มาตรวัดการบำรุงรักษาซอฟต์แวร์ : กรณีศึกษา ระบบการสร้างรายงานอัตโนมัติเพื่อสนับสนุนงานตรวจรับฝุ่นละออง ในอุตสาหกรรมการผลิตฮาร์ดดิสก์

SOFTWARE MAINTENANCE METRICS: CASE STUDY OF AUTOMATED REPORT GENERATION SYSTEM FOR PARTICLE MONITORING IN HARD DISK DRIVE INDUSTRY

อภัสรา วิโรจน์ชยกิจ

APHATSARA WIROTYAKIJ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

สาขาวิชาเทคโนโลยีสารสนเทศ

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2555

KMITL-2012-IT-M-001-004

มาตรวัดการบำรุงรักษาซอฟต์แวร์ : กรณีศึกษา ระบบการสร้างรายงานอัตโนมัติเพื่อสนับสนุนงานตรวจจับฝุ่นละออง ในอุตสาหกรรมการผลิตฮาร์ดดิสก์

SOFTWARE MAINTENANCE METRICS: CASE STUDY OF AUTOMATED REPORT GENERATION SYSTEM FOR PARTICLE MONITORING IN HARD DISK DRIVE INDUSTRY

อภัสรา วิโรจน์ยะกูล

APHATSARA WIROTYAKUN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาเทคโนโลยีสารสนเทศ
คณะเทคโนโลยีสารสนเทศ
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2555

KMITL-2012-IT-M-001-004

SOFTWARE MAINTENANCE METRICS: CASE STUDY OF AUTOMATED REPORT GENERATION SYSTEM FOR PARTICLE MONITORING IN HARD DISK DRIVE INDUSTRY

APHATSARA WIROTYAKUN

A THESIS SUBMITTED IN PATIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
MASTER OF SCIENCE IN INFORMATION TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2012

KMITL-2012-IT-M-001-004

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์ มาตรวัดการบำรุงรักษาซอฟต์แวร์: กรณีศึกษา ระบบการสร้างรายงานอัตโนมัติเพื่อสนับสนุนงานตรวจจับฝุ่นละอองในอุตสาหกรรมการผลิตฮาร์ดดิสก์

Software Maintenance Metrics: Case Study of Automated Report Generation System for Particle Monitoring in Hard Disk Drive Industry

นักศึกษา นางสาวอภัสรา วิโรจน์ยะกูล

รหัสประจำตัว 51066447

ปริญญา วิทยาศาสตรมหาบัณฑิต

สาขาวิชา เทคโนโลยีสารสนเทศ

อาจารย์ที่ปรึกษาวิทยานิพนธ์ ผู้ช่วยศาสตราจารย์ ดร.พรฤดี เนติโสภากุล

คณะกรรมการสอบวิทยานิพนธ์	ลายมือชื่อ
ผู้ช่วยศาสตราจารย์ ดร.สมเกียรติ วังศิริพิทักษ์	
ดร.เทพชัย ทรัพย์นิธิ	
ผู้ช่วยศาสตราจารย์ ดร.ชุติเมษฎ์ ศรีนิลทา	
ผู้ช่วยศาสตราจารย์ ดร.พรฤดี เนติโสภากุล	

วัน / เดือน/ ปี ที่สอบ วันอังคารที่ 15 พฤษภาคม 2555 เวลา 09.30 น.

สถานที่สอบ ณ ห้อง 328 (ชั้น3) คณะเทคโนโลยีสารสนเทศ

คณะเทคโนโลยีสารสนเทศรับรองแล้ว

(รองศาสตราจารย์ ดร.จันทร์บูรณ์ สถิตวิริยวงศ์)

คณบดีคณะเทคโนโลยีสารสนเทศ

วันที่ 19 **เดือน** พฤษภาคม **พ.ศ.** 2555

หัวข้อวิทยานิพนธ์	มาตรวัดการบำรุงรักษาซอฟต์แวร์: กรณีศึกษา ระบบการสร้างรายงานอัตโนมัติเพื่อสนับสนุนงานตรวจจับฝุ่นละอองในอุตสาหกรรมการผลิตฮาร์ดดิสก์
นักศึกษา	นางสาวอภัสรา วิโรจน์ยะกูล
รหัสนักศึกษา	51066447
ปริญญา	วิทยาศาสตรมหาบัณฑิต
สาขาวิชา	เทคโนโลยีสารสนเทศ
แขนงวิชา	วิทยาการสารสนเทศ
พ.ศ.	2554
อาจารย์ที่ปรึกษา	ผศ.ดร. พรฤดี เนติโสภากุล

บทคัดย่อ

วิทยานิพนธ์ฉบับนี้นำเสนอการวัดการบำรุงรักษาซอฟต์แวร์ โดยใช้กรณีศึกษาของระบบการสร้างรายงานปริมาณฝุ่นละอองเพื่อสนับสนุนงานตรวจจับฝุ่นละอองในอุตสาหกรรมการผลิตฮาร์ดดิสก์ โดยระบบดังกล่าวจะต้องง่ายต่อการบำรุงรักษาและปรับขยายระบบในอนาคต เพื่อตอบสนองความต้องการของผู้ใช้งานที่หลากหลาย และประเภทรายงานที่จะเพิ่มขึ้นในอนาคต โดยวิทยานิพนธ์ฉบับนี้นำเสนอมาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์เพิ่มเติม จำนวน 4 มาตรวัด จากเดิมที่พิจารณาจากจำนวนบรรทัดของซอร์ดโค้ดเป็นการพิจารณาคลาสและเมธอด ซึ่งถือเป็นส่วนสำคัญสำหรับการเขียนโปรแกรมเชิงวัตถุ จากผลการศึกษาพบว่ามาตรวัดที่นำเสนอขึ้น โดยพิจารณาจากจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนัก (MS-WMC) ให้ค่าประมาณของเวลาในการบำรุงรักษาซอฟต์แวร์ได้ใกล้เคียงกับเวลาจริง นอกจากนี้ได้ทำการวัดโครงสร้างและความซับซ้อนของการบำรุงรักษาซอฟต์แวร์โดยใช้มาตรวัดเชิงวัตถุ โดยได้แยกประเภทการบำรุงรักษาซอฟต์แวร์ ประกอบด้วย 1) การเพิ่มเติมความต้องการของผู้ใช้งาน และ 2)การบำรุงรักษาซอฟต์แวร์โดยการจัดระเบียบซอร์ดโค้ด พบว่า งานทางด้านการเพิ่มเติมความต้องการของผู้ใช้งานจะทำให้โครงสร้างและความซับซ้อนของซอฟต์แวร์มีมากขึ้นและความสามารถในการบำรุงรักษาซอฟต์แวร์จะลดลง ขณะที่การจัดระเบียบซอร์ดโค้ดจะช่วยให้ซอฟต์แวร์มีความซับซ้อนน้อยลง สามารถบำรุงรักษาได้ง่ายขึ้น จากนั้นได้ทำการวัดผลการเปรียบเทียบโครงสร้างและความซับซ้อนของซอฟต์แวร์ระหว่างการออกแบบของระบบเดิม และการปรับปรุงการออกแบบโดยใช้ดีไซน์แพตเทิร์นชนิดอ็อบเซิร์ฟเวอร์แพตเทิร์น ผลการศึกษาพบว่า การปรับปรุงการออกแบบ จะทำให้ความซับซ้อนทางด้านการเรียกใช้เมธอดและความสัมพันธ์ระหว่างคลาสลดลงมากกว่าการออกแบบของระบบเดิม

Thesis	Software Maintenance Metrics: Case Study of Automated Report Generation System for Particle Monitoring in Hard Disk Drive Industry
Student	Miss.Aphatsara Wirotyakun
Student ID.	51066447
Degree	Master of Science
Program	Information Technology
Major	Information Science
Year	2011
Thesis Advisor	Asst.Prof. Dr. Ponrudee Netisopakul

ABSTRACT

This thesis proposed software maintenance metrics using a case study of an Automated Report Generation System for Particle Monitoring in Hard Disk Drive Industry (ARGS_PMS). The maintainability of the ARGS_PMS is important because the system must be able to support the need for various requirements in the future. The traditional maintenance size metric only relies on line of source code (LOC), which is hardly suitable for object-oriented software. This research proposed four new maintenance size metrics based on number of classes and methods changed during maintenance process. We found that, on average the maintenance size based on an average number of weighted methods per class (MS-WMC) gave the best performance in predicting maintenance time. Then we measured the results of using six object-oriented metrics for measuring structure and complexity. In this case study, there were two types of software improvement: 1) functional enhancement tasks and 2) refactoring task. We found that, for functional enhancement tasks, such as adding more reports, the maintainability declines with increasing complexity. On the other hand, the task of refactoring source code increases the maintainability. The design of old ARGS_PMS (original design) was compared to the redesign of an existing ARGS_PMS using observer pattern (redesign version). The result of comparing the original design and redesign version showed that in some metrics the redesign version has more cohesion between class methods and its attributes, and less coupling between objects.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ด้วยความกรุณาจาก ผศ.ดร.พรฤดี เนติโสภากุล อาจารย์ที่ปรึกษาวิทยานิพนธ์ที่ได้ให้คำเสนอแนะ แนวคิด ตลอดจนแก้ไขข้อบกพร่องต่างๆ มาโดยตลอด จนวิทยานิพนธ์ฉบับนี้เสร็จสมบูรณ์ ซึ่งข้าพเจ้าขอขอบพระคุณเป็นอย่างสูง

ขอขอบคุณ ศูนย์วิจัยร่วมเฉพาะทางด้านส่วนประกอบฮาร์ดดิสก์ไดรฟ์ I/UCRC (Industry/University Cooperative Research Center) ภายใต้ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (National Electronics and Computer Technology Center: NECTEC) ขอขอบคุณวิทยาลัยร่วมด้านเทคโนโลยีการบันทึกข้อมูลและการประยุกต์ใช้งาน (Data Storage Technology and Applications: DSTAR) สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง และบริษัทฮิตาชิ โกลบอล สตอเรจ เทคโนโลยีส์ (ประเทศไทย) จำกัด ที่ให้การสนับสนุนทางด้านทุนวิจัยสำหรับโครงการวิจัย

ขอขอบพระคุณ บิดา มารดา และครอบครัว ที่ให้คำปรึกษาและเป็นกำลังใจที่ดีเสมอมา

ขอขอบพระคุณคณาจารย์คณะเทคโนโลยีสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ทุกๆ ท่านที่ได้ประสิทธิ์ประสาทวิชาให้กับข้าพเจ้า ขอบคุณเพื่อนๆ พี่ๆ น้องๆ ในคณะเทคโนโลยีสารสนเทศ และหน่วยปฏิบัติการการจัดการองค์ความรู้และวิศวกรรมความรู้ (KMAKE LAB) ขอบคุณเจ้าหน้าที่คณะเทคโนโลยีสารสนเทศทุกๆ คนที่คอยให้ความช่วยเหลือและคำแนะนำต่างๆ เสมอมา รวมทั้งเจ้าหน้าที่ของบริษัทฮิตาชิ โกลบอล สตอเรจ เทคโนโลยีส์ (ประเทศไทย) จำกัด ที่ให้ข้อมูลอันเป็นประโยชน์ต่อการจัดทำวิทยานิพนธ์จนเสร็จสมบูรณ์

และความดีอันเกิดจากการศึกษาค้นคว้าครั้งนี้ ผู้เขียนขอมอบแด่บิดา มารดา ครู อาจารย์ และผู้มีพระคุณทุกท่าน ผู้เขียนมีความซาบซึ้งในความกรุณาอันดียิ่งจากทุกท่านที่ได้กล่าวนามมา และขอกราบขอบพระคุณมา ณ โอกาสนี้

อภัสรา วิโรจน์ยะกูล

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

สารบัญรูป (ต่อ)

บทที่1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในอุตสาหกรรมการผลิตฮาร์ดดิสก์ มนุษย์และเครื่องจักรจะทำงานภายในห้องที่มีการควบคุมสภาพแวดล้อม หรือ ห้องคลีนรูม (Cleanroom) [1-3] ปริมาณฝุ่นละอองจากห้องคลีนรูมจะจัดเก็บลงสู่ระบบฐานข้อมูลและนำมาสร้างรายงานอัตโนมัติ ซึ่งรายงานปริมาณฝุ่นละอองจะเพิ่มขึ้นตามความต้องการของผู้ใช้งาน ดังนั้น ระบบการสร้างรายงานอัตโนมัติจะต้องง่ายต่อการบำรุงรักษา เพื่อรองรับประเภทรายงานที่หลากหลายตามความต้องการของผู้ใช้งานในอนาคต

วิทยานิพนธ์ฉบับนี้จะเลือกมาตรวัดที่บ่งบอกคุณภาพของซอฟต์แวร์ทางด้านการบำรุงรักษาซอฟต์แวร์เป็นหลัก โดยมาตรวัดที่สำคัญ ได้แก่ ขนาดของการบำรุงรักษาซอฟต์แวร์ (Software Maintenance Size) และมาตรวัดทางด้านโครงสร้างและความซับซ้อนของซอฟต์แวร์ จึงใช้มาตรวัดเชิงวัตถุ (Object-Oriented Metrics) จำนวน 6 มาตรวัด เพื่อใช้วัดการบำรุงรักษาซอฟต์แวร์ของกรณีศึกษา ระบบการสร้างรายงานอัตโนมัติเพื่อสนับสนุนงานตรวจจับฝุ่นละอองในอุตสาหกรรมการผลิตฮาร์ดดิสก์ (Automated Report Generation System for Particle Monitoring in Hard Disk Drive Industry)

ในปัจจุบันมาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์จะพิจารณาจากจำนวนบรรทัดของการเขียนโปรแกรม (Line of Code) ที่เปลี่ยนแปลงไปในระหว่างที่ทำการบำรุงรักษา [4] งานวิจัยนี้จึงนำเสนอการปรับปรุงมาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์ โดยมาตรวัดที่ปรับปรุงขึ้นจะพิจารณาถึงคลาสและเมธอดที่เปลี่ยนแปลงไปเพื่อให้เหมาะสมกับโครงสร้างการเขียนโปรแกรมเชิงวัตถุ ซึ่งผลลัพธ์จากการทดสอบการบำรุงรักษาซอฟต์แวร์จะทำให้ทราบว่ามาตรวัดใดเหมาะสมสำหรับการวัดขนาดของการบำรุงรักษาซอฟต์แวร์ในขั้นตอนของการออกแบบโครงสร้างคลาส โดยการเปรียบเทียบมาตรวัดที่เหมาะสมจะใช้ค่าประมาณของเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์จากขนาดของการบำรุงรักษาซอฟต์แวร์ ที่ใกล้เคียงกับเวลาจริงที่ใช้ในระหว่างการบำรุงรักษาซอฟต์แวร์

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

1.2.1 นำเสนอมาตรวัดใหม่สำหรับวัดขนาดของการบำรุงรักษาซอฟต์แวร์ที่ใช้การพิจารณาคลาสและเมธอดที่เปลี่ยนแปลงไป จำนวน 4 มาตรวัด นอกเหนือจากมาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์ที่พิจารณาจากจำนวนบรรทัดที่เปลี่ยนแปลงไปของซอฟต์แวร์

1.2.2 วิเคราะห์เปรียบเทียบความสามารถในการบำรุงรักษาซอฟต์แวร์สำหรับการบำรุงรักษาซอฟต์แวร์สำหรับงาน 2 ลักษณะ คือ การบำรุงรักษาโดยการเพิ่มเติมความต้องการของผู้ใช้งาน และการบำรุงรักษาโดยการจัดระเบียบซอร์ดโค้ด โดยใช้มาตรวัดเชิงวัตถุ

1.2.3 เปรียบเทียบโครงสร้างและความซับซ้อนของซอฟต์แวร์จากการออกแบบระบบของกรณีศึกษา และการปรับปรุงการออกแบบระบบโดยใช้ดีไซน์แพตเทิร์นชนิดอ็อบเซิร์ฟเวอร์แพตเทิร์น โดยใช้มาตรวัดเชิงวัตถุ

1.3 ทฤษฎีและแนวคิดที่ใช้ในการวิจัย

คุณภาพของซอฟต์แวร์ (Software Quality) เป็นปัจจัยที่บ่งบอกว่าซอฟต์แวร์มีคุณภาพหรือไม่ประกอบด้วยหลายปัจจัย ซึ่งปัจจัยสำคัญที่จะนำเสนอในวิทยานิพนธ์ ได้แก่ ความสามารถด้านการบำรุงรักษา และ ความสามารถด้านการเพิ่มขยายระบบ เพื่อรองรับความต้องการของระบบที่เปลี่ยนแปลงไป จากการเพิ่มเติมความต้องการรายงานของผู้ใช้งาน

1.3.1 ความสามารถด้านการบำรุงรักษา (Maintainability) การบำรุงรักษาซอฟต์แวร์จะเกิดขึ้นหลังจากที่มีการพัฒนาและติดตั้งระบบแล้ว โดยระบบที่จะบำรุงรักษาได้ง่ายเป็นผลมาจากการออกแบบที่ดี ซึ่งมาตรวัดที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ เช่น ความพยายามที่ใช้ในการบำรุงรักษาซอฟต์แวร์ (Maintenance Effort) ซึ่งจะต้องใช้กำลังคนจำนวนน้อย

1.3.2 ความสามารถด้านการเพิ่มขยายระบบ (Scalability) การปรับขยายระบบจะเกี่ยวข้องโดยตรงกับการออกแบบโครงสร้างและความซับซ้อนของซอฟต์แวร์ เช่น การเพิ่มเติมคลาส การแก้ไขเปลี่ยนแปลงคลาสและเมธอดเดิม เป็นต้น ดังนั้นหากต้องการที่จะให้ซอฟต์แวร์ปรับขยายได้ง่าย ค่าความซับซ้อนของซอฟต์แวร์ในแต่ละเมธอดและแต่ละคลาสที่ทำการปรับปรุงจะต้องมีค่าต่ำ โดยมาตรวัดโครงสร้างและค่าความซับซ้อน เช่น ค่า Coupling Cohesion หรือ การสืบทอดคุณสมบัติ (Inheritance) เป็นต้น

ขนาดของซอฟต์แวร์ (Software Size) เป็นมาตรวัดเบื้องต้นและมาตรวัดที่สำคัญสำหรับการวัดซอฟต์แวร์ในกระบวนการพัฒนาระบบ การวัดขนาดของซอฟต์แวร์สามารถนำไปใช้ในการประมาณการค่าใช้จ่ายสำหรับการพัฒนาซอฟต์แวร์ [4] หรือการสร้างแบบจำลองของการประมาณการซอฟต์แวร์ เช่น แบบจำลองโคโคโม่ (COCOMO) และโคโคโม่เวอร์ชั่นสอง (COCOMO II) [5][21] โดยการวัดขนาดของซอฟต์แวร์สามารถวัดได้หลายวิธี เช่น จำนวนบรรทัดของซอร์ดโค้ด (LOC) หรือจากฟังก์ชั่นพอยต์ (Function Point) ซึ่งเป็นการวัดแบบดั้งเดิม (Traditional Metrics) และการวัดจากจำนวนเมธอด จำนวนคลาส หรือจำนวนเมธอดต่อคลาส ซึ่งเป็นการวัดเชิงวัตถุ (Object-Oriented Metrics)

สำหรับกระบวนการบำรุงรักษาซอฟต์แวร์ (Maintenance Phase) ปัจจุบันจะใช้การพิจารณาขนาดของการบำรุงรักษาซอฟต์แวร์จะวัดจากจำนวนบรรทัดของซอฟต์แวร์ที่เปลี่ยนแปลงไป [4]

ดังนั้นเพื่อให้มาตรวัดขนาดของซอฟต์แวร์เหมาะสมกับการเขียนโปรแกรมเชิงวัตถุจึงนำเสนอมาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์โดยพิจารณาคลาสและเมธอดที่เปลี่ยนแปลงไป โดยมาตรวัดใหม่ที่นำเสนอจะพิจารณามาตรวัดที่เหมาะสมจากค่าประมาณของเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์ เนื่องจาก We Li [6] ได้ทำการวิเคราะห์ความสัมพันธ์ระหว่าง ความพยายามในการบำรุงรักษาซอฟต์แวร์ (Maintenance Effort) และจำนวนบรรทัดของซอฟต์แวร์ที่เปลี่ยนแปลงไปจากระบบเดิม พบว่าปัจจัยทั้งสองมีความสัมพันธ์กัน ดังนั้นจึงมีแนวคิดที่จะหาค่าประมาณของเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์จาก จำนวนบรรทัด จำนวนคลาสและเมธอดที่เปลี่ยนแปลงไป

1.4 ขอบเขตการวิจัย

1.4.1 ปรับปรุงมาตรวัด ขนาดของการบำรุงรักษาซอฟต์แวร์ (Maintenance Size)

1.4.2 เปรียบเทียบค่าความคลาดเคลื่อนของมาตรวัดขนาดของการบำรุงรักษาโดยพิจารณาจากจำนวนบรรทัดของซอฟต์แวร์ (MS-LOC) และมาตรวัดใหม่ที่นำเสนอจำนวน 4 มาตรวัด ได้แก่

1.4.2.1 พิจารณาจากจำนวนเมธอดที่เปลี่ยนแปลงไป (MS-TM)

1.4.2.2 พิจารณาจากจากจำนวนคลาสที่เปลี่ยนแปลงไป (MS-TC)

1.4.2.3 พิจารณาจากจำนวนเมธอดต่อคลาสที่เปลี่ยนแปลงไป (MS-MC)

1.4.2.4 พิจารณาจากจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนักที่เปลี่ยนแปลงไป (MS-WMC)

โดยการเปรียบเทียบค่าความคลาดเคลื่อนของมาตรวัด จากการพิจารณามาตรวัดที่เหมาะสมจากค่าประมาณของเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์

1.4.3 เปรียบเทียบ ความสามารถในการบำรุงรักษาซอฟต์แวร์ สำหรับงาน 2 ลักษณะ ได้แก่ 1) การเพิ่มเติมความต้องการของผู้ใช้งาน และ 2) การบำรุงรักษาปรับเปลี่ยนโดยการจัดระเบียบซอร์ดโค้ด โดยพิจารณาความสามารถในการบำรุงรักษาซอฟต์แวร์จากมาตรวัดเชิงวัตถุจำนวน 6 มาตรวัด ประกอบด้วย

1.4.3.1 มาตรวัดค่าความเข้าคู่กันระหว่างวัตถุ (CBO)

1.4.3.2 มาตรวัดการขาดระดับความสัมพันธ์ภายในคลาส (LCOM)

1.4.3.3 มาตรวัดระดับความลึกของการสืบทอดคุณสมบัติของคลาส (DIT)

1.4.3.4 มาตรวัดจำนวนคลาสลูก (NOC)

1.4.3.5 มาตรวัดจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนัก (WMC)

1.4.3.6 มาตรวัดการตอบสนองต่อคลาส (RFC)

1.4.4 เปรียบเทียบโครงสร้างและความซับซ้อน ระหว่าง โครงสร้างของระบบเดิม และโครงสร้างที่ปรับปรุงการออกแบบโดยใช้ดีไซน์แพตเทิร์นชนิดอ็อบเซิร์ฟเวอร์แพตเทิร์น โดยใช้มาตรวัดเชิงวัตถุจำนวน 6 มาตรวัดในหัวข้อ 1.4.3.1 ถึง 1.4.3.6

1.5 ขั้นตอนการดำเนินงานวิจัย

1.5.1 ออกแบบและพัฒนา ระบบการสร้างรายงานในอุตสาหกรรมการผลิตฮาร์ดดิสก์

1.5.2 ออกแบบมาตรวัดเพื่อใช้สำหรับวัดขนาดของการบำรุงรักษาซอฟต์แวร์สำหรับงาน 2 ลักษณะ ได้แก่ 1) การบำรุงรักษาซอฟต์แวร์โดยการเพิ่มเติมความต้องการ และ 2) การปรับปรุงโครงสร้างของระบบโดยการจัดระเบียบซอร์ดโค้ด (Refactoring Source Code)

1.5.3 ออกแบบและจัดทำแบบฟอร์มสำหรับการบันทึกข้อมูลระหว่างที่ทำการพัฒนาและบำรุงรักษาซอฟต์แวร์

1.5.4 ออกแบบโครงสร้างคลาส โดยจะทำการปรับเปลี่ยนคลาสและเมธอดจากโครงสร้างคลาสเบื้องต้นของกรณีศึกษา โดยแบ่งโครงสร้างคลาสในส่วนของการบำรุงรักษาซอฟต์แวร์ออกเป็น 6 โครงสร้าง ประกอบด้วย

1.5.4.1 โครงสร้างคลาสที่ปรับปรุงจากการเพิ่มเติมรายงานระดับง่าย ครั้งที่ 1

1.5.4.2 โครงสร้างคลาสที่ปรับปรุงโดยการเพิ่มเติมรายงานระดับง่าย ครั้งที่ 2

1.5.4.3 โครงสร้างคลาสที่ปรับปรุงโดยการเพิ่มเติมรายงานระดับปานกลางครั้งที่ 1

1.5.4.4 โครงสร้างคลาสที่ปรับปรุงโดยการเพิ่มเติมรายงานระดับปานกลางครั้งที่ 2

1.5.4.5 โครงสร้างคลาสสำหรับการบำรุงรักษาโดยการจัดระเบียบซอร์ดโค้ด

1.5.4.6 โครงสร้างคลาสการปรับปรุงการออกแบบโดยใช้ดีไซน์แพตเทิร์น

1.5.5 ปรับปรุงระบบในแต่ละเวอร์ชั่นของซอฟต์แวร์ ซึ่งการปรับปรุงจะแก้ไขโครงสร้างคลาสภายใน เช่น กรณีของการเพิ่มเติมความต้องการประเภทรายงาน การปรับปรุงโครงสร้างคลาสโดยส่วนใหญ่จะเป็นการเพิ่มคลาสและเมธอดที่เกี่ยวข้องกับรายงานที่เพิ่มขึ้น กรณีการบำรุงรักษาซอฟต์แวร์โดยการจัดระเบียบซอร์ดโค้ดจะปรับปรุงโครงสร้างคลาสโดยการจัดระเบียบคลาสและเมธอดที่ไม่ใช้งานหรือใช้งานร่วมกัน เป็นต้น และการปรับปรุงการออกแบบโดยใช้ดีไซน์แพตเทิร์นจะเปลี่ยนแปลงโครงสร้างคลาสและเมธอดโดยการสืบทอดคุณสมบัติจากการเฝ้าสังเกตการณ์ข้อมูลที่เปลี่ยนแปลงไป โดยระหว่างที่ทำการปรับปรุงระบบจะทำการบันทึกข้อมูล ได้แก่ เวลาที่ใช้ในการพัฒนาและบำรุงรักษาซอฟต์แวร์ จำนวนบรรทัดของซอฟต์แวร์ คลาส และเมธอดที่เปลี่ยนแปลงไปในแต่ละเวอร์ชั่นของการบำรุงรักษาซอฟต์แวร์

1.5.6 ในการปรับปรุงแต่ละเวอร์ชั่นจะทำการวัดโครงสร้างของระบบโดยใช้มาตรวัดเชิงวัตถุ ได้แก่ มาตรวัดจำนวนเมธอดต่อคลาส (WMC) มาตรวัดจำนวนคลาสลูก (NOC) มาตรวัดระดับความลึกของการสืบทอดคุณสมบัติ (DIT) มาตรวัดการตอบสนองต่อคลาส (RFC) มาตรวัดความเข้าคู่ระหว่างวัตถุ (CBO) และมาตรวัดระดับการขาดความสัมพันธ์ภายในคลาส (LCOM)

1.5.7 คำนวณขนาดของการบำรุงรักษาซอฟต์แวร์จากจำนวนบรรทัดของซอฟต์แวร์ที่มีอยู่เดิม และมาตรวัดใหม่ที่นำเสนอ ได้แก่ มาตรวัดที่พิจารณาจากจำนวนเมธอด (MS-TM) มาตรวัดที่

พิจารณาจากจากจำนวนคลาส (MS-TC) มาตรวัดที่พิจารณาจากจำนวนเมธอดต่อคลาส (MS-MC) และมาตรวัดที่พิจารณาจากจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนัก (MS-WMC)

1.5.8 หลังจากคำนวณค่าขนาดของการบำรุงรักษาซอฟต์แวร์ (MS) ของแต่ละมาตรวัดแล้ว จะนำค่า MS ที่ได้มาหาค่าประมาณของเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์ (Estimated Maintenance Time) เพื่อพิจารณาว่า มาตรวัดใดเป็นมาตรวัดที่เหมาะสมสำหรับวัดขนาดของการบำรุงรักษาซอฟต์แวร์ สำหรับการหาค่าประมาณของเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์สามารถคำนวณได้จากการพิจารณาถึงสัดส่วนของ จำนวนคลาส/เมธอดที่ใช้ในการพัฒนาต่อเวลาที่ใช้ในการพัฒนา เท่ากับจำนวนคลาส/เมธอดที่เปลี่ยนแปลงไป ต่อเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์

1.5.9 เปรียบเทียบค่าความคลาดเคลื่อนของแต่ละมาตรวัด โดยการพิจารณาจากค่าความผิดพลาดของค่าประมาณของเวลากับเวลาจริงที่ใช้ในการบำรุงรักษาซอฟต์แวร์

1.5.10 สรุปและวิเคราะห์ผลการวัดการบำรุงรักษาซอฟต์แวร์ จากมาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์และจากมาตรวัดเชิงวัตถุ สำหรับงาน 2 ลักษณะ ได้แก่ 1) การบำรุงรักษาซอฟต์แวร์โดยการเพิ่มเติมความต้องการ และ 2) การบำรุงรักษาโดยการจัดระเบียบซอร์ดโค้ด

1.5.11 สรุปและวิเคราะห์ผลการเปรียบเทียบโครงสร้างและความซับซ้อนของการออกแบบระบบของระบบเดิม และการปรับปรุงโครงสร้างการออกแบบของระบบ ที่ใช้เป็นกรณีศึกษา

1.6 เค้าโครงวิทยานิพนธ์

วิทยานิพนธ์แบ่งออกเป็น 5 บทได้แก่

บทที่ 1 กล่าวถึง ความเป็นมาและความสำคัญของงานวิจัย ความมุ่งหมายและวัตถุประสงค์ของการศึกษา ทฤษฎีและแนวคิดที่ใช้ในงานวิจัย ขอบเขตและขั้นตอนของการศึกษางานวิจัย

บทที่ 2 กล่าวถึง งานวิจัยที่เกี่ยวข้องกับการวัดการบำรุงรักษาซอฟต์แวร์ มาตรวัดซอฟต์แวร์เชิงวัตถุ และงานวิจัยทางด้านการวัดซอฟต์แวร์เชิงวัตถุและความสัมพันธ์ระหว่างคุณภาพของซอฟต์แวร์ด้านการบำรุงรักษาและมาตรวัดเชิงวัตถุ

บทที่ 3 กล่าวถึง ขั้นตอนและวิธีการดำเนินงานวิจัย การออกแบบโครงสร้างคลาสในการบำรุงรักษาซอฟต์แวร์ของแต่ละเวอร์ชั่น การออกแบบมาตรวัดซอฟต์แวร์ การออกแบบประเภทของรายงาน การออกแบบแบบฟอร์มการบันทึกข้อมูลระหว่างการพัฒนาและปรับปรุงซอฟต์แวร์

บทที่ 4 กล่าวถึง ผลการทดลองจากการวัดการบำรุงรักษาซอฟต์แวร์จากมาตรวัด ขนาดของการบำรุงรักษาซอฟต์แวร์ และมาตรวัดเชิงวัตถุ

บทที่ 5 กล่าวถึง ดีไซน์แพตเทิร์นและการปรับปรุงโครงสร้างการออกแบบระบบการสร้างรายงานอัตโนมัติ และผลการทดลองจากการปรับปรุงการออกแบบโดยใช้อ็อบเซิร์ฟเวอร์แพตเทิร์น

บทที่ 6 กล่าวถึงบทสรุปของงานวิจัยและข้อเสนอแนะสำหรับงานวิจัยทางด้านการวัดการบำรุงรักษาซอฟต์แวร์

บทที่ 2
งานวิจัยทางด้านการออกแบบ และการวัดการบำรุงรักษาซอฟต์แวร์

ในบทนี้จะกล่าวถึง งานวิจัยทางด้านการออกแบบและการวัดการบำรุงรักษาซอฟต์แวร์ โดยหัวข้อ 2.1 จะกล่าวถึง ความสำคัญของการบำรุงรักษาซอฟต์แวร์ หัวข้อที่ 2.2 จะกล่าวถึงการวัดประสิทธิภาพของการบำรุงรักษาซอฟต์แวร์ด้วยมาตรวัดประเภทต่างๆ และหัวข้อที่ 2.3 กล่าวถึงงานวิจัยที่เกี่ยวข้อง ทั้งงานวิจัยทางด้านมาตรวัดที่ใช้สำหรับวัดการบำรุงรักษาซอฟต์แวร์ และงานวิจัยทางด้านการวิเคราะห์ความสัมพันธ์ของมาตรวัดและการบำรุงรักษาซอฟต์แวร์

2.1 ความสำคัญของการบำรุงรักษาซอฟต์แวร์

การบำรุงรักษาซอฟต์แวร์มีความสำคัญไม่น้อยกว่ากระบวนการพัฒนาซอฟต์แวร์ ซอฟต์แวร์ที่ดีจะต้องบำรุงรักษาได้ง่ายในอนาคต นั่นหมายถึงต้องมีความสามารถในการบำรุงรักษาซอฟต์แวร์

2.1.1 ความสามารถในการบำรุงรักษาซอฟต์แวร์

ความสามารถในการบำรุงรักษาซอฟต์แวร์ [7] หมายถึง "ระบบซอฟต์แวร์หรือส่วนประกอบของซอฟต์แวร์สามารถปรับเปลี่ยน แก้ไขข้อผิดพลาด ปรับปรุงประสิทธิภาพ หรือปรับเปลี่ยนคุณลักษณะให้เข้ากับสภาพแวดล้อมที่เปลี่ยนแปลงไปได้อย่างง่าย" โดยการบำรุงรักษาซอฟต์แวร์แบ่งออกเป็น 4 ประเภท [8] ได้แก่ การบำรุงรักษาเพื่อความถูกต้อง (Corrective Maintenance) การบำรุงรักษาเพื่อการปรับเปลี่ยน (Adaptive Maintenance) การบำรุงรักษาเพื่อความสมบูรณ์ (Perfective Maintenance) และการบำรุงรักษาเพื่อการป้องกัน (Preventive Maintenance) ซึ่งโดยทั่วไปค่าใช้จ่ายไม่น้อยกว่า 50% จะถูกนำไปใช้ในส่วนของการบำรุงรักษาซอฟต์แวร์ [9] ซึ่งงานวิจัยของ Schach และคณะ [10] ได้สำรวจเกี่ยวกับความพยายามในการบำรุงรักษาซอฟต์แวร์ พบว่า กว่า 18% ของความพยายามในการบำรุงรักษาซอฟต์แวร์จะใช้ในการบำรุงรักษาเพื่อการปรับเปลี่ยน (Adaptive Maintenance) ซึ่งเป็นอันดับสองรองจากการบำรุงรักษาเพื่อความสมบูรณ์

ซึ่งการบำรุงรักษาเพื่อการปรับเปลี่ยน [8] เป็นการบำรุงรักษาในการดัดแปลงขั้นตอนการทำงานบางส่วนของระบบตามความต้องการใช้งานที่เพิ่มขึ้นตามสถานการณ์ในการดำเนินงาน

ระบบการสร้างรายงานอัตโนมัติเพื่อสนับสนุนงานตรวจจับฝุ่นละอองในอุตสาหกรรมการผลิตฮาร์ดดิสก์ จะต้องสามารถรองรับการเพิ่มขยายหรือการบำรุงรักษาซอฟต์แวร์ในอนาคต ทั้งนี้การวัดการบำรุงรักษาซอฟต์แวร์ของกรณีศึกษาดังกล่าว จะแยกประเภทการบำรุงรักษาออกเป็นงาน

2 ลักษณะ ได้แก่ 1) การเพิ่มเติมความต้องการของผู้ใช้งาน และ 2) การบำรุงรักษาปรับเปลี่ยนโดยการจัดระเบียบซอร์ดโค้ด นอกจากนี้จะมีการวัดโครงสร้างหลังจากที่ทำการบำรุงรักษาซอฟต์แวร์ในแต่ละเวอร์ชั่นของการบำรุงรักษาซอฟต์แวร์ ด้วยมาตรวัดเชิงวัตถุเพื่อวิเคราะห์ถึงความสามารถในการบำรุงรักษาซอฟต์แวร์ ซึ่งสิ่งที่จะเป็นตัวชี้วัดด้านปัจจัยคุณภาพของซอฟต์แวร์ ได้แก่ มาตรวัดซอฟต์แวร์ อธิบายรายละเอียดในหัวข้อ 2.2

2.2 การวิเคราะห์คุณภาพของซอฟต์แวร์จากมาตรวัดซอฟต์แวร์

เนื่องจากระบบที่ใช้เป็นกรณีศึกษาเป็นระบบที่รองรับผู้ใช้งานที่หลากหลายและประเภทรายงานที่ต้องเพิ่มขึ้นในอนาคต ดังนั้น ความสามารถด้านการบำรุงรักษา (Maintainability) และความสามารถด้านการเพิ่มขยายระบบ (Scalability) จึงเป็นปัจจัยคุณภาพที่สำคัญ

2.2.1 คุณภาพของซอฟต์แวร์ (Software Quality)

คุณภาพของซอฟต์แวร์ จะขึ้นอยู่กับความต้องการของผู้ใช้งาน แยกประเภทของความต้องการออกเป็น ความต้องการทางด้านการดำเนินการ (Operational needs) และความต้องการทางด้านการบำรุงรักษา (Maintenance needs) [18] โดยทางด้านการทำงานอาจมีการพิจารณาในเรื่องของประสิทธิภาพของซอฟต์แวร์ (Performance) หรือหน้าที่การทำงาน (Functional) ด้านการบำรุงรักษาจะเกี่ยวข้องกับการปรับเปลี่ยน ปรับปรุงซอฟต์แวร์ (Modifying) ตามความต้องการของผู้ใช้งาน

โดยปัจจัยทางด้านคุณภาพ (Quality Factors) ที่มีความสัมพันธ์กับความต้องการของผู้ใช้งานในด้านของการบำรุงรักษา แสดงดังตารางที่ 2.1

ตารางที่ 2.1 ความสัมพันธ์ระหว่างปัจจัยคุณภาพและความต้องการของผู้ใช้งาน [18]

ความต้องการของผู้ใช้ (User's Need)	ความกังวลของผู้ใช้ (User's Concern)	ปัจจัยทางด้านคุณภาพ (Quality Factor)
การเปลี่ยนแปลง (Change)	- ง่ายต่อการซ่อมแซม/ปรับปรุงหรือไม่? - ง่ายต่อการเพิ่มขยายหรือไม่? - ง่ายต่อการเปลี่ยนแปลงหรือไม่? - ง่ายต่อการย้ายระบบหรือติดตั้งใหม่หรือไม่? - สามารถนำมาใช้ใหม่กับระบบอื่นได้หรือไม่?	Maintainability Scalability Flexibility Portability Reusability
การจัดการ (Management)	- สามารถตรวจสอบประสิทธิภาพได้ง่ายหรือไม่? - สามารถจัดการซอฟต์แวร์ได้ง่ายหรือไม่	Verifiability Manageability

จากตารางที่ 2.1 แสดงให้เห็นปัจจัยคุณภาพที่จะตอบสนองต่อความต้องการของผู้ใช้งาน โดยวิทยานิพนธ์ฉบับนี้จะพิจารณาปัจจัยคุณภาพทางด้าน ความสามารถในการบำรุงรักษา (Maintainability) และ ความสามารถด้านการเพิ่มขยายระบบ (Scalability) ซึ่งจะใช้มาตรวัดซอฟต์แวร์เป็นตัววัดปัจจัยคุณภาพดังกล่าว

2.2.2 มาตรวัดซอฟต์แวร์ (Software Metrics)

มาตรวัดซอฟต์แวร์ (Software Metrics) เป็นการกำหนดตัวเลขเชิงปริมาณให้กับผลิตภัณฑ์ซอฟต์แวร์ (Software Product) หรือกระบวนการในการพัฒนาผลิตภัณฑ์ซอฟต์แวร์ ซึ่งตัวชี้วัดอาจจะวัดได้โดยตรงหรือไม่ก็ได้ หรืออาจวัดได้จากการคำนวณค่าอื่น เช่น จำนวนบรรทัดของซอฟต์แวร์ (Source Lines of Code: SLOC) เป็นตัวชี้วัดขนาดของซอฟต์แวร์ (Software size) ซึ่งจะเรียก SLOC นี้ว่า ตัวชี้วัด เป็นต้น [19]

โดยทั่วไปมาตรวัดซอฟต์แวร์แบ่งออกเป็น 2 ประเภท [20] ได้แก่ มาตรวัดซอฟต์แวร์แบบดั้งเดิม (Traditional) และมาตรวัดซอฟต์แวร์เชิงวัตถุ (Object Oriented) ดังนั้นมาตรวัดที่จะใช้วัดประสิทธิภาพของซอฟต์แวร์ในขั้นตอนของการบำรุงรักษาซอฟต์แวร์ในงานวิจัยนี้ ประกอบด้วย

2.2.2.1 มาตรวัดขนาดการบำรุงรักษาซอฟต์แวร์ (Software Maintenance Size)

2.2.2.2 มาตรวัดความพยายามในการบำรุงรักษาซอฟต์แวร์ (Software Maintenance Effort)

2.2.2.3 มาตรวัดเชิงวัตถุเพื่อใช้วัดโครงสร้างของระบบ

ขนาดของซอฟต์แวร์ (Software Size) เป็นมาตรวัดเบื้องต้นและมาตรวัดที่สำคัญสำหรับการวัดซอฟต์แวร์ การวัดขนาดของซอฟต์แวร์สามารถนำไปใช้ในการประมาณการค่าใช้จ่ายสำหรับการพัฒนาซอฟต์แวร์ หรือการสร้างแบบจำลองของการประมาณการซอฟต์แวร์ เช่น แบบจำลองโคโคโม่ (COCOMO) และโคโคโม่เวอร์ชั่นสอง (COCOMO II) [5][21] โดยการวัดขนาดของซอฟต์แวร์สามารถวัดได้หลายวิธี เช่น จำนวนบรรทัดของซอร์ดโค้ด (LOC) หรือจากฟังก์ชั่นพอยต์ (Function Point) ซึ่งเป็นการวัดแบบดั้งเดิม (Traditional Metrics) และการวัดจากจำนวนเมธอด จำนวนคลาส หรือจำนวนเมธอดต่อคลาส ซึ่งเป็นการวัดเชิงวัตถุ (Object-Oriented Metrics)

หากพิจารณาขนาดของซอฟต์แวร์ในส่วนของการบำรุงรักษาซอฟต์แวร์ ปัจจุบันการวัดขนาดของการบำรุงรักษาซอฟต์แวร์ (Software Maintenance Size) จะพิจารณาจากจำนวนบรรทัดที่เปลี่ยนแปลงไป [4] จากระบบเดิมเท่านั้น วิทยานิพนธ์ฉบับนี้จึงนำเสนอมาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์จำนวน 4 มาตรวัด ได้แก่ การพิจารณาจากจำนวนคลาส การพิจารณาจากจำนวนเมธอด การพิจารณาจากจำนวนเมธอดต่อคลาส และการพิจารณาจากจำนวนเมธอดต่อคลาสโดยคิดถ่วงน้ำหนัก จากจำนวนคลาส/เมธอด ที่เปลี่ยนแปลงไปจากเวอร์ชั่นก่อนการบำรุงรักษาซอฟต์แวร์ ซึ่งการนำเสนอมาตรวัดใหม่นี้ จะทำให้มีมาตรวัดทางด้านขนาดของซอฟต์แวร์ใน

กระบวนการบำรุงรักษาซอฟต์แวร์ที่เหมาะสำหรับการเขียนโปรแกรมเชิงวัตถุ ซึ่งจะเป็นพื้นฐานสำหรับงานวิจัยในกระบวนการบำรุงรักษาซอฟต์แวร์ในส่วนอื่นต่อไป

