอลอินเตอร์เฟสไฟล์นี้จะเป็นอินเทอร์นอลลอจิคอลไฟล์ของซอฟต์แวร์อื่นที่ดูแลไฟล์นี้นั่นเอง




รูปที่ 2.1 โมเดลของฟังก์ชันพอยต์ [3]

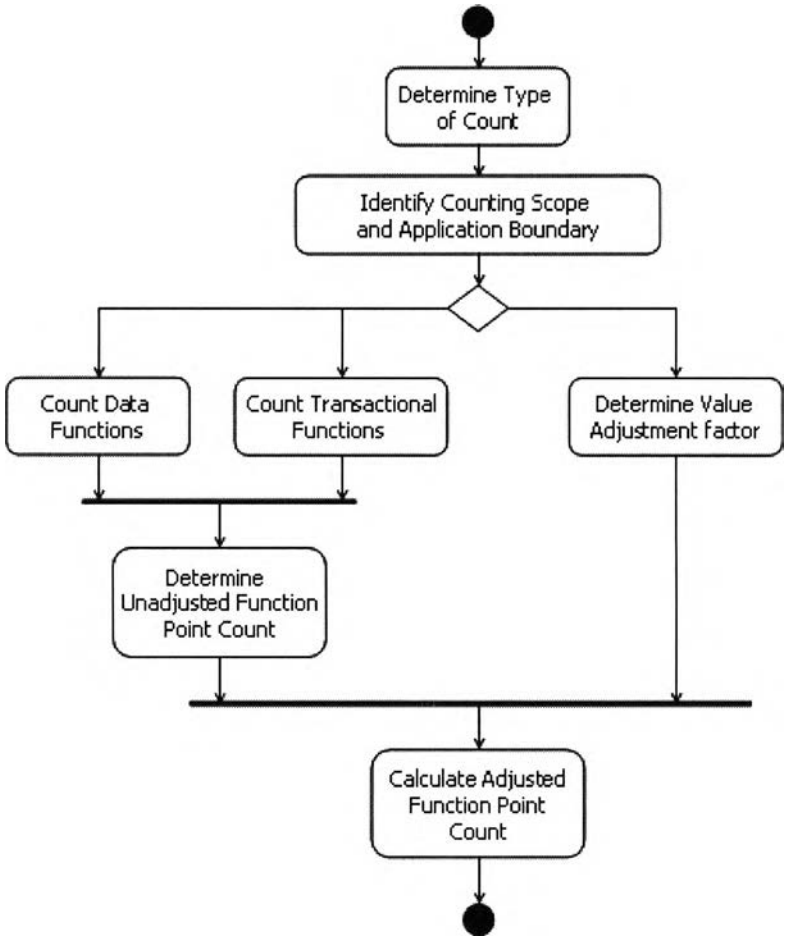
A.J. Albrecht ได้ให้น้ำหนัก (Weight Factor) องค์ประกอบพื้นฐานทั้ง 5 แตกต่างกัน ค่าดังกล่าวนี้จะแทนความซับซ้อนโดยประมาณขององค์ประกอบพื้นฐานที่กล่าวข้างต้น ซึ่งองค์ประกอบใดมีความซับซ้อนมากค่าน้ำหนักที่ให้ก็จะมีค่ามาก และในทางกลับกันถ้าองค์ประกอบใดมีความซับซ้อนน้อยค่าน้ำหนักที่ให้ก็จะมีค่าน้อยเช่นกัน ค่าน้ำหนักต่อไปนี้ได้จากการทดลองและวิจัยของ A.J. Albrecht ดังแสดงในตารางที่ 2.2

ตารางที่ 2.2 แสดงค่าน้ำหนักสำหรับแต่ละองค์ประกอบพื้นฐานของซอฟต์แวร์ [2]

| **องค์ประกอบ** | **ค่าน้ำหนัก (Weight Factor)** |
|---|---|
| External Inputs | 4 |
| External Outputs | 5 |
| External Inquiries | 4 |
| Internal Logical Files | 10 |
| External Interface Files | 7 |

วิธีการวัดขนาดซอฟต์แวร์ที่กล่าวข้างต้นนี้มีชื่อว่าฟังก์ชันพอยต์เป็นเทคนิคการวัดขนาดซอฟต์แวร์วิธีแรกที่นำเอาความสามารถที่มีอยู่ของซอฟต์แวร์ มาใช้ในการวิเคราะห์ได้จริง กล่าวได้ว่าฟังก์ชันพอยต์ได้เสนอวิธีการวัดขนาดซอฟต์แวร์โดยการวิเคราะห์งานที่มีของระบบจากมุมมองของผู้ใช้ที่สามารถมองเห็นและโต้ตอบได้ ผลลัพธ์จากการใช้เทคนิคการวัดแบบฟังก์ชันพอยต์นี้ สามารถนำไปประยุกต์ใช้เพื่อประมาณจำนวนบรรทัดของโค้ดในการพัฒนาซอฟต์แวร์ได้ ซึ่งข้อดีที่ชัดเจนของฟังก์ชันพอยต์คือภาษาคอมพิวเตอร์ที่ใช้ในการพัฒนาซอฟต์แวร์จะไม่มีผลกระทบต่อค่าที่ได้จากการวัด ถึงแม้ว่าในปัจจุบันเครื่องมือในการพัฒนาซอฟต์แวร์และภาษาคอมพิวเตอร์จะก้าวหน้าและแตกต่างกันเพียงใด แต่คุณสมบัติและความสามารถของซอฟต์แวร์ที่พัฒนานั้น

ยังคงเหมือนเดิม ดังนั้นค่าฟังก์ชันพอยต์ที่ได้จึงไม่แตกต่างกัน รูปที่ 2.2 แสดงขั้นตอนการวัดแบบฟังก์ชันพอยต์ (Function Point Counting Procedure) ซึ่งมีรายละเอียดดังต่อไปนี้




รูปที่ 2.2 แผนภาพกิจกรรมการนับของฟังก์ชันพอยต์ [5]

**Determine Type of Count** – ในรูปที่ 2.2 เป็นขั้นตอนการกำหนดประเภทของการนับ ซึ่งเป็นขั้นตอนแรกในกระบวนการนับของฟังก์ชันพอยต์ ประเภทของการนับมีอยู่ด้วยกัน 3 แบบ