จากการศึกษางานวิจัยในส่วนของการบำรุงรักษาซอฟต์แวร์ งานวิจัยหลายงานวิจัย [6][19][22] ได้ทำการวิเคราะห์ความสัมพันธ์ระหว่าง ความพยายามในการบำรุงรักษาซอฟต์แวร์ (Maintenance Effort) และจำนวนบรรทัดของซอฟต์แวร์ที่เปลี่ยนแปลงไปจากระบบเดิม พบว่าปัจจัยทั้งสองมีความสัมพันธ์กัน ดังนั้นการพิจารณาถึงความเหมาะสมหรือประสิทธิภาพของมาตรวัดที่นำเสนอ จำนวน 4 มาตวัด และมาตรวัดขนาดของการบำรุงรักษาโดยพิจารณาจากจำนวนบรรทัดของซอฟต์แวร์ที่เปลี่ยนแปลงไป (มาตรวัดเดิม) จะพิจารณาจากการคำนวณสัดส่วนระหว่างอัตราการเปลี่ยนแปลงของคลาสหรือเมธอด ในแต่ละมาตรวัดที่กำลังพิจารณา (ขนาดของการบำรุงรักษาซอฟต์แวร์) และอัตราส่วนระหว่างเวลาที่ใช้ในการพัฒนาเมื่อเปรียบเทียบกับเวลาที่ประมาณการในการบำรุงรักษาซอฟต์แวร์

โดยรายละเอียดของมาตรวัดซอฟต์แวร์แต่ละประเภท ได้แก่ มาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์ มาตรวัดความพยายามในการบำรุงรักษาซอฟต์แวร์ และมาตรวัดเชิงวัตถุที่ใช้สำหรับวัดโครงสร้างของระบบที่ใช้ในวิทยานิพนธ์จะกล่าวรายละเอียดและตัวอย่างในบทที่ 3

2.3 งานวิจัยที่เกี่ยวข้อง

เนื่องจากกระบวนการบำรุงรักษาซอฟต์แวร์เป็นขั้นตอนที่สำคัญหลังจากเสร็จสิ้นขั้นตอนของการพัฒนาระบบ และเพื่อให้ทราบถึงความสามารถของการบำรุงรักษาซอฟต์แวร์จึงต้องใช้มาตรวัดเพื่อตรวจสอบความสามารถในการบำรุงรักษา (Maintainability) โดยงานวิจัยที่เกี่ยวข้องจะแบ่งออกเป็น 2 ด้าน ได้แก่ หัวข้อ 2.3.1 งานวิจัยทางด้านมาตรวัดที่ใช้สำหรับการบำรุงรักษาซอฟต์แวร์ และ หัวข้อ 2.3.2 งานวิจัยทางด้านการวิเคราะห์ความสัมพันธ์ของมาตรวัดและการบำรุงรักษาซอฟต์แวร์ แสดงรายละเอียด ดังนี้

2.3.1 งานวิจัยทางด้านมาตรวัดที่ใช้สำหรับการบำรุงรักษาซอฟต์แวร์

Masuda, Sakamoto และ Ushijima [23] ได้ทำการปรับปรุงการออกแบบจากซอฟต์แวร์ที่มีอยู่เดิมโดยใช้ดีไซน์แพตเทิร์น (Redesigning of an Existing Software using Design Patterns) ของระบบการเรียนรู้ต้นไม้ตัดสินใจ (Decision Tree Learning System) โดยระบบดังกล่าวจะจำแนกประเภท (Classifier) จากการแทนความรู้ที่อยู่ในรูปของต้นไม้ ระบบที่ดีจะต้องมีความยืดหยุ่น (Flexibility) และปรับขยายได้ (Extensibility) ดังนั้นจึงได้นำเสนอแนวทางการเพิ่มประสิทธิภาพในแต่ละส่วนของระบบที่คาดว่าจะมีการปรับขยายในอนาคตด้วยดีไซน์แพตเทิร์นชนิดต่างๆ โดยแต่ละส่วนย่อยของระบบจะเรียกว่า hot-spot โดยแต่ละ hot-spot จะแบ่งออกเป็นระบบย่อย และใช้ดีไซน์แพตเทิร์นชนิดต่างๆ ดังนี้

2.3.1.1 การจัดการการสร้างชุดข้อมูล (Data set creation handling) ระบบจะต้องรองรับการสร้างชุดข้อมูลที่ประกอบด้วยข้อมูลจากหลายแหล่งข้อมูล เช่น ข้อมูลจากฐานข้อมูลเชิงสัมพันธ์ (Relational Databases) และข้อมูลที่อยู่ในรูปแบบของเท็กซ์ไฟล์ เป็นต้น

2.3.1.2 การจัดการประเภทของแอตทริบิวต์ (Attribute type handling) ระบบจะต้องรองรับการจัดการประเภทของแอตทริบิวต์นอกเหนือจาก ข้อมูลแบบไม่ต่อเนื่อง (discrete attribute) โดยข้อมูลแบบไม่ต่อเนื่อง ประกอบด้วยข้อมูลที่นับได้ สามารถจัดกลุ่มหรือหมวดได้ แทนด้วยเลขจำนวนเต็ม เช่น ข้อมูลแบบไบนารีที่มีค่าได้เพียง 0 และ 1 ชายหรือหญิง เป็นต้น

2.3.1.3 การจัดการโครงสร้างการตัดสินใจ (Decision tree structure handling) ที่มีลักษณะโครงสร้างเป็นลำดับชั้น

2.3.1.4 การจัดการกระบวนการทดสอบ (Test method handling) ระบบจะต้องสนับสนุนประเภทการทดสอบที่จะใช้กับแต่ละประเภทของแอตทริบิวต์

2.3.1.5 การจัดการการเลือกวิธีการทดสอบ (Test selection method handling)

2.3.1.6 การจัดการข้อมูล Noise (Noise data handling method handling)

2.3.1.7 การจัดการกระบวนการ Pruning (Decision tree pruning method handling) ระบบจะต้องมีกระบวนการการจัดการปรับแต่งข้อมูลที่ได้จาก decision tree ที่หลากหลาย โดยการเปลี่ยนแปลงจะต้องไม่มีผลกระทบต่อส่วนอื่นของระบบ

2.3.1.8 การจัดการการประเมินผล (Decision tree evaluation method handling)

จากนั้นจะทำการเลือกดีไซน์แพตเทิร์นที่เหมาะสมในแต่ละ hot-spot โดยแพตเทิร์นที่เลือกใช้ในแต่ละส่วน แสดงดังตารางที่ 2.2

ตารางที่ 2.2 ดีไซน์แพตเทิร์นที่เลือกใช้ในแต่ละส่วน [23]

Hot-spot	**ดีไซน์แพตเทิร์น**
(1) การจัดการการสร้างชุดข้อมูล	Builder, Template Method, Strategy
(2) การจัดการประเภทของแอตทริบิวต์	Factory Method, Abstract Factory
(3) การจัดการโครงสร้างต้นไม้การตัดสินใจ	Composite, Visitor, Abstract Factory
(4) การจัดการกระบวนการทดสอบ	Command
(5) การจัดการการเลือกวิธีทดสอบ	Template Method
(6) การจัดการกระบวนการจัดการข้อมูล Noise	Strategy, Abstract Factory
(7) การจัดการกระบวนการ Pruning	Visitor, Template Method
(8) การจัดการการประเมินผล	Strategy

จากนั้นจะประเมินประสิทธิภาพการประยุกต์ใช้ดีไซน์แพตเทิร์นเชิงปริมาณในการปรับการออกแบบด้วยดีไซน์แพตเทิร์น โดยใช้มาตรวัดซีเค (C&K Metrics) ในการประเมินผล ซึ่งมาตร

วัดซีเค [25] เป็นมาตรวัดเชิงวัตถุที่ใช้สำหรับวัดโครงสร้างและความซับซ้อนของซอฟต์แวร์ ค่าที่ได้จากมาตรวัดสามารถบ่งบอกปัจจัยคุณภาพต่างๆ ได้ เช่น ความยืดหยุ่นของระบบ การปรับขยายระบบ เป็นต้น โดยการเก็บข้อมูลจากมาตรวัดซีเคจะแบ่งออกเป็น 2 เวอร์ชั่น ได้แก่

Release1: เวอร์ชั่นที่ออกแบบโดยไม่ใช้ดีไซน์แพตเทิร์น และ

Release2: เวอร์ชั่นที่ปรับปรุงการออกแบบโดยใช้ดีไซน์แพตเทิร์น

ผลจากการจัดเก็บข้อมูลของทั้ง 2 เวอร์ชั่น ประกอบด้วย ขนาด ได้แก่ จำนวนบรรทัด (Lines) จำนวนเมธอด (Method) และจำนวนคลาส (Classes) ซึ่งขนาดของจำนวนคลาส เมธอด และจำนวนบรรทัดของซอฟต์แวร์ เป็นการนับจำนวน ณ เวอร์ชั่นของการบำรุงรักษาซอฟต์แวร์เวอร์ชั่นนั้นๆ เปรียบเสมือนการนับขนาดของซอฟต์แวร์ที่ไม่ได้คำนึงถึงอัตราการเปลี่ยนไปของคลาสและเมธอด ในส่วนของการบำรุงรักษาซอฟต์แวร์ แสดงดังตารางที่ 2.3

ตารางที่ 2.3 ขนาดของระบบการเรียนรู้ต้นไม้ตัดสินใจทั้ง 2 เวอร์ชั่น [23]

	Lines	Classes	Methods
Release1	2627	25	291
Release2	4992	89	871

เมื่อวัดขนาดของซอฟต์แวร์ ของระบบการเรียนรู้ต้นไม้ตัดสินใจ (Decision Tree Learning System) ใน Release ที่ 1 และหลังจากการปรับปรุงการออกแบบโดยการใช้ดีไซน์แพตเทิร์นใน Release ที่ 2 พบว่า การปรับโครงสร้างของการออกแบบโดยใช้ดีไซน์แพตเทิร์น จำนวนคลาสและเมธอดจะเพิ่มขึ้น หมายถึง ขนาดของซอฟต์แวร์จะมีขนาดใหญ่กว่าระบบเดิม (การออกแบบที่ไม่ใช้ดีไซน์แพตเทิร์น) ในขณะที่การพิจารณาโครงสร้างและความซับซ้อนของซอฟต์แวร์ที่ได้จากมาตรวัดเชิงวัตถุ ได้แก่ มาตรวัดซีเค แสดงดังตารางที่ 2.4

ตารางที่ 2.4 ผลลัพธ์การวัดค่าจากมาตรวัดซีเค [23]

มาตรวัด	Release1			Release2		
	Min	Median	Max	Min	Median	Max
จำนวนเมธอดต่อคลาส (WMC)	0.56	9.53	30.56	0	3.75	44.59
ระดับความลึกของการสืบทอดคุณสมบัติของคลาส (DIT)	1	1	5	1	2	5
จำนวนคลาสลูก (NOC)	0	0	3	0	0	5
ความเข้าคู่กันระหว่างวัตถุ (CBO)	0	1	8	0	2	20
การตอบสนองของคลาส (RFC)	2	28	83	0	19	143
ระดับการขาดความสัมพันธ์ภายในคลาส (LCOM)	0	21	187	0	1	1469

จากตารางที่ 2.4 แสดงผลลัพธ์จากมาตรวัดซีเคทั้ง 6 มาตรวัดจะเห็นว่าการออกแบบใหม่โดยใช้ดีไซน์แพตเทิร์นที่เหมาะสมสามารถเพิ่มประสิทธิภาพด้านการปรับขยาย (Extensibility) และความยืดหยุ่น (Flexibility) ของซอฟต์แวร์เห็นได้จากค่า WMC หรือมาตรวัดจำนวนเมธอดต่อคลาสใน Release2 ที่มีการปรับการออกแบบมีค่าต่ำกว่า นั่นหมายถึงมีความซับซ้อนน้อยกว่า เช่นเดียวกับค่า RFC หรือมาตรวัดการตอบสนองต่อคลาส และค่า LCOM หรือมาตรวัดระดับการขาดความสัมพันธ์ภายในคลาส มีค่าต่ำกว่าเช่นกัน จากผลการทดลองดังกล่าวแสดงให้เห็นว่า การออกแบบโดยใช้ดีไซน์แพตเทิร์นทำให้ซอฟต์แวร์มีความยืดหยุ่นและปรับขยายได้ง่ายกว่าการออกแบบที่ไม่ได้ใช้ดีไซน์แพตเทิร์น กรณีศึกษาระบบการเรียนรู้ต้นไม้ตัดสินใจ

สำหรับงานวิทยานิพนธ์ฉบับนี้จะวัดการปรับปรุงการออกแบบโดยการเปรียบเทียบประสิทธิภาพการออกแบบของระบบเช่นเดียวกับงานวิจัยข้างต้น โดยจะเปรียบเทียบกับระบบที่ออกแบบโดยไม่ใช้ดีไซน์แพตเทิร์นในส่วนของโครงสร้างและความซับซ้อนของซอฟต์แวร์ โดยใช้มาตวัดจำนวน 6 มาตรวัด ได้แก่ WMC, DIT, NOC, CBO, RFC และ LCOM เช่นเดียวกับงานวิจัยข้างต้น สำหรับกรณีศึกษา ระบบการสร้างรายงานอัตโนมัติ

2.3.2 งานวิจัยด้านการวิเคราะห์ความสัมพันธ์ของมาตรวัดและการบำรุงรักษาซอฟต์แวร์

การวิเคราะห์ความสัมพันธ์ของมาตรวัดและการบำรุงรักษาซอฟต์แวร์เป็นสิ่งสำคัญ เนื่องจาก จะทำให้สามารถเลือกมาตรวัดที่เหมาะสมเพื่อใช้สำหรับการวัดการบำรุงรักษาซอฟต์แวร์ โดยงานวิจัยทางด้านการวิเคราะห์ความสัมพันธ์ระหว่างมาตรวัด และ การบำรุงรักษาซอฟต์แวร์ประกอบด้วย

2.3.2.1 Sarah [11] ได้ทำการศึกษาและสำรวจมาตรวัดเชิงวัตถุที่สำคัญที่มีผลกระทบหรือความสัมพันธ์กับปัจจัยคุณภาพการออกแบบ (Quality Attributes) ในงานวิจัยเรื่อง “Object Oriented Design Metrics and Tools A Survey” โดยระบุว่า

Quality Attributes หรือ ปัจจัยคุณภาพการออกแบบ เช่น ความน่าเชื่อถือ (reliability) ความมีประสิทธิภาพ (efficiency) การใช้งานง่าย (usability) ความสามารถด้านการบำรุงรักษา (Maintainability) เป็นต้น

Measurable Attributes: จะเกี่ยวกับการวัด เช่น การวัด Coupling Cohesion การวัดความความซับซ้อน (Complexity) และขนาด (Sizing) ซึ่งการวัดเหล่านี้จะมีผลต่อ ปัจจัยคุณภาพการออกแบบ จากนั้นจะวิเคราะห์ความสัมพันธ์ระหว่างตัวแปรอิสระ ว่ามีความสัมพันธ์กันหรือไม่

ตัวแปรอิสระ (Independent variables) ได้แก่ มาตรวัดที่สามารถวัดคุณลักษณะของสิ่งที่สนใจ ประกอบด้วย ขนาด (Size) ความสัมพันธ์ระหว่างคลาส (Coupling) ความเป็นเนื้อเดียวกัน (Cohesion) การสืบทอดคุณสมบัติ (Inheritance) และความซับซ้อน (Complexity)

ตัวแปรตาม (Dependent variables) ได้แก่ คุณลักษณะของการออกแบบ ประกอบด้วย ความยืดหยุ่น (Flexibility) ความน่าเชื่อถือ (Reliability) ความเข้าใจง่าย (Understandability) ความสามารถในการบำรุงรักษา (Maintainability) และแนวโน้มการเกิดความผิดพลาด (Fault-Proneness) เป็นต้น

โดยผลการศึกษามาตรวัดที่มีผลกระทบหรือความสัมพันธ์ต่อคุณภาพของการออกแบบ ผู้วิจัยได้เลือกมาตรวัดบางมาตรวัดที่มีความสำคัญโดยสามารถสรุปความสัมพันธ์ระหว่างมาตรวัดเชิงวัตถุและคุณภาพการออกแบบ แสดงดังตารางที่ 2.5

ตารางที่ 2.5 ตารางสรุปความสัมพันธ์ระหว่างมาตวัดเชิงวัตถุและคุณภาพการออกแบบ [11]

Quality Attributes / Measurable Attributes	ความยืดหยุ่น Flexibility	การบำรุงรักษา Maintainability	ความเข้าใจง่าย Understandability	นำกลับมาใช้ใหม่ Reusability	แนวโน้มความผิดพลาด Fault-Proneness
ขนาด (Sizing)					✓
ความสัมพันธ์ระหว่างคลาส (Coupling)		✓	✓	✓	✓
ความเป็นเนื้อเดียวกัน (Cohesion)		✓	✓	✓	
การสืบทอด (Inheritance)	✓		✓	✓	✓
ความซับซ้อน (Complexity)	✓		✓	✓	✓

จากตารางที่ 2.5 แสดงความสัมพันธ์ระหว่างมาตรวัดเชิงวัตถุ (ตัวแปรอิสระ) และคุณภาพการออกแบบ (ตัวแปรตาม) พบว่าแต่ละมาตรวัดจะมีความสัมพันธ์กับคุณภาพการออกแบบ

ขนาด (Sizing) มีผลกระทบต่อแนวโน้มการเกิดข้อผิดพลาด (Fault-Proneness) โดยแนวโน้มความผิดพลาดจะมากขึ้นตามขนาดที่ใหญ่ขึ้น (เช่น มีขนาดคลาสหรือขนาดของแพคเกจที่ใหญ่) ดังนั้นระบบที่มีขนาดใหญ่อาจเกิดจากการออกแบบที่ไม่ดี

มาตรวัด Coupling มีส่วนสำคัญในการเกิดแนวโน้มของความผิดพลาด เช่นเดียวกับมาตรวัด Cohesion ดังนั้น Coupling และ Cohesion จะมีผลกระทบต่อ ความสามารถในการบำรุงรักษาอย่างมาก ดังนั้นการที่ระหว่างคลาสมีความสัมพันธ์กันอย่างหลวมๆ จะช่วยในเรื่องของความสามารถในการเข้าใจและการนำกลับมาใช้ใหม่ได้มากขึ้น ดังนั้นการออกแบบที่ดีควรมีความสัมพันธ์กันต่ำ (Loose Coupling)

ค่าที่ได้จากมาตรวัดการสืบทอดคุณสมบัติ (Inheritance) และมาตรวัดความซับซ้อน (Complexity) จะบ่งบอกคุณภาพถึงคุณภาพการออกแบบทั้งหมด ยกเว้น การบำรุงรักษา

จากงานวิจัยดังกล่าว สรุปได้ว่า ค่าที่ได้จากการวัด coupling และ cohesion สามารถบอกถึงความสามารถด้านการบำรุงรักษา (Maintainability) ได้ ดังนั้นวิทยานิพนธ์ฉบับนี้จะ

เลือกใช้มาตรวัด Coupling และ Cohesion เพื่อเปรียบเทียบประสิทธิภาพการออกแบบในด้านการบำรุงรักษาระบบ แต่จากผลการศึกษาข้างต้นแม้ว่า การวัดขนาด (Sizing) จะไม่มีผลโดยตรงกับความสามารถด้านการบำรุงรักษา แต่ซอฟต์แวร์ที่มีขนาดใหญ่ (จำนวนคลาสหรือแพคเกจมีมาก) หมายถึง จะมีแนวโน้มความซับซ้อนมากซึ่งจะมีผลทางด้านการบำรุงรักษา และการเพิ่มขยายหรือปรับปรุงระบบในอนาคตลดลง ดังนั้นวิทยานิพนธ์ฉบับนี้จะเลือกการวัดขนาดด้วย นอกเหนือจากการวัด coupling และ cohesion แต่ขนาดที่เลือกเป็นมาตรวัดจะใช้ขนาดของการบำรุงรักษาซอฟต์แวร์ ที่พิจารณาถึงสัดส่วนของการเปลี่ยนแปลงของ จำนวนบรรทัด คลาส และเมธอด

2.3.2.2 Yang [4] ได้ทำการศึกษาและวิเคราะห์ความสัมพันธ์ระหว่างความพยายามในแต่ละกิจกรรมของการบำรุงรักษาและความสามารถในการบำรุงรักษา ในงานวิจัยเรื่อง "An Empirical Analysis on Distribution Patterns of Software Maintenance Effort" โดยได้เก็บข้อมูลเกี่ยวกับความพยายามที่ใช้ในแต่ละกระบวนการสำหรับการพัฒนาซอฟต์แวร์ เป็นพื้นฐานสำคัญสำหรับการวางแผนโครงการพัฒนา โดยงานวิจัยส่วนใหญ่จะมุ่งเน้นในการประมาณการ (Estimate) และความสามารถในการวางแผนสำหรับกิจกรรมกระบวนการพัฒนาซอฟต์แวร์ ซึ่งมีงานวิจัยน้อยมากที่เน้นเกี่ยวกับ การวิเคราะห์ความพยายามในแต่ละขั้นตอนหรือกิจกรรมในส่วนของการบำรุงรักษาซอฟต์แวร์

Yang ได้ทำการศึกษาและวิเคราะห์รูปแบบการวัดความพยายามในขั้นตอนของการบำรุงรักษาซอฟต์แวร์ โดยแบ่งการวัดความพยายามในแต่ละกิจกรรม ได้แก่ ความพยายามในการจัดทำข้อกำหนดซอฟต์แวร์ (Requirement) ความพยายามในการออกแบบ (Design) ความพยายามในการเขียนโปรแกรม (Coding) และความพยายามในการทดสอบ (Testing)

โดยได้ทำการวัดความพยายาม (Effort) จากการเก็บข้อมูลจริงจากการบำรุงรักษาซอฟต์แวร์ทั้ง 8 โครงการ ที่เป็นโครงการขนาดกลางเกี่ยวกับ Software Quality Management Platform (QMP) Product ในประเทศจีน โดย QMP เวอร์ชั่น1 (V1) เป็น QMP ที่พัฒนาขึ้นในรุ่นแรก จากนั้น QMP เวอร์ชั่น2 ถึง เวอร์ชั่น9 เป็นรุ่นที่พัฒนาต่อจากรุ่นแรก (นับเป็นเวอร์ชั่นที่บำรุงรักษาเพิ่มเติม) โดยการแบ่งกิจกรรมจะแบ่งตามทฤษฎีของแบบจำลองน้ำตก (Water Fall Model) แสดงมาตรวัดดังตารางที่ 2.6

ตารางที่ 2.6 มาตรวัดที่ใช้สำหรับวัดกระบวนการและผลผลิตของซอฟต์แวร์ [4]

ตัวชี้วัด (Metrics)	หน่วย (Unit)	คำอธิบาย (Description)
ขนาด (Size)	KSLOC	จำนวนบรรทัดทั้งหมดของซอฟต์แวร์
ขนาดของการบำรุงรักษา (Maintenance Size: MS)	KSLOC	จำนวนบรรทัดของซอร์ดโค้ดที่เปลี่ยนแปลงและเพิ่มขึ้นในระหว่างที่ทำการปรับเปลี่ยนหรือบำรุงรักษาระบบ
ความพยายาม (Effort)	Person-Hour	ความพยายามที่ใช้ทั้งหมดในการทำงาน
ข้อบกพร่อง (Defect)	Number	จำนวนข้อบกพร่องที่พบหรือตรวจสอบได้
ความพยายามการศึกษาความต้องการ การออกแบบ การเขียนโปรแกรม และการทดสอบ (E_Req ,E_Des ,E_Cod ,E_Tes)	Person-Hour	ความพยายามที่ใช้ในขั้นตอนของการวิเคราะห์ความต้องการ การออกแบบระบบ การเขียนโปรแกรม และการทดสอบซอฟต์แวร์
ข้อบกพร่องในการวิเคราะห์ความต้องการ การออกแบบ การเขียนโปรแกรมและการทดสอบ (Def_Req ,Def_Des ,Def_Cod ,Def_Tes)	Number	จำนวนข้อบกพร่องในระหว่างการวิเคราะห์ความต้องการ การออกแบบระบบ การเขียนโปรแกรม และการทดสอบซอฟต์แวร์

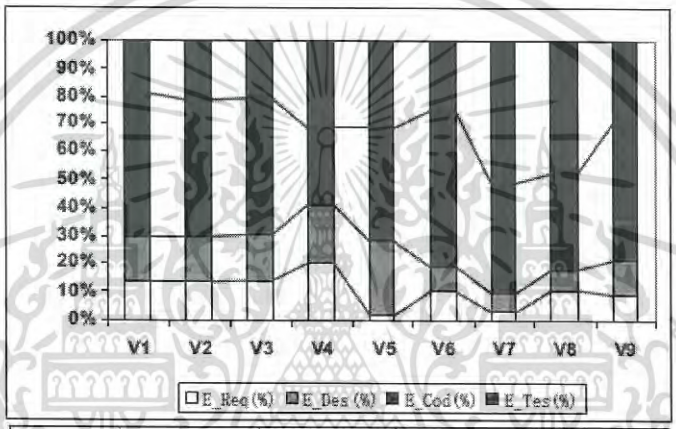
โดยผลจากการวัดขนาดของซอฟต์แวร์ในเวอร์ชั่น1 และ ขนาดของการบำรุงรักษาในทั้ง 8 เวอร์ชั่น ของแต่ละขั้นตอนการพัฒนาทั้งในด้านของความพยายามและข้อบกพร่องของแสดงดังตารางที่ 2.7

ตารางที่ 2.7 ขนาดของซอฟต์แวร์ (Size) ความพยายาม (Effort) และ ข้อบกพร่อง (Defect) ในแต่ละกิจกรรมในขั้นตอนของการบำรุงรักษาซอฟต์แวร์ [4]

Ver.	Size	Eff_Req	Eff_Des	Eff_Cod	Eff_Tes	Def_Req	Def_Des	Def_Cod
1	69.1	953	1185	3694	1363	NA	NA	NA
2	64	1240	1480	4400	1960	18	36	99
3	19	923	1108	3231	1385	19	41	128
4	21.2	1641	1728	2236	2587	365	165	478
5	18	10	158	240.5	187	56	44	114
6	85	516	451	2784	1271	NA	NA	NA
7	207	804	2086	12003	15466	166	325	2113
8	65	1760	1113	5555	7788	261	12	651
9	8	109	154	590	365.5	61	43	81

จากตารางที่ 2.7 แสดงข้อมูลเบื้องต้น ของขนาดของซอฟต์แวร์ ความพยายามและข้อบกพร่องในแต่ละกิจกรรมการบำรุงรักษาของแต่ละเวอร์ชั่น โดยหากพิจารณาการบำรุงรักษาในเวอร์ชั่นที่ 7 และเวอร์ชั่นที่ 9 จะเห็นว่าขนาดของการบำรุงรักษามีความสัมพันธ์ในลักษณะแปรผันตรงกับ ความพยายาม (Effort) แลข้อบกพร่อง (Defect) ในแต่ละขั้นตอนของการบำรุงรักษา

ซึ่งหากพิจารณาลักษณะของความพยายาม (Effort) ที่ใช้ในแต่ละกิจกรรมการบำรุงรักษา ได้แสดงความพยายามในแต่ละขั้นตอนของการบำรุงรักษา ได้แก่ ความพยายามในการกำหนดความต้องการ (Requirement Effort) ความพยายามในการออกแบบ (Design Effort) ความพยายามในการเขียนโปรแกรม (Coding Effort) และความพยายามในการทดสอบ (Testing Effort) ในการพัฒนาในเวอร์ชั่นแรก และการบำรุงรักษาทั้ง 8 เวอร์ชั่น แสดงดังรูปที่ 2.1



	E_Req(%)	E_Des(%)	E_Cod(%)	E_Tes(%)
Mean	10.2%	14.5%	42.8%	32.5%
StDev	6%	7%	9.2%	11.3%
Median	10.6%	14.5%	44.4%	30.7%

รูปที่ 2.1 ความพยายามของแต่ละกิจกรรมในการพัฒนาซอฟต์แวร์ทั้ง 9 เวอร์ชั่น [4]

จากรูปที่ 2.1 จะสังเกตได้ว่าในทุกเวอร์ชั่น ความพยายามด้านการเขียนโปรแกรม (E_Cod) มีค่า Effort มากที่สุด ซึ่งมีค่าเฉลี่ยถึง 42.8% ของความพยายามทั้งหมด โดยความพยายามที่ใช้สำหรับการทดสอบและการออกแบบ มีค่ารองลงมาตามลำดับ โดยที่ความพยายามในการกำหนดความต้องการ (E_Req) มีค่าน้อยที่สุดในขั้นตอนของการบำรุงรักษาซอฟต์แวร์เมื่อเทียบกับความพยายามทั้งหมด จากนั้นจึงได้ทำการวิเคราห์ความสัมพันธ์ระหว่างขนาดของการบำรุงรักษา (Maintenance Size) และความพยายาม (Effort) ซึ่งหากพิจารณาความสัมพันธ์ของขนาดของการบำรุงรักษาและความพยายามของแต่ละกิจกรรม แสดงความสัมพันธ์ได้ดังตารางที่ 2.8

ตารางที่ 2.8 ความสัมพันธ์ระหว่างขนาดของการบำรุงรักษาและความพยายามในการบำรุงรักษา [4]

	E_Req	E_Des	E_Cod	E_Tes
Maintenance Size	0.10	0.59	0.93	0.89

จากการศึกษางานวิจัยดังกล่าว สรุปได้ว่า ขนาดของการบำรุงรักษามีความสัมพันธ์กับความพยายามในแต่ละขั้นตอนของการบำรุงรักษา โดยเฉพาะในส่วนของความพยายามในการเขียนโปรแกรมนั่นหมายถึง หากขนาดของการบำรุงรักษาซอฟต์แวร์มีขนาดใหญ่ความพยายามในการเขียนโปรแกรมจะมากขึ้นตามไปด้วยทำให้ ความสามารถในการบำรุงรักษาลดลง ดังนั้นวิทยานิพนธ์ฉบับนี้จะ เลือกใช้มาตรวัดเกี่ยวกับ ความพยายามในแต่ละกิจกรรมของการบำรุงรักษาเพื่อใช้เปรียบเทียบประสิทธิภาพจากการออกแบบทั้งสองรูปแบบ

2.3.2.3 Wei Li และ Sallie Henry [20] ได้ทำการตรวจสอบความสัมพันธ์ระหว่างมาตรวัดเชิงวัตถุและความพยายามในการบำรุงรักษาซอฟต์แวร์ โดยประกอบด้วยมาตรวัดซีเค ได้แก่ DIT, NOC, MPC, RFC, LCOM, DAC, WMC, NOM, Size1 and Size2 โดย Size1 คือขนาดของจำนวนบรรทัดของซอฟต์แวร์ (Line of code) และ Size2 หมายถึง จำนวนแอตทริบิวต์ และจำนวนเมธอดที่มีภายในคลาส ซึ่งข้อมูลของความพยายามในการบำรุงรักษาซอฟต์แวร์จะใช้ข้อมูลของระบบซอฟต์แวร์เชิงพาณิชย์จำนวน 2 ระบบ โดยให้ความพยายามในการบำรุงรักษาซอฟต์แวร์หมายถึง การเปลี่ยนแปลงจำนวนของบรรทัดของซอฟต์แวร์ในการบำรุงรักษาซอฟต์แวร์ จากการวิเคราะห์มาตรวัดเชิงวัตถุดังกล่าวพบว่ามาตรวัดเชิงวัตถุ (Object Oriented Metrics) สามารถใช้ในการคาดการณ์ความพยายามในการบำรุงรักษาซอฟต์แวร์ (Maintenance Effort) ได้

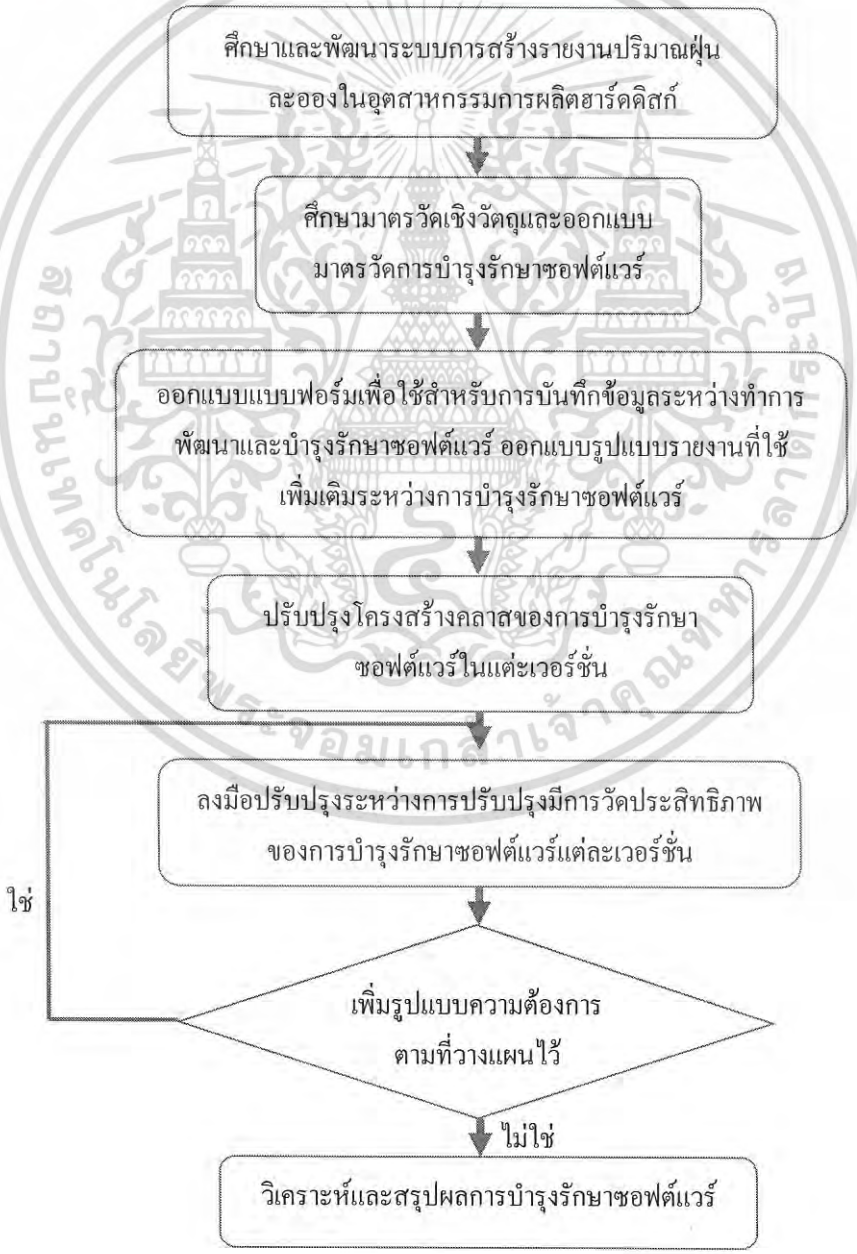
จากงานวิจัยที่กล่าวมาข้างต้นสามารถสรุปและเปรียบเทียบแนวคิดเพื่อปรับปรุงงานวิจัยสำหรับการวัดการบำรุงรักษาซอฟต์แวร์ ที่นำเสนอในวิทยานิพนธ์ฉบับนี้ ดังตารางที่ 2.9

ตารางที่ 2.9 เปรียบเทียบแนวคิดเพื่อปรับปรุงงานวิจัยทางด้านการบำรุงรักษาซอฟต์แวร์

ชื่องานวิจัย	ปี ค.ศ.	ผู้วิจัย	แนวคิดที่นำเสนอ	แนวคิดและการปรับปรุงสำหรับวิทยานิพนธ์
Redesigning of an Existing Software using Design Patterns	2000	Gou Masud, Norihiro Sakamot, Kazuo Ushijima	- นำเสนอการปรับปรุงซอฟต์แวร์ที่มีอยู่เดิมโดยใช้ดีไซน์แพตเทิร์น - วัดประสิทธิภาพโดยใช้มาตรวัดซีเค และขนาดของซอฟต์แวร์	- งานวิจัยดังกล่าวนำเสนอเพียงมาตรวัดซีเค และขนาดซึ่งอาจไม่ครอบคลุมในส่วนของการบำรุงรักษาและการเพิ่มขยายซอฟต์แวร์ - ดังนั้นนอกเหนือจากมาตรวัดซีเคแล้วจะทำการเพิ่มเติมมาตรวัดอื่นเพื่อให้ครอบคลุมมากขึ้น โดยจะเน้นการวัดขนาดของการบำรุงรักษาซอฟต์แวร์ (Maintenance Size)
An Empirical Analysis on Distribution Patterns of Software Maintenance Effort	2008	Ye Yang, Qi Li, Mingshu Li, Qing Wang	- นำเสนอ การวิเคราะห์ความพยายามในแต่ละขั้นตอนของกระบวนการบำรุงรักษาซอฟต์แวร์ - ขนาดของการบำรุงรักษาแปรผันตรงกับความพยายามในการบำรุงรักษา - มีการวัดขนาดของการบำรุงรักษาจาก จำนวนบรรทัดของซอฟต์แวร์ (Source Line of Code: SLOC)	- มาตรวัดทางด้าน การบำรุงรักษาซอฟต์แวร์ จะวัดจากจำนวนบรรทัดของซอฟต์แวร์ (SLOC) ซึ่งอาจจะไม่เหมาะสำหรับการเขียนโปรแกรมเชิงวัตถุ - ดังนั้นจึงเพิ่มเติม การวัดขนาดของการบำรุงรักษาซอฟต์แวร์ โดยพิจารณาด้านอื่นๆ เช่น การพิจาณาจำนวนโมดูล หรือ จำนวนคลาสเพิ่มเติม แต่ขณะเดียวกันจะยังคงมีการวัดโดยใช้ LOC แบบเดิมไว้
Object Oriented Design Metrics and Tools A Survey	2010	Sahar R. Ragab, Hany H. Ammar	- นำเสนอความสัมพันธ์ระหว่างปัจจัยคุณภาพและมาตรวัดซอฟต์แวร์ - มาตรวัดทางด้าน Coupling และ Cohesion สามารถบ่งบอกปัจจัยคุณภาพทางด้าน Maintainability - ค่าของความซับซ้อนและการสืบทอดคุณสมบัติ สามารถบ่งบอกปัจจัยคุณภาพเกือบทุกด้าน เช่น ความสามารถด้านความยืดหยุ่น ความเข้าใจง่าย เป็นต้น	- งานวิจัยดังกล่าว ได้นำเสนอความสัมพันธ์ระหว่างมาตรวัดและปัจจัยคุณภาพด้าน Maintainability ซึ่งเป็นปัจจัยที่วิทยานิพนธ์ต้องการวัด ดังนั้น วิทยานิพนธ์นี้จึงเลือกมาตรวัดที่บ่งบอกถึง ปัจจัยคุณภาพ ได้แก่ มาตรวัดค่าของ Coupling และ Cohesion

บทที่ 3
วิธีดำเนินการวิจัย

ในบทนี้จะกล่าวถึงการวัดการบำรุงรักษาซอฟต์แวร์ ประกอบด้วย 3.1 อธิบายถึงลักษณะเบื้องต้นของซอฟต์แวร์ที่ใช้เป็นกรณีศึกษาและรายละเอียดของโครงสร้างคลาสดั้งเดิมและโครงสร้างคลาสที่ปรับปรุงระหว่างบำรุงรักษาซอฟต์แวร์ หัวข้อ 3.2 กล่าวถึง มาตรวัดที่จะใช้สำหรับวัดการบำรุงรักษาซอฟต์แวร์ ทั้งมาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์ และมาตรวัดซอฟต์แวร์เชิงวัตถุ และหัวข้อ 3.3 กล่าวถึงการออกแบบการทดลองเพื่อวัดการบำรุงรักษาซอฟต์แวร์ ประกอบด้วยการออกแบบประเภทรายงานและแบบฟอร์มที่ใช้สำหรับบันทึกข้อมูล

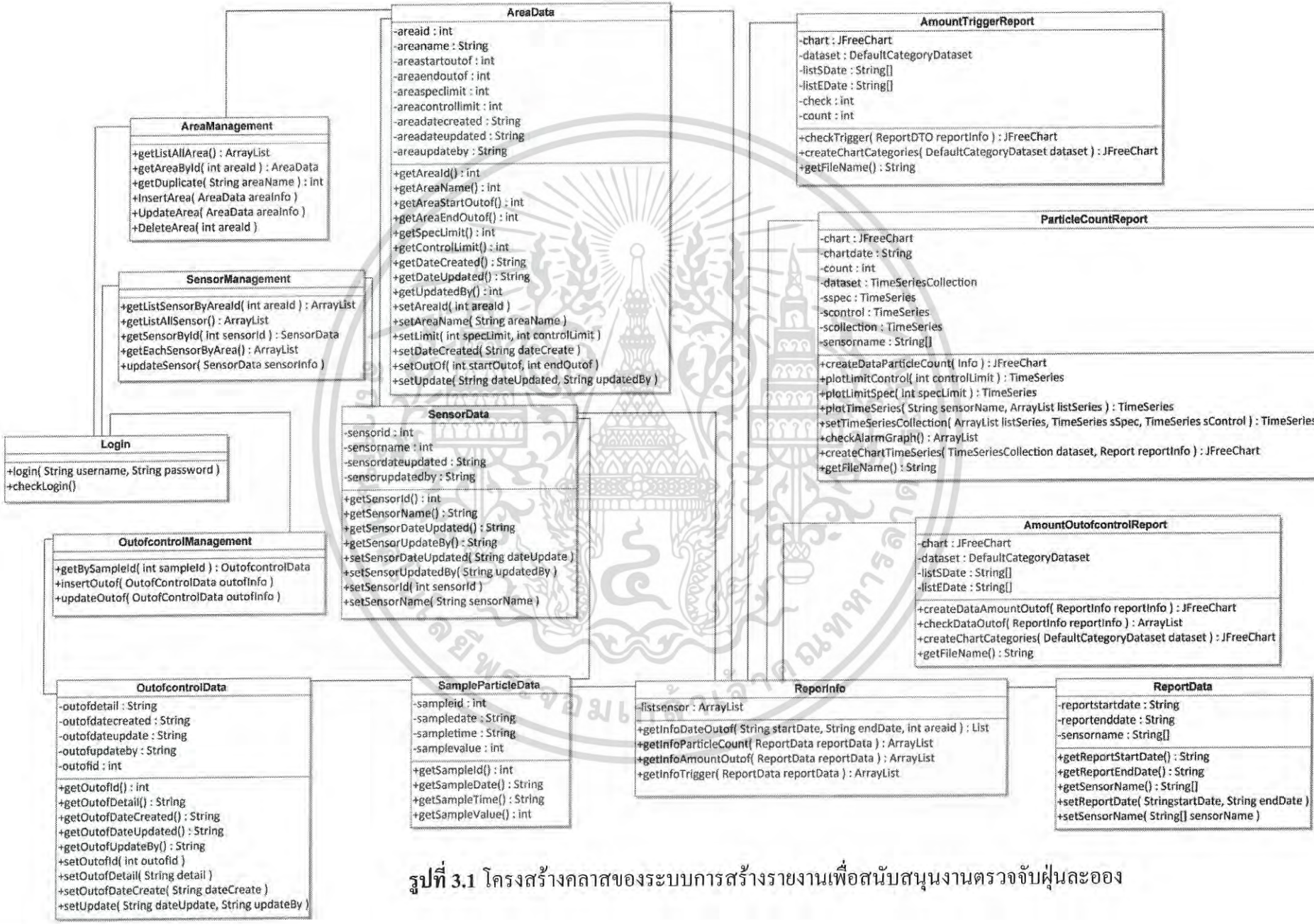


3.1 การออกแบบโครงสร้างคลาสสำหรับซอฟต์แวร์ดั้งเดิมและการบำรุงรักษาซอฟต์แวร์

สำหรับการบำรุงรักษาซอฟต์แวร์ เป็นการปรับปรุงซอฟต์แวร์ที่มีอยู่เดิมหลังจากที่มีการติดตั้งหรือส่งมอบซอฟต์แวร์ไปแล้ว [11] โดยวิทยานิพนธ์ฉบับนี้จะวิเคราะห์ปัจจัยคุณภาพทางด้านความสามารถในการบำรุงรักษา สำหรับงาน 2 ลักษณะ ประกอบด้วย 1) การบำรุงรักษาซอฟต์แวร์โดยการเพิ่มเติมรายงานตามความต้องการของผู้ใช้งาน (Functional Enhancement Maintenance) และ 2) การบำรุงรักษาโดยการจัดระเบียบซอร์ดโค้ด (Refactoring Source Code)

3.1.1 ลักษณะเบื้องต้นของซอฟต์แวร์ที่ใช้เป็นกรณีศึกษา

ระบบที่ใช้เป็นกรณีศึกษา ได้แก่ ระบบการสร้างรายงานอัตโนมัติเพื่อสนับสนุนงานตรวจจับฝุ่นละอองในอุตสาหกรรมการผลิตฮาร์ดดิสก์ (Automated Report Generation System for Particle Monitoring in Hard Disk Drive Industry) มีวัตถุประสงค์เพื่อสร้างรายงานปริมาณฝุ่นละอองตามความต้องการของผู้ใช้งาน ปัจจุบันการทำงานของระบบจะเน้นการสร้างรายงานพื้นฐานเป็นหลัก เช่น รายงานเปรียบเทียบปริมาณฝุ่นละอองของแต่ละเซ็นเซอร์ รายงานสรุปจำนวนที่เกินมาตรฐาน รายงานสรุปจำนวนครั้งที่เกิดทริกเกอร์ เป็นต้น ซึ่งระบบนี้พัฒนาขึ้นเมื่อปี พ.ศ. 2552 เป็นระบบขนาดเล็ก แสดงโครงสร้างคลาส (Class Diagram) ที่ได้จากการออกแบบซอฟต์แวร์ ดังรูปที่ 3.1



รูปที่ 3.1 โครงสร้างคลาสของระบบการสร้างรายงานเพื่อสนับสนุนงานตรวจจับฝุ่นละออง

จากรูปที่ 3.1 แสดงโครงสร้างคลาสของระบบการสร้างรายงานอัตโนมัติเพื่อสนับสนุนงานตรวจจับฝุ่นละอองในอุตสาหกรรมการผลิตฮาร์ดดิสก์ ประกอบด้วยรายละเอียดของการทำงานในแต่ละคลาส ดังนี้

1. คลาสข้อมูลเซ็นเซอร์ (SensorData) เป็นคลาสที่ทำหน้าที่จัดเก็บข้อมูลของอุปกรณ์เซ็นเซอร์ทั้งหมดในทุกพื้นที่การผลิต โดยจะเก็บข้อมูล รหัสเซ็นเซอร์ (sensorid) ชื่อเซ็นเซอร์ (sensorname) รหัสพื้นที่การผลิตที่เซ็นเซอร์นั้นทำงานอยู่ (areaid)

2. คลาสการจัดการเซ็นเซอร์ (SensorManagement) ใช้สำหรับจัดการข้อมูลของเซ็นเซอร์ ประกอบด้วย เมธอดสำหรับทำการเปลี่ยนแปลงแก้ไขข้อมูลของอุปกรณ์เซ็นเซอร์ ได้แก่ เมธอด *updateSensor()*

3. คลาสข้อมูลพื้นที่การผลิต (AreaData) เป็นคลาสที่ทำหน้าที่จัดเก็บข้อมูลของพื้นที่การผลิต ประกอบด้วยแอตทริบิวต์ที่เกี่ยวกับพื้นที่การผลิต เช่น รหัสพื้นที่การผลิต (areaid) ชื่อพื้นที่การผลิต (areaname) ค่าควบคุมของแต่ละพื้นที่ (areacontrollimit) ค่าสเปคของแต่ละพื้นที่ (areaspeclimit) และค่าเกินมาตรฐาน (areaoutofcontrol)

4. คลาสการจัดการพื้นที่การผลิต (AreaManagement) ทำหน้าที่สำหรับจัดการข้อมูลของพื้นที่การผลิต โดยประกอบด้วย เมธอด *addArea()* สำหรับเพิ่มพื้นที่การผลิต เมธอด *editArea()* สำหรับแก้ไขพื้นที่การผลิตและเมธอด *deleteArea()* ใช้ลบพื้นที่การผลิต

5. คลาสข้อมูลปริมาณฝุ่นละออง (SampleParticleData) เป็นคลาสที่ทำหน้าที่เก็บข้อมูลปริมาณฝุ่นละออง ได้แก่ วันที่ที่เกิดฝุ่นละออง (sampledate) เวลาที่เกิดฝุ่น (sampletime) ปริมาณฝุ่นละออง (samplevalue)

6. คลาสข้อมูลเกินควบคุม (OutofcontrolData) ทำหน้าที่เก็บข้อมูลและรายละเอียดในกรณีที่เกิดปริมาณฝุ่นละออง ณ จุดนั้นเกินมาตรฐานที่กำหนดไว้ (Out of Control) เช่น สาเหตุของการเกิด out of control เป็นต้น

7. คลาสจัดการข้อมูลเกินควบคุม (OutofcontrolManagement) ใช้สำหรับจัดการข้อมูลเกินควบคุม หรือ ข้อมูลที่ out of control โดยประกอบด้วยเมธอดที่สามารถ เพิ่มสาเหตุของการเกิดปริมาณฝุ่นละอองเกินมาตรฐานที่กำหนด ด้วยเมธอด *addOutof()* และแก้ไขข้อมูลของการเกิด out of control ด้วยเมธอด *editOutof()*

8. คลาสตรวจสอบสิทธิ์การใช้งานระบบ (Login) ทำหน้าที่รับและตรวจสอบข้อมูลจากผู้ใช้งานเพื่อตรวจสอบสิทธิ์ผู้เข้าใช้งานในการตั้งค่าพื้นที่การผลิตหรือเปลี่ยนแปลงแก้ไขข้อมูลของอุปกรณ์เซ็นเซอร์ ประกอบด้วยเมธอด *login()* และ *checkLogin()*

9. คลาสรายงาน (ReportData) เป็นเอนทิตีคลาสที่ทำหน้าที่เก็บข้อมูลทั้งหมดเพื่อนำมาใช้ในการสร้างรายงาน ประกอบด้วยข้อมูลที่นำมาจากคลาส AreaData SensorData และ SampleParticleData

10. คลาสรายงาน (ReportInfo) เป็นคลาสที่ทำหน้าที่จัดเตรียมข้อมูลเพื่อสร้างรายงาน ประกอบด้วยเมธอดในการจัดการข้อมูลสำหรับสร้างรายงานแต่ละประเภท เช่น เมธอด *getInfoTrigger()* สำหรับสร้างรายงานการเกิดทริกเกอร์ เป็นต้น

11. คลาสรายงานปริมาณฝุ่นละออง (ParticleCountReport) ทำหน้าที่ในการสร้างรายงานปริมาณฝุ่นละอองของแต่ละเซ็นเซอร์ (Particle Count Report) เพื่อเปรียบเทียบกับปริมาณฝุ่นมาตรฐาน ประกอบด้วยแอตทริบิวต์ที่ใช้สำหรับเก็บข้อมูลการสร้างรายงาน เช่น เวลาเริ่มต้น (startDate) และ เวลาสิ้นสุด (endDate) และเมธอดที่ใช้สำหรับการสร้างกราฟ เช่น เมธอด *generateChartTimeSeries()* เป็นต้น

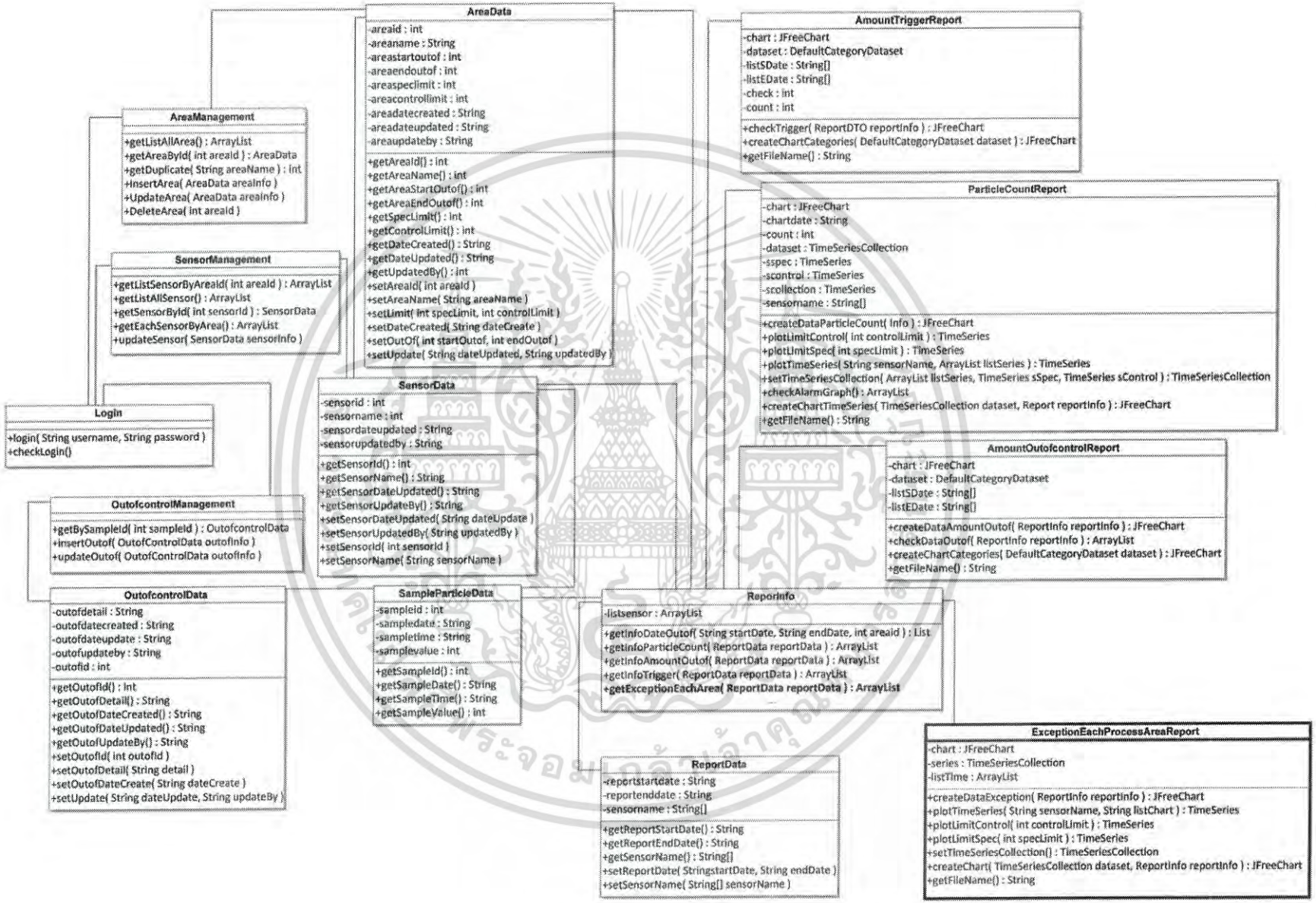
12.คลาสรายงานสรุปจำนวนครั้งที่ปริมาณฝุ่นเกินมาตรฐาน (AmountOutofControlReport) ทำหน้าที่ตรวจสอบและสร้างรายงาน จำนวนครั้งของการเกิดปริมาณฝุ่นเกินมาตรฐานในพื้นที่การผลิต (Amount out of control Report) ประกอบด้วย แอตทริบิวต์ ที่ใช้สำหรับเก็บข้อมูลการสร้างรายงาน เช่น อุปกรณ์เซ็นเซอร์ที่ต้องการนับจำนวนครั้งที่ปริมาณฝุ่นเกินมาตรฐานที่กำหนด (sensorgroup) หรือ ชุดวันที่สำหรับแบ่งกลุ่ม (category) และเมธอดที่ใช้สำหรับการสร้างกราฟ เช่น เมธอด *generateChartCategories()* เป็นต้น

13. คลาสรายงานสรุปจำนวนครั้งที่เกิดทริกเกอร์ (AmountTriggerReport) ข้อมูลในแต่ละพื้นที่จะมีการกำหนดค่า Out of Control เช่น การกำหนด “3 out of 6” หมายถึง หากมีปริมาณฝุ่นละอองเกินมาตรฐานต่อเนื่องกัน 3 ครั้งใน 6 ครั้ง ณ เวลานั้นจะเรียกว่าเกิด ทริกเกอร์ (Trigger) ซึ่งคลาสดังกล่าวจะเป็นคลาสรายงานประเภทหนึ่ง ประกอบด้วยแอตทริบิวต์ที่ใช้ในการสร้างรายงาน เช่น อุปกรณ์เซ็นเซอร์ที่ต้องการ (sensorgroup) และ จำนวนครั้งการเกิดทริกเกอร์ (count) เป็นต้น

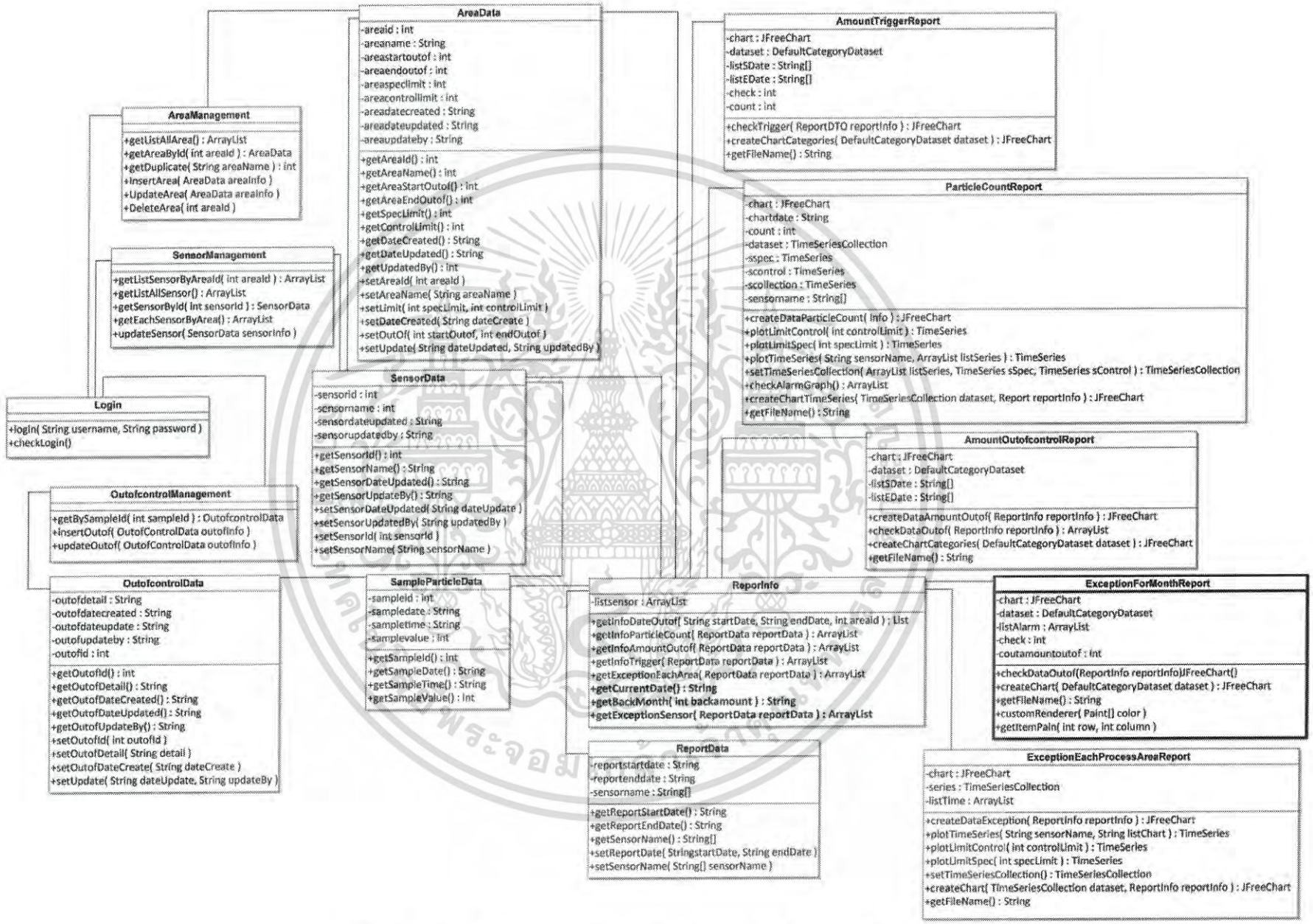
3.1.2 รายละเอียดโครงสร้างคลาสที่ใช้สำหรับบำรุงรักษาซอฟต์แวร์

หลังจากที่ออกแบบระบบการสร้างรายงานจะทำการพัฒนาตามโครงสร้างที่ได้ออกแบบซึ่งในระหว่างการพัฒนาจะทำการบันทึกข้อมูล ความพยายามในการพัฒนา (Development Effort) และหลังจากพัฒนาเสร็จสิ้นจะทำการวัดผลผลิตของซอฟต์แวร์เพื่อตรวจสอบโครงสร้างของการออกแบบ ได้แก่ การวัดค่าจำนวนเมธอดต่อคลาส (WMC) ค่าความเข้าคู่ระหว่างวัตถุ (CBO) ค่าระดับการขาดความสัมพันธ์ภายในคลาส (LCOM) ค่าการตอบสนองต่อคลาส (RFC) ค่าจำนวนคลาสลูก (NOC) และค่าระดับความลึกของการถ่ายทอดคุณสมบัติ (DIT)

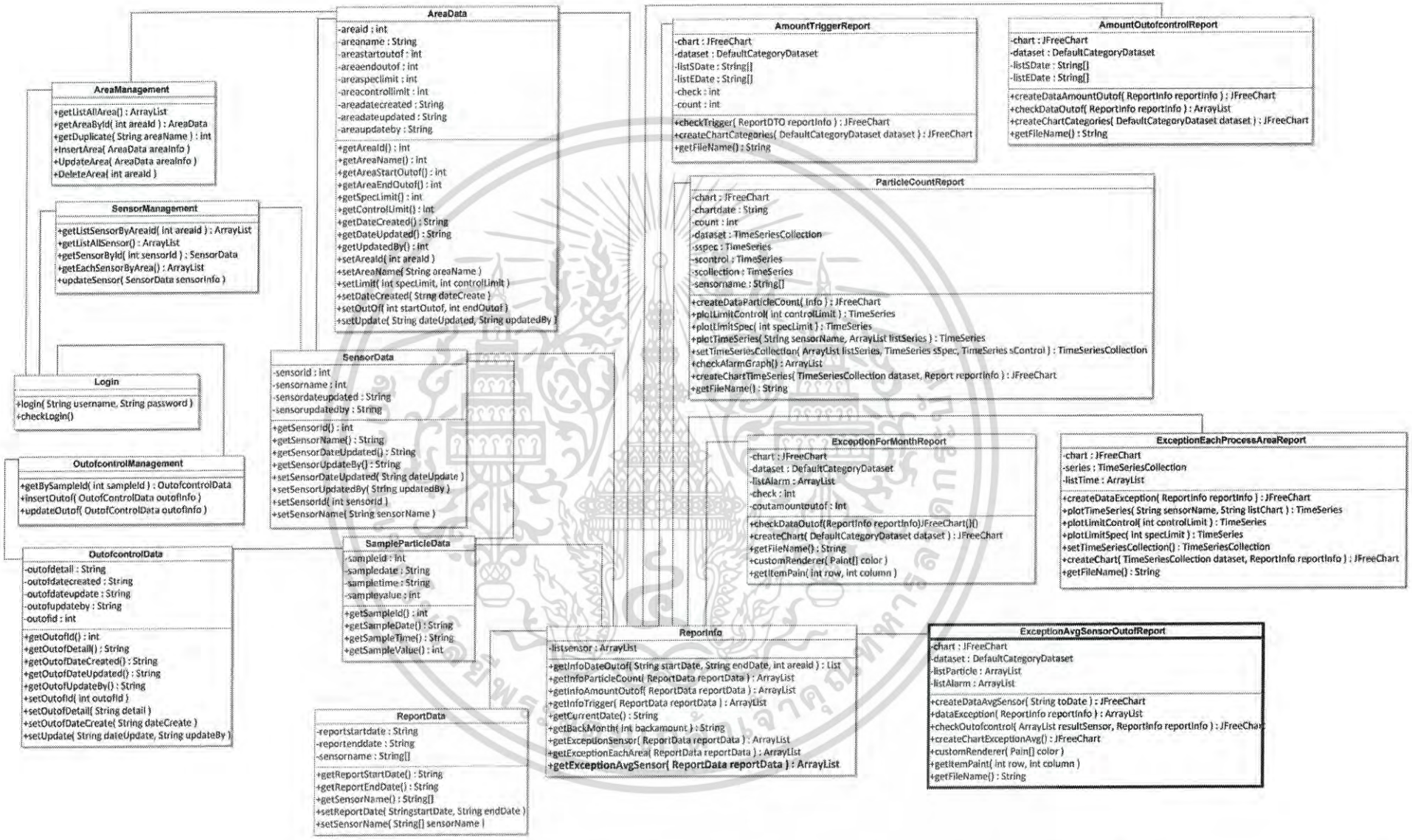
หลังจากนั้นจะออกแบบโครงสร้างคลาสในส่วนของการบำรุงรักษาซอฟต์แวร์ในแต่ละเวอร์ชั่น แบ่งออกการบำรุงรักษาออกเป็น 2 ลักษณะงาน ได้แก่ การเพิ่มเติมประเภทรายงานตามความต้องการของผู้ใช้งาน (บำรุงรักษาครั้งที่ 1- 4) แสดงดังรูปที่ 3.2 – 3.5 และการบำรุงรักษาโดยการจัดระเบียบซอร์ดโค้ด (บำรุงรักษาครั้งที่ 5) แสดงดังรูปที่ 3.6



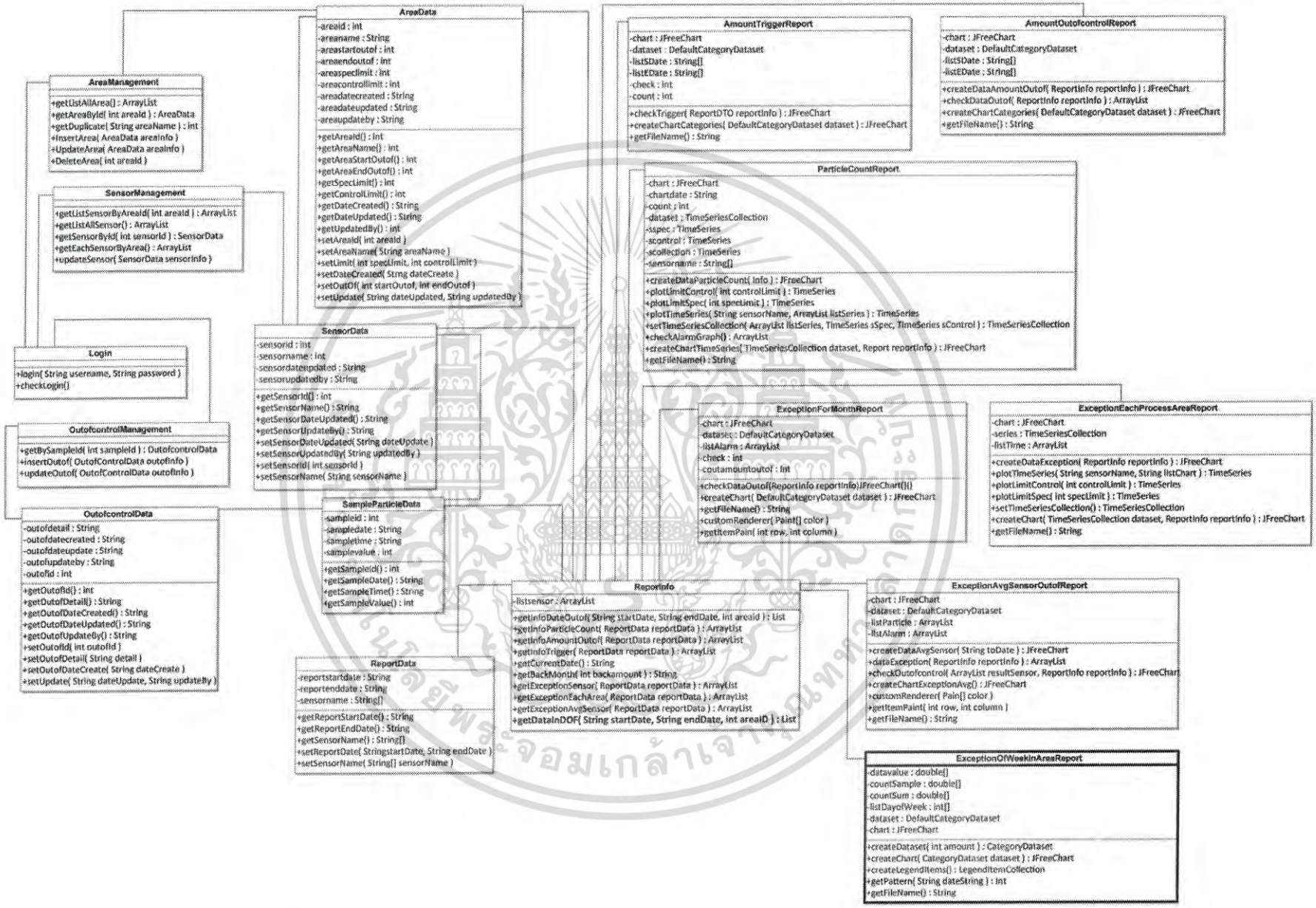
รูปที่ 3.2 โครงสร้างคลาสของการบำรุงรักษาซอฟต์แวร์โดยการเพิ่มรายงานระดับง่ายครั้งที่ 1



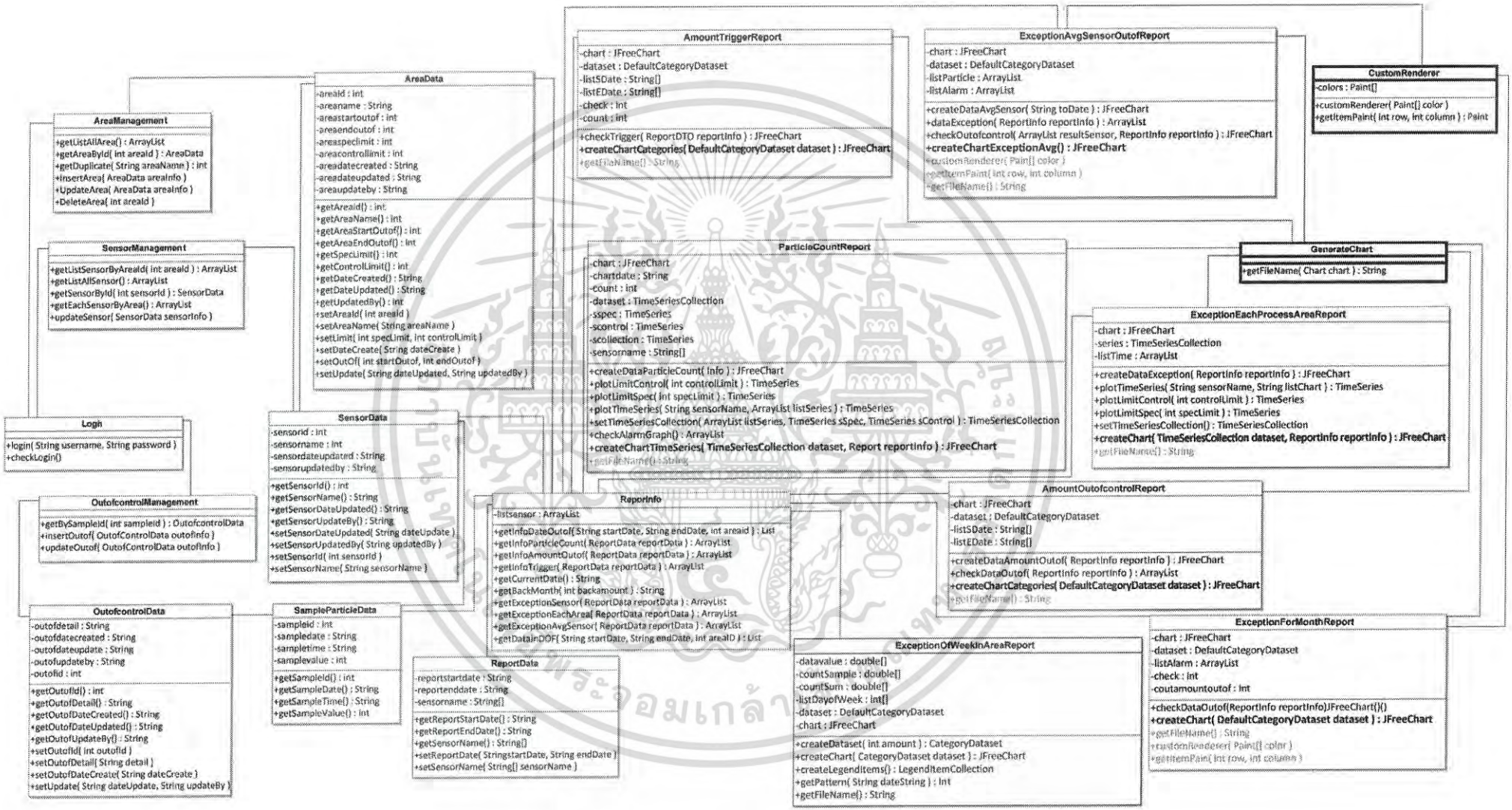
รูปที่ 3.3 โครงสร้างคลาสของการบำรุงรักษาซอฟต์แวร์โดยการเพิ่มรายงานระดับง่าย ครั้งที่ 2



รูปที่ 3.4 โครงสร้างคลาสของการบำรุงรักษาซอฟต์แวร์โดยการเพิ่มรายงานระดับปานกลาง ครั้งที่ 1



รูปที่ 3.5 โครงสร้างคลาสของการบำรุงรักษาซอฟต์แวร์โดยการเพิ่มรายงานระดับปานกลาง ครั้งที่ 2



รูปที่ 3.6 โครงสร้างคลาสของการบำรุงรักษาซอฟต์แวร์โดยการจัดเรียงซอร์ดโค้ด

จากรูปที่ 3.2 ถึง 3.5 แสดงโครงสร้างคลาส (Class Diagram) ของการปรับปรุงซอฟต์แวร์โดยการเพิ่มเติมประเภทรายงานตามความต้องการของผู้ใช้งาน ประกอบด้วย 4 รายงาน เป็นรายงานระดับง่าย 2 รายงาน และ รายงานระดับปานกลาง 2 รายงาน แสดงรายละเอียดคลาส รายงาน ดังนี้

รายงานระดับง่าย

1. คลาสรายงานแสดงความผิดปกติของฝุ่นละอองของเซ็นเซอร์ทั้งหมดที่อยู่ในแต่ละพื้นที่การผลิต (ExceptionEachProcessAreaReport) ทำหน้าที่ในการสร้างรายงานความของปริมาณฝุ่นละอองของเซ็นเซอร์ที่ผิดปกติในแต่ละพื้นที่การผลิต ประกอบด้วยเมธอด *createDataException()* ที่ใช้สำหรับการตรวจหาข้อมูลของเซ็นเซอร์ที่ผิดปกติ เป็นต้น

2. คลาสรายงานความผิดปกติของจำนวนครั้งการเกิดปริมาณฝุ่นละอองเกินกำหนดของแต่ละเซ็นเซอร์ย้อนหลัง 1 เดือน (ExceptionForMonthReport) ทำหน้าที่ในการตรวจหาข้อมูลของเซ็นเซอร์ที่ผิดปกติ ย้อนหลัง 1 เดือน ประกอบด้วยแอตทริบิวต์ที่ใช้สำหรับเก็บข้อมูลการนับจำนวนการเกิดจำนวนครั้งการผิดปกติ (countamountoutof) และเมธอด *createDataOutof()* ที่ใช้ในการสร้างรายงานความผิดปกติย้อนหลัง เป็นต้น

รายงานระดับปานกลาง

3. คลาสแสดงรายงานการเปรียบเทียบจำนวนของเซ็นเซอร์ที่ผิดปกติระหว่างพื้นที่การผลิต (ExceptionAvgSensorOutofReport) ทำหน้าที่ในการเปรียบเทียบข้อมูลปริมาณฝุ่นละอองที่เกินค่ามาตรฐานที่กำหนดของแต่ละพื้นที่การผลิต ประกอบด้วยเมธอด *checkOutofcontrol()* ที่ใช้ในการตรวจสอบค่าที่เกินมาตรฐาน เมธอด *dataException()* ที่ใช้ในการตรวจจับค่าที่ผิดปกติ เป็นต้น

4. คลาสรายงานการเปรียบเทียบปริมาณฝุ่นละอองเฉลี่ยในแต่ละวันของพื้นที่การผลิต (ExceptionOfWeekInAreaReport) ทำหน้าที่ในการสร้างรายงานปริมาณฝุ่นละอองของแต่ละพื้นที่การผลิตแยกเป็นสัปดาห์ ทำให้สามารถทราบได้ว่า ในหนึ่งพื้นที่การผลิตวันใดที่ปริมาณฝุ่นละอองให้ค่าเฉลี่ยสูงสุดในหนึ่งสัปดาห์ ประกอบด้วย แอตทริบิวต์ที่เก็บค่าปริมาณฝุ่นเฉลี่ยภายใน 1 วันของสัปดาห์ (datavalue) และเมธอด *createDataset()* ที่ใช้สำหรับคำนวณและแยกปริมาณฝุ่นแต่ละวันในแต่ละพื้นที่การผลิต เป็นต้น

จากรูปที่ 3.6 เป็นการปรับปรุงและจัดระเบียบซอร์ดโค้ด โดยเพิ่มคลาสและแยกเมธอดที่เรียกใช้งานจากคลาสหลายคลาส (Re-Engineering) เพื่อให้ง่ายต่อการเพิ่มเติมรายงานในอนาคตมากยิ่งขึ้น ซึ่งคลาสที่เพิ่มเติม ได้แก่

5. คลาสการสร้างรายงาน (GenerateChart) เป็นคลาสที่ทำหน้าที่ในการสร้างรายงานและแสดงผลรายงานให้ผู้ใช้งาน ประกอบด้วยเมธอดที่ใช้สำหรับเรียกไฟล์รายงานที่ทำการสร้างผ่านเซสชั่นได้แก่ เมธอด *getFileName()*

6. คลาสการสร้างสีของรายงานอัตโนมัติ (CustomRenderer) เป็นคลาสที่ทำหน้าที่กำหนดสีของรายงานอัตโนมัติเพื่อให้เห็นความแตกต่างของรายงานแต่ละประเภท

3.2 มาตรวัดที่ใช้สำหรับวัดการบำรุงรักษาซอฟต์แวร์

การออกแบบการทดลองเพื่อวัดการบำรุงรักษาซอฟต์แวร์ ประกอบด้วย หัวข้อ 3.2.1 มาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์ โดยจะอธิบายถึงมาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์เดิมที่มีอยู่และมาตรวัดที่นำเสนอขึ้นใหม่ หัวข้อ 3.2.2 อธิบายมาตรวัดความพยายามในการบำรุงรักษาซอฟต์แวร์ หัวข้อ 3.2.3 อธิบายมาตรวัดซอฟต์แวร์เชิงวัตถุ ประกอบด้วย 6 มาตรวัด หัวข้อ 3.2.4 อธิบายการออกแบบรายงานที่ใช้เป็นกรณีศึกษาสำหรับการบำรุงรักษาซอฟต์แวร์ และหัวข้อที่ 3.2.5 แสดงแบบฟอร์มเพื่อบันทึกข้อมูลระหว่างที่ทำการพัฒนาและบำรุงรักษาซอฟต์แวร์

โดยวิทยานิพนธ์ฉบับนี้ จะพิจารณาความสามารถในการบำรุงรักษาซอฟต์แวร์จาก ขนาดของการบำรุงรักษาซอฟต์แวร์ (Maintenance Size) และมาตรวัดซอฟต์แวร์เชิงวัตถุจำนวน 6 มาตรวัด เนื่องจาก ขนาดของการบำรุงรักษาซอฟต์แวร์มีความสัมพันธ์อย่างมากกับความพยายามในการบำรุงรักษาซอฟต์แวร์ [4] ดังนั้นจึงมีแนวคิดว่าเมื่อขนาดของการบำรุงรักษาซอฟต์แวร์มีขนาดใหญ่หรืออัตราการเปลี่ยนแปลงของซอฟต์แวร์มีมากจะทำให้ต้องใช้ความพยายามในการบำรุงรักษาซอฟต์แวร์มากขึ้นไปด้วย ซึ่งความพยายามในการบำรุงรักษาซอฟต์แวร์เป็นปัจจัยหนึ่งที่ส่งผลถึงประสิทธิภาพของความสามารถในการบำรุงรักษาซอฟต์แวร์ สำหรับมาตรวัดเชิงวัตถุเป็นตัวชี้วัดที่ใช้วัดความซับซ้อนของซอฟต์แวร์ทางด้านโครงสร้าง โดยค่าที่ได้จากมาตรวัดสามารถบ่งบอกถึงความสามารถในการบำรุงรักษาซอฟต์แวร์ได้ ดังนั้นวิทยานิพนธ์นี้จึงเลือก มาตรวัดทั้ง 3 กลุ่ม ได้แก่ มาตรวัดความพยายามในการบำรุงรักษาซอฟต์แวร์ (Maintenance Effort) ซึ่งได้จากการเก็บข้อมูลจริงระหว่างที่ทำการบำรุงรักษาซอฟต์แวร์ มาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์ (Maintenance Size) และมาตรวัดเชิงวัตถุ (Object-Oriented Metrics) แสดงรายละเอียด ดังนี้

3.2.1 มาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์ (Software Maintenance Size)

มาตรวัดขนาดของซอฟต์แวร์ (Software Size Metrics) เป็นมาตรวัดที่จะใช้วัดผลผลิตของซอฟต์แวร์ ซึ่งมาตรวัดขนาดของซอฟต์แวร์ส่วนใหญ่จะวัดจากจำนวนบรรทัดของซอฟต์แวร์ (Source Line of Code) ซึ่งถึงแม้จะเป็นมาตรวัดแบบดั้งเดิม แต่สำหรับการเขียนโปรแกรมเชิงวัตถุในปัจจุบันก็ยังมีการใช้มาตรวัดนี้สำหรับวัดระบบซอฟต์แวร์

โดยส่วนใหญ่ขนาดของซอฟต์แวร์จะพิจารณาจากจำนวนบรรทัดของซอฟต์แวร์ รวมถึงการวัดขนาดของการบำรุงรักษาซอฟต์แวร์เช่นกัน จะมีการพิจารณาขนาดของการบำรุงรักษาซอฟต์แวร์จาก การพิจารณาจำนวนบรรทัดที่เปลี่ยนแปลงไป (คือจำนวนบรรทัดของซอร์ดโค้ดเพิ่มขึ้นและเปลี่ยนแปลงไป) [4] ดังสมการที่ (3.1)

3.2.1.1 มาตรวัดขนาดการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจากจำนวนบรรทัดที่เปลี่ยนแปลงไปของซอฟต์แวร์ (MS-LOC) แสดงดังสมการ (3.1)

$$\text{MS} - \text{LOC} = \frac{LOC_{Added} + LOC_{Modified}}{Total\ Line\ of\ Code\ in\ the\ Initial\ Sytem} \quad (3.1)$$

จากสมการที่ (3.1) มาตรวัด MS-LOC ใช้เป็นมาตรวัดเบื้องต้นในการวัดขนาดการบำรุงรักษาของซอฟต์แวร์ โดยการหาอัตราส่วนระหว่างจำนวนบรรทัดที่เพิ่มขึ้น (LOC_{Added}) และจำนวนบรรทัดที่เปลี่ยนแปลง ($LOC_{Modified}$) ของซอฟต์แวร์เปรียบเทียบกับจำนวนบรรทัดทั้งหมดในเวอร์ชั่นดั้งเดิม

ในวิทยานิพนธ์นี้จะนำเสนอมาตรวัดการบำรุงรักษาซอฟต์แวร์โดยพิจารณาปัจจัยอื่นนอกเหนือจากจำนวนบรรทัดของซอฟต์แวร์ ได้แก่ จำนวนคลาส จำนวนเมธอด ดังแสดงรายละเอียดดังมาตรวัดที่ 3.2.1.2 – 3.2.1.5

อนึ่งแนวคิดการนับจำนวนเมธอด/คลาส ที่ลบออกจากระบบนั้นมีแนวคิด 3 วิธี ได้แก่

1. การนับจำนวนเมธอด/คลาส ที่ลบออก
2. การนำเมธอด/คลาส ที่ลบออก หักออกจากคลาสที่เพิ่ม
3. ไม่คำนึงถึงเมธอด/คลาสที่ลบออก

สำหรับวิทยานิพนธ์ฉบับนี้จะใช้วิธีคำนวณจำนวนเมธอด/คลาส ตามแนวคิดที่ 2 โดยการนับจำนวนคลาสหรือเมธอดโดยพิจารณาคลาสหรือ เมธอดที่เปลี่ยนแปลงไป คำนวณได้จากจำนวนคลาสหรือเมธอดที่เพิ่มขึ้นลบออกและเปลี่ยนแปลงไปจากระบบเดิม แสดงรายละเอียดของแต่ละมาตรวัด ดังนี้

3.2.1.2 มาตรวัดขนาดการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจากจำนวนคลาสของซอฟต์แวร์ (MS-TC) จะคำนวณจากอัตราส่วนระหว่างคลาสที่เปลี่ยนแปลงไปและจำนวนคลาสจากการพัฒนาในเวอร์ชั่นตั้งต้น แสดงดังสมการ (3.2)

$$\text{MS} - \text{TC} = \frac{|Class_{Added} - Class_{Deleted}| + Class_{Modified}}{Total\ Class\ in\ the\ Initial\ Sytem} \quad (3.2)$$

3.2.1.3 มาตรวัดขนาดการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจากจำนวนเมธอดของซอฟต์แวร์ (MS-TM) พิจารณาขนาดของการบำรุงรักษาซอฟต์แวร์จากจำนวนเมธอดที่เปลี่ยนไป คำนวณได้จาก อัตราส่วนระหว่างเมธอดที่เปลี่ยนแปลงไปต่อจำนวนเมธอดทั้งหมด ดังสมการ (3.3)

$$\text{MS} - \text{TM} = \frac{|Method_{Added} - Method_{Deleted}| + Method_{Modified}}{Total\ Method\ in\ the\ Initial\ Sytem} \quad (3.3)$$

3.2.1.4 การวัดขนาดการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจากจำนวนเมธอดต่อคลาส (MS-MC) คำนวณจากอัตราส่วนระหว่างค่าเฉลี่ยของจำนวนเมธอดต่อคลาสที่เปลี่ยนแปลงไปต่อจำนวนเมธอดต่อคลาสทั้งหมดในเวอร์ชั่นดั้งเดิม ดังสมการ (3.4)

$$\mathrm{MS-MC} = \frac{(Method_{Changed})/Number\ of\ Class\ Change}{\frac{\sum_{I \in IS} Method(i)}{Total\ Class\ in\ the\ Initial\ System}} \tag{3.4}$$

โดยที่ i เป็นสมาชิกในเซต IS (Initial System)

3.2.1.5 มาตรวัดขนาดการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจากจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนัก (MS- WMC) จะพิจารณาเพิ่มเติมจากมาตรวัดที่ (3.4) โดยคำนวณจากอัตราส่วนระหว่างจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนักที่เปลี่ยนแปลงไปต่อจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนักในเวอร์ชั่นตั้งต้นทั้งหมด แสดงดังสมการ (3.5)

$$\mathrm{MS-WMC} = \frac{(WeightedMethod_{Changed})/Number\ of\ Class\ Change}{\frac{\sum_{I \in IS} Complexity(i)}{Total\ Class\ in\ the\ Initial\ System}} \tag{3.5}$$

โดยที่ i เป็นสมาชิกในเซต IS (Initial System)

จำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนัก (Weighted Methods per Class: WMC) จะคำนวณจากค่าความซับซ้อนทั้งหมดของทุกเมธอดภายในคลาส ซึ่งจะพิจารณาจากค่าความซับซ้อนของไซโคลเมติก (Cyclomatic Complexity) ของทุกเมธอดภายในระบบ

โดยค่าความซับซ้อนไซโคเมติก [24] สามารถคำนวณได้จากภายในแต่ละเมธอด โดยจะวัดจากจากเส้นทางการเชื่อมต่อในกราฟควบคุมกระแส (Flow Graph) โดยแต่ละโหนด (node: n) จะแทนคำสั่งของโปรแกรมหนึ่งคำสั่ง และแต่ละคำสั่งจะเชื่อมต่อกันด้วยเส้นเชื่อมโหนด (edge: e) แสดงการคำนวณค่าความซับซ้อนของไซโคเมติก ดังสมการที่ (3.6)

$$\mathrm{Cyclomatic_i} = e - n + 2 \tag{3.6}$$

ขนาดของการบำรุงรักษาโดยพิจารณาจาก จำนวนเมธอด จำนวน จำนวนเมธอดต่อคลาสทั้งที่คิดค่าถ่วงน้ำหนักและไม่คิดค่าถ่วงน้ำหนักนั้น จะนำมาใช้คำนวณค่าประมาณของเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์ เนื่องจาก ขนาดของซอฟต์แวร์มีความสัมพันธ์กับความพยายามในการบำรุงรักษาซอฟต์แวร์ [4][20]

3.2.2 มาตรวัดความพยายามของการบำรุงรักษาซอฟต์แวร์

ความพยายามของการบำรุงรักษาซอฟต์แวร์ (Maintenance Effort) เป็นการวัดกำลังคนหรือแรงงานที่ใช้ในกระบวนการพัฒนาหรือในกระบวนการบำรุงรักษาซอฟต์แวร์ อาจมีหน่วยเป็น คน