- ดีเวลล็อปเมนต์โปรเจ็ค (Development project) เป็นการนับจำนวนฟังก์ชันการทำงานของซอฟต์แวร์ที่ได้รับการติดตั้งใช้งานเป็นครั้งแรกภายหลังจากการพัฒนาเสร็จสิ้น กล่าวคือเป็นซอฟต์แวร์เวอร์ชันแรกสุดที่ให้กับผู้ใช้ จำนวนฟังก์ชันพอยต์ของดีเวลล็อปเมนต์โปรเจ็คเป็นฟังก์ชันพอยต์เริ่มต้นของการนับแบบแอพลิเคชัน
- เอนฮานซ์เมนต์โปรเจ็ค (Enhancement project) เป็นการนับจำนวนฟังก์ชันการทำงานของซอฟต์แวร์ที่พัฒนาต่อจากเวอร์ชันก่อนหน้า โดยการเปลี่ยนแปลงของซอฟต์แวร์อาจเป็น การเพิ่มฟังก์ชันการทำงาน การแก้ไขปรับปรุงฟังก์ชันที่มีอยู่เดิม หรือกำจัดฟังก์ชันที่ไม่จำเป็นหรือผู้ใช้ไม่ต้องการออกไป เป็นต้น จำนวนฟังก์ชันพอยต์ของเอนฮานซ์เมนต์โปรเจ็คจะถูกนำไปปรับปรุงจำนวนฟังก์ชันพอยต์ของแอพลิเคชันเพื่อสะท้อนให้เห็นถึงการเปลี่ยนแปลงของซอฟต์แวร์

- แอพลิเคชัน (Application) เป็นจำนวนฟังก์ชันพอยต์สุทธิของซอฟต์แวร์ จำนวนฟังก์ชันพอยต์ของดีเวลล็อปเมนต์โปรเจ็คเป็นฟังก์ชันพอยต์เริ่มต้นของแอพลิเคชัน และจะถูกปรับปรุงทุกครั้งเมื่อจำนวนฟังก์ชันพอยต์ของเอนฮานซ์เมนต์โปรเจ็คได้เกิดขึ้น

ขั้นตอนกระบวนการนับฟังก์ชันพอยต์จะเหมือนกันทั้งหมดไม่ว่าจะเป็นประเภทการนับแบบใดก็ตาม แต่สิ่งที่ต่างกันมีเพียงสูตรที่ใช้คำนวณค่าฟังก์ชันพอยต์สุดท้ายในขั้นตอน Calculate Adjusted Function Point Count ของรูปที่ 2.2

**Identify Counting Scope and Application Boundary** – ในรูปที่ 2.2 เป็นขั้นตอนการกำหนดขอบเขตการนับว่ามีคุณสมบัติและความสามารถของซอฟต์แวร์ใดบ้างที่เข้าข่ายในเงื่อนไขการนับ นอกจากนี้ยังต้องกำหนดขอบเขตของแอพลิเคชันซึ่งแสดงให้เห็นถึงเส้นแบ่งระหว่างซอฟต์แวร์ที่ถูกวัดและผู้ใช้งาน

**Count Data Functions** – ในรูปที่ 2.2 เป็นขั้นตอนการนับจำนวนอินเทอร์นอลลอจิคอลไฟล์ และเอ็กซ์เทอร์นอลอินเตอร์เฟสไฟล์

- อินเทอร์นอลลอจิคอลไฟล์ เป็นไฟล์ข้อมูลที่ถูกแก้ไขเปลี่ยนแปลง หรือเพิ่มเติม โดยตัวของซอฟต์แวร์เอง
- เอ็กซ์เทอร์นอลอินเตอร์เฟสไฟล์ เป็นไฟล์ข้อมูลที่มีจุดประสงค์เพียงแค่ใช้อ้างอิงในโปรแกรมเท่านั้น ไม่มีการแก้ไขเปลี่ยนแปลง หรือเพิ่มเติมใดๆโดยซอฟต์แวร์ แฟ้มข้อมูลนี้จะถูกดูแลโดยซอฟต์แวร์อื่น ดังนั้นเอ็กซ์เทอร์นอลอินเตอร์เฟสไฟล์นี้จะเป็นอินเทอร์นอลลอจิคอลไฟล์ ของซอฟต์แวร์อื่นที่ดูแลไฟล์นี้

**Count Transactional Functions** – ในรูปที่ 2.2 เป็นขั้นตอนการนับจำนวนฟังก์ชันการทำงานที่ทำหน้าที่ดังต่อไปนี้

- เอ็กซ์เทอร์นอลอินพุต เป็นการรับข้อมูลจากภายนอก ซึ่งซอฟต์แวร์ต้องการใช้เพื่อประมวลผล
- เอ็กซ์เทอร์นอลเอาพุต เป็นผลของการทำงานของซอฟต์แวร์ที่ออกสู่ภายนอก
- เอ็กซ์เทอร์นอลอินไควรี เป็นการรับการสอบถามข้อมูลของซอฟต์แวร์จากภายนอก

**Determine Unadjusted Function Point Count (UFPC)** – ในรูปที่ 2.2 เป็นขั้นตอนที่สะท้อนให้เห็นถึงจำนวนฟังก์ชันการทำงานทั้งหมดที่ซอฟต์แวร์เตรียมไว้ให้กับผู้ใช้งาน ซึ่งอยู่ในรูปแบบของฟังก์ชันการทำงานอะไรที่มีให้กับผู้ใช้ แต่มิใช่ฟังก์ชันนั้นทำงานอย่างไร โดยฟังก์ชันการทำงานมีอยู่ด้วยกัน 2 ประเภทคือ ฟังก์ชันทำงานเกี่ยวกับข้อมูล (Data Functions) และฟังก์ชันทำงานเกี่ยวกับรายการติดต่อ (Transactional Functions)

**Determine Value Adjustment Factor (VAF)** – ในรูปที่ 2.2 เป็นขั้นตอนการกำหนดค่าแวลูแอดจัสเมนต์แฟกเตอร์ (Value Adjustment Factor: VAF) ให้กับซอฟต์แวร์ ซึ่งค่าดังกล่าวถูกแบ่งตามลักษณะและประเภทของ

ซอฟต์แวร์ ซึ่งมีอยู่ด้วยกัน 14 ประเภท แต่ละประเภทจะมีค่า Degree of Influence ตั้งแต่ 0 ถึง 5 และค่านี้จะใช้ในการคำนวณฟังก์ชันพอยต์สุดท้ายต่อไป

**Calculate Adjusted Function Point Count** – ในรูปที่ 2.2 เป็นขั้นตอนสุดท้ายในการคำนวณหาจำนวนของฟังก์ชันพอยต์ โดยใช้สูตรคำนวณเฉพาะ ซึ่งข้อมูลต่างๆที่ใช้ประกอบการคำนวณได้จากขั้นตอนการทำงานข้างต้น และสูตรที่ใช้จะเจาะจงสำหรับประเภทของโครงงาน ได้แก่ ดีเวลล็อปเมนต์โปรเจ็ค เอนฮานซ์เมนต์โปรเจ็ค หรือแอพลิเคชัน