ต่อชั่วโมง (Person Hour) คนต่อวัน (Person Day) คนต่อเดือน (Person Month) สำหรับวิทยานิพนธ์ฉบับนี้จะวัดความพยายามในการบำรุงรักษาซอฟต์แวร์ มีหน่วยเป็นคนต่อชั่วโมง (Person Hour) คำนวณได้จากสมการที่ (3.7)

$$\text{Effort} = \frac{Time}{People} \tag{3.7}$$

โดยที่ Effort หมายถึง ความพยายามที่ใช้ในการบำรุงรักษาซอฟต์แวร์

$Time$ หมายถึง ระยะเวลาที่ใช้ในการปรับปรุง มีหน่วยเป็นชั่วโมง

$People$ หมายถึง กำลังคนที่ใช้ในการปรับปรุงซอฟต์แวร์

จากสมการที่ (3.7) ความพยายามในการพัฒนาหรือบำรุงรักษาซอฟต์แวร์จะพิจารณาจากอัตราส่วนระหว่างเวลาที่ใช้ในการพัฒนาหรือบำรุงรักษาซอฟต์แวร์และจำนวนโปรแกรมเมอร์ หรือกำลังคน แต่เนื่องจากระบบที่ใช้เป็นกรณีศึกษามีผู้พัฒนาหรือโปรแกรมเมอร์หนึ่งคน ดังนั้นความพยายามในการพัฒนาหรือการบำรุงรักษาซอฟต์แวร์ (Development/Maintenance Effort) จึงเท่ากับเวลาในการพัฒนาหรือการบำรุงรักษาซอฟต์แวร์ (Development/Maintenance Time)

ซึ่งการวัดความพยายามของการบำรุงรักษาซอฟต์แวร์อาจทำได้หลายวิธี เช่น งานวิจัยของ Yang [4] จะวัดจาก ผลรวมของความพยายามในแต่ละขั้นตอน ได้แก่ ขั้นตอนการจัดทำข้อกำหนด การออกแบบ การเขียนโปรแกรมและการทดสอบ แต่สำหรับวิทยานิพนธ์ฉบับนี้จะวัดความพยายามของการจัดทำข้อกำหนด การออกแบบโครงสร้าง และความพยายามในการอิมพลีเมนท์ซึ่งการอิมพลีเมนท์จะรวมถึงขั้นตอนของการออกแบบ (Design) การเขียนโปรแกรม (Coding) การทดสอบ (Testing) และการแก้ไขข้อบกพร่อง (Debug)

ความพยายามรวมของแต่ละรายงานเท่ากับ ผลรวมของความพยายามในการจัดทำข้อกำหนด ความพยายามในการออกแบบโครงสร้างและความพยายามในการอิมพลีเมนท์ แสดงดังสมการที่ (3.8)

$$\text{Report} = Effort\_Req + Effort\_DesStr + Effort\_Imp \tag{3.8}$$

โดยที่ Report หมายถึง ความพยายามทั้งหมดของการปรับปรุงรายงานแต่ละประเภท

$Effort\_Req$ หมายถึง ความพยายามในการจัดทำข้อกำหนดของรายงาน

$Effort\_DesStr$ หมายถึง ความพยายามในการออกแบบโครงสร้างของรายงาน

$Effort\_Imp$ หมายถึง ความพยายามในการอิมพลีเมนท์ ประกอบด้วย ขั้นตอนการออกแบบ (Design) เขียนโปรแกรม (Coding) ทดสอบ (Testing) และแก้ไขข้อบกพร่อง (Debug)

โดยจะการวัดความพยายามในการบำรุงรักษาซอฟต์แวร์ จะวัดจากทุกๆ เวอร์ชั่นของการบำรุงรักษาซอฟต์แวร์

3.2.3 มาตรวัดซอฟต์แวร์เชิงวัตถุ

นอกจากจะวัดการบำรุงรักษาซอฟต์แวร์จากขนาดของการบำรุงรักษาซอฟต์แวร์แล้ว ในแต่ละขั้นตอนของการบำรุงรักษาซอฟต์แวร์ จะวัดค่าจากมาตรวัดซอฟต์แวร์เชิงวัตถุ จำนวนทั้งหมด 6 มาตรวัด ซึ่งประกอบด้วย

3.2.3.1 มาตรวัดจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนัก (Weighted Methods per Class: WMC) เป็นมาตรวัดที่ใช้วัดความซับซ้อนภายในคลาสของแต่ละเมธอด โดยวิธีวัดจะคำนวณจากผลรวมของจำนวนเมธอดต่อคลาส โดยแต่ละเมธอดจะคำนวณค่าความซับซ้อนของแต่ละเมธอดแตกต่างกันไป ซึ่งเราเรียกค่าความซับซ้อนแต่ละเมธอดว่า "ค่าความซับซ้อนของไซโคลเมติก (Cyclomatic Complexity)" โดยมาตรวัดจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนัก (WMC) [25] คำนวณได้ดังสมการที่ (3.9)

$$\mathrm{WMC} = \sum_{i=0}^{n} Cyclomatic_i \tag{3.9}$$

โดยที่ เมื่อพิจารณาในแต่ละคลาส จะประกอบด้วยเมธอด $M1, M2, \ldots Mn$

WMC หมายถึง จำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนักในแต่ละคลาส

$Cyclomatic_i$ หมายถึง ค่าความซับซ้อนไซโคลเมติกของแมคแคบในเมธอดที่ i

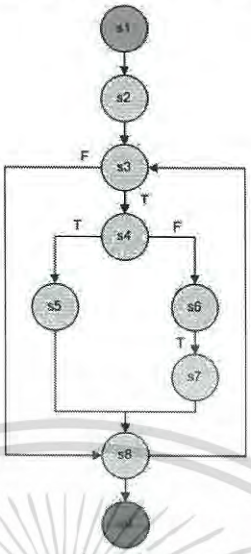
ค่า WMC ที่ได้จะเป็นค่า จำนวนเมธอดต่อคลาสของซอฟต์แวร์ทั้งระบบ โดยสามารถคำนวณค่าความซับซ้อนของไซโคลเมติกได้จากภายในแต่ละเมธอด โดยจะวัดจากจากเส้นทางการเชื่อมต่อในกราฟควบคุมกระแส (Flow Graph) โดยแต่ละโหนด (node: n) จะแทนคำสั่งของโปรแกรมหนึ่งคำสั่ง และแต่ละคำสั่งจะเชื่อมต่อกันด้วยเส้นเชื่อมโหนด (edge: e) แสดงการคำนวณค่าความซับซ้อนของไซโคเมติก เช่นเดียวกับสมการที่ (3.6)

แสดงตัวอย่างการคำนวณค่าความซับซ้อนของไซโคลเมติกแสดงดังรูปที่ 3.7 โดยจะคำนวณค่าภายในเมธอด *CalculatePoint()*

```
//s1  public static int CalculatePoint(List<CartItem> list)
//    {
//s2        int p = 0;
//s3        foreach (CartItem c in list)
//        {
//s4              if (c.redeemType == CartItem.redeemTypeEnum.byPoint)
//              {
//s5                    p += c.product.fullPoint;
//              }
//s6              else if (c.redeemType
//        ==CartItem.redeemTypeEnum.byPointAndPrice)
//              {
//s7                    p += c.product.partPoint;
//              }
//s8        }
//s9        return p;
//    }
```

รูปที่ 3.7 แสดงตัวอย่างเมธอด CalculatePoint

จากรูปที่ 3.7 สามารถนำมาเขียนกราฟควบคุมกระแส (Flow Graph) ดังรูปที่ 3.8



รูปที่ 3.8 กราฟควบคุมกระแสที่ได้จากเมธอด CalculatePoint

จากรูปที่ 3.8 แสดงกราฟที่ได้จากการแปลงโปรแกรมในเมธอด *CalculatePoint()* จากนั้นจะนำมาคำนวณตามสมการที่ (4) โดย e มีค่า 11, n มีค่า 9 ดังนั้นค่าความซับซ้อนของเมธอด *CalculatePoint()* มีค่าเท่ากับ 11-9+2 เท่ากับ 4

โดยค่าของ WMC จะมีความสัมพันธ์กับความสามารถในการบำรุงรักษา (Maintainability) ในลักษณะแปรผกผัน ซึ่งหากมีจำนวนเมธอดต่อคลาสสูงความสามารถในการบำรุงรักษาจะลดลง

3.2.3.2 มาตรวัดความเข้าคู่กันระหว่างวัตถุ (Coupling Between Objects: CBO) มาตรวัดความเข้าคู่กันระหว่างวัตถุหรือ CBO คำนวณได้จากผลรวมของจำนวนคลาสอื่นที่เข้ามาเรียกใช้คลาสที่กำลังพิจารณาและจำนวนคลาสอื่นที่ถูกเรียกใช้โดยคลาสที่กำลังพิจารณา แสดงตัวอย่างการคำนวณค่า CBO ดังรูปที่ 3.9

```
1  public double calculateVAT(double rate){
2      double price = calculateTotal();
3      return price * rate;
4  }
5  public double calculateTotal(){
6      double total = 0D;
7      for(Item item: getCart())
8          price += item.getPrice();
9      return total;
10 }
11 private boolean isCartValid(){
12     return !getCart().isEmpty();
13 }
```

รูปที่ 3.9 ตัวอย่างการคำนวณค่า CBO

จากรูปที่ 3.9 สามารถคำนวณค่า CBO ได้เท่ากับ 2 ดังนี้ (1) พิจารณาจาก เมธอด *CalculateTotal ()* มีการเรียกใช้คลาส *Item* และคลาส *Item* เองมีการ return กลับมายังเมธอด *getCart()* ในคลาสที่กำลังพิจารณา ดังนั้นจากตัวอย่างนี้จึงคำนวณค่า CBO เท่ากับ 2 เป็นต้น

ซึ่งค่าที่ได้คำนวณได้จากมาตรวัด CBO ควรจะมีค่าน้อยเนื่องจากจะทำให้การบำรุงรักษาทำได้ง่ายขึ้น ตรงกันข้ามหากค่า CBO มีค่ามาก ความสามารถในการบำรุงรักษาซอฟต์แวร์จะลดลง

3.2.3.3 มาตรวัดการตอบสนองต่อคลาส (Response for a Class: RFC) เป็นมาตรวัดที่ใช้วัดความซับซ้อนของคลาสในแง่ของการเรียกใช้เมธอด วิธีการวัดจะวัดโดยนับจำนวนเมธอดที่สร้างไว้ใช้งานภายในคลาส และจำนวนเมธอดที่เรียกใช้งานจากคลาสภายนอก ตัวอย่างการคำนวณค่าของ RFC แสดงดังรูปที่ 3.9

จากรูปที่ 3.9 เมื่อนับจำนวนเมธอดที่สร้างขึ้นภายในคลาสจำนวน 3 เมธอด ประกอบด้วย *calculateVAT() calculateTotal()* และ *isCartValid()* และมีการเรียกใช้เมธอดภายในคลาส 2 เมธอด ได้แก่ *calculateTotal()* และ *getCart()* การเรียกใช้เมธอดภายนอกได้แก่ *getPrice()* ในคลาส Item จำนวน 1 เมธอด ดังนั้นค่า RFC ในคลาสนี้มีค่าเท่ากับ 6 เป็นต้น

โดยค่าที่ได้จากมาตรวัด RFC จะเป็นค่าที่บ่งบอกถึงความซับซ้อนระหว่างเมธอดภายในคลาสดังนั้นหากค่าที่คำนวณได้จากมาตรวัด RFC มีค่ามากจะทำให้การบำรุงรักษาซอฟต์แวร์ลดลงเนื่องจากคลาสนั้นมีความซับซ้อนมาก ในทางกลับกันหากค่าที่คำนวณได้มีค่าน้อยแสดงว่าคลาสนั้นง่ายต่อการบำรุงรักษา

3.2.3.4 มาตรวัดระดับการขาดความสัมพันธ์ภายในคลาส (Lack of Cohesion in Methods: LCOM) มาตรวัดระดับการขาดความสัมพันธ์ภายในคลาสเป็นมาตรวัดที่ใช้ในการวัด Cohesion โดยจะวัดความสัมพันธ์ระหว่างเมธอดและตัวแปรที่ประกาศไว้ภายในคลาส (Instance Variable) โดยคำนวณตามสมการที่ (3.10)

$$\mathrm{LCOM} = \frac{\langle P \rangle - |M|}{1 - |M|} \tag{3.10}$$

กำหนดให้ M คือ เมธอดภายในคลาส

F คือ จำนวนตัวแปรที่ประกาศภายในคลาส (Instance Variable)

$P\langle f \rangle$ คือ จำนวนเมธอดที่เรียกใช้ตัวแปรที่อยู่ภายในคลาส f เป็นสมาชิกของ F

$\langle P \rangle$ คือ ค่าเฉลี่ยของ $P\langle f \rangle$ ต่อ F

ตัวอย่างการคำนวณค่า LCOM แสดงดังรูปที่ 3.10

```
1   package com.pms.pm.util;
2   import java.awt.Paint;
3   import org.jfree.chart.renderer.category.BarRenderer;
4   public class CustomRenderer extends BarRenderer{
5   // /** The colors. */
6           private Paint[] colors;
7           public CustomRenderer(final Paint[] colors) {
8               this.colors = colors;
9           }
10          @Override
11          public Paint getItemPaint(final int row, final int column) {
12              return this.colors[column % this.colors.length];
13          }
14  }
```

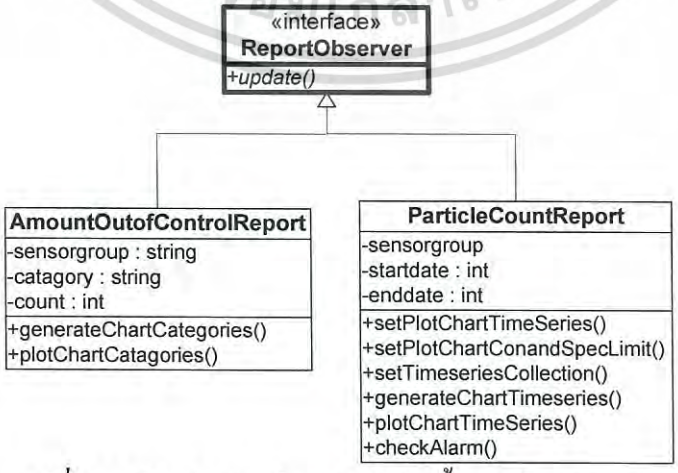
รูปที่ 3.10 ตัวอย่างการคำนวณค่าของการขาดระดับความสัมพันธ์ภายในคลาส

จากรูปที่ 3.10 แสดงการคำนวณค่า LCOM ได้โดยเมธอดภายในคลาส มีจำนวน 2 เมธอด ได้แก่ *CunstomRenderer()* และ *getItemPaint()* ตัวแปรที่ประกาศภายในคลาส (instance variable) มีจำนวน 1 ตัวแปร ได้แก่ color ดังนั้นสามารถคำนวณค่าของ LCOM ได้ดังนี้

$M = 2, F = 1, \langle P \rangle = 1+1/2 = 1$ ดังนั้นค่าของ LCOM มีค่าเท่ากับ 1.00 เป็นต้น

โดยค่า LCOM ที่คำนวณได้จะเกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์ หากค่า LCOM มีค่าสูงความสามารถในการบำรุงรักษาจะลดลงดังนั้น ดังนั้นซอฟต์แวร์ที่สามารถบำรุงรักษาได้ง่ายควรจะมีค่าของ LCOM ต่ำ

3.2.3.5 มาตรวัดระดับความลึกของการสืบทอดคุณสมบัติของคลาส (Depth of Inheritance Tree: DIT) เป็นมาตรวัดเพื่อหาระดับความลึกของการสืบทอดคุณสมบัติของคลาสโดยจะพิจารณาในแต่ละคลาส โดยค่าของระดับความลึกของการสืบทอดคุณสมบัติจะได้จาก การนับจำนวนของระดับชั้น (Level) การสืบทอดคุณสมบัติของแต่ละคลาสที่กำลังพิจารณา ตัวอย่างแสดงดังรูปที่ 3.11



รูปที่ 3.11 ตัวอย่างการพิจารณาลำดับชั้นการสืบทอดคุณสมบัติ

จากรูปที่ 3.11 แสดงลำดับชั้นของการสืบทอดคุณสมบัติ พิจารณาคลาส *ReportObserver* เป็นคลาสแม่ คลาส *AmountOutfoControlReport* และคลาส *ParticleCountReport* เป็นคลาสลูก ซึ่งสามารถคำนวณค่าระดับความลึกของการสืบทอดคุณสมบัติ (DIT) ดังตารางที่ 3.1

ตารางที่ 3.1 คำนวณค่าระดับความลึกของการสืบทอดคุณสมบัติของแต่ละคลาส

คลาสที่ทำการพิจารณา	**ค่า DIT**	**หมายเหตุ**
ReportObserver	0	เนื่องจาก *ReportObserver* เป็น root คลาส
AmountOutofControlReport	1	เนื่องจากความยาวจากคลาส *AmountOutofControlReport* ถึง root คลาสมี 1 ลำดับชั้น
ParticleCountReport	1	เนื่องจากความยาวจากคลาส *ParticleCountReport* ถึง root คลาสมี 1 ลำดับชั้น
** *หมายเหตุ* ในกรณีที่คลาสหนึ่งทำการสืบทอดคุณสมบัติจากหลายคลาส จะเลือกคลาสที่มีเส้นทางจากคลาสนั้นถึง root คลาสยาวที่สุด		

ค่าที่ได้จากมาตรวัด DIT แสดงถึงความสามารถในการบำรุงรักษา โดยที่ค่า DIT มีค่ามากหมายถึงคลาสมีความซับซ้อนมากขึ้น ส่งผลให้ความสามารถด้านการบำรุงรักษาและการปรับขยายระบบในอนาคตลดลง

3.2.3.6 มาตรวัดจำนวนคลาสลูก (Number of Children: NOC) มาตรวัด NOC จะคำนวณโดยการนับจำนวนคลาสลูกทั้งหมดที่สืบทอดมาจากคลาสแม่ (Root Class) ที่กำลังพิจารณาในขณะนั้น

จากรูปที่ 3.11 หากพิจารณาคลาส *ReportObserver* สามารถคำนวณค่า NOC ได้เท่ากับ 2 เนื่องจาก คลาส *ReportObserver* ประกอบด้วยคลาสลูกจำนวน 2 คลาส ได้แก่ คลาส *AmountOutofControlReport* และคลาส *ParticleCountReport* เป็นต้น

ค่าที่ได้จากมาตรวัด NOC แสดงถึงความสามารถในการนำกลับมาใช้ใหม่ โดยหากค่าของ NOC มีค่ามากหมายถึงจะมีการนำคลาสกลับมาใช้ใหม่ได้น้อยลงซึ่งจะส่งผลถึงความสามารถในการบำรุงรักษาระบบจะลดลดเช่นกัน

3.3 การออกแบบการทดลองสำหรับการวัดการบำรุงรักษาซอฟต์แวร์

การออกแบบการทดลองสำหรับการวัดการบำรุงรักษาซอฟต์แวร์ ประกอบด้วย หัวข้อ 3.3.1 การออกแบบประเภทรายงาน โดยแบ่งออกเป็นรายงานระดับง่ายและรายงานระดับปานกลาง หัวข้อ 3.3.2 แบบฟอร์มที่ใช้สำหรับการบันทึกข้อมูลระหว่างการพัฒนาและการบำรุงรักษาซอฟต์แวร์

3.3.1 การออกแบบประเภทรายงาน

การศึกษาจะแบ่งประเภทรายงานตามความต้องการของผู้ใช้งาน ออกเป็น 2 ประเภท ได้แก่

3.3.1.1 รายงานระดับง่าย (Easy Level) เป็นการสร้างรายงานโดยนำข้อมูลมาสร้างรายงานโดยไม่ผ่านการ Process เพิ่มเติม มีลักษณะดังนี้

- มีการใช้คำสั่งจากฐานข้อมูลเบื้องต้น เช่น คำสั่ง Select
- มีการ Join ตารางเพื่อให้ได้ข้อมูลสำหรับการนำมาสร้างกราฟ
- เป็นรายงานระดับรายละเอียดสำหรับเจ้าหน้าที่ปฏิบัติการ

3.3.1.2 รายงานระดับปานกลาง (Medium Level) เป็นรายงานที่ไม่สามารถนำข้อมูลมาสร้างรายงานได้ทันที อาจต้องผ่านการคำนวณค่าหรือ Process เพิ่มเติม มีลักษณะดังนี้

- มีการใช้คำสั่งจากฐานข้อมูลเบื้องต้น เช่น คำสั่ง Select
- มีการใช้บางฟังก์ชั่นสำหรับคำนวณข้อมูลเพิ่มเติม เช่น ฟังก์ชั่น sum ฟังก์ชั่น count เป็นต้น
- เป็นรายงานระดับภาพรวมของเจ้าหน้าที่ปฏิบัติการและรายละเอียดสำหรับเจ้าหน้าที่ระดับบริหาร

โดยการออกแบบเพื่อเพิ่มเติมรายงาน ประกอบด้วยรายงาน แสดงดังตารางที่ 3.2

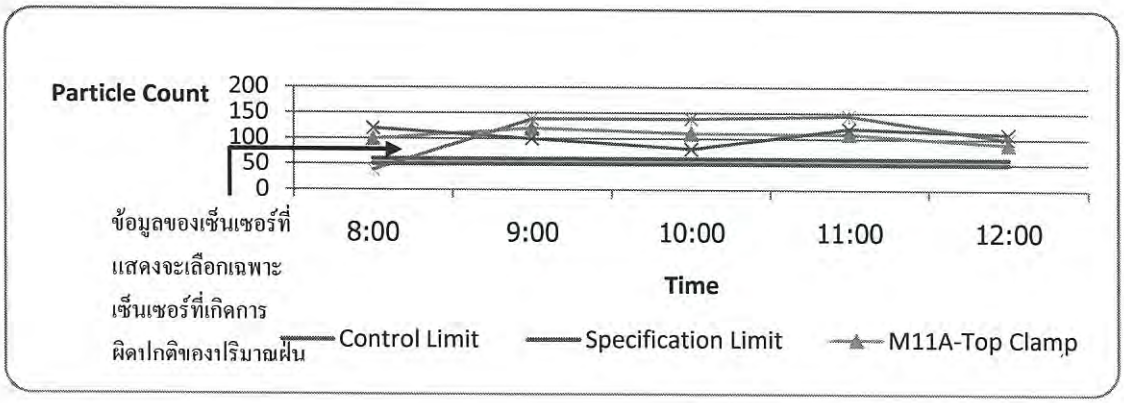
ตารางที่ 3.2 ประเภท รายชื่อและรายละเอียดของรายงานที่ใช้เป็นกรณีศึกษา

ชื่อประเภทรายงาน	**ชื่อรายงาน**
รายงานระดับง่าย (Easy Level)	รายงานความผิดปกติของเซ็นเซอร์ในแต่ละพื้นที่การผลิต
	รายงานความผิดปกติของจำนวนครั้งการเกิดปริมาณฝุ่นเกินกำหนดของแต่ละเซ็นเซอร์ย้อนหลัง 1 เดือน
รายงานระดับปานกลาง (Medium Level)	รายงานเปรียบเทียบจำนวนที่ผิดปกติระหว่างพื้นที่การผลิต
	รายงานการเปรียบเทียบปริมาณฝุ่นเฉลี่ยในแต่ละวันของพื้นที่การผลิต

จากตารางที่ 3.2 แสดงรายละเอียดของรายงานแต่ละประเภทดังนี้

1) รายงานความผิดปกติของเซ็นเซอร์ในแต่ละพื้นที่การผลิต

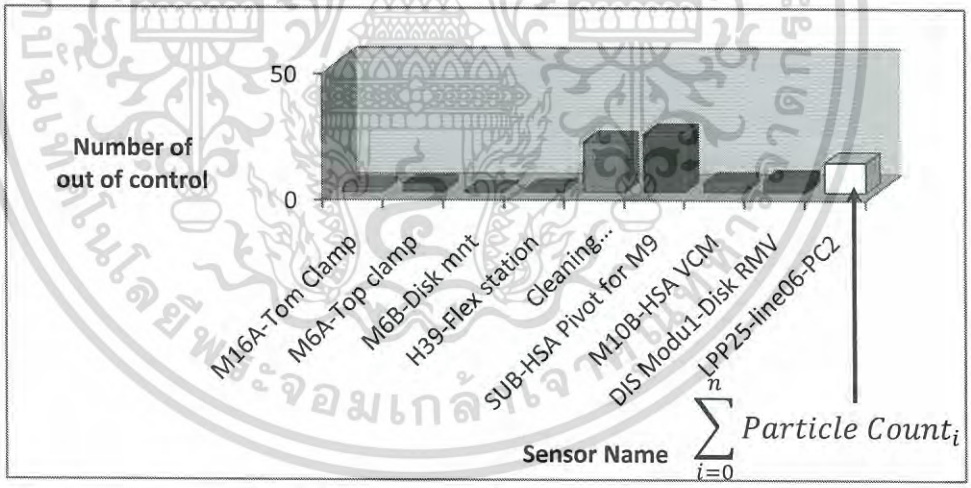
ผู้ใช้งานจะเลือกพื้นที่การผลิต (Process Area) เพื่อแสดงรายงานความผิดปกติของการเกิดปริมาณฝุ่นละอองของเซ็นเซอร์ (Sensor) ที่มีค่าปริมาณฝุ่นเกินค่ามาตรฐาน แสดงดังรูปที่ 3.12



รูปที่ 3.12 รายงานความผิดปกติของเซ็นเซอร์จากการเลือกพื้นที่การผลิต

2) รายงานความผิดปกติของจำนวนครั้งการเกิดปริมาณฝุ่นเกินกำหนดของแต่ละเซ็นเซอร์ย้อนหลัง 1 เดือน

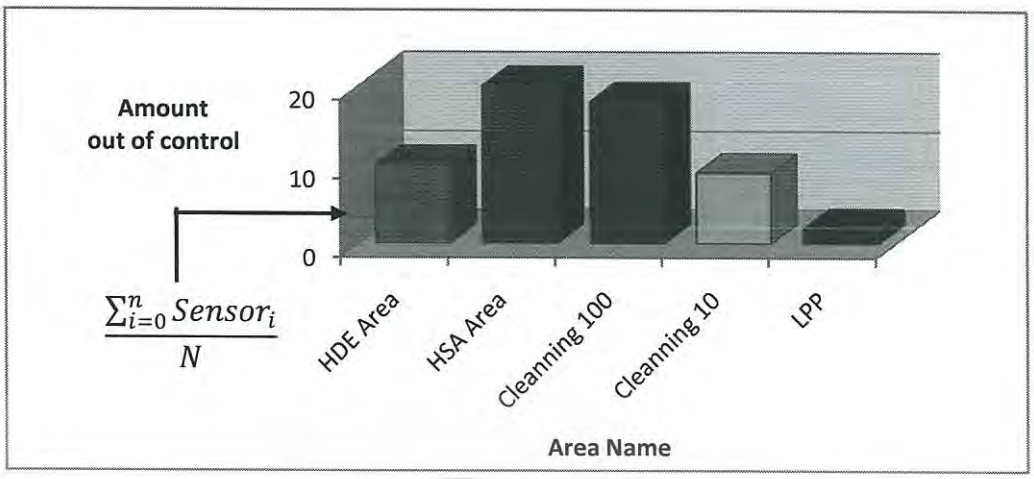
แสดงข้อมูลการเกิดจำนวนครั้งของการเกิดปริมาณฝุ่นเกินกำหนด (out of control) ของแต่ละเซ็นเซอร์โดยรวบรวมข้อมูลย้อนหลัง 1 เดือน เพื่อแสดงข้อมูลของแต่ละเซ็นเซอร์เปรียบเทียบกัน แสดงรายงานดังรูปที่ 3.13



รูปที่ 3.13 ตัวอย่างรายงานความผิดปกติของจำนวนครั้งการเกิดปริมาณฝุ่นเกินกำหนดของแต่ละเซ็นเซอร์ย้อนหลัง 1 เดือน

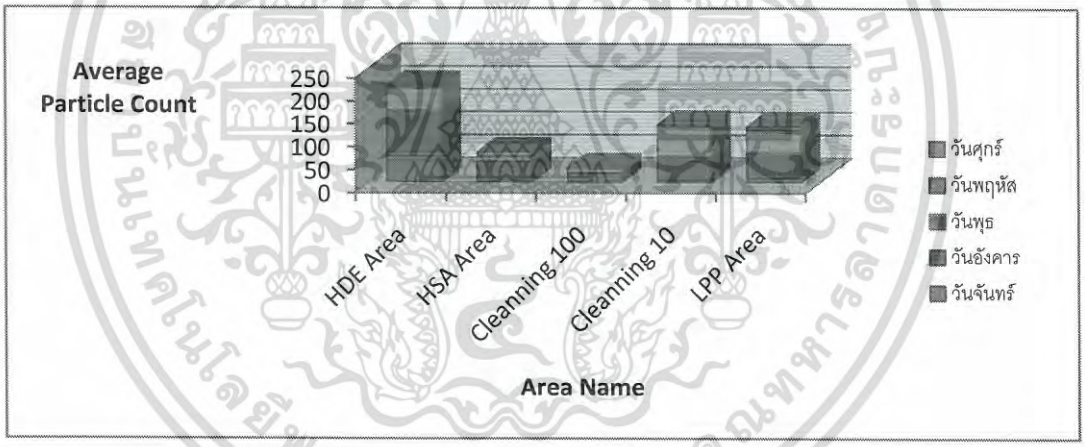
3) รายงานเปรียบเทียบจำนวนที่ผิดปกติระหว่างพื้นที่การผลิต

แสดงรายงานการเปรียบเทียบจำนวนครั้งที่เกิดความผิดปกติ (out of control) ระหว่างพื้นที่การผลิต โดยแสดงเป็นกราฟแท่ง (Bar Chart) โดยสามารถเลือก ช่วงเวลา วัน สัปดาห์ เดือน ได้ แสดงลักษณะของรายงาน ดังรูปที่ 3.14



รูปที่ 3.14 ตัวอย่างรายงานการเปรียบเทียบจำนวนที่ผิดปกติระหว่างพื้นที่การผลิต

4) รายงานการเปรียบเทียบปริมาณฝุ่นเฉลี่ยในแต่ละวันของพื้นที่การผลิต แสดงรายงานข้อมูลปริมาณฝุ่นละอองเฉลี่ยของแต่ละวันภายในสัปดาห์ ในแต่ละพื้นที่การผลิต (Area) แสดงรายงานดังรูปที่ 3.15



รูปที่ 3.15 ตัวอย่างรายงานการเปรียบเทียบจำนวนที่ผิดปกติระหว่างพื้นที่การผลิตภายในสัปดาห์

3.3.2 การออกแบบฟอร์มการบันทึกข้อมูลระหว่างการพัฒนาและบำรุงรักษาซอฟต์แวร์

ในระหว่างการพัฒนาและบำรุงรักษาซอฟต์แวร์จะทำการบันทึกข้อมูลจริง ของการเปลี่ยนแปลงจำนวนบรรทัด คลาสและเมธอดของซอฟต์แวร์ เพื่อจะใช้ในการคำนวณมาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์

ซึ่งรูปแบบสำหรับการเก็บรวบรวมข้อมูลมีหลายเทคนิค เช่น การใช้โปรแกรมวิเคราะห์การสร้างแบบฟอร์มสำหรับให้ผู้วิเคราะห์หรือโปรแกรมเมอร์บันทึกข้อมูลและการสัมภาษณ์หรือสอบถามจากโปรแกรมเมอร์ซึ่งแต่ละวิธีมีข้อดีข้อเสียแตกต่างกันไปโดยการเลือกใช้งานควรเลือก

ให้เหมาะกับโครงการซอฟต์แวร์ของแต่ละโครงการ สำหรับกรณีศึกษานี้จะใช้วิธีการจดบันทึกจากโปรแกรมเมอร์โดยตรงแสดงแบบฟอร์มการเก็บข้อมูล ดังตารางที่ 3.3

ตารางที่ 3.3 ข้อมูลที่ใช้สำหรับการจัดเก็บเพื่อทดสอบประสิทธิภาพการออกแบบทั้งสองรูปแบบ

ลำดับ	ข้อมูลที่เก็บ	รายละเอียด
1	วันที่ทำการปรับปรุง	ข้อมูลของวันที่ทำการปรับปรุงระบบ
2	โมเดลที่ปรับปรุง	ประเภทของการบำรุงรักษาซอฟต์แวร์ แบ่งออกเป็น 2 ประเภท (1) การบำรุงรักษาโดยการเพิ่มเติมความต้องการของผู้ใช้งาน และ (2) การบำรุงรักษาโดยการปรับปรุงโครงสร้างของระบบ
3	ผู้ทำการปรับปรุง	ชื่อของโปรแกรมเมอร์ที่ทำการปรับปรุงระบบ
4	ประเภทรายงาน	ประเภทของรายงาน 1) รายงานระดับง่าย และ 2) รายงานระดับปานกลาง
5	ชื่อรายงาน	ชื่อรายงานที่ทำการปรับปรุง
6	รายละเอียดของรายงาน	อธิบายรายละเอียดการสร้างรายงาน ว่าใช้สำหรับสร้างรายงานใด ลักษณะเป็นอย่างไร
7	ประเภทการบันทึก	ประเภทที่ใช้ในการปรับปรุง แบ่งออกเป็น การจัดทำความต้องการ การออกแบบโครงสร้าง และการ Implement โดยการ Implement จะรวมถึง ขั้นตอนของการออกแบบ (Design) การเขียนโปรแกรม (Coding) การทดสอบ (Testing) และการแก้ไขข้อผิดพลาด (Debug)
8	เวลาเริ่มต้น	เวลาที่เริ่มต้นที่ใช้สำหรับกิจกรรมหรือการทำงานต่างๆ เช่น ใช้ในการจัดทำข้อกำหนด ใช้ในการออกแบบ (Design) ใช้ในการเขียนโปรแกรม (Coding) ใช้ในการทดสอบรายงานให้ใช้ได้ตามความต้องการของผู้ใช้ (Testing)
9	เวลาสิ้นสุด	เวลาที่พัฒนาแต่ละกิจกรรมเสร็จสิ้น ได้แก่ เวลาที่ออกแบบเสร็จสิ้น เวลาที่เขียนโปรแกรมเสร็จสิ้น และเวลาที่ทดสอบเสร็จสิ้น
10	รายละเอียดงาน	รายละเอียดของกิจกรรมหรืองานที่ทำเพื่อการปรับปรุงซอฟต์แวร์ โดยแต่ละกิจกรรมจะแบ่งออกเป็น 2 ประเภท ได้แก่ งานใหม่ (New Work: N) เป็นงานหรือกิจกรรมที่เริ่มทำเป็นครั้งแรก และ งานต่อเนื่อง (Continue Work: C) เป็นงานหรือกิจกรรมที่ทำต่อหรือได้มีการเริ่มงานนั้นไปแล้ว

ลำดับ	ข้อมูลที่เก็บ	รายละเอียด
11	Comment	ใช้สำหรับบันทึกเพิ่มเติม เช่น อาจระบุปัญหาต่างๆ หรือผลกระทบต่างๆ จากการทำงานหรือกิจกรรมนั้นๆ เป็นต้น
12	ชื่อคลาส	เป็นการบันทึกชื่อคลาสที่ทำการปรับปรุง โดยแบ่งประเภทของการปรับปรุงคลาสเป็น 2 ประเภท ได้แก่ การเพิ่มคลาสใหม่ (New Class: N) และการปรับปรุงคลาส (Edit Class: E)
13	ชื่อเมธอด	เป็นการบันทึกชื่อเมอดที่มีการปรับปรุง โดยแบ่งการปรับปรุงประเภทของเมธอดออกเป็น 3 ประเภท ได้แก่ การเพิ่มเมธอดใหม่ การลบเดิมที่มีอยู่ และการเปลี่ยนแปลงแก้ไขเมธอดโดยจะทำการบันทึกจำนวนบรรทัดภายในเมธอดที่มีการเพิ่ม ลบ หรือเปลี่ยนแปลง

แสดงแบบฟอร์มการบันทึกข้อมูลระหว่างทำบำรุงรักษาซอฟต์แวร์ซอฟต์แวร์ และแบบฟอร์มบันทึกข้อมูลการเปลี่ยนแปลงของคลาสและเมธอด ดังนี้

แบบฟอร์มบันทึกข้อมูลการบำรุงรักษาซอฟต์แวร์

วันที่ทำการปรับปรุง.....

☐ บำรุงรักษาโดยการเพิ่มเติมความต้องการของผู้ใช้งาน ☐ บำรุงรักษาโดยการปรับปรุงโครงสร้างของระบบ

ผู้ทำการปรับปรุง.....

ประเภทรายงาน ☐ รายงานระดับง่าย ☐ รายงานระดับปานกลาง

ชื่อรายงาน.....

รายละเอียดของรายงาน.....

.....

.....

ประเภทการบันทึก ☐ การจัดทำความต้องการ ☐ การออกแบบโครงสร้าง ☐ การ Implement

No.	เวลาเริ่มต้น	เวลาสิ้นสุด	รายละเอียดงาน			Comment
			N	C		

หมายเหตุ

N (New Work) หมายถึง เริ่มทำงานนั้นเป็นครั้งแรก

C (Continue work) หมายถึง ทำงานนั้นต่อหรือมีการเริ่มทำงานนั้นไปแล้ว

Implement หมายถึง ขั้นตอนการออกแบบ เขียนโปรแกรม ทดสอบและแก้ไขข้อผิดพลาด

ตัวอย่างแบบฟอร์มบันทึกข้อมูลการเปลี่ยนแปลงของคลาสและเมธอด

No.	คลาส			เมธอด				Comment
	ประเภท		ชื่อ	ชื่อ	จำนวนบรรทัดที่ (LOC)			
	N	E			เพิ่ม	ลบ	เปลี่ยนแปลง	
1	✓		*ParticleReport*	*SunParticleCount()*	*30*			
				GenerateReport()	*27*			
2		✓	*AmountOutofControlReport*	*PlotChartCatagories()*		*50*	*13*	

หมายเหตุ

N (New Class) หมายถึง การเพิ่มคลาสใหม่

E (Edit Class) หมายถึง การเปลี่ยนแปลงแก้ไขคลาส

Comment หมายถึง อาจมีการกระทบกับคลาส หรือ เชื่อมโยงกับเมธอดอื่นๆ

บทที่ 4
ผลการทดลองและวิเคราะห์เปรียบเทียบการบำรุงรักษาซอฟต์แวร์

ในบทนี้จะกล่าวถึงผลการทดลองและการอภิปรายผลการเปรียบเทียบค่าความคลาดเคลื่อนของค่าประมาณของเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์ ของมาตรวัด 5 มาตรวัด ได้แก่

1. มาตรวัดขนาดการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจากจำนวนบรรทัดของซอฟต์แวร์ที่เปลี่ยนแปลงไป (MS-LOC)
2. มาตรวัดขนาดการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจากจำนวนคลาสของซอฟต์แวร์ที่เปลี่ยนแปลงไป (MS-TC)
3. มาตรวัดขนาดการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจากจำนวนเมธอดของซอฟต์แวร์ที่เปลี่ยนแปลงไป (MS-TM)
4. มาตรวัดขนาดการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจากค่าเฉลี่ยของจำนวนเมธอดต่อคลาสของซอฟต์แวร์ที่เปลี่ยนแปลงไป (MS-MC)
5. มาตรวัดขนาดการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจากจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนักของซอฟต์แวร์ที่เปลี่ยนแปลง (MS-WMC)

นอกจากจะเปรียบเทียบความผิดพลาดของค่าประมาณเวลาที่คลาดเคลื่อนไปจากค่าจริงที่ใช้ในการบำรุงรักษาซอฟต์แวร์แล้ว จะเปรียบเทียบผลการวัดการบำรุงรักษาซอฟต์แวร์จากมาตรวัดซอฟต์แวร์เชิงวัตถุ ได้แก่ มาตรวัดจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนัก (Weighted Methods per Class: WMC) มาตรวัดการตอบสนองต่อคลาส (Response for a Class: RFC) มาตรวัดความเข้าคู่กันระหว่างวัตถุ (Coupling between Object: CBO) มาตรวัดการขาดระดับความสัมพันธ์ภายในคลาส (Lack of Cohesion in Method: LCOM) มาตรวัดระดับความลึกของการสืบทอดคุณสมบัติของคลาส (Depth of Inheritance Tree: DIT) และมาตรวัดจำนวนคลาสลูก (Number of Children: NOC) ที่นำมาใช้วัดการบำรุงรักษาซอฟต์แวร์ในแต่ละเวอร์ชั่น (Maintenance Version)

การทดสอบจะใช้กรณีศึกษาของ "ระบบการออกรายงานอัตโนมัติเพื่อสนับสนุนงานตรวจจับฝุ่นละอองในอุตสาหกรรมการผลิตฮาร์ดดิสก์ (Automated Report Generation System for Particle Monitoring in Hard Disk Drive Industry)" โดยหัวข้อ 4.1 กล่าวถึงการบันทึกข้อมูลการเปลี่ยนแปลงของจำนวนบรรทัด คลาส และเมธอดในระหว่างที่ทำการพัฒนาและบำรุงรักษาซอฟต์แวร์ หัวข้อที่ 4.2 จะนำค่าที่ได้จากการบันทึกมาคำนวณขนาดของการบำรุงรักษาซอฟต์แวร์ (Maintenance Size) ของแต่ละมาตรวัด หัวข้อที่ 4.3 จะหาค่าประมาณของเวลาที่ใช้ในการบำรุงรักษาที่ได้จากมาตรวัดแต่ละมาตรวัดและเปรียบเทียบกับเวลาจริงที่ใช้ในการบำรุงรักษาซอฟต์แวร์เพื่อหามาตรวัดที่เหมาะสมจากขนาดของการบำรุงรักษาซอฟต์แวร์ หัวข้อ 4.4 สรุปมาตร

วัดที่เหมาะสมโดยพิจารณาจากค่าเปอร์เซ็นต์ความผิดพลาดของแต่ละมาตรวัด และหัวข้อ 4.5 จะแสดงค่าที่ได้จากมาตรวัดโครงสร้างและความซับซ้อนของคลาสในแต่ละเวอร์ชั่นของการบำรุงรักษาซอฟต์แวร์

4.1 การบันทึกข้อมูลระหว่างการพัฒนาและบำรุงรักษาซอฟต์แวร์

ในระหว่างที่พัฒนาและบำรุงรักษาซอฟต์แวร์ในแต่ละเวอร์ชั่นจะทำการเก็บข้อมูลจริง ได้แก่ จำนวนบรรทัด คลาสและเมธอดที่เปลี่ยนแปลงไปในการปรับปรุงระบบแต่ละครั้ง รวมทั้งจะวัดซอฟต์แวร์ในแต่ละเวอร์ชั่นโดยใช้มาตรวัดเชิงวัตถุ (Object-Oriented Metrics) จำนวน 6 มาตรวัด โดยแบ่งประเภทของการบำรุงรักษาออกเป็นงาน 2 ลักษณะ ได้แก่ 1) การเพิ่มเติมความต้องการของผู้ใช้งาน (Functional Enhancement) และ 2) การบำรุงรักษาปรับเปลี่ยนโดยการจัดระเบียบซอร์ดโค้ด (Refactoring Source Code) แสดงรายละเอียด ดังนี้

4.1.1 การบำรุงรักษาซอฟต์แวร์โดยการเพิ่มเติมความต้องการของผู้ใช้งาน (Functional Enhancement)

Maintenance Version1 การปรับปรุงระบบโดยการเพิ่มเติมรายงานระดับง่าย ได้แก่ รายงานแสดงปริมาณฝุ่นละอองของแต่ละเซ็นเซอร์

Maintenance Version2 การปรับปรุงระบบโดยการเพิ่มเติมรายงานระดับง่าย ได้แก่ รายงานความผิดปกติของจำนวนครั้งการเกิดปริมาณฝุ่นของแต่ละเซ็นเซอร์ย้อนหลัง 1 เดือน

Maintenance Version3 การปรับปรุงระบบโดยการเพิ่มเติมรายงานระดับปานกลาง ได้แก่ รายงานเปรียบเทียบจำนวนที่ผิดปกติระหว่างพื้นที่การผลิต

Maintenance Version4 การปรับปรุงระบบโดยการเพิ่มเติมรายงานระดับปานกลาง ได้แก่ รายงานการเปรียบเทียบปริมาณฝุ่นเฉลี่ยในแต่ละวันของพื้นที่การผลิต

4.1.2 การบำรุงรักษาซอฟต์แวร์โดยจัดระเบียบซอร์ดโค้ด (Refactoring Source Code)