นอกจากนี้ความสัมพันธ์ระหว่างจำนวนบรรทัดของโค้ดและฟังก์ชันพอยต์ A.J. Albrecht ได้ทำการวิจัยหาค่าเฉลี่ยโดยประมาณของจำนวนบรรทัดของโค้ดต่อการสร้าง 1 ฟังก์ชันพอยต์ สำหรับภาษาคอมพิวเตอร์ต่างๆ ดังตารางที่ 2.3 ดังนั้นผู้พัฒนาสามารถประมาณจำนวนบรรทัดของโค้ดได้ถ้าทราบจำนวนฟังก์ชันพอยต์ทั้งหมดของซอฟต์แวร์

ตารางที่ 2.3 แสดงจำนวนบรรทัดเฉลี่ยของโค้ดต่อ 1 ฟังก์ชันพอยต์สำหรับภาษาต่างๆ [1]

| **Programming Language** | **LOC/FP (Average)** |
|---|---|
| Assembly language | 320 |
| C | 128 |
| Cobol | 105 |
| Fortran | 105 |
| Pascal | 90 |
| Ada | 70 |
| Object-oriented languages | 30 |
| Fourth generation languages (4GLs) | 20 |
| Code generators | 15 |
| Spreadsheets | 6 |
| Graphical language (icons) | 4 |

2.1.3.2 ฟังก์ชันพอยต์เชิงวัตถุ (Object-Oriented Function Point) [2]

ในอดีตเทคนิคการวัดขนาดซอฟต์แวร์แบบฟังก์ชันพอยต์ ถูกนำไปประยุกต์ใช้กับการออกแบบและพัฒนาซอฟต์แวร์ที่ใช้แนวคิดแบบโครงสร้าง ซึ่งองค์ประกอบสำคัญของซอฟต์แวร์ที่ใช้ในการวัดขนาดคือไฟล์ข้อมูล (Logical Files) และรายการติดต่อ (Transactions) ซึ่งหมายถึง ข้อมูลเข้าจากภายนอก งานที่เป็นเอาพุตของซอฟต์แวร์ และการสอบถามข้อมูลจากภายนอก แต่เมื่อนำเทคนิคการวัดแบบฟังก์ชันพอยต์มาประยุกต์ใช้กับแนวคิดการออกแบบและพัฒนาซอฟต์แวร์เชิงวัตถุ องค์ประกอบสำคัญของซอฟต์แวร์ที่ใช้วัดจะไม่ใช่ไฟล์ข้อมูล

และรายการติดต่อข้างต้นอีกต่อไป แต่จะเป็นสิ่งที่เรียกว่าวัตถุ โดยมุ่งความสนใจไปยังคลาสและเมทธอดของคลาสเป็นสำคัญ

จากที่ทราบแล้วว่าไฟล์เป็นกลุ่มของข้อมูลที่เกี่ยวข้องกัน และเมื่อพิจารณาถึงคลาสก็มีลักษณะการเก็บข้อมูลต่างๆที่จำเป็นและเกี่ยวข้องกันเพื่อใช้ในคลาสเช่นเดียวกับไฟล์ ดังนั้นคลาสและไฟล์ในความหมายของเทคนิคการวัดแบบฟังก์ชันพอยต์มีความใกล้เคียงกันในตัวอยู่แล้ว และเมื่อพิจารณาถึงวัตถุ ซึ่งเป็นสิ่งที่ทำงานได้ของคลาส (Instance) มีความหมายเป็นนัยใกล้เคียงกับเรคคอร์ดของไฟล์นั่นเอง เทคนิคการวัดแบบฟังก์ชันพอยต์ได้แบ่งไฟล์ออกเป็น 2 ลักษณะคือ ไฟล์ที่ถูกแก้ไขเปลี่ยนแปลง หรือเพิ่มเติมโดยซอฟต์แวร์เอง และไฟล์ที่ใช้งานเพียงเพื่ออ้างอิงในซอฟต์แวร์ แต่ไม่มีการเปลี่ยนแปลงแก้ไขใดๆ ไฟล์นี้จะถูกดูแลโดยซอฟต์แวร์ภายนอกอื่นๆ การแบ่งประเภทของไฟล์ในลักษณะนี้แสดงให้เห็นถึงขอบเขตของโปรแกรม (Application Boundary) ได้อย่างชัดเจน และเมื่อนำขอบเขตของโปรแกรมมาประยุกต์เข้ากับแนวคิดเชิงวัตถุแล้ว ขอบเขตดังกล่าวจะถูกแบ่งตามลักษณะของคลาส กล่าวคือ คลาสภายนอก หมายถึงคลาสที่ไม่ได้เป็นส่วนประกอบของระบบ เช่น คลาสที่มีหน้าที่ให้บริการต่างๆ คลาสไลบรารีต่างๆ โดยไม่คำนึงว่าจะใช้งานคลาสเหล่านี้ในลักษณะใด เช่น สร้างวัตถุของคลาสเหล่านั้นขึ้นมาโดยตรงหรือใช้วิธีสืบทอดคลาส คลาสเหล่านี้จะถูกมองเป็นคลาสภายนอกทั้งสิ้น ดังนั้นพิจารณาได้ว่าคลาสที่อยู่ภายในขอบเขตของซอฟต์แวร์เป็นอินเทอร์นอลลอจิคอลไฟล์ และคลาสที่อยู่ภายนอกขอบเขตของซอฟต์แวร์เป็นเอ็กซ์เทอร์นอลอินเตอร์เฟสไฟล์ ของฟังก์ชันพอยต์นั่นเอง และจากที่กล่าวข้างต้นรายการติดต่อ ในฟังก์ชันพอยต์แบ่งออกเป็น 3 ประเภทได้แก่ ข้อมูลเข้าสู่ระบบจากภายนอก งานที่เป็นเอาพุตของระบบที่ออกสู่ภายนอก และการสอบถามข้อมูลของระบบจากภายนอก เมื่อนำมาประยุกต์เข้ากับแนวความคิดเชิงวัตถุ รายการติดต่อ ทั้ง 3 ประเภทข้างต้นสามารถถูกพิจารณาเป็นประเภทของเมทธอดในคลาสได้ โดยมีเมทธอดที่ทำงานเกี่ยวกับการนำข้อมูลเข้าสู่ระบบ เมทธอดที่ให้ผลการดำเนินงานออกสู่ระบบภายนอก และเมทธอดที่จัดการเกี่ยวกับการสอบถามข้อมูลระบบจากภายนอก ซึ่งเมทธอดทั้ง 3 ประเภทนี้จะเรียกว่าเซอร์วิสรีเควส (Service Request: SRs) ของคลาส ดังนั้นกล่าวได้ว่าคลาสเทียบได้กับลอจิคอลไฟล์และเมทธอดเทียบได้กับรายการติดต่อสำหรับฟังก์ชันพอยต์นั่นเอง

2.1.3.2.1 กระบวนการวัดของฟังก์ชันพอยต์เชิงวัตถุ (Measurement Process) [4]

สมมติให้ Object-Oriented Function Point (OOFP) เป็นฟังก์ชันหนึ่งของวัตถุในโมเดลเชิงวัตถุ D (Object Model) ที่กำหนด โมเดลเชิงวัตถุ D อาจถูกสร้างขึ้นในขณะทำการออกแบบ (Design Phase) หรือวิเคราะห์และสร้างจากโค้ดต้นฉบับของซอฟต์แวร์ที่มีอยู่ก่อนแล้วก็ได้ [4] OOFP สามารถคำนวณได้ดังนี้

$OOFP = OOFP_{ILF} + OOFP_{EIF} + OOFP_{SR}$ ซึ่ง

$$OOFP_{ILF} = \sum_{O \in A} W_{ILF}(DET_O, RET_O)$$

$$OOFP_{EIF} = \sum_{O \notin A} W_{EIF}(DET_O, RET_O)$$

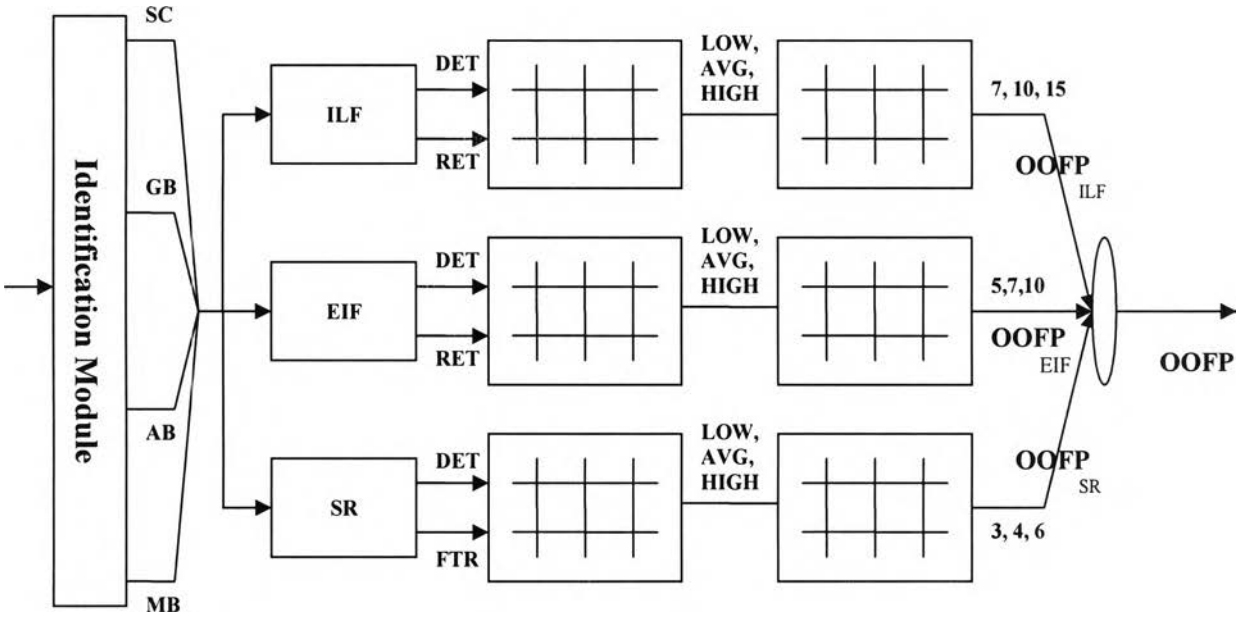
$$OOFP_{SR} = \sum W_{SR}(DET_O, FTR_O)$$

$o \in A$

$A$ คือ กลุ่มของวัตถุ (Object) ที่มีอยู่ในโปรแกรม

$O$ คือ วัตถุทั่วไปในโมเดลเชิงวัตถุ $D$

ดาต้าอิเลเมนต์ไทป์ (Data Element Types: DET) เรคคอร์ดอิเลเมนต์ไทป์ (Record Element Types: RET) และไฟล์ไทป์รีเฟอร์เรนซ์ (File Types Referenced: FTR) เป็นค่าที่เกิดจากการวัด ซึ่งได้จากการนับจำนวนลอจิคอลไฟล์ และเซอร์วิสรีเควส โดยที่ค่าเหล่านี้ถูกใช้เพื่อกำหนดความซับซ้อนของวัตถุโดยแสดงผ่านทางตารางวัดความซับซ้อน W (Complexity Metrics)




รูปที่ 2.3 โมเดลกระบวนการนับของฟังก์ชันพอยต์ [4]

รูปที่ 2.3 แสดงกระบวนการนับจำนวนฟังก์ชันพอยต์เชิงวัตถุ (Measurement Process) โดยมีรายละเอียดดังต่อไปนี้

1) โมเดลเชิงวัตถุ (Object Model) ถูกนำมาใช้ในการวิเคราะห์เพื่อพิจารณาคลาสที่มีทั้งหมดในระบบสำหรับใช้ในการนับ ซึ่งมีอยู่ด้วยกัน 4 วิธี

(1) ซิงเกิลคลาส (Single Class: SC)

เป็นวิธีการนับในลักษณะหนึ่งคลาสเปรียบเสมือนหนึ่งลอจิคอลไฟล์ จะเป็นการแยกแต่ละคลาสออกจากกัน โดยที่ไม่มีการพิจารณาความสัมพันธ์ของคลาสไม่ว่าจะเป็นแบบแอกกรีเกชัน หรือแบบสืบทอดก็ตาม จากรูปที่ 2.4 เป็นแผนภาพคลาสแสดงกิจกรรมเกี่ยวกับไพ่ ซึ่งประกอบด้วย 6 คลาสได้แก่ Collection of Cards, Card, Deck, Hand, Discard Pile และ Draw Pile เมื่อใช้วิธีการนับแบบซิงเกิ้ลคลาสแล้ว นับได้ 6 ลอจิคอลไฟล์ กล่าวคือ