Refactoring Source Code Version การปรับปรุงและจัดระเบียบซอร์ดโค้ด

โดยแต่ละการทดลอง จะแสดงโครงสร้างคลาส แบ่งออกเป็น 2 เวอร์ชั่นได้แก่ เวอร์ชั่นตั้งต้น (Initial Version) และเวอร์ชั่นบำรุงรักษา (Maintenance Version) จากนั้นจะคำนวณขนาดของการบำรุงรักษาซอฟต์แวร์ (Maintenance Size) เพื่อพิจารณามาตรวัดที่เหมาะสมกับโครงสร้างคลาสแต่ละประเภท โดยจะหามาตรวัดที่เหมาะสมจากการเปรียบเทียบจากเวลาที่ประมาณการได้จากขนาดของการบำรุงรักษาซอฟต์แวร์ (Predict Maintenance Time) จากมาตรวัดขนาดการบำรุงรักษาทั้ง 5 ประเภท

แสดงรายละเอียดและโครงสร้างคลาสของการบำรุงรักษาซอฟต์แวร์ ของแต่ละเวอร์ชั่นการบำรุงรักษา ดังนี้

Maintenance Version 1 การปรับปรุงระบบโดยการเพิ่มรายงานระดับง่ายครั้งที่ 1

Initial Version เป็นเวอร์ชั่นดั้งเดิมของระบบการสร้างรายงาน ประกอบด้วยรายงานพื้นฐาน ได้แก่ รายงานแสดงปริมาณฝุ่นละอองของแต่ละเซ็นเซอร์ (ParticleCountReport) รายงานสรุปจำนวนครั้งที่ปริมาณฝุ่นละอองเกินมาตรฐาน (AmountOutofControlReport) และรายงานสรุปจำนวนครั้งที่เกิดทริกเกอร์ (AmountTriggerReport)

Maintenance Version1 เป็นเวอร์ชั่นที่ได้ทำการปรับปรุง (เวอร์ชั่นการบำรุงรักษา) ต่อจากเวอร์ชั่นดั้งเดิม (Initial Version) โดยเพิ่มเติมรายงานระดับง่าย ตามความต้องการของผู้ใช้งาน ได้แก่ รายงานความผิดปกติของเซ็นเซอร์ในแต่ละพื้นที่การผลิต (ExceptionEachProcessAreaReport)

Maintenance Version 2 การปรับปรุงระบบโดยการเพิ่มรายงานระดับง่ายครั้งที่ 2

Maintenance Version 1 เป็นเวอร์ชั่นของระบบการสร้างรายงานที่ประกอบด้วยรายงานพื้นฐานจำนวน 3 รายงาน และรายงานระดับง่ายจำนวน 1 รายงาน

Maintenance Version 2 เป็นเวอร์ชั่นที่ได้ทำการปรับปรุง (เวอร์ชั่นการบำรุงรักษา) ต่อจาก Maintenance Version1 โดยเพิ่มเติมรายงานระดับง่าย ตามความต้องการของผู้ใช้งาน 1 รายงาน ได้แก่ รายงานความผิดปกติของจำนวนครั้งการเกิดปริมาณฝุ่นของแต่ละเซ็นเซอร์ย้อนหลัง 1 เดือน (ExceptionForMonthReport)

Maintenance Version 3 การปรับปรุงระบบโดยการเพิ่มรายงานระดับปานกลางครั้งที่ 1

Maintenance Version2 เป็นเวอร์ชั่นของระบบการสร้างรายงานที่ประกอบด้วยรายงานพื้นฐานและรายงานระดับง่าย

Maintenance Version3 เป็นเวอร์ชั่นที่ได้ทำการปรับปรุง (เวอร์ชั่นการบำรุงรักษา) ต่อจาก Initial Version โดยเพิ่มเติมรายงานระดับปานกลาง ได้แก่ รายงานเปรียบเทียบจำนวนที่ผิดปกติระหว่างพื้นที่การผลิต (ExceptionAvgSensorOutofReport)

Maintenance Version4 การปรับปรุงระบบโดยการเพิ่มรายงานระดับปานกลางครั้งที่ 2

Maintenance Version3 เป็นเวอร์ชั่นดั้งเดิมของระบบการออกรายงาน ให้มีหน้าที่การทำงานโดยสร้างรายงานระดับพื้นฐานจำนวน 3 รายงาน รายงานระดับง่ายจำนวน 2 รายงานและระดับปานกลางจำนวน 1 รายงาน

Maintenance Version4 เป็นเวอร์ชั่นที่ได้ทำการปรับปรุง (เวอร์ชั่นการบำรุงรักษา) ต่อจาก Maintenance Version3 โดยเพิ่มเติมรายงานระดับปานกลาง ได้แก่ รายงานการเปรียบเทียบปริมาณฝุ่นเฉลี่ยในแต่ละวันของพื้นที่การผลิต (ExceptionOfWeekInAreaReport)

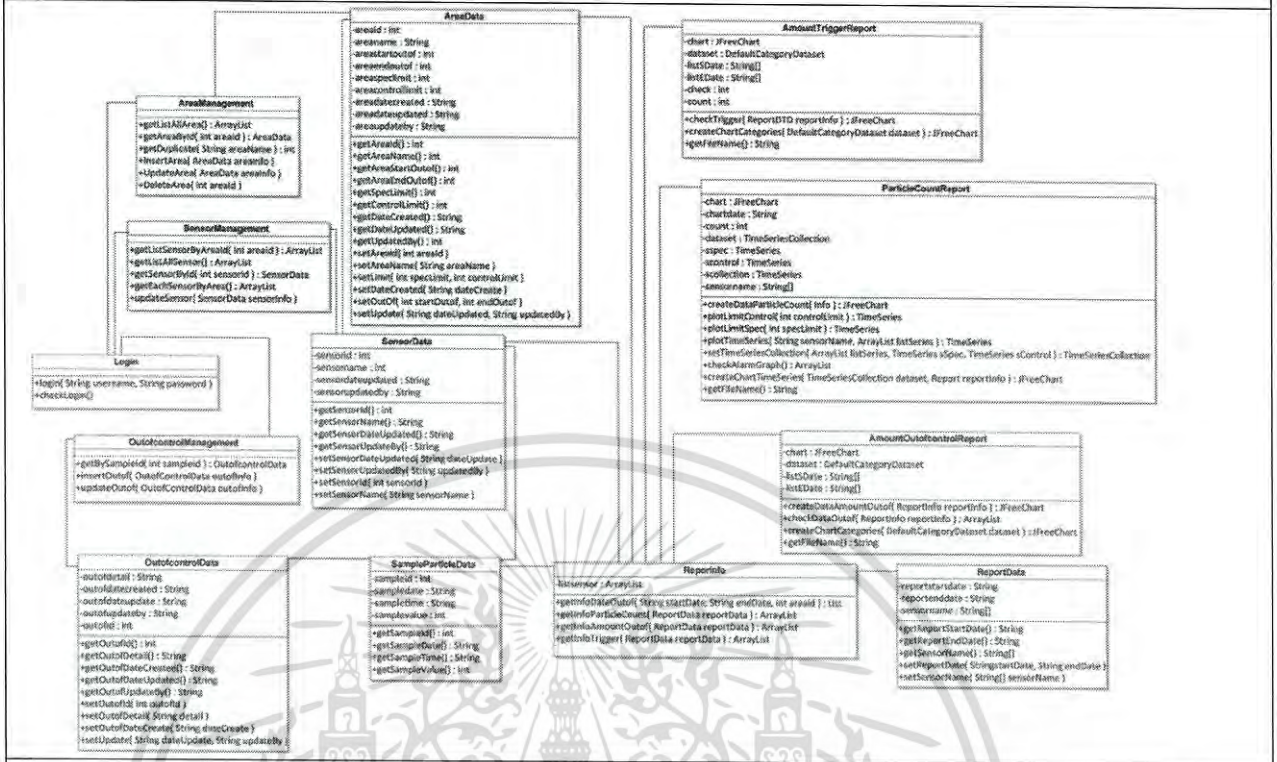
Refactoring Source Code Version การปรับปรุงและจัดระเบียบซอร์ดโค้ด

Maintenance Version4 เป็นเวอร์ชั่นดั้งเดิมของระบบการออกรายงาน สามารถสร้างรายงานระดับพื้นฐาน รายงานระดับง่ายและระดับปานกลางได้

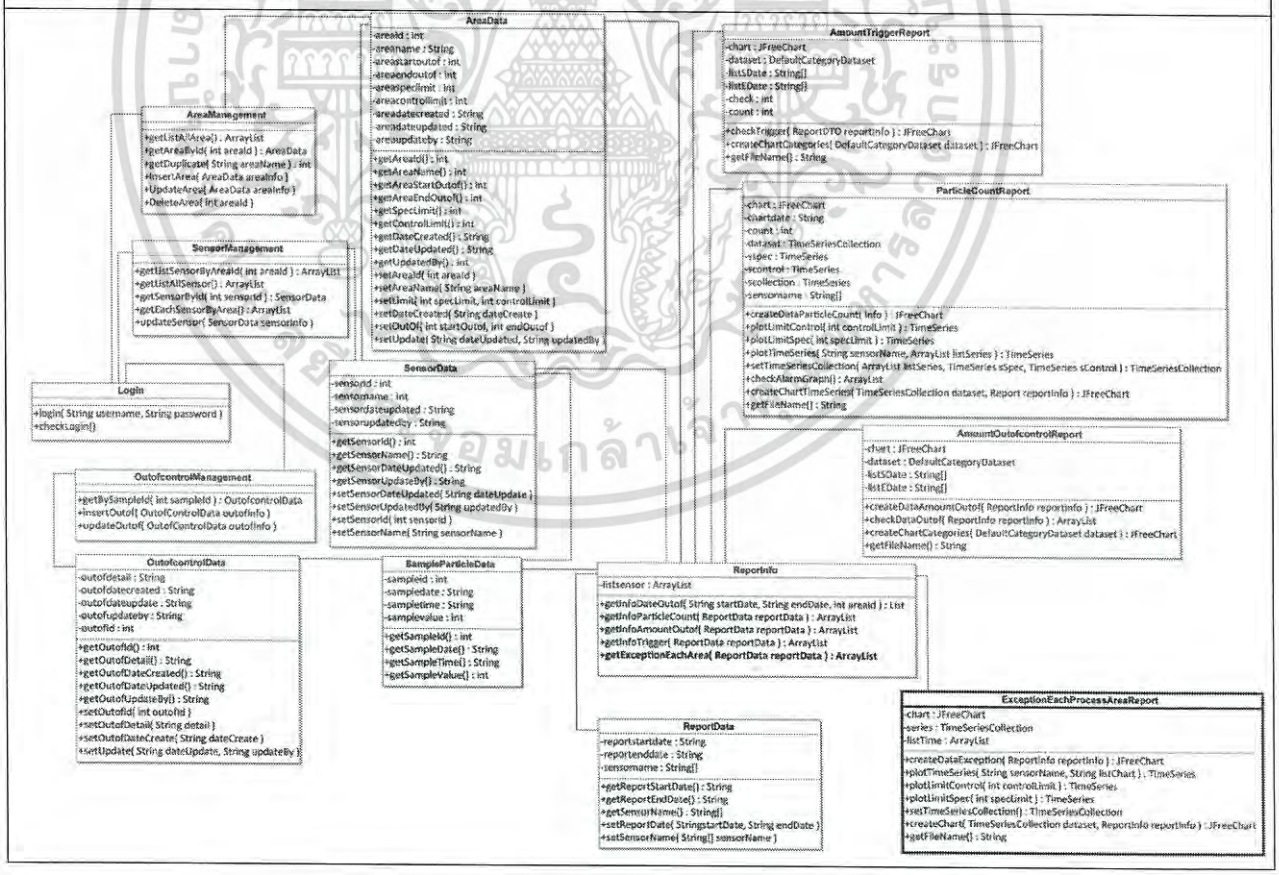
Refactoring Source Code Version เป็นเวอร์ชั่นที่ได้ทำการปรับปรุง (เวอร์ชั่นการบำรุงรักษา) ต่อจากเวอร์ชั่นดั้งเดิม โดยการปรับปรุงโครงสร้างคลาส โดยเพิ่มคลาส และแยกเมธอดที่เรียกใช้งานจากคลาสหลายคลาส (Re-Engineering) เพื่อให้ง่ายต่อการเพิ่มเติมรายงานในอนาคตมากยิ่งขึ้น

โดยแสดงการเปรียบเทียบระหว่างโครงสร้างคลาสเบื้องต้นและโครงสร้างคลาสหลังจากการบำรุงรักษา ดังรูปที่ 4.1 ถึง 4.5

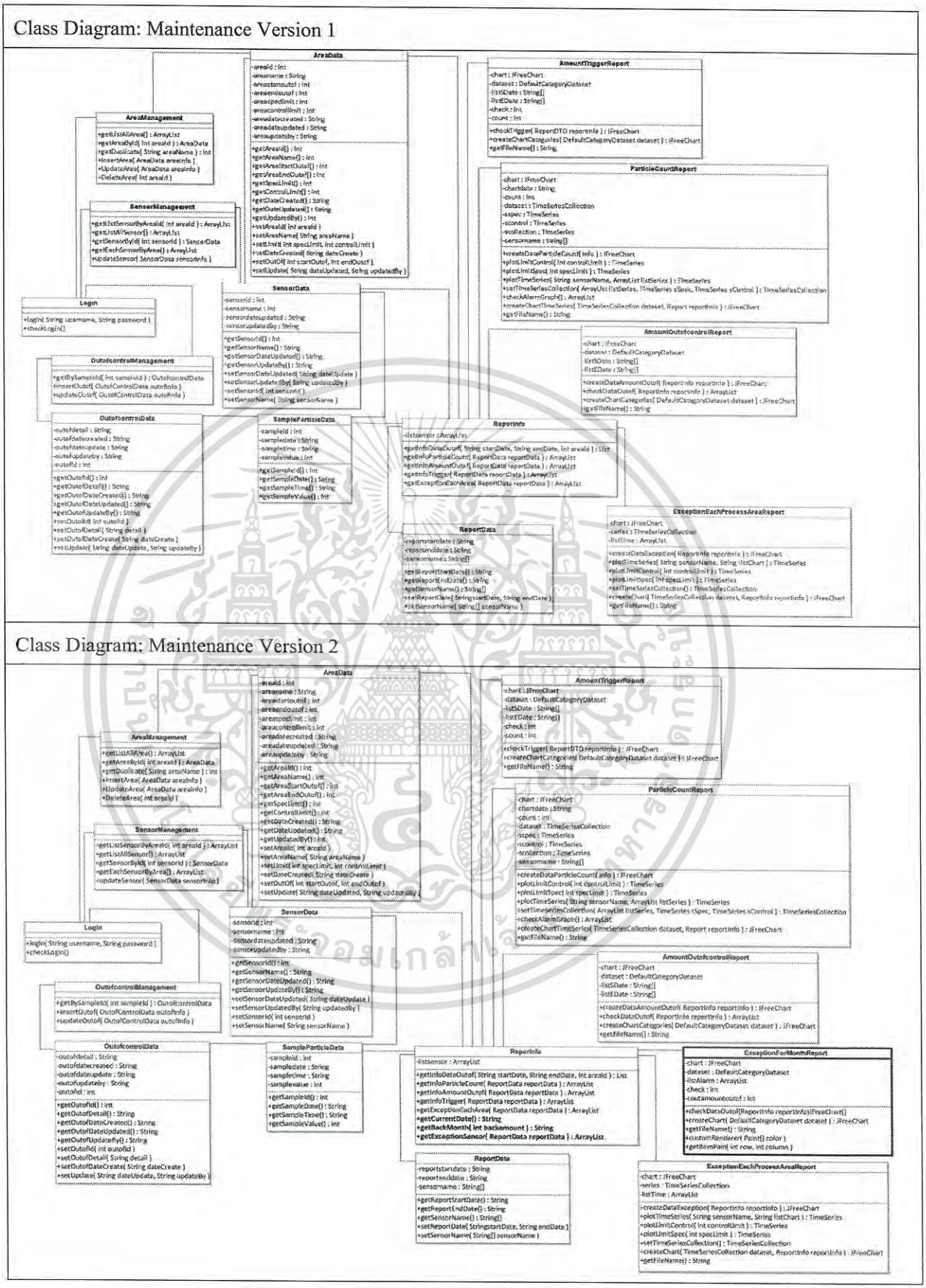
Class Diagram: Initial Version



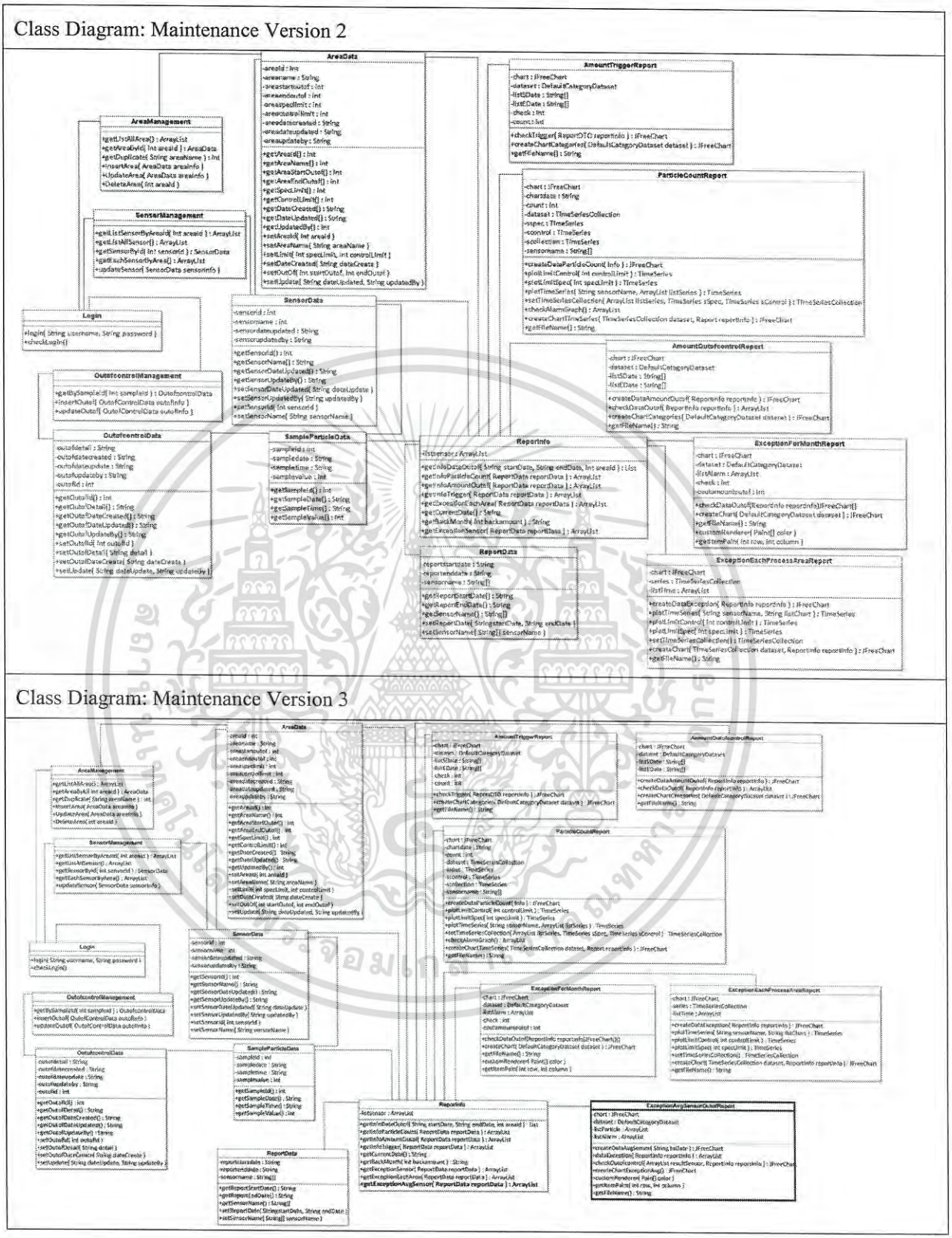
Class Diagram: Maintenance Version 1



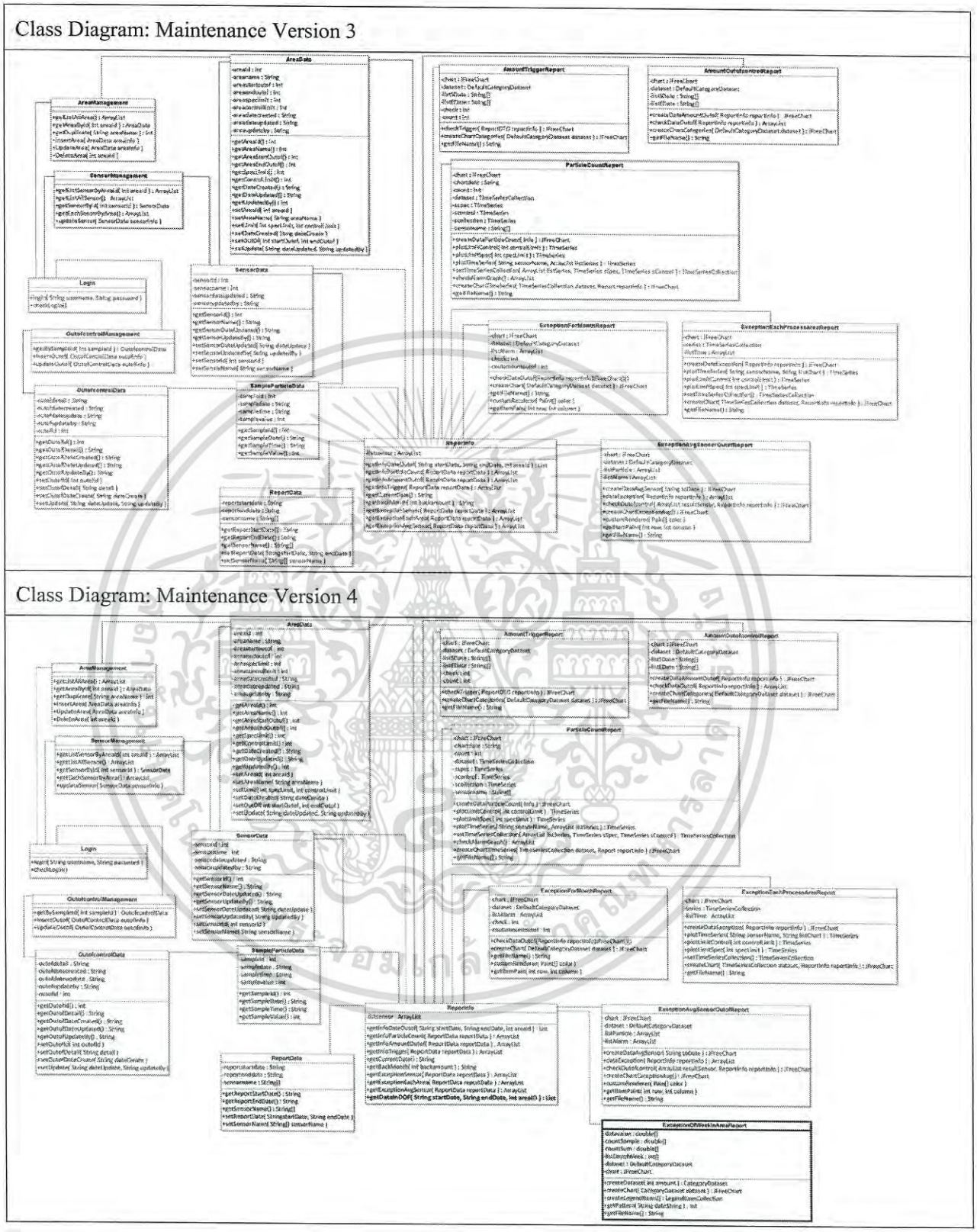
รูปที่ 4.1 การเปรียบเทียบความแตกต่างโครงสร้างคลาสในการบำรุงรักษาซอฟต์แวร์ครั้งที่ 1



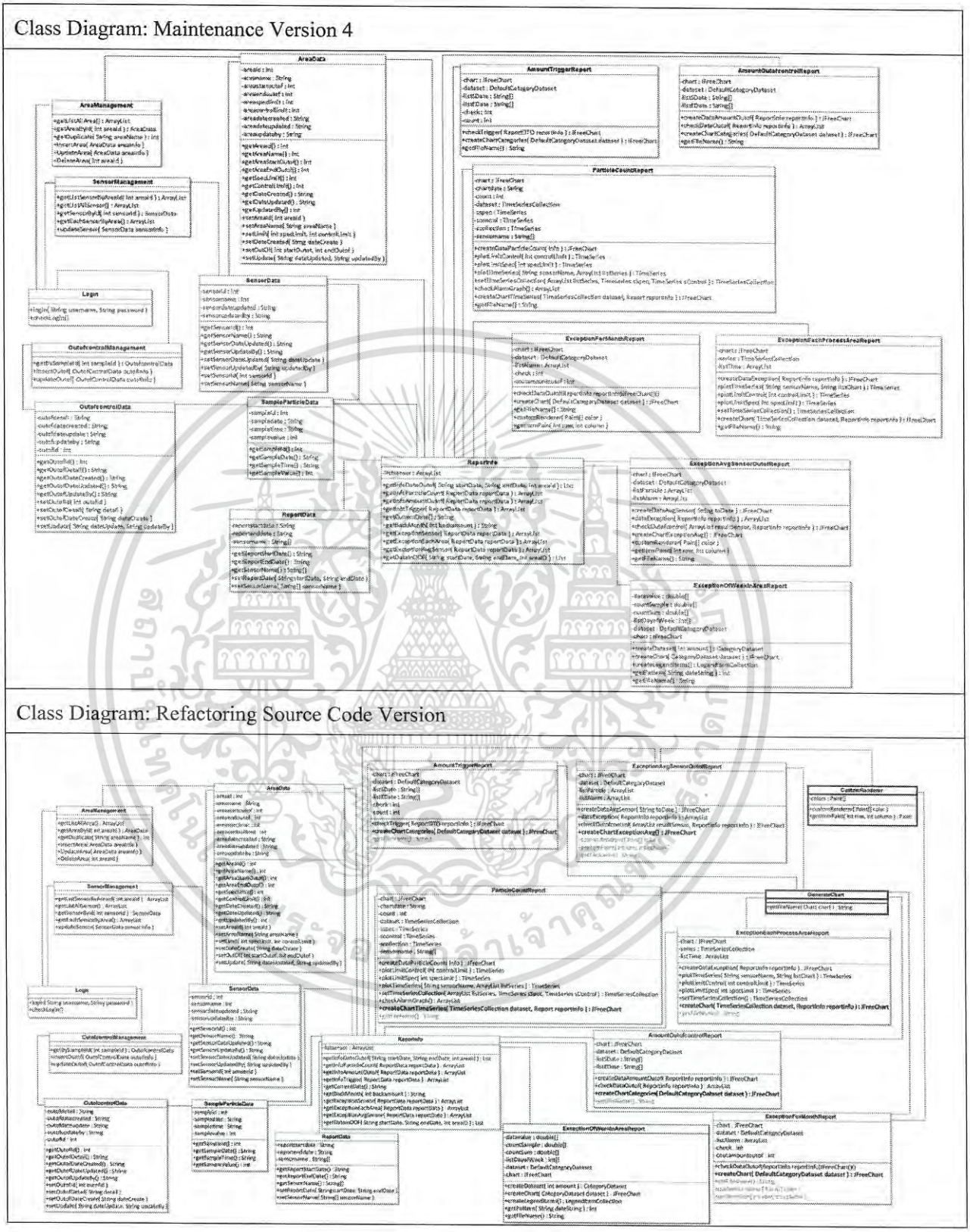
รูปที่ 4.2 การเปรียบเทียบความแตกต่างโครงสร้างคลาสในการบำรุงรักษาซอฟต์แวร์ครั้งที่ 2



รูปที่ 4.3 การเปรียบเทียบความแตกต่างโครงสร้างคลาสในการบำรุงรักษาซอฟต์แวร์ครั้งที่ 3



รูปที่ 4.4 การเปรียบเทียบความแตกต่างโครงสร้างคลาสในการบำรุงรักษาซอฟต์แวร์ครั้งที่ 4



รูปที่ 4.5 การเปรียบเทียบความแตกต่างโครงสร้างคลาสของการจัดเรียงซอร์ดโค้ด

ซึ่งในระหว่างที่ทำการพัฒนาและปรับปรุงระบบในแต่ละเวอร์ชั่นจะทำการบันทึกข้อมูล เช่น จำนวนบรรทัด จำนวนคลาส จำนวนเมธอด ของซอฟต์แวร์ที่เปลี่ยนแปลงไปในแต่ละเวอร์ชั่นของการบำรุงรักษาซอฟต์แวร์ แสดงข้อมูลการเปลี่ยนแปลง จำนวนบรรทัด คลาส และเมธอดของการบำรุงรักษาซอฟต์แวร์แต่ละเวอร์ชั่น ดังตารางที่ 4.1

ตารางที่ 4.1 ข้อมูลจากการบันทึกค่าระหว่างการพัฒนาและบำรุงรักษาซอฟต์แวร์

	จำนวนบรรทัด (LOC)					**จำนวนคลาส (Class)**					**จำนวนเมธอด (Method)**				
	Ini.	Tot.	A.	M.	D.	Ini.	Tot.	A.	M.	D.	Ini.	Tot.	A.	M.	D.
Maintenance Version 1 เพิ่มรายงานระดับง่าย ครั้งที่ 1															
Mtn. Ver1	1,221	1,402	181	-	-	13	14	1	1	-	76	84	8	-	-
Maintenance Version 2 เพิ่มรายงานระดับง่าย ครั้งที่ 2															
Mtn. Ver2	1,402	1,609	207	-	-	14	15	1	1	-	84	92	5	3	-
Maintenance Version 3 เพิ่มรายงานระดับปานกลาง ครั้งที่ 1															
Mtn. Ver3	1,609	1,778	140	29	-	15	16	1	1	-	92	100	7	1	-
Maintenance Version 4 เพิ่มรายงานระดับปานกลาง ครั้งที่ 2															
Mtn. Ver4	1,778	1,956	178	-	-	16	17	1	1	-	100	105	5	-	-
Refactoring Source Code															
Refac	1,956	1,862	52	15	123	17	19	2	7	-	105	93	3	7	11

Ini. หมายถึง เวอร์ชั่นตั้งต้น

Mtn Ver. หมายถึง เวอร์ชั่นการบำรุงรักษา

Tot. หมายถึง ผลรวมของเวอร์ชั่นบำรุงรักษา

A. หมายถึง จำนวนที่เพิ่มขึ้น

M. หมายถึง จำนวนที่เปลี่ยนแปลง/แก้ไข

D. หมายถึง จำนวนที่ลบออก

4.2 การเปรียบเทียบขนาดของการบำรุงรักษาซอฟต์แวร์ในแต่ละมาตรวัด

เมื่อบันทึกค่าในระหว่างที่ทำการพัฒนาและบำรุงรักษาซอฟต์แวร์จะนำค่าที่ได้จากการบันทึกมาคำนวณขนาดของการบำรุงรักษาซอฟต์แวร์ (Maintenance Size) โดยแบ่งมาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์ ออกเป็น 5 ประเภท ได้แก่ การวัดการบำรุงรักษาซอฟต์แวร์จากจำนวนบรรทัด (MS-LOC) การวัดจากจำนวนคลาส (MS-TC) การวัดจากเมธอด (MS-TM) การวัดจากจำนวนเมธอดต่อคลาส (MS-MC) และการวัดการบำรุงรักษาจากจำนวนคลาสต่อเมธอดโดยคิดค่าถ่วงน้ำหนัก (MS-WMC) แสดงรายละเอียดมาตรวัดและค่าที่ได้จากการวัดของแต่ละมาตรวัดการบำรุงรักษาซอฟต์แวร์ในแต่ละ Maintenance Version ดังตารางที่ 4.2

ตารางที่ 4.2 การคำนวณขนาดของการบำรุงรักษาซอฟต์แวร์ของ 5 มาตรวัดในแต่ละเวอร์ชั่น

ประเภทของมาตรวัด	บำรุงรักษาโดยการเพิ่มเติมความต้องการของผู้ใช้งาน				บำรุงรักษาโดยการจัดเรียงซอร์ดโค้ด
	Maintenance Version 1	Maintenance Version 2	Maintenance Version 3	Maintenance Version 4	
$MS-LOC=\frac{LOC_{Added}+LOC_{Modified}}{\text{Total Line of Code in the Initial System}}$	$\frac{181}{1,221}$ $= 0.1482$	$\frac{207}{1,402}$ $= 0.1476$	$\frac{140+29}{1,609}$ $= 0.1050$	$\frac{178}{1,778}$ $= 0.1001$	$\frac{\lvert 52-123\rvert+15}{1,956}$ $= 0.0435$
$MS-TC=\frac{\lvert Class_{Added}-Class_{Deleted}\rvert+Class_{Modified}}{\text{Total Class in the Initial System}}$	$\frac{1+1}{13}$ $= 0.1538$	$\frac{1+1}{14}$ $= 0.1429$	$\frac{1+1}{15}$ $= 0.1333$	$\frac{1+1}{16}$ $= 0.1250$	$\frac{7+2}{17}$ $= 0.5294$
$MS-TM=\frac{\lvert Method_{Added}-Method_{Deleted}\rvert+Method_{Modified}}{\text{Total Method in the Initial System}}$	$\frac{8}{76}$ $= 0.1053$	$\frac{5+3}{84}$ $= 0.0952$	$\frac{7+1}{92}$ $= 0.0870$	$\frac{5}{100}$ $= 0.0500$	$\frac{\lvert 3-11\rvert+7}{105}$ $= 0.1429$
$MS-MC=\frac{(Method_{Changed})/\text{Number of Class Change}}{\text{Total Class in the Initial System}}$	$\frac{8/14}{76/13}$ $= 0.0977$	$\frac{(5+3)/15}{84/14}=$ 0.0888	$\frac{(7+1)/16}{92/15}$ $= 0.0815$	$\frac{5/17}{100/16}$ $= 0.0471$	$\frac{(\lvert 3-11\rvert+7)/19}{105/17}$ $= 0.1278$
$MS-WMC=\frac{(\text{WeightedMethodChanged})/\text{Number of Class Change}}{\text{Total Class in the Initial System}}$	$\frac{20/14}{159/13}$ $= 0.1168$	$\frac{24/15}{191/14}$ $= 0.1173$	$\frac{20/16}{203/15}$ $= 0.0924$	$\frac{16/17}{223/16}$ $= 0.0675$	$\frac{12/19}{239/17}$ $= 0.0449$

จากนั้นจะนำค่าขนาดของการบำรุงรักษาซอฟต์แวร์ในแต่ละมาตรวัดของแต่ละเวอร์ชั่นการบำรุงรักษาซอฟต์แวร์ในตารางที่ 4.2 มาหาค่าประมาณของเวลาที่ใช้สำหรับการบำรุงรักษาซอฟต์แวร์ (Estimated Maintenance Time)

เนื่องจาก We Li [6] ได้ทำการวิเคราะห์ความสัมพันธ์ระหว่าง ความพยายามในการบำรุงรักษาซอฟต์แวร์ (Maintenance Effort) และจำนวนบรรทัดของซอฟต์แวร์ที่เปลี่ยนแปลงไปจากระบบเดิม พบว่าปัจจัยทั้งสองมีความสัมพันธ์กัน ดังนั้นจึงมีแนวคิดที่จะหาค่าประมาณของเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์จาก จำนวนบรรทัด จำนวนคลาสและเมธอดที่เปลี่ยนแปลงไปด้วย โดยคำนณ EMT ได้จากสัดส่วนระหว่าง ขนาดของการบำรุงรักษาซอฟต์แวร์ (MS) และเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์ แสดงตัวอย่างดังนี้

กำหนดให้ขนาดของการบำรุงรักษา (MS) มีค่าเท่ากับ 0.50 หมายถึง หากมีการพัฒนาระบบโดยการสร้างคลาส 100 คลาสในขณะที่ใช้เวลาในการพัฒนา 20 ชั่วโมงแล้วจะใช้เวลาในการบำรุงรักษาซอฟต์แวร์ (Maintenance Time) เท่ากับครึ่งหนึ่งของเวลาที่ใช้ในการพัฒนาจริง (20 x 50% เท่ากับ 10 ชั่วโมง) เป็นต้น แสดงการคำนวณดังสมการที่ (4.1)

$$MS = \text{สัดส่วนของการเปลี่ยนแปลงระหว่างบำรุงรักษา} = 100:20 = 50:x = \frac{100}{20} = \frac{50}{x} = 10 \text{ ชั่วโมง} \quad (4.1)$$

x หมายถึงเวลาประมาณที่ใช้ในการบำรุงรักษาซอฟต์แวร์

แสดงค่าของขนาดของการบำรุงรักษาซอฟต์แวร์ (Maintenance Size: MS) และค่าจากการประมาณเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์ (Estimated Maintenance Time: EMT)

หมายเหตุ: เวลาที่ใช้ในการพัฒนาจริง (Development Time) จะเป็นเวลาที่ใช้ในการพัฒนาตอนแรกสำหรับกรณีศึกษานี้เท่านั้น (เบื้องต้นเวลาในการพัฒนาเท่ากับ 55 ชั่วโมง) ดังนั้นการคำนวณค่า EMT ควรเปลี่ยนแปลงค่าของการพัฒนาจริงตามกรณีศึกษาที่นำไปปรับใช้

แสดงการคำนวณค่าของ MS และ EMT ดังตารางที่ 4.3

ตารางที่ 4.3 ค่าของขนาดของการบำรุงรักษาซอฟต์แวร์และค่าประมาณของเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์ของ 5 มาตรวัดในแต่ละเวอร์ชั่นการบำรุงรักษาซอฟต์แวร์

ประเภทของ MS	บำรุงรักษาโดยการเพิ่มเติมความต้องการของผู้ใช้งาน								บำรุงรักษาโดยการจัดเรียงซอร์ดโค้ด	
	Maintenance Version 1		Maintenance Version 2		Maintenance Version 3		Maintenance Version 4			
	MS	EMT	MS	EMT	MS	EMT	MS	EMT	MS	EMT
1. MS-LOC	0.1482	8 h. 09 m.	0.1476	8 h. 07 m.	0.1050	5 h. 46 m.	0.1001	5 h. 30 m.	0.0435	2 h. 23 m.
2. MS-TC	0.1538	8 h. 27 m.	0.1429	7 h. 51 m.	0.1333	7 h. 19 m.	0.1250	6 h. 52 m.	0.5294	29 h. 07 m.
3. MS-TM	0.1053	5 h. 47 m.	0.0952	5 h. 01 m.	0.0870	4 h. 47 m.	0.0500	2 h. 45 m.	0.1429	7 h. 51 m.
4. MS-MC	0.0977	5 h. 22 m.	0.0888	4 h. 53 m.	0.0815	4 h. 28 m.	0.0471	2 h. 35 m.	0.1278	7 h. 01 m.
5. MS-WMC	0.1168	6 h. 25 m.	0.1173	6 h. 27 m.	0.0924	5 h. 04 m.	0.0675	3 h. 42 m.	0.0449	2 h. 28 m.
Actual Maintenance Time	3 h. 34 min.		5 h. 47 min.		3 h. 56 min.		12 h. 15 min.		1 h. 48 min.	

4.3 เปรียบเทียบความผิดพลาด (%) จากค่าประมาณของเวลาที่ใช้บำรุงรักษาซอฟต์แวร์

หลังจากคำนวณค่าประมาณของเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์แล้วจะเปรียบเทียบกับเวลาจริงที่ใช้ในการบำรุงรักษาซอฟต์แวร์ (Actual Maintenance Time) จะได้เวลาที่คำนวณผิดพลาดไปจากเวลาจริงของแต่ละมาตรวัดซอฟต์แวร์ เพื่อหามาตรวัดที่เหมาะสมที่จะใช้ในการบำรุงรักษาซอฟต์แวร์ โดยจะแยกประเภทของการบำรุงรักษาออกเป็นงาน 2 ลักษณะ ได้แก่ 1) การบำรุงรักษาซอฟต์แวร์โดยการเพิ่มเติมความต้องการ (Functional Enhancement) และ 2) การปรับปรุงโครงสร้างของระบบโดยการจัดระเบียบซอร์ดโค้ด (Refactoring Source Code)

แสดงความผิดพลาดจากการประมาณการเวลาของแต่ละมาตรวัดของการบำรุงรักษาซอฟต์แวร์โดยการเพิ่มเติมความต้องการประเภทรายงาน ดังตารางที่ 4.4

ตารางที่ 4.4 ความผิดพลาดของค่าประมาณของเวลาเมื่อเปรียบเทียบกับเวลาจริงที่ใช้ในการบำรุงรักษาซอฟต์แวร์ (ชั่วโมง/นาที) สำหรับการบำรุงรักษาซอฟต์แวร์โดยการเพิ่มเติมความต้องการ

ประเภทของ MS	ค่าความผิดพลาดเมื่อเปรียบเทียบกับเวลาจริงที่ใช้ในการบำรุงรักษาซอฟต์แวร์โดยการเพิ่มเติมความต้องการของผู้ใช้งาน			
	Maintenance Version 1	Maintenance Version 2	Maintenance Version 3	Maintenance Version 4
1. MS-LOC	4 h. 35 m.	2 h. 19 m.	1 h. 50 m.	- 6 h. 45 m.
2. MS-TC	4 h. 53 m.	2 h. 04 m.	3 h. 23 m.	- 5 h. 23 m.
3. MS-TM	2 h. 13 m.	- 46 m.	51 m.	- 9 h. 30 m.
4. MS-MC	1 h. 48 m.	- 55 m.	32 m.	- 9 h. 40 m.
5. MS-WMC	2 h. 51 m.	40 m.	1 h. 08 m.	- 8 h. 33 m.