แต่ละคลาส อาทิ คลาส Card จะถูกนับเป็น 1 ลอจิคอลไฟล์ โดยไม่คำนึงถึงความสัมพันธ์ของคลาสไม่ว่าจะเป็นแบบแอกกรีเกชันหรือแบบสืบทอด

(2) แอกกรีเกชัน (Aggregation: AB)

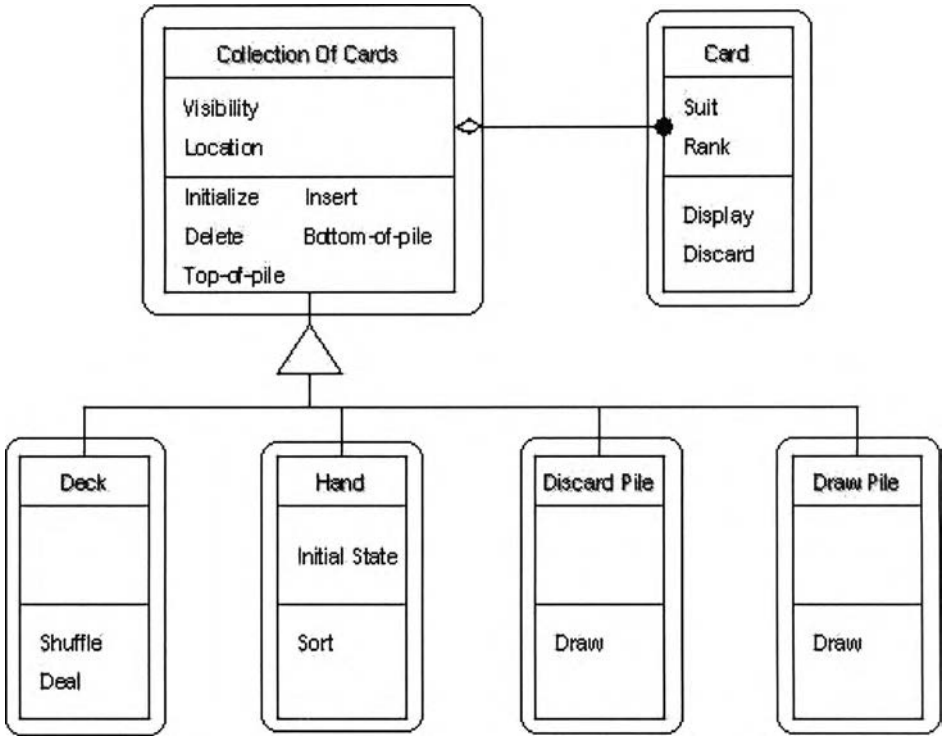
เป็นวิธีการนับที่พิจารณาความสัมพันธ์ของคลาสในแบบรวม กล่าวคือถ้ามีความสัมพันธ์ของคลาสเกิดขึ้นในลักษณะนี้ คลาสที่เกี่ยวข้องในลักษณะดังกล่าวทั้งหมดจะถูกนับเสมือนเป็น1 ลอจิคอลไฟล์ จากรูปที่ 2.5 เป็นแผนภาพคลาสแสดงกิจกรรมเกี่ยวกับไพ่ ซึ่งประกอบด้วย 6 คลาสได้แก่ Collection of Cards, Card, Deck, Hand, Discard Pile และ Draw Pile จากแผนภาพคลาส Collection of Cards มีความสัมพันธ์กับคลาส Card ในลักษณะแอกกรีเกชัน ดังนั้นเมื่อใช้วิธีการนับแบบแอกกรีเกชันแล้ว จะนับได้ 5 ลอจิคอลไฟล์ กล่าวคือคลาส Collection of Cards และ คลาส Card จะถูกนับรวมเป็น 1 ลอจิคอลไฟล์ ส่วนคลาสที่เหลือที่ไม่ได้มีความสัมพันธ์แบบแอกกรีเกชัน แต่ละคลาสจะถูกนับเป็น 1 ลอจิคอลไฟล์

(3) เจนเนอร์รัลไลเซชัน / สเปเชี่ยลไลเซชั่น (Generalization/Specialization: GB)

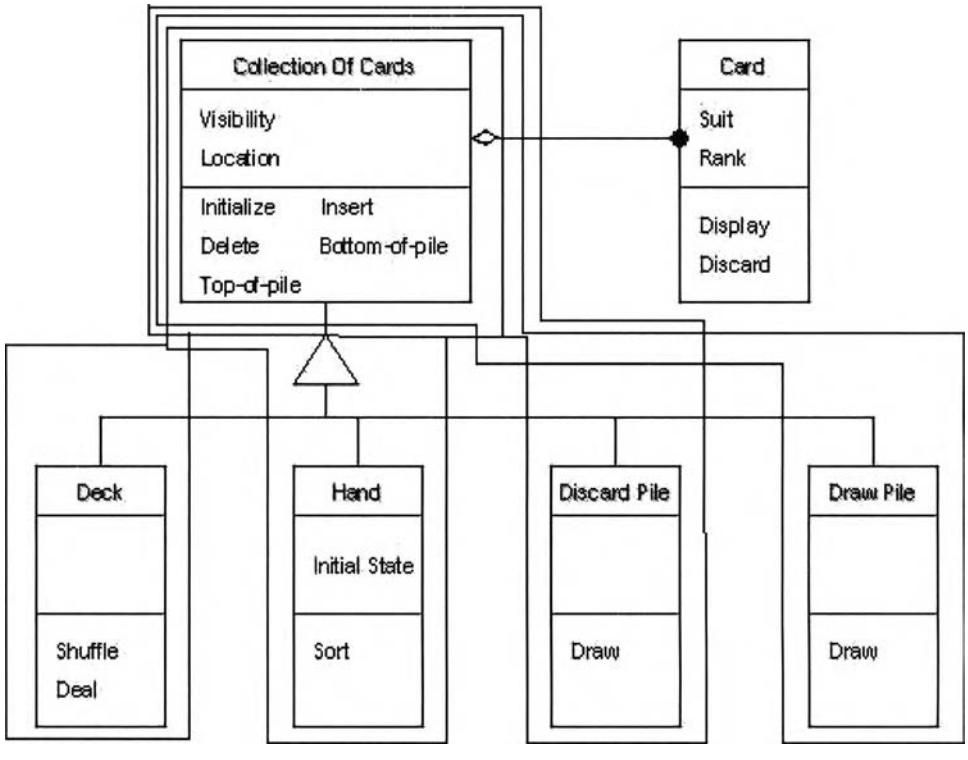
เป็นวิธีการนับที่พิจารณาความสัมพันธ์ของคลาสในลักษณะสืบทอด กล่าวคือถ้ามีความสัมพันธ์ของคลาสเกิดขึ้นในลักษณะนี้ คลาสต่างๆที่มีการสืบทอดกันมาเริ่มจากรากจนกระทั่งถึงปลายสุดของแต่ละเส้นทางการสืบทอดจะถูกนับเป็น 1 ลอจิคอลไฟล์ จากรูปที่ 2.6 เป็นแผนภาพคลาสแสดงกิจกรรมเกี่ยวกับไพ่ ซึ่งประกอบด้วย 6 คลาสได้แก่ Collection of Cards, Card, Deck, Hand, Discard Pile และ Draw Pile จากแผนภาพ คลาส Collection of Cards มีความสัมพันธ์กับคลาส Deck คลาส Hand คลาส Discard Pile และคลาส Draw Pile ในลักษณะสืบทอด ดังนั้นเมื่อใช้วิธีการนับแบบเจนเนอร์รัลไลเซชันแล้ว จะนับได้ 5 ลอจิคอลไฟล์ กล่าวคือคลาส Collection of Cards และ คลาส Deck ถูกนับรวมเป็น 1 ลอจิคอลไฟล์ นับเช่นนี้กับทุกๆคลาสที่มีความสัมพันธ์กันแบบเจนเนอร์รัลไลเซชัน ส่วนคลาสที่เหลือที่ไม่ได้มีความสัมพันธ์ดังกล่าว แต่ละคลาสจะถูกนับเป็น 1 ลอจิคอลไฟล์

(4) มิกซ์ (Mixed: MB)

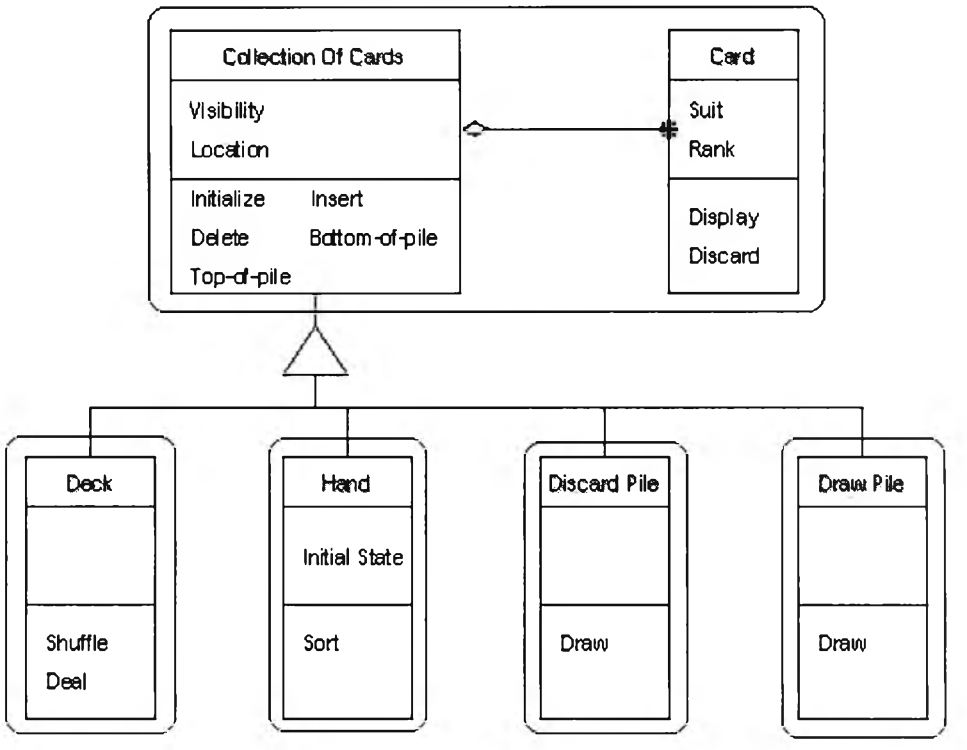
เป็นวิธีการนับที่ใช้ทั้งวิธีแอกกรีเกชันและเจนเนอร์รัลไลเซชัน /สเปเชี่ยลไลเซชั่นร่วมกัน จากรูปที่ 2.6 คลาส Collection of Cards และคลาส Card มีความสัมพันธ์แบบแอกกรีเกชัน และคลาส Collection of Cards และคลาส Deck มีความสัมพันธ์แบบเจนเนอร์รัลไลเซชัน ด้วยวิธีการนับแบบมิกซ์ทำให้ คลาส Collection of Cards คลาส Card และคลาส Deck ถูกนับเป็น 1 ลอจิคอลไฟล์ นับเช่นนี้กับทุกๆคลาสที่มีความสัมพันธ์กันแบบเจนเนอร์รัลไลเซชันและแอกกรีเกชัน ดังนั้นเมื่อใช้วิธีการนับแบบมิกซ์แล้ว จะนับได้ 4 ลอจิคอลไฟล์




รูปที่ 2.4 ซิงเกิ้ลคลาส




รูปที่ 2.5 แอกกรีเกชัน




รูปที่ 2.6 เจนเนอร์รัลไลเซชัน / สเปเชี่ยลไลเซชั่น

2) พิจารณาความความซับซ้อนของคลาสและเมทธอด

(1) เมธอด / เซอร์วิสรีเควส

นับจำนวน DET และ FTR ของเมธอดแต่ละเมธอดของทุกคลาสในระบบ โดยมีวิธีการนับดังนี้

- เมธอดที่เป็นแอบสแต็กจะไม่ถูกนับ
- เมธอดจะถูกพิจารณาเพียงครั้งเดียว ถึงแม้ว่าเมธอดนั้นจะถูกสืบทอดไปสู่คลาสลูก
- แต่ละพารามิเตอร์ของเมธอดที่เป็นชนิดข้อมูลพื้นฐานจะถูกพิจารณาเป็น 1 DET
- แต่ละตัวของตัวแปรโกลบอลที่เป็นชนิดข้อมูลพื้นฐานซึ่งถูกอ้างอิงในเมธอดจะถูกพิจารณาเป็น 1 DET
- แต่ละพารามิเตอร์ของเมธอดที่มีชนิดข้อมูลเป็นคลาสจะถูกพิจารณาเป็น 1 FTR
- แต่ละตัวของตัวแปรโกลบอลที่มีชนิดข้อมูลเป็นคลาสซึ่งถูกอ้างอิงในเมธอดจะถูกพิจารณาเป็น 1 FTR