จากตารางที่ 4.4 แสดงความผิดพลาดของค่าประมาณของเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์ จากขนาดของการบำรุงรักษาซอฟต์แวร์ โดยค่าความผิดพลาด หมายถึง มาตรวัดแต่ละมาตรวัดให้ค่าประมาณของเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์ผิดพลาดไปจากเวลาจริงเท่าใด ซึ่งค่าความผิดพลาดเป็นลบ (-) หมายถึง ค่าประมาณของเวลาที่คำนวณได้จากมาตรวัดต่ำกว่าค่าจริงที่ใช้ในการบำรุงรักษาซอฟต์แวร์ ในขณะที่ค่าความผิดพลาดเป็นบวก (+) หมายถึง ค่าประมาณเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์ของมาตรวัดนั้นได้ค่าสูงกว่าเวลาจริงที่ใช้ในการบำรุงรักษา

ซอฟต์แวร์ โดยแสดงค่าความผิดพลาดของค่าประมาณของเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์แต่ละเวอร์ชั่น จากการเพิ่มเติมความต้องการของผู้ใช้งาน (Functional Enhancement) ดังนี้

1. การเพิ่มเติมรายงานระดับง่าย ครั้งที่ 1 มาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจำนวนเมธอดต่อคลาส (MS-MC) ให้ค่าประมาณของเวลาใกล้เคียงกับเวลาจริงที่ใช้ในการบำรุงรักษาซอฟต์แวร์ โดยให้ค่าความผิดพลาดน้อยที่สุดคือ 1 ชั่วโมง 48 นาทีเมื่อเปรียบเทียบกับเวลาจริง รองลงมาคือ มาตรวัดที่พิจารณาจากจำนวนเมธอด (MS-TM) ซึ่งคำนวณคลาดเคลื่อนไปจากเวลาจริง 2 ชั่วโมง 13 นาที และมาตรวัดที่ให้ค่าใกล้เคียงลำดับที่ 3 คือมาตรวัดที่พิจารณาจากจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนัก (MS-WMC) ให้ค่าความผิดพลาด 2 ชั่วโมง 51นาที

2. การเพิ่มเติมรายงานระดับง่าย ครั้งที่ 2 มาตรวัดของการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนัก (MS-WMC) ให้ค่าใกล้เคียงกับเวลาจริงที่ใช้ในการบำรุงรักษาซอฟต์แวร์ในการบำรุงรักษาซอฟต์แวร์ ให้ค่าความผิดพลาดน้อยที่สุดคือ 40 นาที เมื่อเปรียบเทียบกับเวลาจริง รองลงมาคือ มาตรวัดที่พิจารณาจากจำนวนเมธอด (MS-TM) ให้ค่าความผิดพลาด 46 นาที และมาตรวัดที่ให้ค่าใกล้เคียงลำดับที่ 3 คือ มาตรวัดที่พิจารณาจากจำนวนเมธอดต่อคลาส (MS-MC) ให้ค่าความผิดพลาด 55 นาที ซึ่งทั้งสองมาตรวัดให้ค่าประมาณของเวลาที่น้อยกว่าเวลาจริงที่ใช้ในการบำรุงรักษาซอฟต์แวร์ (จึงทำให้ค่าเป็น -)

3. การเพิ่มเติมรายงานระดับปานกลาง ครั้งที่ 1 มาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจำนวนเมธอดต่อคลาส (MS-MC) ให้ค่าใกล้เคียงกับเวลาจริงที่ใช้ในการบำรุงรักษาซอฟต์แวร์ในการบำรุงรักษาซอฟต์แวร์ โดยให้ค่าความผิดพลาดน้อยที่สุดคือ 32 นาที เมื่อเปรียบเทียบกับเวลาจริง รองลงมาคือ มาตรวัดที่พิจารณาจากจำนวนเมธอด (MS-TM) ซึ่งคำนวณคลาดเคลื่อนไปจากเวลาจริง 51 นาที และมาตรวัดที่ให้ค่าใกล้เคียงลำดับที่ 3 คือมาตรวัดที่พิจารณาจากจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนัก (MS-WMC) ให้ค่าความผิดพลาด 1 ชั่วโมง 8 นาที

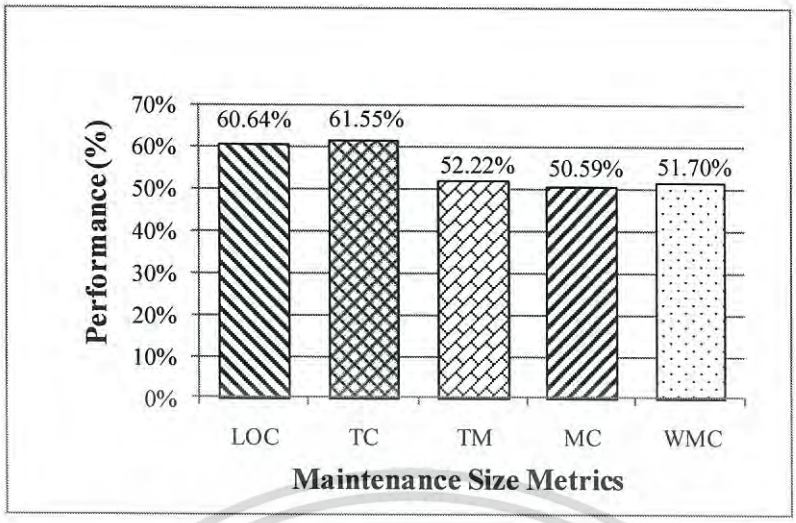
4. การเพิ่มเติมรายงานระดับปานกลาง ครั้งที่ 2 มาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจำนวนคลาส (MS-TC) ให้ค่าใกล้เคียงกับเวลาจริงที่ใช้ในการบำรุงรักษาซอฟต์แวร์ในการบำรุงรักษาซอฟต์แวร์ โดยให้ค่าความผิดพลาดน้อยที่สุดคือ 5 ชั่วโมง 23 นาที เมื่อเปรียบเทียบกับเวลาจริง รองลงมาคือ มาตรวัดที่พิจารณาจากจำนวนบรรทัดของซอร์ดโค้ด (MS-LOC) ซึ่งคำนวณคลาดเคลื่อนไปจากเวลาจริง 6 ชั่วโมง 45 นาที และมาตรวัดที่ให้ค่าใกล้เคียงลำดับที่ 3 คือมาตรวัดที่พิจารณาจากจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนัก (MS-WMC) ให้ค่าความผิดพลาด 8 ชั่วโมง 33 นาที จากการพิจารณาค่าประมาณของเวลาในทุกมาตรวัดจะพบว่าค่าประมาณของเวลาน้อยกว่าเวลาจริงที่ใช้ในการบำรุงรักษาซอฟต์แวร์ในทุกๆ มาตรวัด

ตารางที่ 4.5 ความผิดพลาดของค่าประมาณของเวลาเมื่อเปรียบเทียบกับเวลาจริงที่ใช้ในการบำรุงรักษาซอฟต์แวร์ (ชั่วโมง/นาที) สำหรับการบำรุงรักษาซอฟต์แวร์จากการจัดระเบียบซอร์ดโค้ด

ประเภทของ MS	ค่าความผิดพลาดเมื่อเปรียบเทียบกับเวลาจริงที่ใช้ในการบำรุงรักษาซอฟต์แวร์ ของบำรุงรักษาซอฟต์แวร์จากการจัดระเบียบซอร์ดโค้ด
1. MS-LOC	1 h. 35 m.
2. MS-TC	27 h. 19 m.
3. MS-TM	6 h. 03 m.
4. MS-MC	5 h. 13 m.
5. MS-WMC	40 m.

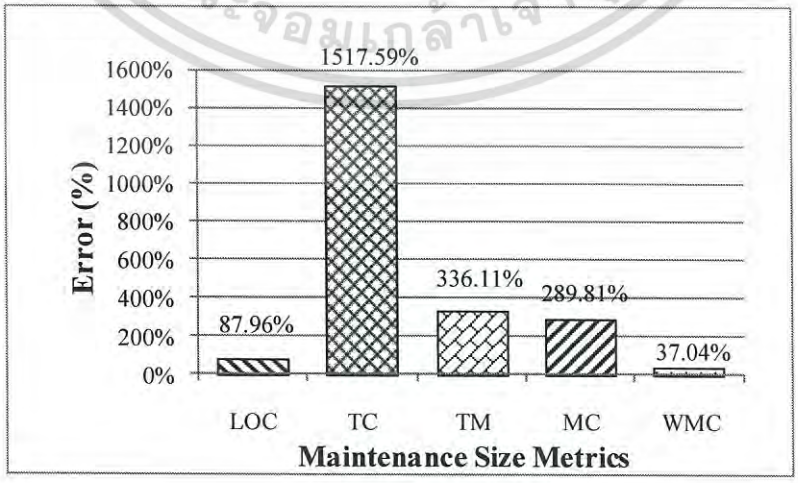
จากตารางที่ 4.5 แสดงค่าความผิดพลาดของค่าประมาณของเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์ จากการจัดระเบียบซอร์ดโค้ด (Refactoring Source Code) มาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนัก (MS-WMC) ให้ค่าใกล้เคียงกับเวลาจริงที่ใช้ในการบำรุงรักษาซอฟต์แวร์ในการบำรุงรักษาซอฟต์แวร์ โดยให้ค่าความผิดพลาดน้อยที่สุดคือ 40 นาทีเมื่อเปรียบเทียบกับเวลาจริง รองลงมาคือ มาตรวัดที่พิจารณาจากจำนวนบรรทัดของซอร์ดโค้ด (MS-LOC) ซึ่งคำนวณคลาดเคลื่อนไปจากเวลาจริง 1 ชั่วโมง 35 นาที และมาตรวัดที่ให้ค่าใกล้เคียงลำดับที่ 3 คือมาตรวัดที่พิจารณาจากจำนวนเมธอดต่อคลาส (MS-MC) ให้ค่าความผิดพลาด 5 ชั่วโมง 13 นาที

ซึ่งเมื่อพิจารณาการบำรุงรักษาซอฟต์แวร์ออกเป็น 2 ลักษณะงาน ได้แก่ (1) การบำรุงรักษาซอฟต์แวร์จากการเพิ่มเติมความต้องการจากผู้ใช้งาน (Functional Enhancement) และ (2) การบำรุงรักษาซอฟต์แวร์จากการจัดระเบียบซอร์ดโค้ด (Refactoring Source Code) แสดงค่าความผิดพลาดของมาตรวัดซอฟต์แวร์แต่ละประเภท แยกตามลักษณะการบำรุงรักษาซอฟต์แวร์ จากการพิจารณาค่าประมาณเวลาต่อเวลาจริงที่ใช้ในการบำรุงรักษาซอฟต์แวร์ ดังรูปที่ 4.6 และ รูปที่ 4.7



รูปที่ 4.6 เปอร์เซ็นต์เฉลี่ยของความผิดพลาดในแต่ละมาตรวัดสำหรับการเพิ่มเติมความต้องการของระบบ

จากรูปที่ 4.6 แสดงค่าเปอร์เซ็นต์เฉลี่ยของค่าความผิดพลาดของค่าประมาณเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์เมื่อเปรียบเทียบกับเวลาจริงที่ใช้ในการบำรุงรักษาซอฟต์แวร์แต่ละประเภท โดยผลการศึกษาพบว่า ในการบำรุงรักษาซอฟต์แวร์โดยการเพิ่มเติมความต้องการของผู้ใช้งาน การประมาณการเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์ที่ได้จากการคำนวณขนาดของการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจากจำนวนเมธอดต่อคลาส (MS-MC) ให้ค่าความผิดพลาดเฉลี่ยน้อยที่สุดเมื่อเทียบกับเวลาจริง คือ 50.59% รองลงมาคือ มาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจากจำนวนคลาส (MS-WMC) ให้ค่าความผิดพลาด 51.70% ซึ่งเมื่อเปรียบเทียบกับการค่าประมาณโดยการพิจารณาจากจำนวนบรรทัดของซอฟต์แวร์ (MS-LOC) ให้ค่าความผิดพลาดน้อยกว่า 10.05% และ 8.94% ตามลำดับ

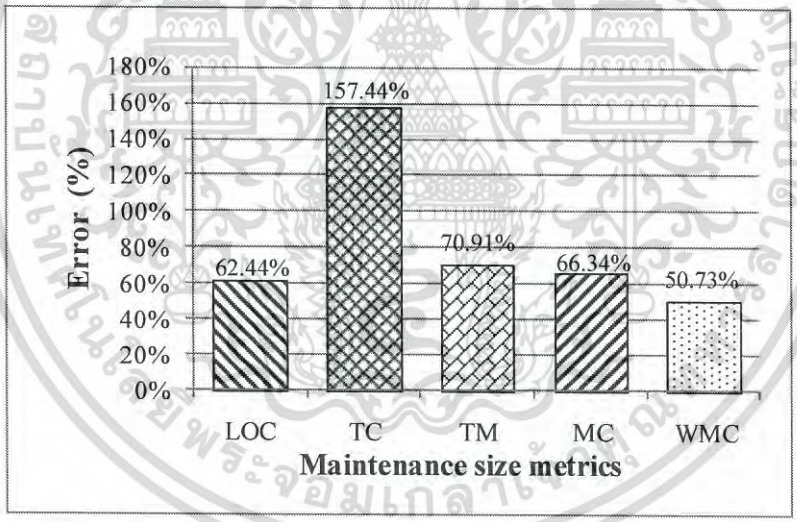


รูปที่ 4.7 เปอร์เซ็นต์เฉลี่ยของความผิดพลาดแต่ละมาตรวัดสำหรับการจัดระเบียบซอร์ดโค้ด

จากรูปที่ 4.7 แสดงค่าเปอร์เซ็นต์เฉลี่ยของความผิดพลาดของค่าประมาณเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์โดยผลการศึกษาพบว่า ในการบำรุงรักษาซอฟต์แวร์สำหรับการจัดระเบียบซอร์ดโค้ด ค่าประมาณของเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์ที่ได้จากการคำนวณขนาดของการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจากจำนวนเมธอดต่อคลาส (MS-WMC) ให้ค่าความผิดพลาดน้อยที่สุด คือ 37.04% เมื่อเทียบกับเวลาจริง รองลงมาคือ การพิจารณาจากจำนวนบรรทัดของซอร์ดโค้ด (MS-LOC) ให้ค่าความผิดพลาด 87.96% ซึ่งเมื่อเปรียบเทียบ แล้วมาตรวัด MS-WMC ให้ความผิดพลาดของค่าประมาณเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์ MS-LOC น้อยกว่า 50.92%

เมื่อพิจารณาค่าความผิดพลาดในส่วนของการจัดระเบียบซอร์ดโค้ด จะพบว่าค่าความผิดพลาดค่อนข้างมาก โดยปัจจัยหนึ่งที่มีผลสำหรับค่าประมาณของเวลาสำหรับการจัดเรียงซอร์ดโค้ด ได้แก่ ลักษณะการ Refactoring เช่น หากมีการปรับเปลี่ยนหรือปรับปรุงคลาสหลายๆ คลาสแต่ใช้เวลาการปรับปรุงคลาสแต่ละคลาสเพียงเล็กน้อยหรือเปลี่ยนแปลงข้อมูลในคลาสหรือเมธอดเพียงเล็กน้อย จะส่งผลต่อการพิจารณาในมาตรวัด MS-TC เป็นต้น

เมื่อพิจารณาเปอร์เซ็นต์เฉลี่ยของความผิดพลาดของแต่ละมาตรวัดเมื่อเปรียบเทียบกับเวลาจริงของการบำรุงรักษาซอฟต์แวร์ในทุกๆ เวอร์ชั่นแล้ว ให้ค่าความผิดพลาด ดังตารางที่ 4.8



รูปที่ 4.8 ค่าความผิดพลาดเฉลี่ย (%) ของแต่ละมาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์

จากรูปที่ 4.8 แสดงค่าความผิดพลาดเฉลี่ยของแต่ละมาตรวัดในการบำรุงรักษาซอฟต์แวร์ โดยผลการศึกษาพบว่า มาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจากจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนัก (MS-WMC) ให้ความผิดพลาดน้อยที่สุดเฉลี่ย 50.73% รองลงมาคือมาตรวัดที่พิจารณาจากจำนวนบรรทัดของซอฟต์แวร์ (MS-LOC) ให้ค่าความผิดพลาดเฉลี่ย 62.44% ตามลำดับ ถัดมาคือมาตรวัดที่พิจารณาจากจำนวนเมธอดต่อคลาสให้ค่าความผิดพลาดเฉลี่ย 66.34% โดยมาตรวัด MS-WMC ให้ค่าความผิดพลาดน้อยกว่า MS-LOC 12.71% ในขณะที่ MS-MC ให้ค่าความผิดพลาดน้อยกว่า MS-LOC 3.90%

4.4 การเปรียบเทียบมาตรวัดเชิงวัตถุของการวัดการบำรุงรักษาซอฟต์แวร์

ในแต่ละครั้งที่ทำการบำรุงรักษาซอฟต์แวร์ จะมีการวัดโครงสร้างของซอฟต์แวร์หลังจากบำรุงรักษาเสร็จแล้ว โดยใช้มาตรวัดเชิงวัตถุ โดยวิทยานิพนธ์ฉบับนี้จะเลือกใช้ มาตรวัดจำนวน 6 มาตรวัด ได้แก่ มาตรวัดจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนัก (WMC) มาตรวัดการตอบสนองต่อคลาส (RFC) มาตรวัดความเข้าคู่ระหว่างวัตถุ (CBO) มาตรวัดการขาดระดับความสัมพันธ์ภายในคลาส (Lack of Cohesion in Method: LCOM) มาตรวัดระดับความลึกของการสืบทอดคุณสมบัติของคลาส (Depth of Inheritance Tree: DIT) และมาตรวัดจำนวนคลาสลูก (Number of Children: NOC) แสดงค่าที่ได้จากการวัดในแต่ละเวอร์ชั่น ดังตารางที่ 4.6

ตารางที่ 4.6 ค่าที่ได้จากมาตรวัดเชิงวัตถุจำนวน 6 มาตรวัดโดยวัดจากซอฟต์แวร์ที่ได้รับการเปลี่ยนแปลงแล้ว ในกรณีของการบำรุงรักษาโดยการเพิ่มเติมความต้องการของผู้ใช้งาน

ประเภทของมาตรวัด	ค่าที่ได้จากมาตรวัดเชิงโครงสร้างของแต่ละเวอร์ชั่นการบำรุงรักษาซอฟต์แวร์				
	Int Ver.	Mtn Ver. 1	Mtn Ver. 2	Mtn Ver. 3	Mtn Ver. 4
1. WMC	12.23	13.64	13.73	13.93	14.06
2. RFC	34.26	34.36	36.67	38.44	39.41
3. CBO	2.83	3.17	3.50	3.83	4.83
4. LCOM	0.30	0.29	0.27	0.26	0.241
5. DIT	1	1	1	1	1
6. NOC	0	0	0	0	0

จากตารางที่ 4.6 แสดงค่าที่ได้จากมาตรวัดเชิงวัตถุ จำนวน 6 มาตรวัด โดยผลของการบำรุงรักษาซอฟต์แวร์ในแต่ละเวอร์ชั่น โดย การเพิ่มเติมความต้องการของระบบ (Functional Enhancement) เมื่อพิจารณาทางด้านความสามารถในการบำรุงรักษาซอฟต์แวร์ (Maintainability) ในด้านของการเพิ่มเติมความต้องการของระบบพบว่า ในแต่ละครั้งของการเพิ่มประเภทรายงาน จะทำให้ความสามารถในการบำรุงรักษาซอฟต์แวร์ลดลง พิจารณาได้จากค่าของ WMC, RFC และ CBO ที่มีการเพิ่มขึ้นทุกครั้งที่มีการเพิ่มเติมประเภทรายงาน แม้ว่าค่า LCOM จะมีค่าลดลงก็ตาม ในส่วนของค่า DIT และค่า NOC คงที่เนื่องจากการเพิ่มเติมประเภทรายงานไม่ได้มีผลต่อโครงสร้างของระบบโดยรวม ดังนั้นแต่เดิมที่ไม่ได้มีการถ่ายทอดคุณสมบัติ (Inheritance) และการสืบทอดคุณสมบัติจากคลาสแม่ ค่าของ DIT และ NOC จึงไม่มีการเปลี่ยนแปลง ในทุกๆ เวอร์ชั่นของการบำรุงรักษา

ตารางที่ 4.7 ค่าที่ได้จากมาตรวัดเชิงวัตถุจำนวน 6 มาตรโดยวัดจากซอฟต์แวร์ที่ได้รับการเปลี่ยนแปลงแล้วในกรณีของการบำรุงรักษาซอฟต์แวร์โดยการจัดระเบียบซอร์ดโค้ด

ประเภทของมาตรวัด	ค่าที่ได้จากมาตรวัดเชิงโครงสร้างสำหรับการจัดเรียงซอร์ดโค้ด	
	Mtn Ver. 4	Refactoring Source Code
1. WMC	14.06	11.95
2. RFC	39.41	34.26
3. CBO	4.83	4.5
4. LCOM	0.241	0.242
5. DIT	1	1
6. NOC	0	0

จากตารางที่ 4.7 แสดงค่าที่ได้จากมาตรวัดเชิงวัตถุ จำนวน 6 มาตรวัด เพื่อเปรียบเทียบความสามารถ (ความยากง่าย) ในการบำรุงรักษาของซอฟต์แวร์ ก่อนและหลังทำการจัดระเบียบซอร์ดโค้ด โดยผลของการบำรุงรักษาโดยการบำรุงรักษาซอฟต์แวร์โดยการจัดระเบียบซอร์ดโค้ด (Refactoring Source Code) จากการพิจารณาค่าของ WMC, CBO และ RFC มีผลลัพธ์ไปในทิศทางเดียวกัน นั่นคือค่าในเวอร์ชั่นของการบำรุงรักษาลดลง นั่นหมายถึง การจัดระเบียบซอร์ดโค้ดทำให้สามารถบำรุงรักษาและปรับขยายได้ง่ายขึ้น ถึงแม้ว่าค่าของ LCOM จะเพิ่มขึ้นแต่เพิ่มขึ้นเพียงเล็กน้อยเท่านั้น ในขณะที่ค่าของ DIT และ NOC คงที่ ซึ่งโดยรวมแล้ว การจัดระเบียบซอร์ดโค้ดทำให้การบำรุงรักษาและเพิ่มขยายระบบทำได้ง่ายขึ้น

บทที่ 5
ดีไซน์แพตเทิร์นและการปรับปรุงการออกแบบโดยใช้ดีไซน์แพตเทิร์น

ในบทนี้จะกล่าวถึงการศึกษาดีไซน์แพตเทิร์น การปรับปรุงการออกแบบโดยใช้ดีไซน์แพตเทิร์นชนิด อ็อบเซิร์ฟเวอร์แพตเทิร์น (Redesign using Observer Pattern) และผลของการวัดโครงสร้างและความซับซ้อนของซอฟต์แวร์จากการปรับปรุงการออกแบบจากระบบเดิม เป็นการปรับปรุงการออกแบบโดยใช้ดีไซน์แพตเทิร์น ในกรณีศึกษาของ "ระบบการสร้างรายงานอัตโนมัติเพื่อสนับสนุนงานตรวจจับฝุ่นละอองในอุตสาหกรรมการผลิตฮาร์ดดิสก์ (Software Maintenance Metrics: Case Study of Automated Report Generation System for Particle Monitoring in Hard Disk Drive Industry)"

โดยหัวข้อ 5.1 แสดงรายละเอียดของดีไซน์แพตเทิร์นและแพตเทิร์นที่ใช้ในกรณีศึกษา ได้แก่ อ็อบเซิร์ฟเวอร์แพตเทิร์น หัวข้อ 5.2 แสดงรายละเอียดและโครงสร้างของการปรับปรุงการออกแบบจากซอฟต์แวร์ที่มีอยู่เดิมโดยประยุกต์ใช้ดีไซน์แพตเทิร์น และหัวข้อ 5.3 แสดงค่าที่ได้จากมาตรวัดโครงสร้างและความซับซ้อนของคลาสในการปรับปรุงการออกแบบจากระบบเดิม

5.1 การปรับปรุงการออกแบบโดยใช้ดีไซน์แพตเทิร์น

ปัจจุบันการออกแบบได้มีการนำดีไซน์แพตเทิร์นมาช่วยให้การออกแบบมีประสิทธิภาพมากขึ้น โดยดีไซน์แพตเทิร์น เกิดจากแนวคิดของ Erich Gamma และคณะ[12] ที่ได้ผ่านการพิสูจน์และทดสอบแล้วว่าสามารถใช้แก้ไขปัญหาต่างๆ ตามความเหมาะสม โดยทั่วไปการออกแบบตามดีไซน์แพตเทิร์นที่ถูกต้องจะมีข้อบกพร่องน้อยกว่าการแก้ปัญหาแบบแอดฮอก (Adhoc) [13] Prechelt และคณะ [14] ได้ทำการทดลองเกี่ยวกับดีไซน์แพตเทิร์น โดยผลการทดลองแสดงว่าดีไซน์แพตเทิร์นมีผลต่อความสามารถด้านการบำรุงรักษา โดยจะทำให้การบำรุงรักษาทำได้ง่ายกว่าและมีข้อผิดพลาดจำนวนน้อยลงสำหรับบางแพตเทิร์น Vokáč และคณะ[15] ได้ทำการทดสอบจากการเขียนโปรแกรมจริงเพื่อเปรียบเทียบความสามารถทางด้านการบำรุงรักษา (Maintainability) ผลลัพธ์จากการทดลองพบว่าการใช้ดีไซน์แพตเทิร์นบางแพตเทิร์นจะช่วยในเรื่องของการบำรุงรักษาทั้งในแง่ของเวลาและจำนวนข้อผิดพลาด

วิทยานิพนธ์ฉบับนี้จะทำการเปรียบเทียบโครงสร้างและความซับซ้อนของการออกแบบที่มีการประยุกต์ใช้ดีไซน์แพตเทิร์น เพื่อให้ทราบว่า การออกแบบโดยใช้ดีไซน์แพตเทิร์นจะช่วยในเรื่องการลดความซับซ้อนหรือไม่ ซึ่งหากซอฟต์แวร์มีโครงสร้างของความซับซ้อนในลักษณะที่เป็น

Low Coupling และ High Cohesion นับว่าเป็นการออกแบบที่จะนำไปสู่ความสามารถในการบำรุงรักษาซอฟต์แวร์ต่อไปในอนาคต โดยเบื้องต้นจะศึกษาดีไซน์แพตเทิร์นที่สามารถนำมาประยุกต์ใช้กับกรณีศึกษาได้ (อธิบายในหัวข้อ 5.1.2) ได้แก่ อ็อบเซิร์ฟเวอร์แพตเทิร์น เพื่อสร้างอ็อบเจ็กต์สำหรับตรวจสอบรายงานที่เปลี่ยนแปลงไป และสร้างรายงานตามเหตุการณ์ที่เปลี่ยนแปลงไป (อธิบายในหัวข้อ 5.1.2) โดยสามารถใช้อ็อบเซิร์ฟเวอร์ในจาวาเอพีไอ ซึ่งข้อดีของการออกแบบด้วยอ็อบเซิฟร์เวอร์แพตเทิร์น คือ ทำให้การออกแบบมีลักษณะเป็น low coupling และ high cohesion (อธิบายในหัวข้อ 5.1.3) ซึ่งเป็นรูปแบบการออกแบบที่ดี

5.1.1 แนวคิดการออกแบบโดยใช้ดีไซน์แพตเทิร์น

ดีไซน์แพตเทิร์นเริ่มต้นจากแนวคิดของ Erich Gamma, Richard Helm, Ralph Johnson และ John Vlissides หรือรู้จักในนามของแก๊งออฟโฟร์ (Gang of four: GoF) [12] โดยได้แบ่งหมวดหมู่ของรูปแบบการออกแบบดังตารางที่ 5.1

ตารางที่ 5.1 ประเภทของดีไซน์แพตเทิร์นตามการแยกประเภทของ GoF [12]

แพตเทิร์นการสร้างอ็อบเจ็กต์ (Creational Pattern)	แพตเทิร์นโครงสร้าง (Structural Pattern)	แพตเทิร์นพฤติกรรม (Behavioral Pattern)
Adapter	Abstract Factory	Command
Flyweight	Builder	Observer
Bridge	Factory Method	Visitor
Composite	Prototype	Interpreter
Decorator	Singleton	Iterator
Proxy		Memento
Façade		Template Method
		Strategy
		Mediator
		State
		Chain of Responsibility

จากตารางที่ 5.1 แสดงการแบ่งประเภทของกลุ่มแพตเทิร์นออกเป็น 3 กลุ่มแต่ละกลุ่มแพตเทิร์นจะประกอบด้วยแพตเทิร์นหลายประเภท แสดงดังนี้

1. รูปแบบการออกแบบสำหรับสร้างวัตถุ (Creational Patterns) เป็นกลุ่มแพตเทิร์นที่ช่วยในการสร้างวัตถุของคลาส (Instance) ในรูปแบบต่างๆ

2. รูปแบบการออกแบบโครงสร้าง (Structural Patterns) เป็นกลุ่มแพตเทิร์นที่ช่วยในเรื่องการวางโครงสร้างระหว่างคลาส ทำให้แต่ละคลาสมีความสัมพันธ์กันอย่างเหมาะสม โดยส่วนใหญ่มักใช้ในลักษณะเป็นตัวต่อประสานของกลุ่มของคลาสนั้นกับคลาสอื่น

3. รูปแบบการออกแบบพฤติกรรม (Behavioral Patterns) เป็นกลุ่มแพตเทิร์นที่ใช้สำหรับการติดต่อสื่อสารกันระหว่างคลาส ซึ่งจะช่วยควบคุมการไหลของข้อมูลได้อย่างเหมาะสม

โดยแพตเทิร์นแต่ละชนิดนั้นสามารถนำไปประยุกต์ได้กับการออกแบบระบบต่างๆ ตามความเหมาะสม สำหรับวิทยานิพนธ์ฉบับนี้ได้นำดีไซน์แพตเทิร์นมาประยุกต์ใช้ในส่วนของการตรวจสอบข้อมูลปริมาณฝุ่นละอองเพื่อนำมาสร้างรายงาน ซึ่งแพตเทิร์นที่เหมาะสมคืออ็อบเซิร์ฟเวอร์แพตเทิร์น (Observer Pattern) หรือ รูปแบบเฝ้าสังเกตการณ์ ซึ่งเป็นกลุ่มแพตเทิร์นสำหรับแก้ปัญหาเกี่ยวกับพฤติกรรมการทำงานระหว่างอ็อบเจ็กต์ด้วยกัน

5.1.2 อ็อบเซิร์ฟเวอร์แพตเทิร์น (Observer Pattern)

โดยทั่วไปการนำดีไซน์แพตเทิร์นมาใช้งานจะนำมาเพื่อแก้ไขปัญหาการออกแบบให้เป็นแบบแผนมากขึ้น ซึ่งอ็อบเซิร์ฟเวอร์แพตเทิร์นเป็นดีไซน์แพตเทิร์นรูปแบบหนึ่ง อาจเรียกได้ว่า พับลิช/ซับสไครบ์ (Public/Subscribe) ใช้สำหรับการแจ้งเตือนเหตุการณ์หรือสถานะของอ็อบเจ็กต์ที่มีการเปลี่ยนแปลงไป โดยจะมีตัวเฝ้าสังเกตการณ์ (Observer) สังเกตการเปลี่ยนแปลงของสถานะหรือ อ็อบเจ็กต์นั้นๆ โดยแสดงโครงสร้างการแก้ปัญหาดังตารางที่ 5.2

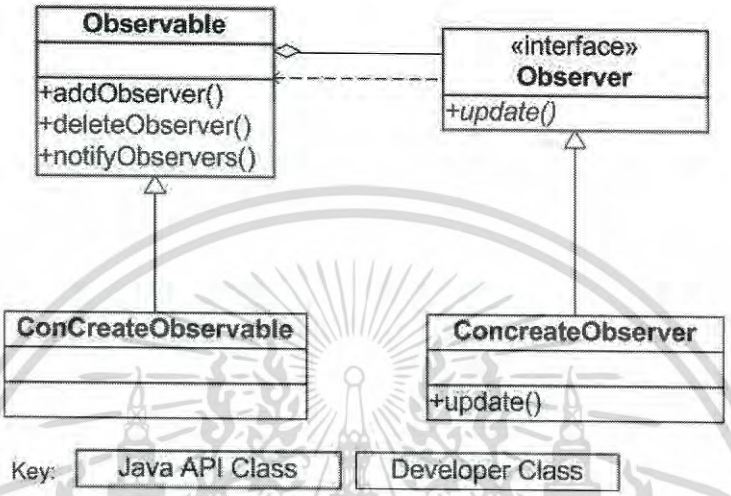
ตารางที่ 5.2 โครงสร้างการแก้ปัญหาโดยอ็อบเซิร์ฟเวอร์แพตเทิร์น [12]

แพตเทิร์น (Pattern)	**อ็อบเซิร์ฟเวอร์แพตเทิร์น (Observer Pattern)**
ปัญหา:	เมื่อเกิดเหตุการณ์ที่เปลี่ยนแปลงไปกับคลาสหนึ่ง คลาสอื่นที่อยู่ในสภาพแวดล้อมเดียวกันจะทราบได้อย่างไรว่ามีเหตุการณ์ที่เกิดขึ้นกับคลาสดังกล่าว ดังนั้นจะทำอย่างไรในการที่จะสังเกตพฤติกรรมที่เปลี่ยนแปลงไปโดยไม่มีผลกระทบต่อความสัมพันธ์ระหว่างคลาส
แนวทางการแก้ปัญหา:	คลาสที่ต้องการทราบเกี่ยวกับพฤติกรรมหรือเหตุการณ์ที่เปลี่ยนแปลงไปของคลาสอื่นจะต้องทำการลงทะเบียน กับคลาสที่ต้องการเฝ้าดู จากนั้นหากมีเหตุการณ์ใดเกิดขึ้น คลาสที่มีพฤติกรรมเปลี่ยนแปลงไปจะแจ้งเตือนไปยังคลาสที่ได้ทำการเฝ้าดู ** แพตเทิร์นนี้อาจเรียกว่า listener pattern และคลาส observer บางครั้งจะเรียกว่า listener คลาส
ประโยชน์และผลกระทบ:	➢ การออกแบบโดยใช้อ็อบเซิร์ฟเวอร์แพตเทิร์นจะช่วยทำให้ความสัมพันธ์ระหว่างคลาส (Coupling) ลดลง ➢ สามารถเพิ่มหรือลด Observable ได้หลายตัวในการตรวจสอบหรือเฝ้าดูการเปลี่ยนแปลง ➢ สามารถเพิ่มหรือลด Observer หรือตัวเฝ้าสังเกตการณ์ได้หลายตัวเช่นเดียวกัน

จาวาเอพีไอ (Java Application Program Interface: Java API) เป็นกลุ่มของคลาสที่เขียนด้วยภาษาจาวา (Java) จะช่วยอำนวยความสะดวกต่างๆ เช่น การจัดการที่เกี่ยวข้องกับเครือข่าย การติดต่อกับระบบฐานข้อมูล การจัดการกราฟิก หรือการรับส่งอีเมล์ เป็นต้น โดยจะรวบรวมเป็น

ไลบรารีของคลาสและอินเทอร์เฟสที่สัมพันธ์กันในรูปแบบของแพ็กเกจที่สามารถนำมาใช้โดยผู้ใช้งานไม่ต้องเขียนขึ้นเอง เอพีไอของจาวา เช่น เอพีไอที่ช่วยสำหรับการสร้างส่วนต่อประสานผู้ใช้อยู่ใน java.awt แพ็คเกจ เป็นต้น

สำหรับดีไซน์แพตเทิร์นชนิดอ็อบเซิร์ฟเวอร์แพตเทิร์น ในจาวาเอพีไอ จะอยู่ในคลาสของ *Observer* โครงสร้างของอ็อบเซิร์ฟเวอร์ในจาวาเอพีไอ [16] แสดงดังรูปที่ 5.1



รูปที่ 5.1 โครงสร้างของอ็อบเซิร์ฟเวอร์ในจาวาเอพีไอ

จากรูปภาพที่ 2.1 แสดงโครงสร้างของคลาสในจาวาเอพีไอ ประกอบด้วยคลาสหลัก 2 คลาส ได้แก่ คลาส *Observer* ที่เป็นอินเทอร์เฟสคลาส (Interface Class) และคลาส *Observable* โดยเมธอด *update()* เป็น callback เมธอดเนื่องจากเมธอดนี้จะสังเกตอ็อบเจ็กต์ที่เปลี่ยนแปลงไปจากคลาสของ Observable โดยหากมีเหตุการณ์หรือพฤติกรรมเปลี่ยนแปลง คลาสที่มีพฤติกรรมเปลี่ยนแปลงจะแจ้งเตือนไปยังคลาสที่ทำการเฝ้าดูผ่านเมธอด *notifyObservers()* จากนั้นคลาสที่ทำการเฝ้าดูจะนำข้อมูลที่เปลี่ยนแปลงไปใช้งานต่อไป อย่างไรก็ตามนอกจากอ็อบเซิร์ฟเวอร์แพตเทิร์นแล้ว ยังมีดีไซน์แพตเทิร์นประเภทอื่นที่สามารถเรียกใช้จาวาเอพีไอได้

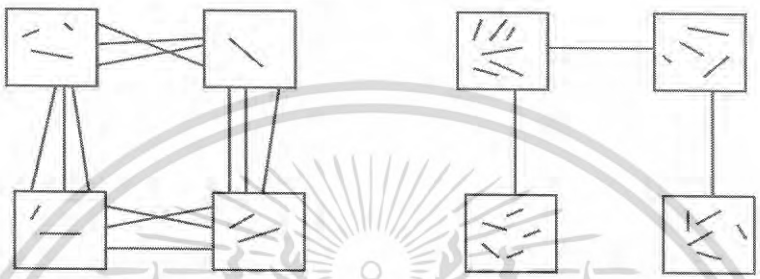
การออกแบบโดยใช้ดีไซน์แพตเทิร์นชนิดอ็อบเซิร์ฟเวอร์แพตเทิร์นช่วยให้เกิด Loose coupling ระหว่างโมดูลและ High cohesion ภายในโมดูล ทั้งนี้ทำให้การบำรุงรักษาและการปรับขยายระบบทำได้ง่ายขึ้น

5.1.3 Coupling และ Cohesion ในการออกแบบ

จากการศึกษาดีไซน์แพตเทิร์นชนิดอ็อบเซิร์ฟเวอร์แพตเทิร์น ข้อดีประการหนึ่งของการออกแบบด้วย อ็อบเซิร์ฟเวอร์ คือ ทำให้การออกแบบซอฟต์แวร์มีลักษณะเป็น low coupling และ high cohesion ซึ่งเป็นรูปแบบของการออกแบบที่ดี

Coupling [17] เป็นการอ้างถึงการติดต่อกันภายในระหว่างโมดูล หาก coupling น้อยการทำงานของโมดูลจะเป็นอิสระมากขึ้น เนื่องจากจำนวนการเชื่อมต่อระหว่างโมดูลลดลง จะช่วยให้โอกาสความผิดพลาดจากโมดูลหนึ่งมีผลกระทบน้อยลงกับโมดูลอื่น

Cohesion [17] การวัดการยึดเหนี่ยวที่เกี่ยวข้องกับ Elements ภายในโมดูล ซึ่งการวัดค่าการยึดเกาะจะแสดงให้เห็นว่าระบบมีการแบ่งส่วนที่อยู่ในระดับที่ดีหรือไม่ โดยส่วนใหญ่แล้วจะวัดควบคู่ไปกับ coupling โดยค่า cohesion ที่มากแสดงถึงการออกแบบที่ดี ความสัมพันธ์ระหว่างค่า coupling และ cohesion ที่ดีแสดงดังรูปที่ 5.2 (ข)



(ก) High Coupling and Low Cohesion (ข) Low Coupling and High Cohesion

รูปที่ 5.2 ความสัมพันธ์ระหว่าง coupling และ cohesion

มาตรวัดที่บ่งบอกถึงระดับความสัมพันธ์ภายในโมดูล (Cohesion) และความสัมพันธ์ระหว่างโมดูล (Coupling) ในระดับเมธอดหรือคลาส ประกอบด้วยหลายมาตรวัด เช่น มาตรวัดความเข้าคู่ระหว่างวัตถุ (Coupling between Object: CBO) มาตรวัดการขาดระดับความสัมพันธ์ภายในคลาส (Lack of Cohesion in Method: LCOM) หรือมาตรวัดการตอบสนองต่อคลาส (Response for a Class: RFC)

5.2 โครงสร้างคลาสการปรับปรุงการออกแบบโดยใช้อ็อบเซิร์ฟเวอร์แพตเทิร์น

การปรับปรุงโครงสร้างโดยใช้ดีไซน์แพตเทิร์นชนิดอ็อบเซิร์ฟเวอร์แพตเทิร์น เป็นการปรับปรุงการออกแบบจากซอฟต์แวร์ที่มีอยู่เดิม มีรายละเอียดดังนี้

Initial Version เป็นเวอร์ชั่นดั้งเดิมของระบบการสร้างรายงาน ประกอบด้วยรายงานพื้นฐาน ได้แก่ รายงานแสดงปริมาณฝุ่นละอองของแต่ละเซ็นเซอร์ (ParticleCountReport) รายงานสรุปจำนวนครั้งที่ปริมาณฝุ่นละอองเกินมาตรฐาน (AmountOutofControlReport) และรายงานสรุปจำนวนครั้งที่เกิดทริกเกอร์ (AmountTriggerReport)

Redesign Version เป็นเวอร์ชั่นที่ได้ทำการปรับปรุง (เวอร์ชั่นการบำรุงรักษา) มาจากระบบเดิม โดยการปรับปรุงโครงสร้างการออกแบบโดยการประยุกต์ใช้ดีไซน์แพตเทิร์นชนิดอ็อบเซิร์ฟเวอร์แพตเทิร์น โดยการทำงานของระบบการสร้างรายงานจะใช้อ็อบเซิร์ฟเวอร์ในการตรวจสอบและสร้างรายงานตามเหตุการณ์ที่เปลี่ยนแปลงไป

รูปที่ 5.3 เป็นการปรับปรุงโครงสร้างของระบบการสร้างรายงาน โดยการปรับปรุงการออกแบบแบบเดิมเป็นการออกแบบโดยประยุกต์ใช้ดีไซน์แพตเทิร์น ชนิด อ็อบเซิร์ฟเวอร์แพตเทิร์น (Observer Pattern) ประกอบด้วยคลาสเพิ่มเติมจากโครงสร้างคลาสในเวอร์ชั่นดั้งเดิม ดังนี้

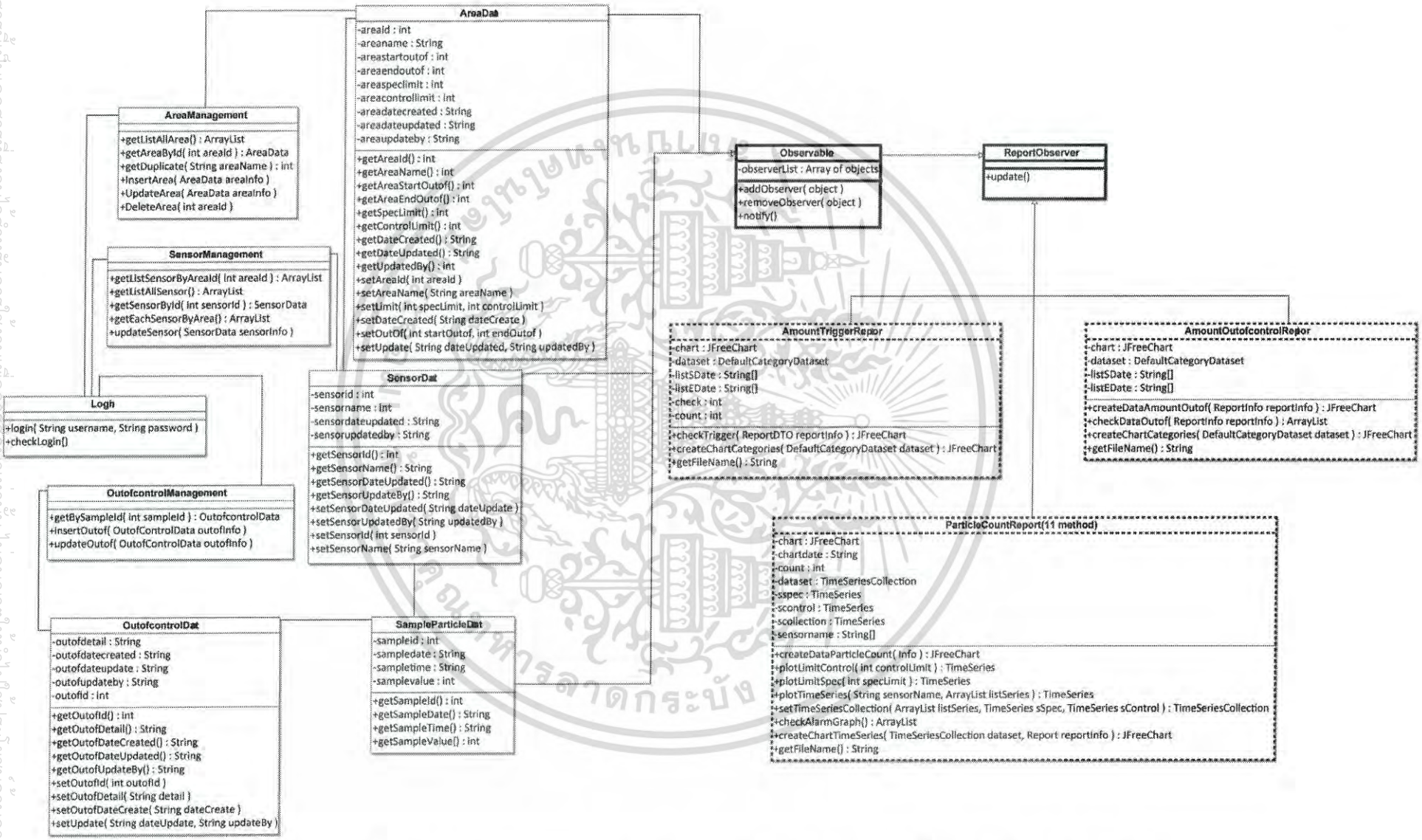
1. คลาสจัดการตัวเฝ้าสังเกตการณ์ (Observable) เป็นอินเทอร์เฟสคลาส มีเมธอดสำหรับการเพิ่ม Observer เมธอดสำหรับการลบ Observer และเมธอดสำหรับการแจ้งเตือนได้แก่ เมธอด *notify()* ประกอบด้วยคลาสลูก 3 คลาส ได้แก่ คลาส *SensorData* คลาส *AreaData* และคลาส *SampleParticleData*

2. คลาสการเฝ้าสังเกตรายงาน (ReportObserver) เป็นอินเทอร์เฟสคลาส มีเมธอดที่ทำให้เกิดการตรวจสอบสถานะของ Object โดยเมธอด *update()* จะถูกเรียกใช้งานทันทีหากมีการแจ้งเตือนการเปลี่ยนแปลงสถานะจาก เมธอด *notify()* ประกอบด้วยคลาสลูก 2 คลาส (คลาสลูกหมายถึงคลาสที่ทำการลงทะเบียนเพื่อจะทำงานตามความเปลี่ยนแปลงของข้อมูลซึ่งอาจเพิ่มได้ตามความต้องการ) โดยโมเดลต้นแบบประกอบด้วยคลาสรายงาน 3 คลาส ได้แก่ คลาส *ParticleCountReport* คลาส *AmountOutofControlReport* และคลาส *AmountTriggerReport*