จากแผนภาพคลาสในรูปที่ 2.4-2.6 ประกอบด้วยคลาส Collection Of Cards, คลาส Card, คลาส Deck, คลาส Hand, คลาส Discard Pile และ คลาส DrawPile เมื่อนับจำนวนเมธอด (เซอร์วิสรีเควส) จะมีทั้งสิ้น 12 เมธอด แต่เนื่องจากแผนภาพคลาสนี้เป็นแผนภาพตัวอย่างซึ่งไม่ได้ระบุชนิดข้อมูลของแอตริบิวของคลาสและพารามิเตอร์ของเมธอด จึงสมมติให้ค่าความซับซ้อนของแต่ละเมธอดมีค่าเท่ากับค่าเฉลี่ยดังนั้น ซึ่งหมายความว่า

แต่ละเมธอดจะมีค่า 4 ฟังก์ชันพอยต์ (ตารางที่ 2.4 แสดงการแปลงค่าความซับซ้อนของเซอร์วิสรีเควสไปสู่ค่าฟังก์ชันพอยต์) ดังนั้นเมธอดหรือเซอร์วิสรีเควสจะให้ค่าจำนวนฟังก์ชันพอยต์ 12 x 4 = 48 OOFPs

ตารางที่ 2.4 แสดงการแปลงค่าความซับซ้อนของเซอร์วิสรีเควสไปสู่ค่าฟังก์ชันพอยต์ [5]

| Function Complexity Rating | Unadjusted Function Points |
|---|---|
| Low | 3 |
| Average | 4 |
| High | 6 |

(2) คลาส / ลอจิคอลไฟล์

นับจำนวน DET และ RET ของแต่ละคลาสในระบบ โดยพิจารณาเป็น 2 รูปแบบ คือ ซิงเกิลลอจิคอลไฟล์ซึ่งเป็นการจัดกลุ่มแบบซิงเกิลคลาส และคอมโพสิตลอจิคอลไฟล์ซึ่งเป็นการจัดกลุ่มคลาสแบบแอกกรีเกชัน เจนเนอร์รัลไลเซชัน และมิกซ์ ในแต่ละคลาสหรือลอจิคอลไฟล์ต้องทำการนับจำนวน DET และ RET โดยวิธีการคำนวณนั้นต่างกันเล็กน้อยระหว่างซิงเกิลลอจิคอลไฟล์และคอมโพสิตลอจิคอลไฟล์ ตารางที่ 2.5 แสดงการแปลงค่าความซับซ้อนของลอจิคอลไฟล์ไปสู่ค่าฟังก์ชันพอยต์

- แต่ละคลาสจะมีค่าเริ่มต้นเป็น 1 RET
- แอตริบิวที่เป็นชนิดข้อมูลพื้นฐานจะถูกพิจารณาเป็น DET
- แอตริบิวที่มีชนิดข้อมูลเป็นคลาสจะถูกพิจารณาเป็น RET
- ถ้ามีความสัมพันธ์แบบแอกรีเกชัน คลาสที่ถูกแอกกรีเกชันต้องถูกเพิ่มค่า DET ขึ้น 1 DET และคลาสที่เป็นคอนเทนเนอร์จะเพิ่มขึ้น 1 RET
- การพิจารณากลุ่มของคลาสในลักษณะคอมโพสิตลอจิคอลไฟล์ ซึ่งได้แก่ แอกกรีเกชัน เจนเนอร์รัลไลเซชัน และมิกซ์ เป็นการรวมค่า DET และ RET ของแต่ละคลาสภายใต้รูปแบบของการรวมกลุ่มคลาสดังที่กล่าวมาข้างต้น

ตารางที่ 2.5 แสดงการแปลงค่าความซับซ้อนของลอจิคอลไฟล์ไปสู่ค่าฟังก์ชันพอยต์ [5]

| Function Complexity Rating | Unadjusted Function Points |
|---|---|
| Low | 7 |
| Average | 10 |
| High | 15 |

จากที่กล่าวข้างต้น แผนภาพคลาสนี้เป็นแผนภาพตัวอย่างซึ่งไม่ได้ระบุชนิดข้อมูลของแอตริบิวของคลาสและพารามิเตอร์ของเมธอด ดังนั้นจึงสมมติให้ทุกๆแอตริบิวของคลาสเป็นชนิดข้อมูลพื้นฐาน เพราะฉะนั้นเมื่อพิจารณา DET และ RET ของแต่ละคลาสแล้วจะได้ผลดังตารางที่ 2.6

ตารางที่ 2.6 แสดงผลการนับจำนวน DET และ RET ของคลาสจากแผนภาพคลาส

| คลาส | Data Element Type (DET) | Record Element Type (RET) |
|---|---|---|
| Collection Of Cards | 2 | 2 |
| Cards | 3 | 1 |
| Deck | 0 | 1 |
| Hand | 1 | 1 |
| Discard Pile | 0 | 1 |
| Draw Pile | 0 | 1 |

คลาส Cards มี 3 DET (2 DET ได้จาก 2 แอตริบิว และอีก 1 DET ได้จากความสัมพันธ์แบบ many-to-one association กับคลาส Collection Of Cards) และ 1 RET (ความเป็นคลาสทำให้มี 1 RET) คลาส Collection Of Cards มี 2 DET (ซึ่งได้จาก 2 แอตริบิว) และ 2 RET (1 RET ได้จาก one-to-many aggregation กับคลาส Cards และอีก 1 RET ได้จากความเป็นคลาสเอง) ส่วนคลาสอื่นๆก็ใช้วิธีการนับเช่นเดียวกันนี้

3) สำหรับขั้นตอนนี้ได้สร้างตารางขึ้น เพื่อเก็บข้อมูลที่เป็นผลจากการนับจำนวนฟังก์ชันพอยต์ และค่าที่ได้ทั้งหมดซึ่งอยู่ในรูปตารางจะถูกนำมารวมกันเพื่อให้ได้ค่า OOFP สุดท้าย ตารางที่ 2.7 แสดงผลการคำนวณจำนวนฟังก์ชันพอยต์ของแผนภาพตัวอย่าง โดยที่ความซับซ้อนของคลาสเป็นค่า Low ซึ่งตามตารางที่ 2.5 จะมีค่าฟังก์ชันพอยต์ (Unadjusted Function Points) เป็น 7

ตารางที่ 2.7 แสดงผลการคำนวณจำนวนฟังก์ชันพอยต์ของแผนภาพตัวอย่าง

| | ซิงเกิลคลาส | แอกกรีเกชัน | เจนเนอร์รัลไลเซชัน | มิกซ์ |
|---|---|---|---|---|
| Collection Of Cards | Low | Low | - | - |
| Cards | Low | - | Low | - |
| Deck | Low | Low | Low | Low |
| Hand | Low | Low | Low | Low |
| Discard Pile | Low | Low | Low | Low |
| Draw Pile | Low | Low | Low | Low |
| **ลอจิคอลไฟล์ (OOFP)** | **42** | **35** | **35** | **28** |
| **เซอร์วิสรีเควส (OOFP)** | **48** | **48** | **48** | **48** |
| **ผลรวม (OOFP)** | **90** | **83** | **83** | **76** |

การจัดกลุ่มคลาสแต่ละแบบ ไม่ว่าจะเป็น แบบซิงเกิลคลาส แอกกรีเกชัน เจนเนอร์รัลไลเซชัน และมิกซ์ จะมีอิทธิพลต่อจำนวนฟังก์ชันพอยต์เชิงวัตถุ ซึ่งในปัจจุบันจำนวนฟังก์ชันพอยต์เชิงวัตถุที่ได้จากการจัดกลุ่มคลาส

แบบเจนเนอร์รัลไลเซชันสำหรับใช้ในการคำนวณได้รับการยอมรับจากกลุ่มงานวิจัยฟังก์ชันพอยต์ว่ามีความถูกต้องมากที่สุด

เพื่อแสดงการประยุกต์ใช้ฟังก์ชันพอยต์เชิงวัตถุสำหรับการประมาณจำนวนบรรทัดของโค้ดต้นฉบับสำหรับภาษาคอมพิวเตอร์เชิงวัตถุ จากตารางที่ 2.7 จำนวนฟังก์ชันพอยต์เชิงวัตถุของการจัดกลุ่มคลาสแบบเจนเนอร์รัลไลเซชัน นับได้ 83 ฟังก์ชันพอยต์ และจากตารางที่ 2.3 จำนวนบรรทัดของโค้ดต้นฉบับสำหรับภาษาเชิงวัตถุมีค่าเฉลี่ย 30 บรรทัดต่อ 1 ฟังก์ชันพอยต์ ดังนั้นสามารถประมาณจำนวนบรรทัดของโค้ดต้นฉบับสำหรับโปรแกรมเชิงวัตถุตัวอย่างได้เป็นจำนวน 2,490 บรรทัด

## 2.2 งานวิจัยที่เกี่ยวข้อง

งานวิจัยที่เกี่ยวข้องกับการวัดขนาดซอฟต์แวร์โดยส่วนใหญ่อยู่บนพื้นฐานแนวคิดการออกแบบและพัฒนาซอฟต์แวร์แบบดั้งเดิม กล่าวคือเป็นลักษณะแบบโครงสร้าง ซึ่งในปัจจุบันแนวคิดเชิงวัตถุได้ก้าวเข้ามามีอิทธิพลต่อวงการพัฒนาซอฟต์แวร์มากขึ้นเรื่อยๆ ดังนั้นการทราบถึงขนาดซอฟต์แวร์เชิงวัตถุในระหว่างขั้นตอนการวิเคราะห์และออกแบบ ทำให้การจัดสรรทรัพยากรเป็นไปอย่างมีประสิทธิภาพ ซึ่งเทคนิคที่ใช้ในงานวิจัยเพื่อวัดขนาดของซอฟต์แวร์เชิงวัตถุ ได้แก่ฟังก์ชันพอยต์เชิงวัตถุ ซึ่งในปัจจุบันได้มีผู้ทำการวิจัยสนับสนุนไว้บางส่วนแล้วดังนี้

2.2.1 Mapping the OO-Jacobson Approach into Function Point Analysis [3]

เป็นงานวิจัยที่นำเสนอโดย Thomas Fetcke, Alain Abran และ Tho-Hau Nguyen ซึ่งได้หลักเกณฑ์การนับจำนวนฟังก์ชันพอยต์ โดยนำโมเดลการใช้งาน ในภาษายูเอ็มแอล นำมาประยุกต์เข้ากับแนวคิดของฟังก์ชันพอยต์โดยใช้ขั้นตอนการนับจาก IFPUG Counting Practices Manual [5] เป็นหลัก โดยมี 3 กรณีศึกษาที่ยืนยันว่าหลักเกณฑ์ที่สร้างขึ้นนั้นสามารถใช้งานได้จริง

2.2.2 Definition and Experimental Evaluation of Function Points for Object-Oriented Systems [4]

เป็นงานวิจัยที่นำเสนอโดย G. Caldiera, G.Antoniol และ C. Lokan ซึ่งได้เสนอวิธีวัดขนาดของโครงงานพัฒนาซอฟต์แวร์เชิงวัตถุ โดยการประยุกต์ใช้แนวคิดเชิงวัตถุกับวิธีฟังก์ชันพอยต์แบบดั้งเดิม ซึ่งเรียกวิธีการนี้ว่าฟังก์ชันพอยต์เชิงวัตถุ ผู้วิจัยเลือกใช้แผนภาพคลาส ในภาษายูเอ็มแอลเป็นเครื่องมือในการนับจำนวนฟังก์ชันพอยต์ แต่ในงานวิจัยยังไม่มีการพัฒนาเครื่องมือวัดสำหรับการนับจำนวนฟังก์ชันพอยต์เชิงวัตถุที่ประยุกต์แนวคิดและวิธีการนับที่สามารถใช้งานได้จริง

สำหรับงานวิจัยฉบับนี้เป็นการพัฒนาต่อจากงานวิจัยของ G. Caldiera, G.Antoniol และ C. Lokan โดยได้ใช้แนวทางการใช้งานแผนภาพคลาสมาเป็นเครื่องมือในการนับจำนวนของฟังก์ชันพอยต์ กฎเกณฑ์วิธีนับ และการคำนวณจะเป็นไปตามงานวิจัยดังกล่าว โดยสิ่งที่ได้ทำเพิ่มเติมในงานวิจัยคือได้ทำการออกแบบและพัฒนาเครื่องมือวัดสำหรับการนับจำนวนของฟังก์ชันพอยต์เชิงวัตถุแบบอัตโนมัติซึ่งสามารถใช้งานได้จริง ค่าวัดซึ่งเป็นผลลัพธ์ ได้แก่จำนวนฟังก์ชันพอยต์ของคลาสและของซอฟต์แวร์ทั้งหมดโดยใช้แผนภาพคลาสในภาษายูเอ็มแอลเป็นข้อมูลเข้าสำหรับการนับ