3. คลาสรายงานปริมาณฝุ่นละออง (ParticleCountReport) เป็นคลาสที่ Implement จากคลาสของ Observer ทำหน้าที่ในการตรวจสอบและสร้างรายงานปริมาณฝุ่นละอองของแต่ละเซ็นเซอร์ (Particle Count Report) เพื่อเปรียบเทียบกับปริมาณฝุ่นมาตรฐาน ประกอบด้วยแอตทริบิวต์ที่ใช้สำหรับเก็บข้อมูลการสร้างรายงาน เช่น เวลาเริ่มต้น (startDate) และ เวลาสิ้นสุด (endDate) และเมธอดที่ใช้สำหรับการสร้างกราฟ เช่น เมธอด *generateChartTimeSeries()* เป็นต้น

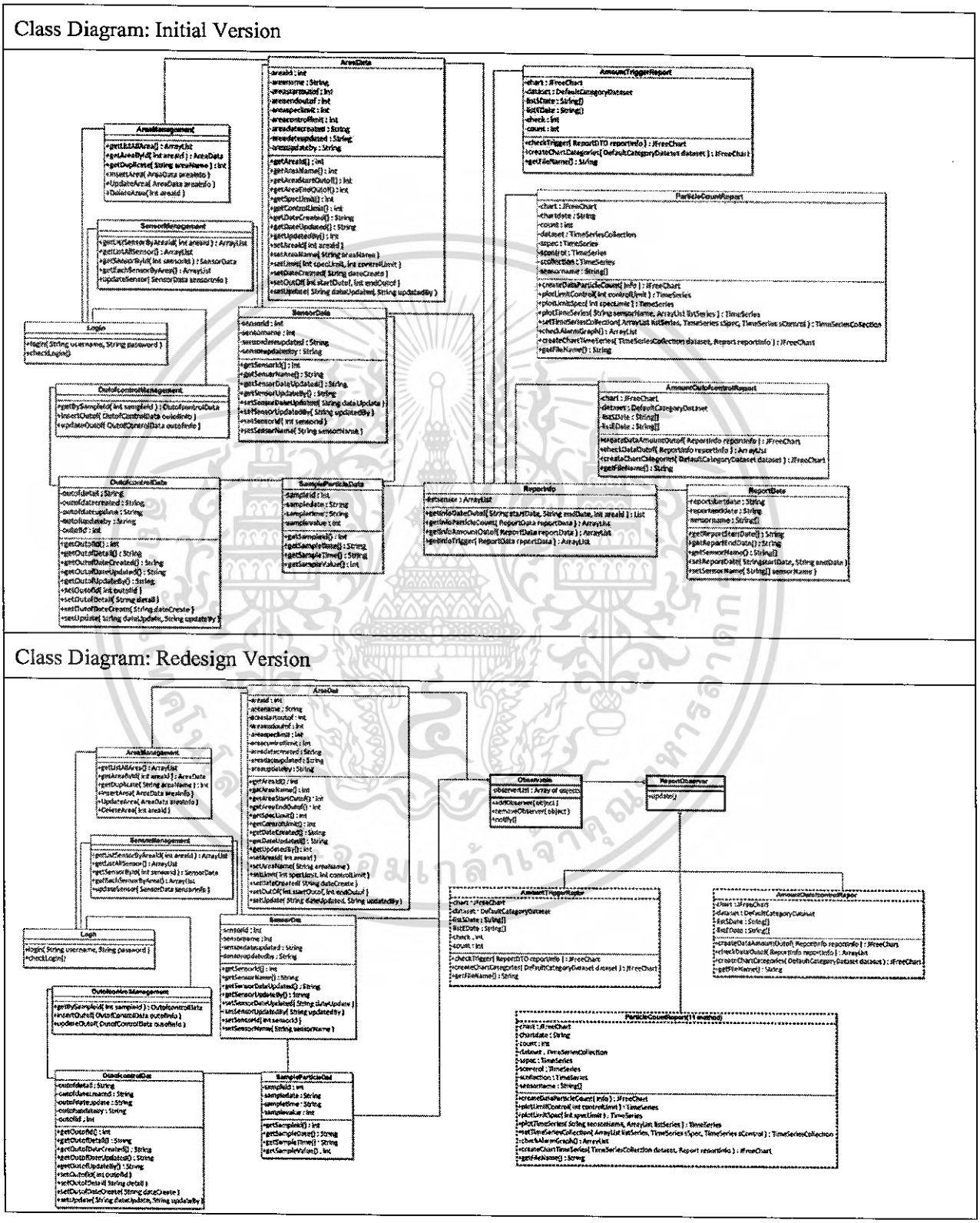
4. คลาสรายงานสรุปจำนวนครั้งที่ปริมาณฝุ่นเกินมาตรฐาน (AmountOutofControlReport) เป็นคลาสที่ Implement จากคลาสของ Observer ทำหน้าที่ตรวจสอบและสร้างรายงาน จำนวนครั้งของการเกิดปริมาณฝุ่นเกินมาตรฐานในพื้นที่การผลิต (Amount out of control Report) ประกอบด้วย แอตทริบิวต์ ที่ใช้สำหรับเก็บข้อมูลการสร้างรายงาน เช่น อุปกรณ์เซ็นเซอร์ที่ต้องการนับจำนวนครั้งที่ปริมาณฝุ่นเกินมาตรฐานที่กำหนด (sensorgroup) หรือ ชุดวันที่สำหรับแบ่งกลุ่ม (category) และเมธอดที่ใช้สำหรับการสร้างกราฟ เช่น เมธอด *generateChartCategories()* เป็นต้น

5. คลาสรายงานสรุปจำนวนครั้งที่เกิดทริกเกอร์ (AmountTriggerReport) เป็นคลาสที่แสดงรายงานปริมาณฝุ่นละอองเกินมาตรฐานต่อเนื่องกันตามที่ได้กำหนดไว้ โดจะทำการ Implement จากคลาสของ *ReportObserver* จากนั้นจะสรุปจำนวนครั้งข้อมูลที่เกิดทริกเกอร์ โดยประกอบด้วยแอตทริบิวต์ที่ใช้ในการสร้างรายงาน เช่น อุปกรณ์เซ็นเซอร์ที่ต้องการ (sensorgroup) และ จำนวนครั้งการเกิดทริกเกอร์ (count) เป็นต้น



รูปที่ 5.3 โครงสร้างคลาสของการบำรุงรักษาซอฟต์แวร์โดยการประยุกต์ใช้อ็อบเซิร์ฟเวอร์แพตเทิร์น

แสดงโครงสร้างคลาสของเวอร์ชั่นตั้งต้น (Initial Version) และโครงสร้างคลาสหลังจากปรับปรุงการออกแบบโดยใช้อ็อบเซิร์ฟเวอร์แพตเทิร์น (Redesign Version) ดังรูปที่ 5.4



รูปที่ 5.4 ความแตกต่างของโครงสร้างคลาสสำหรับการปรับปรุงโครงสร้างโดยใช้อ็อบเซิร์ฟเวอร์แพตเทิร์น

หลังจากทำการปรับปรุงการออกแบบโดยใช้ดีไซน์แพตเทิร์น จะวัดโครงสร้างและความซับซ้อนของซอฟต์แวร์ทั้งของเวอร์ชั่นตั้งต้น (Initial Version) และเวอร์ชั่นการปรับปรุงโดยใช้ดีไซน์แพตเทิร์น (Redesign Version) โดยใช้มาตรวัดซอฟต์แวร์เชิงวัตถุ จำนวน 6 มาตรวัด ได้แก่

1. มาตรวัดจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนัก (WMC)
2. มาตรวัดการตอบสนองต่อคลาส (RFC)
3. มาตรวัดความเข้าคู่กันระหว่างวัตถุ (CBO)
4. มาตรวัดการขาดระดับความสัมพันธ์ภายในคลาส (LCOM)
5. มาตรวัดระดับความลึกของการสืบทอดคุณสมบัติของคลาส (DIT)
6. มาตรวัดจำนวนคลาสลูก (NOC)

อธิบายผลการเปรียบเทียบ การวัดทางด้านโครงสร้างและความซับซ้อนของซอฟต์แวร์จากการปรับปรุงการออกแบบ ในหัวข้อ 5.3

5.3 ผลการเปรียบเทียบมาตรวัดเชิงวัตถุของการปรับปรุงโครงสร้างการออกแบบ

หลังจากการปรับปรุงโครงสร้างการออกแบบแล้ว จะเปรียบเทียบมาตรวัดเชิงวัตถุใน 2 เวอร์ชั่น ได้แก่ Initial Version เวอร์ชั่นการออกแบบแบบดั้งเดิม และ Redesign Version เวอร์ชั่นการปรับปรุงการออกแบบโดยใช้อ็อบเซิร์ฟเวอร์แพตเทิร์น

โดยจะใช้มาตรวัดเชิงวัตถุจำนวน 6 มาตรวัด ได้แก่ มาตรวัดจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนัก (WMC) มาตรวัดการตอบสนองต่อคลาส (RFC) มาตรวัดความเข้าคู่ระหว่างวัตถุ (CBO) มาตรวัดการขาดระดับความสัมพันธ์ภายในคลาส (LCOM) มาตรวัดระดับความลึกของการสืบทอดคุณสมบัติของคลาส (DIT) และมาตรวัดจำนวนคลาสลูก (NOC) แสดงค่าที่ได้จากการวัดในแต่ละเวอร์ชั่น ดังตารางที่ 5.3

ตารางที่ 5.3 เปรียบเทียบค่าที่ได้จากมาตรวัดเชิงวัตถุของการออกแบบทั้ง 2 เวอร์ชั่น

ประเภทของมาตรวัด	ค่าที่ได้จากมาตรวัดเชิงวัตถุของการออกแบบทั้ง 2 โครงสร้าง	
	Initial Version	Redesign Version
1. WMC	12.23	13.38
2. RFC	34.26	27.00
3. CBO	2.83	2.67
4. LCOM	0.30	0.36
5. DIT	1	1.31
6. NOC	0	0.31

จากตารางที่ 5.3 แสดงการเปรียบเทียบค่าที่ได้จากมาตรวัดเชิงวัตถุของการออกแบบทั้ง 2 เวอร์ชั่น ได้แก่ การออกแบบแบบดั้งเดิม (Initial Version) และการปรับปรุงการออกแบบโดยใช้อีอบเซิร์ฟเวอร์แพตเทิร์น (Redesign Version) พบว่าค่าของ RFC และ CBO เป็นไปในทิศทางเดียวกัน นั่นคือ การปรับปรุงการออกแบบโดยใช้ดีไซน์แพตเทิร์นมีจำนวนครั้งในการเรียกใช้เมธอดมี (RFC) น้อย และความเข้าคู่ระหว่างวัตถุ (CBO) มีจำนวนครั้งของการติดต่อกันระหว่างโมดูลน้อย ซึ่งทำให้การทำงานเป็นอิสระมากขึ้น ดังนั้นการปรับปรุงโดยใช้อีอบเซิร์ฟเวอร์แพตเทิร์นจะมีความซับซ้อนน้อยกว่าการออกแบบโดยไม่ใช้ดีไซน์แพตเทิร์น หรือ Initial Version ในขณะที่ค่าของการสืบทอดคุณสมบัติ (DIT) และจำนวนคลาสลูก (NOC) ในเวอร์ชั่นของ Redesign จะเพิ่มขึ้นเล็กน้อยจากเวอร์ชั่น Initial เนื่องจาก การใช้อีอบเซิร์ฟเวอร์แพตเทิร์นจะมีการสืบทอดคุณสมบัติ (Inheritance) จากคลาสของ *ReportObserver* เช่นเดียวกับค่าของจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนัก (WMC) และค่าของระดับการขาดความสัมพันธ์ภายในคลาส (LCOM) ที่แสดงถึงความซับซ้อนภายในเมธอด โดยในเวอร์ชั่นของ Redesign จะมีค่าสูงกว่าเวอร์ชั่นของ Initial เช่นเดียวกัน

จากผลการทดลองข้างต้น แสดงให้เห็นว่า การออกแบบโดยใช้ดีไซน์แพตเทิร์น จากกรณีศึกษาดังกล่าวโดยใช้อีอบเซิร์ฟเวอร์แพตเทิร์น ช่วยลดความซับซ้อนของซอฟต์แวร์ในด้านของการเรียกใช้เมธอดและความสัมพันธ์ระหว่างโมดูล

บทที่ 6
บทสรุปและข้อเสนอแนะ

6.1 บทสรุป

วิทยานิพนธ์ฉบับนี้ได้นำเสนอ การวัดการบำรุงรักษาซอฟต์แวร์ (Software Maintenance Metrics) โดยใช้กรณีศึกษาของ "ระบบการสร้างรายงานอัตโนมัติเพื่อสนับสนุนงานตรวจจับฝุ่นละอองในอุตสาหกรรมการผลิตฮาร์ดดิสก์ (Automated report generation system for particle monitoring in Hard Disk Drive Industry)"

โดยจะพิจารณาความสามารถในการบำรุงรักษาซอฟต์แวร์ จากมาตรวัดขนาดของซอฟต์แวร์ (Maintenance Size Metrics) และ การวัดเชิงโครงสร้างโดยใช้มาตรวัดเชิงวัตถุ (Object-Oriented Metrics) โดยการวัดขนาดของการบำรุงรักษาซอฟต์แวร์ซอฟต์แวร์ (Maintenance Size: MS) จะพิจารณาจากจำนวนบรรทัดของซอฟต์แวร์ (Line of Code: LOC) ที่เปลี่ยนแปลงไป ซึ่งเป็นมาตรวัดที่ใช้อยู่ในปัจจุบัน โดยงานวิจัยนี้นำเสนอมาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์เพิ่มเติม จำนวน 4 มาตรวัด โดยมาตรวัดใหม่ที่นำเสนอนี้จะพิจารณาคลาส และ เมธอด เพื่อให้เหมาะสมกับการเขียนโปรแกรมเชิงวัตถุมากยิ่งขึ้น รวมทั้งสิ้นจำนวน 5 มาตรวัด ประกอบด้วย

1. มาตรวัดขนาดการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจากจำนวนบรรทัดของซอฟต์แวร์ที่เปลี่ยนแปลงไป (MS-LOC)
2. มาตรวัดขนาดการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจากจำนวนคลาสของซอฟต์แวร์ที่เปลี่ยนแปลงไป (MS-TC)
3. มาตรวัดขนาดการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจากจำนวนเมธอดของซอฟต์แวร์ที่เปลี่ยนแปลงไป (MS-TM)
4. มาตรวัดขนาดการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจากค่าเฉลี่ยของจำนวนเมธอดต่อคลาสของซอฟต์แวร์ที่เปลี่ยนแปลงไป (MS-MC)
5. มาตรวัดขนาดการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจากจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนักของซอฟต์แวร์ที่เปลี่ยนแปลงไป (MS-WMC)

ซึ่งในระหว่างที่ทำการบำรุงรักษาซอฟต์แวร์แต่ละเวอร์ชั่นจะวัดโครงสร้างของระบบ โดยใช้มาตรวัดเชิงวัตถุจำนวน 6 มาตรวัด ได้แก่

1. มาตรวัดจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนัก (WMC)
2. มาตรวัดการตอบสนองต่อคลาส (RFC)
3. มาตวัดความเข้าคู่กันระหว่างวัตถุ (CBO)

4. มาตรวัดการขาดระดับความสัมพันธ์ภายในคลาส (LCOM)

5. มาตรวัดระดับความลึกของการสืบทอดคุณสมบัติของคลาส (DIT)

6. และมาตรวัดจำนวนคลาสลูก (NOC)

โดยแบ่งผลการทดลองจากกรณีศึกษาออกเป็น 3 ส่วน ดังนี้

6.1.1 การสรุปผลเปรียบเทียบมาตรวัดใหม่ที่นำเสนอสำหรับการบำรุงรักษาซอฟต์แวร์

ผลจากการศึกษาในส่วนของมาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์พบว่า การเพิ่มเติมความต้องการของผู้ใช้งาน มาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์ที่พิจารณาจากจำนวนเมธอดต่อคลาสที่เปลี่ยนแปลงไป (MS-MC) ให้ค่าความผิดพลาดจากการหาความคลาดเคลื่อนระหว่างค่าประมาณเวลาและค่าจริงที่ใช้ในการบำรุงรักษาซอฟต์แวร์น้อยที่สุด รองลงมาคือมาตรวัดที่พิจารณาจากจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนัก (MS-WMC) ซึ่งทั้งสองมาตรวัดให้ค่าความผิดพลาดน้อยกว่ามาตรวัดที่พิจารณาจากจำนวนบรรทัดของซอฟต์แวร์ (MS-LOC)

สำหรับการบำรุงรักษาซอฟต์แวร์โดยการจัดระเบียบซอร์ดโค้ด มาตรวัดที่ให้ค่าความผิดพลาดจากการหาความคลาดเคลื่อนระหว่างค่าประมาณเวลาและค่าจริงที่ใช้ในการบำรุงรักษาซอฟต์แวร์น้อยที่สุด คือ มาตรวัดที่พิจารณาจากจำนวนเมธอดต่อคลาส โดยคิดค่าถ่วงน้ำหนัก (MS-WMC) รองลงมาคือ มาตรวัด MS-LOC ซึ่งปัจจัยหนึ่งที่มีผลสำหรับค่าประมาณของเวลาสำหรับการจัดเรียงซอร์ดโค้ด ได้แก่ ลักษณะการ Refactoring เช่น หากมีการปรับเปลี่ยนหรือปรับปรุงคลาสหลายๆ คลาสแต่ใช้เวลาการปรับปรุงคลาสแต่ละคลาสเพียงเล็กน้อยหรือเปลี่ยนแปลงข้อมูลในคลาสหรือเมธอดเพียงเล็กน้อย จะส่งผลต่อการพิจารณาในบางมาตรวัด

ซึ่งโดยรวมแล้วมาตรวัดที่ให้ค่าเฉลี่ยดีที่สุดสำหรับการบำรุงรักษาซอฟต์แวร์สำหรับกรณีศึกษา คือ มาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์โดยพิจารณาจากจำนวนเมธอดต่อคลาสโดยคิดค่าถ่วงน้ำหนัก (MS-WMC) เนื่องจากมีการพิจารณาถึงค่าถ่วงน้ำหนักซึ่งในแต่ละเมธอดจะมีค่าที่แตกต่างกันไปขึ้นอยู่กับความซับซ้อนของภายในเมธอดนั้นๆ

6.1.2 การสรุปผลเปรียบเทียบความสามารถการบำรุงรักษาซอฟต์แวร์จากมาตรวัดเชิงวัตถุ

ส่วนผลการศึกษาในด้านของความสามารถในการบำรุงรักษาซอฟต์แวร์ โดยพิจารณาจากมาตรวัดเชิงวัตถุ จะแบ่งลักษณะการบำรุงรักษาซอฟต์แวร์ออกเป็น 2 ลักษณะ ได้แก่

6.1.2.1 การบำรุงรักษาโดยการเพิ่มเติมความต้องการของผู้ใช้งาน (Functional Enhancement) โดยจะเพิ่มเติมประเภทรายงานระดับง่ายและระดับปานกลางในแต่ละเวอร์ชั่น ผลการศึกษาพบว่า การบำรุงรักษาซอฟต์แวร์จากการเพิ่มเติมความต้องของผู้ใช้งาน มาตรวัดส่วนใหญ่จะมีทิศทางไปในทางเดียวกัน คือ ความสามารถในการบำรุงรักษา (Maintainability) จะลดลง

เมื่อซอฟต์แวร์มีฟังก์ชั่น (feature) การทำงานมากขึ้น ซึ่งจะทำให้ซอฟต์แวร์มีความซับซ้อนมากขึ้น โดยปัจจัยหนึ่งอาจเนื่องมากจากความต้องการของผู้ใช้งานที่เพิ่มมากขึ้น

6.1.2.2 การบำรุงรักษาโดยการจัดระเบียบซอร์ดโค้ด (Refactoring Source Code) เป็นการจัดระเบียบซอร์ดโค้ดใหม่ในระหว่างปรับปรุงระบบทำให้บำรุงรักษาได้ง่ายขึ้น

6.1.3 การสรุปผลเปรียบเทียบการปรับปรุงการออกแบบโดยใช้ดีไซน์แพตเทิร์น

สำหรับการเปรียบเทียบโครงสร้างและความซับซ้อนของการปรับปรุงการออกแบบโดยใช้อีอบเซิร์ฟเวอร์แพตเทิร์น พบว่า หากพิจารณาการเรียกใช้งานระหว่างคลาสและการเรียกใช้งานของเมธอด ของการปรับปรุงโดยใช้อีอบเซิร์ฟเวอร์แพตเทิร์นจะมีความซับซ้อนน้อยกว่าการออกแบบโดยไม่ใช้ดีไซน์แพตเทิร์น หรือ Initial Version ในขณะการปรับปรุงการออกแบบ (Redesign) จะมีความซับซ้อนในเกี่ยวกับการสืบทอดคุณสมบัติของคลาสมากกว่า การออกแบบโดยไม่ใช้ดีไซน์แพตเทิร์น

ดังนั้นจากผลการศึกษาของกรณีศึกษาดังกล่าว สามารถวิเคราะห์ผลเบื้องต้นได้ว่าเมื่อพิจารณามาตรวัดขนาดของการบำรุงรักษา MS-MC และมาตรวัดเชิงวัตถุ พบว่า มาตรวัด MS-MC เป็นไปในทิศทางเดียวกับการวัดเชิงโครงสร้าง โดยหากขนาดของการบำรุงรักษาลดลง (ค่าของ MS-MC ลดลงในแต่ละครั้งในการเพิ่มรายงาน) นั่นหมายถึง อัตราการเปลี่ยนแปลงจำนวนเมธอดต่อคลาสในระบบมีมากขึ้นทำให้ระบบมีความซับซ้อนมากยิ่งขึ้น ซึ่งเมื่อพิจารณาจากค่าของ RFC CBO และ WMC พบว่าค่าดังกล่าวเพิ่มมากขึ้น นั่นหมายถึงระบบมีความซับซ้อนมากขึ้น ดังนั้นความสามารถในการบำรุงรักษาซอฟต์แวร์สำหรับกรณีการเพิ่มความต้องการของผู้ใช้งานก็จะลดลงตามไปด้วย

ทั้งนี้การวิเคราะห์ในเบื้องต้นนี้ควรจะมีการอ้างอิงหรือวิเคราะห์ความสัมพันธ์ในเชิงสถิติเพิ่มเติม ระหว่าง มาตรวัด MS-MC และ มาตรวัดเชิงวัตถุ ได้แก่ WMC, RFC, CBO, LCOM, DIT และ NOC ซึ่งหากมีการวิเคราะห์ทางสถิติระหว่างความสัมพันธ์ของ MS-MC และ มาตรวัดเชิงวัตถุทั้ง 6 มาตรวัด แล้วเป็นไปดังข้อสันนิษฐานเบื้องต้น MS-MC จะเป็นมาตรวัดที่บ่งบอกถึงคุณลักษณะในการบำรุงรักษาซอฟต์แวร์มาตรวัดหนึ่ง

6.2 ข้อจำกัดและข้อเสนอแนะ

6.2.1 เนื่องจากการทดสอบจะใช้กรณีศึกษาของ “ระบบการสร้างรายงานอัตโนมัติเพื่อสนับสนุนงานตรวจจับฝุ่นละอองในอุตสาหกรรมการผลิตฮาร์ดดิสก์” เพียงกรณีศึกษาเดียว ดังนั้นเพื่อทดสอบมาตรวัดที่นำเสนอจึงควรนำไปใช้กับกรณีศึกษาอื่นเพื่อทดสอบเพิ่มเติม และทำการศึกษาความสัมพันธ์ระหว่างมาตรวัดขนาดของซอฟต์แวร์ที่นำเสนอและมาตรวัดเชิงวัตถุอื่นๆ

โดยการทำ Regression Analysis โดยใช้กรณีศึกษาหรือเก็บข้อมูลเพิ่มเติมสำหรับระบบที่มีการบำรุงรักษามากกว่า 30 เวอร์ชั่นขึ้นไป

6.2.2 ปัจจัยที่ทำให้การคำนวณค่าประมาณของเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์คลาดเคลื่อนในแต่ละมาตรวัดประกอบด้วยปัจจัยหลายประการ เช่น โปรแกรมเมอร์แต่ละคนอาจจะใช้เวลาในการบำรุงรักษาซอฟต์แวร์แตกต่างกัน หรือ หากเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์ใช้ไปกับขั้นตอนของการ กำหนดความต้องการ (Requirement Effort) หรือการออกแบบ (Design Effort) มากกว่าการเขียนโปรแกรม (Implement Effort) เวลาที่ใช้สำหรับการบำรุงรักษาซอฟต์แวร์จะมาก ขณะที่จำนวนบรรทัดของซอฟต์แวร์ จำนวนคลาสและจำนวนเมธอด จะมีการเปลี่ยนแปลงน้อย การประมาณการเวลาที่ใช้ในการบำรุงรักษาซอฟต์แวร์จากขนาดของการบำรุงรักษาซอฟต์แวร์ก็จะผิดพลาดมากขึ้น

6.2.3 ควรปรับปรุงสูตรของมาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์ (Maintenance Size) โดยการหาปัจจัยที่มีผลกระทบหรือค่าคงที่ ของมาตรวัดแต่ละตัว เพื่อให้ได้ประสิทธิภาพ และพิจารณาถึงความเป็นอิสระ (Dependency) ของมาตรวัดแต่ละมาตรวัด เช่น ความสัมพันธ์ในแต่ละคลาสหรือเมธอด ประกอบกับปัจจัยทางด้านประสบการณ์ของโปรแกรมเมอร์ทั้งในส่วนของการพัฒนา การออกแบบ และการบำรุงรักษาซอฟต์แวร์เป็นส่วนสำคัญ ซึ่งอาจพิจารณาเกี่ยวกับบุคคลที่ทำการบำรุงรักษาซอฟต์แวร์ว่าเป็นบุคคลเดียวกับผู้พัฒนาหรือไม่ เพื่อเป็นงานวิจัยในอนาคตต่อไป

6.2.4 ตัวอย่างที่บันทึกสำหรับนำมาคำนวณเพื่อหาค่าของแต่ละมาตรวัดซอฟต์แวร์ ใช้เฉพาะกับกรณีศึกษานี้เท่านั้น ซึ่งการคำนวณค่าจะอ้างอิงกับเวลาที่พัฒนาในขั้นตอนแรกของระบบที่ใช้เป็นกรณีศึกษา ดังนั้นหากต้องการนำมาตรวัดที่นำเสนอไปใช้ควรพิจารณาถึงค่าของเวลาในการพัฒนาของแต่ละงานวิจัย

6.2.5 วิทยานิพนธ์ฉบับนี้ได้คำนวณมาตรวัดขนาดของการบำรุงรักษาซอฟต์แวร์ (MS) ทั้ง 5 มาตรวัดจากการนับจำนวนเมธอด/คลาส ที่ลบออก ซึ่งในอนาคตควรมีการพิจาณาเพิ่มเติมอีก 2 แนวคิด (ที่นำเสนอไว้ในหัวข้อ 3.2.1) ด้วย ได้แก่ การนำเมธอด/คลาส ที่ลบออก หักออกจากคลาสที่เพิ่ม และ การคำนวณโดยไม่คำนึงถึงเมธอด/คลาสที่ลบออก

บรรณานุกรม

[1] R. W. Welker, **"Guideline for Design and Certification of Cleanroom Tooling"**, Clean room Tooling Document, Revised by Don DeLeo/Tim Karlberg, San Jose Site Contamination Control, August 8 2002.

[2] Manufacturing Environment Engineering Department, **"ESD and Contamination Control"**, Hitachi Global Storage Technologies, April 2009.

[3] Training & Support Department, **"Hard Disk Drive Process Overview"**, Training Document. Hitachi Global Storage Technologies, April 2010.

[4] Ye Yang, Qi Li, Mingshu Li, Qing Wang. **"An empirical analysis on distribution patterns of software maintenance effort"**. In 24th IEEE International Conference on Software Maintenance (ICSM 2008), September 28 - October 4, 2008, Beijing, China. pp.456-459.

[5] B.W. Boehm, E.Horowitz, R. Madachy, D.Reifer, B.K. Clark, B.Steece, A.W. Brown, S. Chulani, and C. Abts, **"Software Cost Estimation with COCOMO II"**, Prentice Hall, 2000.

[6] W. Li and S. Henry, **"Maintenance Metrics for the Object Oriented Paradigm"**, 1st International Software Metrics Symposium. Baltimore, USA, pp. 52-60 , 1993.

[7] IEEE, **"IEEE Standard Glossary of Software Engineering Terminology"**, report IEEE Std 610.12-1990, IEEE, 1990.

[8] G. Canfora and A. Cimitile, **"Software Maintenance"**, University of Sannio, Faculty of Engineering at Benevento Palazzo Bosco Lucarelli, Piazza Roma 82100, Benevento Italy 29 November, 2000.

[9] Lientz B.P.,and Swanson E.B, **"Software Maintenance Management"**, Addison Wesley Publishing Company, 1980.

[10] S. Schach et al., **"Determining the Distribution of Maintenance Categories: Survey versus Empirical Study"**, Kluwer's Empirical Software Eng., vol. 8, no. 4, pp. 351–365, 2003.

[11] S. R. Ragab and H. H. Ammar. **"Object Oriented Design Metrics and Tools a Survey"**, The 7th International Conference on Informatics and Systems (INFOS). Cairo. March. pp. 1-7. 2010.

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. **"Design Patterns: Elements of Reusable Object-Oriented Software"**, Addison Wesley. 1994.

[13] M. Vokáč. **"Defect Frequency and Design Patterns: An Empirical Study of Industrial Code"**. IEEE Transactions on Software Engineering. Vol.30(12). December. pp.904-917. 2004.

[14] L. Prechelt, B. Unger, W.F. Tichy, P. Bro¨ ssler, and L.G. Votta, **"A Controlled Experiment in Maintenance Comparing Design Patterns to Simpler Solutions"**, IEEE Trans. Software Eng. vol. 27, no. 12. Dec. pp. 1134-1144. 2000.

[15] M. Vokáč, W. Tichy, D.I. K. Sjøberg, E. Arisholm, and M. Aldrin. **"A Controlled Experiment Comparing the Maintainability of Programs Designed with aud without Design Patterns: A Replication in a Real Programming Environment"**, Empirical Software Eng. vol. 9, no. 3.pp. 149-195. 2004.

[16] Eric J. Braude. **"Software are design: from programming to architecture"**. INT'L Ed. New York: John Wiley & Sons. 2004.

[17] W. P. Stevens, G. J. Myers, and L. L. Constantine. **Structured design.** IBM Systems Journal, 13(2):115–139, 1974.

[18] Michael S. Deutsch, Ronald R. Willis.Prentice Hall Series in Software Engineering. **Translated from Details of Book: Software Quality Engineering - A Total Technical And Management Approach.** by Randall W.Jensen. Englewood Cliffs, NJ: Prentice Hall. 1988.

[19] Software productivity consortium. **"The Software Measurement Guidebook"**. London: International Thomson Computer Press. 1995.

[20] H. D. Rombach, B. T. Ulery and J. D. Valett. **"Toward Full Life Cycle Control: Adding Maintenance Measurement to the SEL"**. The Journal of Systems and Software, Volume18, Issue 2, pp. 125–138, May 1992.

[21] S.D.Conte, H.E. Dunsmore and V.Y.Shen. **"Software Engineering Metrics and Models"**. Benjamin-Cummings Publishings Company Inc. 1986.

[22] M. Jørgensen. **"An Empirical Study of Software Maintenance Tasks"**. Jounal of Software Research And Practice, Volume 7, Issue 1, pp.27–48, 1995.

[23] Masuda, G. and al., **"Redesigning of an existing software using design patterns"**. International Symposium on Principles of Software Evolution. November.pp. 165-169. 2000.

[24] T. J. McCABE, **"A Complexity Measure"**. IEEE Transactions on Software Engineering, Volum 2(4) .pp 308-320, 1976.

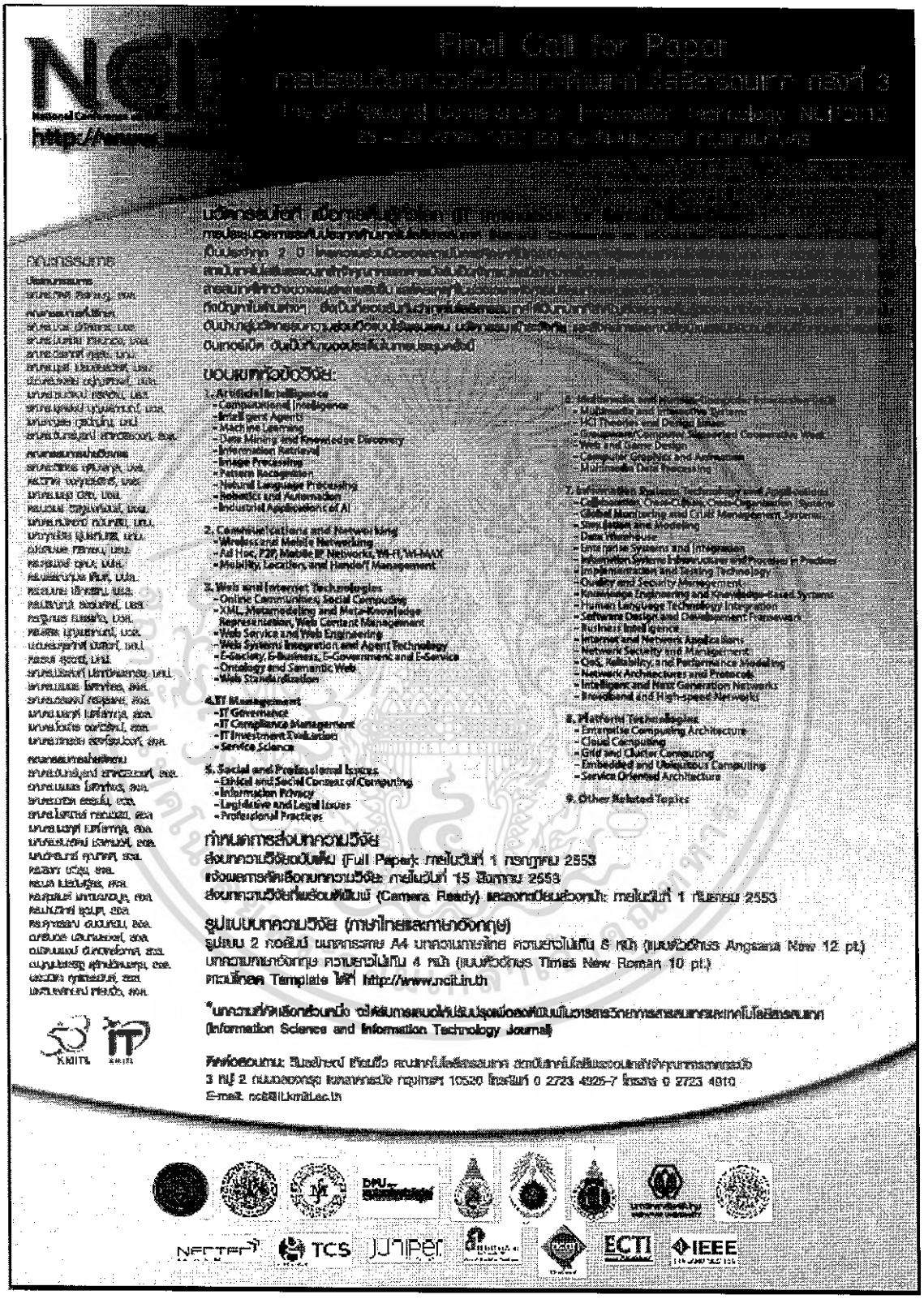
[25] 4.Chidamber, S. R. & Kemerer C. F.**"A Metrics Suite for Object Oriented Design"**, IEEE Transactions on Software Engineering, Vol. 20, No. 6, June. 1994, pp 476-493.

ภาคผนวก

ภาคผนวก

บทความและผลงานวิจัยที่ได้รับการตีพิมพ์

1. อภัสรา วิโรจน์ยะกูล และ พรฤดี เนติโสภากุล. "โครงสร้างคลาสสำหรับการออกรายงานในระบบฐานข้อมูลเพื่อสนับสนุนงานตรวจจับฝุ่นละอองในอุตสาหกรรมการผลิตฮาร์ดดิสก์". **The 3rd National Conference on Information Technology (NCIT2010)**, Thailand, October 28-29, 2010, pp. 465 – 470.
2. อภัสรา วิโรจน์ยะกูล และ พรฤดี เนติโสภากุล. "ระบบสนับสนุนฐานข้อมูลสำหรับการออกรายงานปริมาณข้อมูลฝุ่นละอองบนเว็บ (Web-Based Database Support System for Particle Monitoring Reports)." **วารสารสำนักคณะกรรมการวิจัยแห่งชาติ**, ปีที่ 43, เล่มที่ 1, 2554.
3. Aphatsara Wirotyakun and Ponrudee Netisopakul. "Improving Software Maintenance Size Metrics A Case Study: Automated Report Generation System for Particle Monitoring in Hard Disk Drive Industry". **The 9th International Joint Conference on Computer Science and Software Engineering (JCSSE 2012)**, Thailand, 30 May - 1 June, 2012, pp.335-340.



Final Call for Paper

http://www.ncit.in.th

ขอบเขตหัวข้อวิจัย:

1. Artificial Intelligence
- Computational Intelligence
- Intelligent Agents
- Machine Learning
- Data Mining and Knowledge Discovery
- Information Retrieval
- Image Processing
- Pattern Recognition
- Natural Language Processing
- Robotics and Automation
- Industrial Applications of AI

2. Communications and Networking
- Wireless and Mobile Networking
- Ad Hoc, P2P, Mobile IP Networks, Wi-Fi, Wi-MAX
- Mobility, Locations, and Handoff Management

3. Web and Internet Technologies
- Online Communities, Social Computing
- XML, Metamodeling and Meta-Knowledge Representation, Web Content Management
- Web Service and Web Engineering
- Web Systems Integration and Agent Technology
- E-Society, E-Business, E-Government and E-Service
- Ontology and Semantic Web
- Web Standardization

4. IT Management
- IT Governance
- IT Compliance Management
- IT Investment Evaluation
- Service Science

5. Social and Professional Issues
- Ethical and Social Context of Computing
- Information Privacy
- Legislative and Legal Issues
- Professional Practices

6. Multimedia and Intelligent Computing Environment
- Multimedia and Interactive Systems
- HCI Theories and Design Issues
- Groupware/Computer Supported Cooperative Work
- Web and Game Design
- Computer Graphics and Animation
- Multimedia Data Processing

7. Information Systems Technology and Applications
- Collaboration, Cross-Culture, Cross-Organization Systems
- Global Monitoring and Crisis Management Systems
- Simulation and Modeling
- Data Warehouse
- Enterprise Systems and Integration
- Information Systems Infrastructure and Practice in Practice
- Implementation and Scaling Technology
- Quality and Security Management
- Knowledge Engineering and Knowledge-Based Systems
- Human Language Technology Integration
- Software Design and Development Framework
- Business Intelligence
- Internet and Network Applications
- Network Security and Management
- QoS, Reliability, and Performance Modeling
- Network Architectures and Protocols
- Intelligent and Next Generation Networks
- Broadband and High-speed Networks

8. Platforms Technologies
- Enterprise Computing Architecture
- Cloud Computing
- Grid and Cluster Computing
- Embedded and Ubiquitous Computing
- Service Oriented Architecture

9. Other Related Topics

กำหนดการส่งบทความวิจัย

ส่งบทความวิจัยฉบับเต็ม (Full Paper): ภายในวันที่ 1 กรกฎาคม 2553

แจ้งผลการคัดเลือกบทความวิจัย: ภายในวันที่ 15 สิงหาคม 2553

ส่งบทความวิจัยที่แก้ไขฉบับสมบูรณ์ (Camera Ready) และชำระเงินลงทะเบียน: ภายในวันที่ 1 กันยายน 2553

รูปแบบบทความวิจัย (ภาษาไทยและภาษาอังกฤษ)

รูปแบบ 2 คอลัมน์ ขนาดกระดาษ A4 บทความภาษาไทย ความยาวไม่เกิน 8 หน้า (แบบหัวข้อใช้ font Angsana New 12 pt.)
บทความภาษาอังกฤษ ความยาวไม่เกิน 4 หน้า (แบบหัวข้อใช้ font Times New Roman 10 pt.)
ดาวน์โหลด Template ได้ที่ http://www.ncit.in.th

*บทความที่คัดเลือกส่งตีพิมพ์ จะได้รับการแนะนำให้ปรับปรุงเพื่อตีพิมพ์ในวารสารวิจัยการสารสนเทศและเทคโนโลยีสารสนเทศ (Information Science and Information Technology Journal)

ที่อยู่ติดต่อ: ฝ่ายวิจัย ห้องเรียน คณะวิทยาศาสตร์และเทคโนโลยี สถาบันวิทยาศาสตร์และเทคโนโลยีแห่งประเทศไทย
3 หมู่ 2 ถนนรังสิต-นครนายก ตำบลคลองหก อำเภอคลองหลวง ปทุมธานี 12120 โทรศัพท์ 0 2723 4925-7 โทรสาร 0 2723 4910
E-mail: ncit@it.kmitl.ac.th

โครงสร้างคลาสสำหรับการออกรายงานในระบบฐานข้อมูล เพื่อสนับสนุนงานตรวจจับฝุ่นละอองในอุตสาหกรรมการการผลิตฮาร์ดดิสก์

อภัสรา วิโรจน์ยะกูล และ พรฤดี เนติโสภากุล

คณะเทคโนโลยีสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

Emails: s1066447@kmitl.ac.th, pornudee@it.kmitl.ac.th

บทคัดย่อ

บทความฉบับนี้ได้นำเสนอโครงสร้างคลาสสำหรับการออกรายงานในระบบฐานข้อมูลเพื่อสนับสนุนงานตรวจจับฝุ่นละอองในอุตสาหกรรมการผลิตฮาร์ดดิสก์ เพื่อแก้ปัญหาการออกรายงานโดยมนุษย์ (Manual Report) ที่ต้องมีการโอนย้าย จัดเรียงและสรุปข้อมูลปริมาณฝุ่นละอองที่ได้จากสายการผลิตในกระบวนการประกอบชิ้นส่วนฮาร์ดดิสก์ จากโปรแกรม Microsoft Excel จึงได้ศึกษาและออกแบบโครงสร้างของระบบการออกรายงานโดยการนำ Observer Pattern มามีส่วนช่วยสำหรับการออกแบบระบบในส่วนของการออกรายงาน ทั้งนี้เพื่อให้การออกแบบมีประสิทธิภาพตามหลักการวิเคราะห์และออกแบบระบบเชิงวัตถุ

คำสำคัญ— งานตรวจจับฝุ่นละออง; กระบวนการวิเคราะห์และออกแบบเชิงวัตถุ; Design Pattern

1. บทนำ

กระบวนการทำงานส่วนใหญ่ในการประกอบชิ้นส่วนฮาร์ดดิสก์ (HDD Manufacturing) มนุษย์และเครื่องจักรจะทำงานร่วมกันภายในห้องที่มีการควบคุมสภาพแวดล้อม หรือ เรียกว่า ห้องคลีนรูม (Clean room) ปัจจัยหนึ่งที่ต้องมีการควบคุมภายในห้องดังกล่าว ได้แก่ ปริมาณฝุ่นละอองภายในอากาศ เนื่องจากในระหว่างการประกอบชิ้นส่วนฮาร์ดดิสก์ จะต้องป้องกันไม่ให้ฮาร์ดดิสก์ได้รับความเสียหายจากฝุ่นละอองหรือสิ่งแปลกปลอมอื่น ดังนั้นข้อมูลปริมาณฝุ่นละอองภายในห้องคลีนรูม (Clean room) จึงเป็นข้อมูลสำคัญในการนำมาวิเคราะห์ สรุป หรือทำให้เกิดเป็นองค์ความรู้ที่มีประโยชน์สำหรับการพัฒนาอุตสาหกรรมการผลิตต่อไป

ข้อมูลปริมาณฝุ่นละอองดังกล่าวจะถูกจัดเก็บ จากอุปกรณ์เซ็นเซอร์ (Sensor) ที่ติดตั้งอยู่ในสายการผลิต (Production Line) โดยหลักการทำงานสำหรับการตรวจจับฝุ่นละออง จะตรวจจับอากาศภายในห้องผ่านแสงเลเซอร์หากแสงที่ได้เป็นที่บจะนับปริมาณฝุ่นละอองเป็นหนึ่งจุด ขณะเดียวกันหากแสงเป็นแสงใสแสดงว่าไม่มีฝุ่นละออง ข้อมูลดังกล่าวจะถูกจัดเก็บเป็นเท็กซ์ไฟล์ (Text File) ในรูปแบบของ CSV ไฟล์ ซึ่งสามารถเปิดได้ด้วยโปรแกรม Microsoft Excel จากนั้นวิศวกรที่ทำหน้าที่หลักสำหรับมอนิเตอร์ข้อมูล (Engineer Monitoring) จะนำข้อมูลมาสรุป จัดเรียง และสร้างกราฟด้วยโปรแกรมดังกล่าวอีกครั้ง

ดังนั้นเพื่อปรับเปลี่ยนกระบวนการออกรายงานโดยมนุษย์ซึ่งเป็นการออกรายงานแบบ Manual Report ให้มีประสิทธิภาพมากขึ้น จึงพัฒนาระบบการออกรายงานผ่านเว็บโดยนำแนวคิดของ Design Pattern และกระบวนการวิเคราะห์และออกแบบระบบเชิงวัตถุ (Object Oriented Analysis and Design Methodology: OOADM) เข้ามาช่วยเพื่อให้การออกแบบระบบการรายงานของระบบฐานข้อมูลเพื่อสนับสนุนงานตรวจจับฝุ่นละอองมีประสิทธิภาพเพิ่มมากขึ้น

สำหรับเอกสารฉบับนี้จะกล่าวถึงการออกแบบโครงสร้างคลาสสำหรับการออกรายงาน โดยส่วนที่สองกล่าวถึง กระบวนการจัดเก็บข้อมูลในอุตสาหกรรมการผลิตฮาร์ดดิสก์สำหรับการออกรายงานปริมาณฝุ่นละออง ส่วนที่สามกล่าวถึง การออกรายงานในระบบปัจจุบัน ส่วนที่สี่กล่าวถึง โครงสร้างคลาสสำหรับการออกรายงาน และส่วนสุดท้ายเป็นบทสรุปและงานในอนาคต

2. กระบวนการจัดเก็บข้อมูลในอุตสาหกรรมการผลิตฮาร์ดดิสก์สำหรับการออกรายงานปริมาณฝุ่นละออง

กระบวนการประกอบชิ้นส่วนในอุตสาหกรรมการผลิตฮาร์ดดิสก์ (HDD Manufacturing) การจัดเก็บ จัดการ และวิเคราะห์ข้อมูลที่ได้จากสายการผลิต (Production Line) นับเป็นสิ่งสำคัญสำหรับการทำงานในภาคอุตสาหกรรม โดยเฉพาะข้อมูลปริมาณฝุ่นละอองที่จำเป็นต้องควบคุมเพื่อป้องกันไม่ให้ฮาร์ดดิสก์ได้รับความเสียหายจากฝุ่นละอองระหว่างการดำเนินการผลิตภายในห้องคลีนรูม

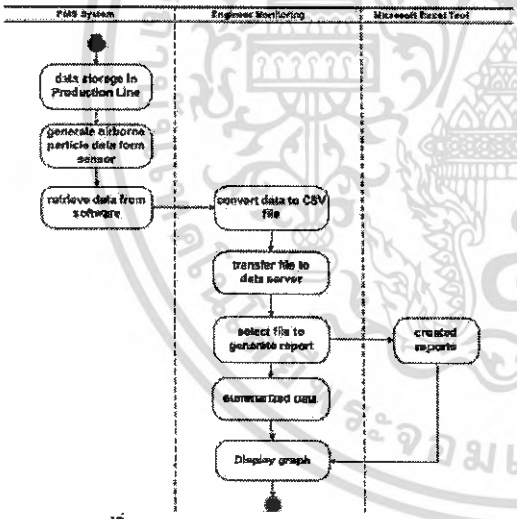
การควบคุมระดับของปริมาณฝุ่นละอองจะแบ่งระดับไว้เป็นคลาส (Class) แต่ละคลาส จะแตกต่างกันตามจำนวนขนาดของอนุภาคที่ยอมให้มีได้ในอากาศ ต่อหนึ่งหน่วยปริมาตรลูกบาศก์เมตร [1] การกำหนดคลาสในแต่ละพื้นที่การทำงาน (Process Area) จะขึ้นอยู่กับ การพิจารณาว่ากระบวนการทำงานในพื้นที่ดังกล่าว มีกระบวนการทำงานที่ฝุ่นละอองจะมีผลกระทบต่อความเสียหายของฮาร์ดดิสก์ เช่น กระบวนการประกอบหัวอ่าน หรือเรียกว่า HSA Process จะต้องกำหนดปริมาณฝุ่นละอองในอากาศระหว่างดำเนินการประกอบชิ้นส่วนเพื่อไม่ให้ฮาร์ดดิสก์ได้รับความ

เสียหายจากฝุ่นละอองภายในอากาศ ซึ่งกำหนดการทำงานที่ระดับ Clean room คลาส 10K (คลีนรูมคลาสหมื่น) ส่วนการทำงานในส่วนของ HDE Process เป็นพื้นที่สำหรับการประกอบ HSA เข้ากับ Base สามารถมีปริมาณฝุ่นในกระบวนการทำงานได้มากกว่า HSA Process จึงทำงานอยู่ที่ระดับ Clean room Class 100 ขณะเดียวกันบางพื้นที่อาจไม่จำเป็นต้องควบคุมปริมาณฝุ่นละอองในกระบวนการทำงาน เช่น การคิดฉลากเพื่อเตรียมส่งมอบให้กับลูกค้า หรือ การ Packaging ที่ทำงานอยู่ใน HDD Process ทำให้กระบวนการนี้ทำงานอยู่ในห้องทั่วไปที่ไม่ได้มีการควบคุมปริมาณฝุ่นละออง [2] เป็นต้น

สำหรับการแบ่งระดับของห้องคลีนรูม ในอุตสาหกรรมการประกอบชิ้นส่วนฮาร์ดดิสก์ที่ทำการวิจัยนั้นได้แบ่งระดับคลาส ออกเป็น 3 ระดับ ได้แก่ Clean room Class 10 (คลีนรูมคลาสสิบ) Class room Class 100 (คลีนรูมคลาสร้อย) และ Class room Class 10K (คลีนรูมคลาสหมื่น) [3] โดยแต่ละระดับจะแบ่งปริมาณฝุ่นตามขนาดของอนุภาค โดยทั่วไปแบ่งเป็นสองขนาด ได้แก่ ปริมาณฝุ่นขนาด 0.5 ไมครอน และปริมาณฝุ่นขนาด 0.3 ไมครอน

3. การออกรายงานระบบปัจจุบัน

ข้อมูลที่ได้จากการจัดเก็บเป็นเท็กซ์ไฟล์ข้างต้น วิศวกรที่ทำหน้าที่หลักสำหรับการตรวจสอบปริมาณฝุ่นละออง จะนำข้อมูลดังกล่าวมาจัดเรียงและปรับเปลี่ยนให้อยู่ในรูปแบบต่างๆ ตามความต้องการของผู้ใช้งาน ด้วยโปรแกรม Microsoft Excel ซึ่งการทำงานดังกล่าวนี้เป็นลักษณะการทำงานแบบ Manual Report แสดงขั้นตอนการสร้างรายงานดังรูปที่ 1



รูปที่ 1 กระบวนการการออกรายงานแบบ Manual Report

จากรูปที่ 1 แสดงลำดับขั้นตอนการทำงานสำหรับการสร้างรายงานแบบ Manual Report เริ่มจากการทำงานของซอฟต์แวร์ในระบบ PMS เดิน จะเก็บข้อมูลปริมาณฝุ่นละอองจากสายการผลิต (Production Line) ผ่านแสงเลเซอร์ในอุปกรณ์เซ็นเซอร์ (Sensor) จากนั้นวิสวกรที่ทำหน้าที่หลักสำหรับมอนิเตอร์ข้อมูล (Engineer Monitoring) จะสร้างไฟล์แบบเท็กซ์ไฟล์ (Text File) จากข้อมูลปริมาณฝุ่นละอองดังกล่าว แล้วจัดเก็บไฟล์ข้อมูลเข้าสู่เครื่อง Data Server จากนั้นผู้ใช้งานจะนำข้อมูลปริมาณฝุ่นละอองที่ได้มาจัดเรียงเพื่อสรุปและสร้างรายงานเชิงกราฟ (Graph-base Report) จากโปรแกรม Microsoft Excel

โดยข้อมูลสำคัญที่จะนำมาสร้างกราฟ ประกอบด้วย

- ข้อมูลวันที่ที่จัดเก็บ (Sample Date)
- ข้อมูลเวลาที่จัดเก็บ (Sample Time)
- ข้อมูลชื่อเซ็นเซอร์ (Sensor Name)
- ข้อมูลปริมาณฝุ่นละออง (Sample Value)

การจัดเก็บข้อมูลดังกล่าวจะมีการจัดเก็บทุก 1 เดือน แต่ละเดือนมีข้อมูลจำนวน 600,000 เรคคอร์ด และไฟล์มีขนาดใหญ่ประมาณ 40-60 เมกะไบต์ (MB)

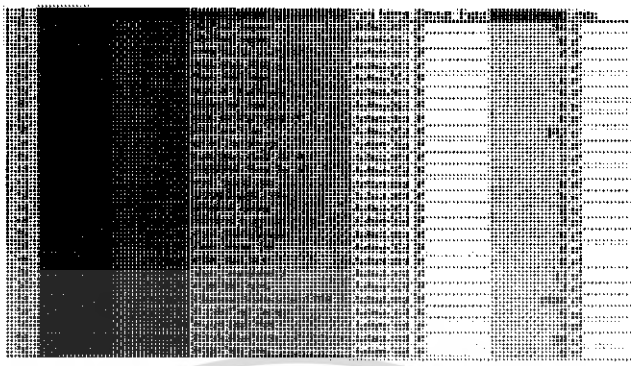
รายงานในปัจจุบันที่ต้องมีการสร้างรายงานแบบ Manual Report เช่น รายงานเปรียบเทียบปริมาณฝุ่นละอองที่เกินมาตรฐานที่กำหนด ดังรูปที่ 2 แสดงการสร้างรายงานเพื่อเปรียบเทียบปริมาณฝุ่นละอองที่เกินมาตรฐานที่กำหนด

จากรูปที่ 2 เป็นการออกรายงานแบบ Manual Report เริ่มจากการนำข้อมูลปริมาณฝุ่นละอองที่จัดเก็บจากสายการผลิตในลักษณะของเท็กซ์ไฟล์ (Text File) มาจัดเรียงใหม่และสรุปด้วยโปรแกรมสำเร็จรูป เช่น Microsoft Excel โดยข้อมูลที่ได้จากการจัดเรียงเป็นข้อมูลรายงานเชิงกราฟของการเปรียบเทียบข้อมูลปริมาณฝุ่นละอองของแต่ละเดือน โดยแต่ละเดือนจะสรุปปริมาณฝุ่นละอองของแต่ละเซ็นเซอร์ ว่ามีกี่ครั้งที่แต่ละเซ็นเซอร์นั้นมีปริมาณฝุ่นเกินมาตรฐานที่กำหนดไว้

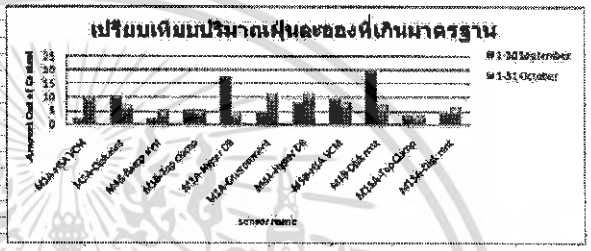
จากการสร้างรายงานดังกล่าวเป็นเพียงการเปรียบเทียบข้อมูลปริมาณฝุ่นละอองเพียง 2 เดือนและเฉพาะพื้นที่การผลิตเดียวเท่านั้น หากต้องมีการเปรียบเทียบข้อมูลในหลายเดือนและหลายพื้นที่ รวมทั้งการสรุปข้อมูลปริมาณฝุ่นละออง จะทำให้เกิดการจัดเรียงข้อมูลขนาดใหญ่สำหรับการสร้างรายงานแบบ Manual Report จึงทำให้การเข้าถึงข้อมูลในเท็กซ์ไฟล์และจัดเรียงข้อมูลยุ่งยากซับซ้อนและมีข้อจำกัดเกี่ยวกับเวลา ดังนั้นการพัฒนาและออกแบบโครงสร้างการออกรายงานใหม่แทนการออกรายงานแบบเดิมจะช่วยลดข้อจำกัดดังกล่าวลงได้

3.1 ความต้องการรายงาน

จากการศึกษาความต้องการการออกรายงานบนเว็บแทนการออกรายงานแบบ Manual Report ประกอบด้วยความต้องการหลัก (Functional Requirement) และความต้องการที่ไม่ใช่หน้าที่หลัก ดังนี้



	1-30 September	1-31 October
M2A-HSA VCN	2	10
M5A-Disk mnt	10	7
M4B-Ramp mnt	2	5
M1B-Top clamp	5	5
M3A-Hyper DB	17	4
M1A-Enviroment	4	11
M4A-Hyper DB	8	11
M5B-HSA VCN	9	8
M3B-Disk mnt	20	7
M15A-Top Clamp	3	3
M15A-Disk mnt	4	6



รูปที่ 2 รายงานเปรียบเทียบปริมาณฝุ่นละอองที่เกินมาตรฐานที่กำหนด โดยการสร้างรายงานแบบ Manual Report

3.1.1 Functional Requirement

- สามารถเชื่อมต่อกับระบบฐานข้อมูล (Database System) DB2 เพื่อตรวจสอบข้อมูลปริมาณฝุ่นละออง
- แสดงกราฟเปรียบเทียบปริมาณฝุ่นละอองในแต่ละ Sensor ตามช่วงเวลาหรือแต่ละวันได้
- แสดงกราฟเปรียบเทียบระหว่างค่า Control Limit และ Specification Limit และค่าปริมาณฝุ่นละอองของแต่ละ Sensor
- แสดงกราฟเปรียบเทียบจำนวนครั้งการเกิด out of control ในแต่ละเดือนได้

3.1.2 Non-Functional Requirement

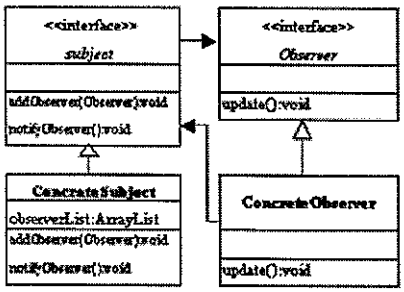
- มีระบบ Authentication สำหรับกำหนดสิทธิ์ผู้เข้าใช้งาน
- รูปแบบการใช้งานการออกรายงานมีส่วนติดต่อกับผู้ใช้งาน (Graphic User Interface: GUI) ที่ง่ายต่อการใช้งาน

4. โครงสร้างคลาสสำหรับการออกรายงาน

จากข้อจำกัดของการออกรายงานแบบ Manual Report จึงมีแนวคิดในการพัฒนาการออกรายงานบนเว็บ โดยมีการจอกแบบระบบการออกรายงานตามกระบวนการวิเคราะห์และออกแบบระบบเชิงวัตถุ (Object Oriented Analysis and Design Methodology: OOADM) และ การออกแบบโครงสร้างคลาส (Class Diagram) ตาม Design Pattern สำหรับ Design Pattern ที่ใช้สำหรับการออกรายงานจะนำหลักการของ Observer Pattern มาประยุกต์ใช้สำหรับการออกรายงานเพื่อให้ระบบออกรายงานมีประสิทธิภาพมากยิ่งขึ้น ทั้งนี้เพื่อรองรับการการเพิ่มประเภทของรายงานในอนาคต ที่ข้อมูลปริมาณฝุ่นละออง สามารถนำมาสรุปหรือวิเคราะห์และสร้างรายงานสำหรับพัฒนาอุตสาหกรรมการผลิตที่หลากหลายมากยิ่งขึ้น

4.1 การออกแบบโครงสร้างโดยใช้ Observer Pattern

Observer Pattern เป็นรูปแบบหนึ่งของ Design Pattern มีความสัมพันธ์ระหว่างคลาสแบบหนึ่งต่อกลุ่ม หรือ One-To-Many เนื่องจากจะมี Object ที่ทำหน้าที่ตรวจสอบและควบคุมสถานะ (State) โดยหนึ่ง Object มีตัวควบคุมได้เพียงหนึ่งสถานะเท่านั้น ในทางตรงกันข้าม ระบบจะมีหลาย Observersขึ้นอยู่กับสถานะของ Subject ที่จะบอกว่าเป็นสถานะใดเมื่อเกิดการเปลี่ยนแปลงขึ้น โดยโครงสร้างทั่วไปของ Observer Pattern ในลักษณะของคลาสไดอะแกรม (Class Diagram) แสดงดังรูปที่ 3



รูปที่ 3 แสดงโครงสร้างคลาสใน Observer Pattern [4]

จากรูปที่ 3 แสดงโครงสร้างทั่วไปของคลาสใน Observer Pattern โครงสร้างคลาสประกอบด้วยคลาส 4 คลาส [4] ได้แก่

- คลาส Subject เป็นอินเทอร์เฟสคลาส ที่ประกอบไปด้วยเมธอดสำหรับการสร้าง Observer การลบ Observer รวมถึงการ แจ้งเตือนให้กับ Observer ในกรณีที่เกิดการเปลี่ยนแปลง
- คลาส Observer เป็นอินเทอร์เฟสคลาส ที่มีเมธอด update ที่จะทำงานเมื่อได้รับการแจ้งเตือนจากคลาส Subject
- คลาส Concrete Observer เป็นคลาสที่ Implement มาจากคลาสของ Observer ประกอบด้วยข้อมูลที่เกิดการเปลี่ยนแปลงโดยจะมีการจัดเก็บสถานะและเชื่อมต่อกับ Observer โดย Concrete Observer สามารถเพิ่มได้ตามต้องการแล้วแต่จำนวนสิ่งที่ต้องการเฝ้าสังเกตการณ์ในแต่ละระบบ
- คลาส Concrete Subject เป็นคลาสที่ Implement จากคลาสของ Subject ประกอบด้วยตัวแปรที่จัดเก็บสถานะเมื่อเกิดการเปลี่ยนแปลงขึ้น

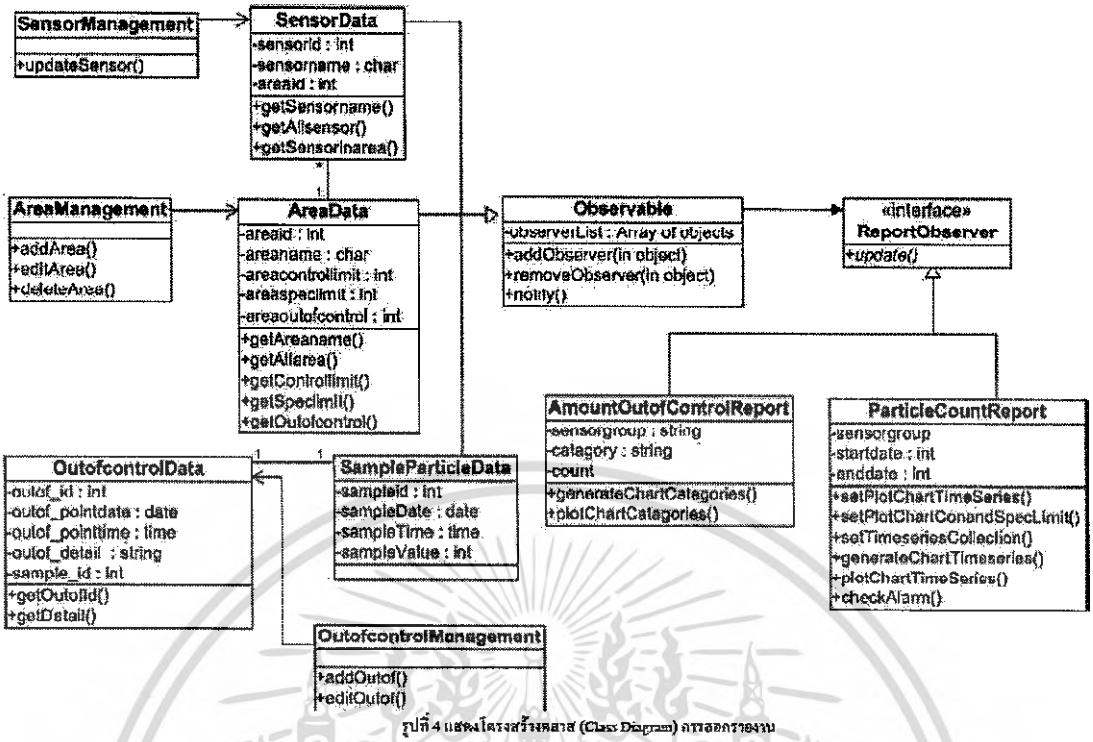
การออกแบบโครงสร้างคลาสของระบบการออกรายงานเพื่อสนับสนุนงานตรวจจับฝุ่นละออง จะใช้ Observer Pattern ซึ่งอยู่ในกลุ่มแพทเทิร์นของการแก้ไขปัญหาพฤติกรรมการทำงานระหว่าง Object โดยจะใช้ตัวเฝ้าสังเกตการณ์ (Observer) ในการตรวจสอบประเภทของรายงาน หลังจากนั้นจะทำงานตามเหตุการณ์ที่ผู้ใช้งานเลือกรายงาน โดยแนวคิดการออกแบบโครงสร้างคลาสแสดงดังรูปที่ 4

จากรูปที่ 4 แสดงโครงสร้างแนวคิดของ Class Diagram และความสัมพันธ์ระหว่างคลาส (Class) ของระบบการออกรายงาน โดยการทำงานหลักในส่วนของการออกรายงานจะเริ่มจากคลาส ReportObserver จะมีหน้าที่ในการสังเกตการเปลี่ยนแปลงความต้องการรายงานจากผู้ใช้งาน จากคลาสของ User โดยการทำงานของเมธอด update ในคลาส ReportObserver จะถูกเรียกใช้งานเมื่อมีการเรียกใช้เมธอด notify จากคลาส Observable สำหรับการทำงานในแต่ละคลาส แสดงดังนี้

- คลาสจัดการตัวเฝ้าสังเกตการณ์ (Observable) เป็นอินเทอร์เฟสคลาส ประกอบด้วยเมธอดสำหรับการเพิ่ม Observer เมธอดสำหรับการลบ Observer และเมธอดสำหรับการแจ้งเตือนได้แก่ เมธอด notify
- คลาสการเฝ้าสังเกตรายงาน (ReportObserver)เป็นอินเทอร์เฟสคลาส ประกอบด้วยเมธอด ในการทำให้เกิดการตรวจสอบสถานะของ Object โดยเมธอด update จะถูกเรียกใช้งานทันทีหากมีการแจ้งเตือนการเปลี่ยนแปลงสถานะ
- คลาสรายงานปริมาณฝุ่นละออง (ParticleCountReport) เป็นคลาสที่ Implement จากคลาสของ ReportObserver ทำหน้าที่ในการตรวจสอบและสร้างรายงานปริมาณฝุ่นละอองของแต่ละเซ็นเซอร์ (Particle Count Report) เพื่อเปรียบเทียบกับปริมาณฝุ่นมาตรฐาน ประกอบด้วยเอตทริบิวต์ที่ใช้สำหรับเก็บข้อมูลการสร้างรายงาน เช่น เวลาเริ่มต้น (startDate) และ เวลาสิ้นสุด (endDate) และเมธอดที่ใช้สำหรับการสร้างกราฟ เช่น เมธอด generateChartTimeSeries เป็นต้น
- คลาสรายงานสรุปจำนวนครั้งที่เกินปริมาณฝุ่นเกินมาตรฐาน (AmountOutofControlReport) เป็นคลาสที่ Implement จากคลาสของ ReportObserver ทำหน้าที่ตรวจสอบและสร้างรายงานจำนวนครั้งของการเกิดปริมาณฝุ่นเกินมาตรฐานในพื้นที่การผลิต (Amount out of control Report) ประกอบด้วย แอตทริบิวต์ ที่ใช้สำหรับเก็บข้อมูลการสร้างรายงาน เช่น อุปกรณ์เซ็นเซอร์ที่ต้องการนับจำนวนครั้งที่ปริมาณฝุ่นเกินมาตรฐานที่กำหนด (sensorgroup) หรือ ชุดวันที่สำหรับแบ่งกลุ่ม (category) และเมธอดที่ใช้สำหรับการสร้างกราฟ เช่น เมธอด generateChartCategories เป็นต้น

นอกจากนี้สามารถอธิบายโครงสร้างคลาสอื่นที่มีความสัมพันธ์เกี่ยวกับการส่วนของการออกรายงานทั้งหมดได้ดังนี้

- คลาสจัดการข้อมูลพื้นที่ (AreaManagement) ทำหน้าที่จัดการพื้นที่ในสายการผลิต โดยสามารถทำการเพิ่มพื้นที่ด้วย เมธอด addArea แก้ไขด้วยเมธอด editArea และลบด้วย เมธอด deleteArea
- คลาสข้อมูลพื้นที่ (AreaData) เป็นคลาสแสดงรายละเอียดของ Area ในสายการผลิต ซึ่งประกอบด้วย หมายเลขพื้นที่ (areaid) ชื่อพื้นที่ (areaname) ค่าปริมาณฝุ่นมาตรฐาน (controllimit) ค่าปริมาณฝุ่นตามที่กำหนด
- คลาสการจัดการเซ็นเซอร์ (SensorManagement) ทำหน้าที่จัดการเซ็นเซอร์ (Sensor) ที่อยู่ในพื้นที่การผลิต โดยผู้ใช้งานจะสามารถทำการแก้ไขหรือเปลี่ยนแปลงพื้นที่การผลิต ในทุกๆ เซ็นเซอร์ได้ผ่านเมธอด updateSensor
- คลาสข้อมูลเซ็นเซอร์ (SensorData) เป็นคลาสแสดงรายละเอียดของเซ็นเซอร์ทั้งหมด ประกอบด้วย รหัสเซ็นเซอร์ (sensorid) ชื่อเซ็นเซอร์ (sensorname) รหัสพื้นที่การผลิตที่เซ็นเซอร์นั้นทำงานอยู่ (areaid) ดังนั้นคลาส SensorData จะมีความสัมพันธ์กับคลาส AreaData แบบหนึ่งต่อกลุ่ม (One-Two-Many)
- คลาสจัดการข้อมูลเกินควบคุม (OutofcontrolManagement) เป็นคลาสที่ทำหน้าที่จัดการข้อมูลที่เกินปริมาณมาตรฐานที่กำหนด โดยสามารถเพิ่มข้อมูลได้ด้วย เมธอด addOutof และแก้ไขข้อมูลได้ด้วย เมธอด editOutof



รูปที่ 4 แสดงโครงสร้างคลาส (Class Diagram) การออกรายงาน

- คลาสข้อมูลเกินควบคุม (OutofcontrolData) เป็นคลาสที่ทำหน้าที่จัดการกับรายละเอียดในกรณีที่เกิด out of control (ปริมาณฝุ่นละออง ณ จุดนั้นเกินมาตรฐานที่กำหนดไว้) เช่น สาเหตุของการเกิด out of control เป็นต้น
- คลาสข้อมูลปริมาณฝุ่นละออง (ParticleData) เป็นคลาสจัดเก็บข้อมูลของปริมาณฝุ่นละอองตามวัน (particledate) และเวลา (particletime) ของแต่ละเซ็นเซอร์ (sensorid) ซึ่งจะมีการจัดเก็บจากข้อมูลปริมาณฝุ่นละอองจากเท็กซ์ไฟล์ (CSV File) ที่มาจากสายการผลิต

จากคลาสทั้งหมดที่กล่าวข้างต้นจะมีความสัมพันธ์ระหว่างคลาสที่ใช้สำหรับการออกรายงาน เนื่องจากจะมีการนำข้อมูลปริมาณฝุ่นละอองมาเพื่อใช้ในการสร้างกราฟ เช่น เปรียบเทียบข้อมูลปริมาณฝุ่นละอองจากแต่ละเซ็นเซอร์ กับปริมาณฝุ่นละอองที่กำหนดไว้ในคลาส หรือ สรุปจำนวนครั้งที่เกิดปริมาณฝุ่นเกินมาตรฐานที่กำหนด เป็นต้น

4.2 ข้อดีสำหรับการออกแบบโดยใช้ Observer Pattern

การศึกษาและออกแบบระบบในส่วนของการออกรายงานโดยใช้ Observer Pattern มีข้อดีหลายประการ ดังนี้

- ไม่จำเป็นต้องมีการแก้ไขหรือเปลี่ยนแปลงข้อมูลสำหรับการออกรายงานเมื่อมีการเพิ่มหรือลด ประเภทรายงาน (ReportObserver) เพียงแต่สามารถสามารถสร้างคลาส (Class) ขึ้นมาเพิ่มเติม จากคลาส ReportObserver
- ความสัมพันธ์ของวัตถุ (Object) ที่สามารถติดต่อสื่อสารกันได้แต่เป็นอิสระไม่ขึ้นต่อกันนี้ทำให้ Observer Pattern มีความสับเซ็นธ์ระหว่างคลาสกันต่ำ (Loose Coupling)
- เมื่อการเปลี่ยนแปลงหรือแก้ไข Objectใด Object หนึ่ง ไม่มีผลกระทบหรือขึ้นต่อกันโดยตรงแล้วจึงทำให้สามารถนำไปใช้ซ้ำได้ หรือเรียกว่า Reusable
- ยืดหยุ่นต่อการเปลี่ยนแปลง แต่ละ Object อย่างมีอิสระ

5. บทสรุปและงานในอนาคต

บทความนี้ได้นำเสนอ การออกแบบโครงสร้างคลาสสำหรับการออกรายงานในระบบฐานข้อมูลเพื่อสนับสนุนงานตรวจจับฝุ่นละอองในอุตสาหกรรมการผลิตฮาร์ดดิสก์ โดยการออกแบบดังกล่าวจะช่วยเพิ่มประสิทธิภาพการออกรายงานโดยนำ Design pattern เข้ามามีส่วนช่วยในการออกแบบซึ่งมีข้อดีหลายประการ เช่น การออกแบบเป็นอิสระไม่ขึ้นต่อกัน มีลักษณะเป็น Loose Coupling เป็นต้น

สำหรับการทำงานในอนาคตจะมีการนำโครงสร้างที่ออกแบบมาพัฒนาระบบฐานข้อมูลเพื่องานตรวจจับฝุ่นละอองในส่วนของการออกรายงานรวมทั้งนำไปใช้ในการทำงานจริงสำหรับออกรายงานปริมาณฝุ่นละอองในอุตสาหกรรมการผลิตฮาร์ดดิสก์ต่อไป

6. กิตติกรรมประกาศ

โครงการพัฒนาระบบงานนี้เป็นส่วนหนึ่งของโครงการพัฒนาทรัพยากรมนุษย์ในอุตสาหกรรมฮาร์ดดิสก์ได้รับทุนจาก I/UCRC (Industry/University Cooperative Research Center) ของศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (National Electronics and Computer Technology Center: NECTEC) ภายใต้การสนับสนุนของวิทยาลัยร่วมด้านเทคโนโลยีการบันทึกข้อมูลและการประยุกต์ใช้งาน (Data Storage Technology and Applications: DSTAR) สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง (King Mongkut's Institute of Technology Ladkrabang: KMITL) และร่วมกับบริษัทฮิตาชิ โกลบอล สตอเรจ ประเทศไทย จำกัด (Hitachi Global Storage Technology: HGST)

เอกสารอ้างอิง

[1] R. W. Welker, "Guideline for Design and Certification of Cleanroom Tooling", Clean room Tooling Document, Revised by Don DeLeo/Tim Karlberg, San Jose Site Contamination Control, August 8 2002.

[2] Training & Support Department, "Hard Disk Drive Process Overview", Training Document. Hitachi Global Storage Technologies, April 2010.

[3] Manufacturing Environment Engineering Department, "ESD and Contamination Control", Hitachi Global Storage Technologies, April 2009.

[4] Z. Yueping, L. Yuefan and X. Kesheng, "The Compound Pattern on the Chain of Responsibility and Observer", *International Forum on Computer Science-Technology and Applications*, vol. 3, pp. 420-422, 2009.

ที่ วช ๐๐๐๑.๕/ ๔๖๗

สำนักอำนวยการกลาง
สำนักงานคณะกรรมการวิจัยแห่งชาติ
๑๙๖ ถนนพหลโยธิน จตุจักร กทม. ๑๐๙๐๐

๑๗ มกราคม ๒๕๕๔

เรื่อง แจ้งผลการพิจารณาบทความวิจัยที่จะลงตีพิมพ์ในวารสารสำนักงานคณะกรรมการวิจัยแห่งชาติ

เรียน นางสาวอภัสรา วิโรจน์ยะกูล

ตามที่ท่านได้ส่งบทความวิจัย เรื่อง **"ระบบสนับสนุนฐานข้อมูลสำหรับการออกรายงานปริมาณข้อมูลฝุ่นละออง"** มาเพื่อขอให้สำนักงานคณะกรรมการวิจัยแห่งชาติ (วช.) พิจารณานำลงพิมพ์ในวารสารสำนักงานคณะกรรมการวิจัยแห่งชาติ นั้น

บัดนี้ บทความของท่านได้รับการพิจารณาจากคณะผู้ทรงคุณวุฒิเรียบร้อยแล้ว โดยมีมติให้พิมพ์ในวารสารสำนักงาน ฯ ได้ ทั้งนี้ส่วนประชาสัมพันธ์และเผยแพร่ สำนักอำนวยการกลางซึ่งได้รับมอบหมายในการจัดทำวารสาร ฯ และเป็นผู้จัดการประจำกองบรรณาธิการวารสารสำนักงาน ฯ จะนำบทความวิจัยของท่านลงพิมพ์ในวารสารสำนักงาน ฯ ฉบับวิทยาศาสตร์ ปีที่ ๔๓ เล่ม ๑ (มกราคม - มิถุนายน ๒๕๕๔) ต่อไป

จึงเรียนมาเพื่อโปรดทราบ และขอขอบคุณที่ท่านให้ความสนใจนำบทความวิจัยมาลงพิมพ์ในวารสารสำนักงานฯ หวังเป็นอย่างยิ่งว่าในโอกาสต่อไปจะได้รับความสนใจจากท่านอีก

ขอแสดงความนับถือ

(นางสาวสุดาวีณ์ นุกูลกิจเสรี)
ผู้อำนวยการสำนักอำนวยการกลาง

ส่วนประชาสัมพันธ์และเผยแพร่
โทร. ๐ ๒๕๗๘ ๐๔๓๑ , ๐ ๒๕๖๑ ๒๔๔๕ ต่อ ๕๓๖
โทรสาร ๐ ๒๕๗๘ ๙๗๗๕
E-mail : info@nrct.go.th

ระบบสนับสนุนฐานข้อมูล
สำหรับการออกรายงานปริมาณข้อมูลฝุ่นละอองบนเว็บ
WEB-BASED DATABASE SUPPORT SYSTEM
FOR PARTICLE MONITORING REPORTS

อภัสรา วิโรจน์ยะกูล

Aphatsara Wirotyakun

พรฤดี เนติโสภากุล

Ponrudee Netisopakul

คณะเทคโนโลยีสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

Faculty of Information Technology, King Mongkut's Institute of Technology Ladkrabang

---

บทคัดย่อ

ปัจจุบันข้อมูลปริมาณฝุ่นละอองภายในห้องสะอาดของอุตสาหกรรมการผลิตฮาร์ดดิสก์ จะถูกโอนย้าย สรุปและสร้างรายงานโดยมนุษย์ ด้วยโปรแกรมไมโครซอฟท์เอ็กซ์เซล ซึ่งขั้นตอนการทำงานโดยมนุษย์ดังกล่าวค่อนข้างเสียเวลา ในการจัดเรียงและสรุปข้อมูลสำหรับสร้างรายงานตามความต้องการที่หลากหลาย ดังนั้น จึงมีความต้องการจะพัฒนาระบบฐานข้อมูลเพื่องานตรวจจับฝุ่นละออง เพื่อคัดกรองและถ่ายโอนข้อมูลดิบจากระบบ PMS เดิม มาเก็บในฐานข้อมูล DB2 โดยอัตโนมัติ และออกรายงานให้มีประสิทธิภาพมากขึ้น โดยเอกสารฉบับนี้จะกล่าวถึงรายละเอียดของการออกแบบระบบย่อยในส่วนของการออกรายงานจากฐานข้อมูล DB2 เพื่อสนับสนุนการทำงานของระบบ PMS เดิม ซึ่งการพัฒนาจะใช้หลักการของกระบวนการวิเคราะห์และออกแบบระบบเชิงวัตถุและเทคโนโลยีไคลเอนต์/เซิร์ฟเวอร์ มาปรับใช้เพื่อให้ระบบมีประสิทธิภาพมากขึ้น

คำสำคัญ: ห้องสะอาด, อุตสาหกรรมการผลิตฮาร์ดดิสก์, ระบบตรวจจับฝุ่นละออง, ระบบฐานข้อมูลสายการผลิต

Abstract

Currently, the quantitative airborne particle data in clean room of Hard Disk Drive (HDD) Manufacturing is manually transferred, summarized and created reports using a Microsoft Excel tool. This human activity takes a long time, because there are a number of reports that need to be created from various requirements. Therefore, we want to develop a database support system for particle monitoring. The system automatically filters and transfers raw data from the existing PMS system to store in a DB2 database in a new system for efficiently generating reports. This paper describes a design of web-based particle monitoring reports from DB2 database subsystem, which extends the functionality of the existing PMS system. The work in this paper applies object-oriented analysis and design methodology, client/server architecture technology to improve the overall PMS system.

Keyword: Clean room, HDD Manufacturing, Particle Monitoring System, Database System Production Line

บทนำ

กระบวนการหรือขั้นตอนการประกอบฮาร์ดดิสก์ในอุตสาหกรรมการผลิตฮาร์ดดิสก์ (Hard disk drive Manufacturing) มนุษย์และเครื่องจักรต้องทำงานร่วมกันภายในห้องสะอาดหรือห้องคลีนรูม (Clean room) หมายถึงห้องที่มีการควบคุมปริมาณฝุ่นละอองไม่ให้เกินมาตรฐานที่กำหนดไว้ โดยข้อมูลของปริมาณฝุ่นละอองจะถูกแสดงผ่านระบบ PMS (Particle Monitoring System) ซึ่งเป็นโปรแกรมสำเร็จรูปที่ติดตั้งมาพร้อมกับอุปกรณ์ฮาร์ดแวร์สำหรับตรวจสอบปริมาณฝุ่นละอองภายในแต่ละวัน โดยจะต้องมีการเฝ้าระวังปริมาณฝุ่นไม่ให้เกินมาตรฐานที่กำหนดไว้ ขณะเดียวกันข้อมูลปริมาณฝุ่นละอองที่มีมากขึ้นในแต่ละวันจะต้องนำมาสรุปเพื่อใช้สำหรับการปรับปรุงทำงานในสายการผลิตต่อไป โดยวิศวกรที่ทำหน้าที่หลักสำหรับมอนิเตอร์ข้อมูลปริมาณฝุ่นละอองจะนำข้อมูลปริมาณฝุ่นละอองที่ได้จากการจัดเก็บจะนำมาสร้างรายงานโดยมนุษย์ (Manually Report) จากโปรแกรมสำเร็จรูป เช่น โปรแกรมไมโครซอฟต์เอ็กซ์เซล (Microsoft Excel) การทำงานโดยมนุษย์ดังกล่าว ค่อนข้างเสียเวลาสำหรับการสร้างรายงานเพื่อสรุปปริมาณฝุ่นละอองตามความต้องการของผู้ใช้งาน เนื่องจากต้องมีการนำข้อมูลดังกล่าวมาจัดเรียงใหม่ในโปรแกรมไมโครซอฟต์เอ็กซ์เซล จากนั้นจึงสร้างรายงานจากโปรแกรมดังกล่าว ดังนั้นเพื่อลดเวลาและขั้นตอนการทำงานการออกรายงานข้างต้น จึงได้มีการพัฒนาระบบฐานข้อมูลสำหรับงานตรวจจับฝุ่นละออง (Database Support System for Particle Monitoring) ในส่วนของการออกรายงานบนเว็บที่มีการเชื่อมต่อกับปริมาณฝุ่นละอองในระบบฐานข้อมูล

งานด้านการประกอบชิ้นส่วนในอุตสาหกรรมการผลิตฮาร์ดดิสก์นั้น ขั้นตอนสำคัญสำหรับการควบคุมการผลิตให้เป็นไปอย่างมีประสิทธิภาพได้แก่ การควบคุมปัจจัยต่างๆ ที่มีผลกระทบต่อการผลิต

ระหว่างการทำงานของมนุษย์และเครื่องจักร เช่น การปนเปื้อน ความชื้น รวมถึงปริมาณฝุ่นละอองภายในห้อง เพื่อให้การควบคุมปัจจัยดังกล่าวมีประสิทธิภาพมากขึ้น งานวิจัยส่วนใหญ่จึงเน้นการปรับปรุงประสิทธิภาพหรือใช้เทคนิคต่างๆ เพื่อศึกษาปัจจัยต่างๆ ที่เกี่ยวข้องกับอากาศภายในห้องคลีนรูมทั้งนี้เนื่องจาก การควบคุมปัจจัยต่างๆ ภายในห้องให้มีประสิทธิภาพมีผลต่องานด้านการผลิตโดยตรง โดยได้มีการศึกษาและวัดปริมาณฝุ่นละออง โดยการจำลองการแสดงภาพการไหลของอากาศภายในห้องคลีนรูม[1] ใช้เทคนิคการแสดงภาพเสมือนภายในห้องคลีนรูม ซึ่งนอกจากการวัดความชื้นอุณหภูมิแล้ว มีการวัดการการกระจายตัวของปริมาณฝุ่นละออง โดยใช้เครื่องนับจำนวนฝุ่นละอองเพื่อตรวจจับปริมาณฝุ่นละอองภายในห้องที่มีการควบคุมปริมาณฝุ่นละออง นอกจากนี้งานวิจัยทางด้านวิศวกรรมเกี่ยวกับการออกแบบและคุณลักษณะของกระบวนการประกอบชิ้นส่วนภายในห้องคลีนรูม[2] ได้มีการจัดลำดับลักษณะการออกแบบสำหรับกระบวนการประกอบชิ้นส่วนภายในห้องคลีนรูม ในกระบวนการประกอบฮาร์ดดิสก์ โดยลักษณะประการหนึ่งที่ควรออกแบบห้องคลีนรูมได้แก่ ควรจะมีการเข้าถึงข้อมูลและสารสนเทศได้ง่าย จากงานวิจัยดังกล่าวข้างต้น จะเห็นว่า ปริมาณฝุ่นละอองภายในห้องคลีนรูม (Clean room) เป็นส่วนสำคัญที่อุตสาหกรรมด้านการประกอบชิ้นส่วนฮาร์ดดิสก์จะต้องคำนึงถึง เนื่องจากข้อมูลปริมาณฝุ่นละอองดังกล่าวมีความสำคัญอย่างมาก ดังเช่น งานวิจัยที่กล่าวถึงระบบการตรวจสอบอย่างต่อเนื่องในอุตสาหกรรมการผลิตฮาร์ดดิสก์[3] ได้กล่าวถึง การตีความข้อมูลและรายงานที่ได้จากห้องคลีนรูม ผ่านเครือข่ายภายในองค์กร (LAN) หรืออินเทอร์เน็ต (Internet) ซึ่งจากงานวิจัยทั้งหมดที่ได้กล่าวมาแล้วนั้นข้อมูลในสายการผลิต ที่อยู่ภายในห้องคลีนรูม เป็นส่วนสำคัญต่ออุตสาหกรรมการผลิต ดังนั้นจึงมีการจัดเก็บข้อมูลดังกล่าวในระบบฐานข้อมูล

ระบบฐานข้อมูลที่ใช้สำหรับจัดเก็บและเชื่อมต่อเพื่อพัฒนาส่วนย่อยของระบบการออกรายงานปริมาณฝุ่นละออง ได้แก่ ฐานข้อมูล DB2 ข้อดีประการหนึ่งของการเลือกใช้ฐานข้อมูล DB2 ได้แก่ การรองรับการทำงานของหลายระบบปฏิบัติการ เช่น งานวิจัยที่ศึกษาคุณลักษณะของฐานข้อมูล DB2 ที่สนับสนุนการทำงานของลีนุกซ์ (Linux) ยูนิกส์ (UNIX) และวินโดวส์ (Windows)[4] ซึ่งได้กล่าวถึงการบริหารจัดการฐานข้อมูลเชิงสัมพันธ์ (Relational Database) ในระบบปฏิบัติการอื่นนอกเหนือจากวินโดวส์ โดยใช้คุณสมบัติที่สำคัญบางประการของฐานข้อมูล DB2

สำหรับเนื้อหาในบทความจะประกอบด้วย ส่วนที่สองกล่าวถึง อุปกรณ์และวิธีการที่ใช้สำหรับการวิจัย ได้แก่ การศึกษาระบบการทำงานเดิมภายในห้องคลีนรูม กระบวนการออกรายงานที่ต่อยอดจากระบบ PMS โดยใช้หลักการ การวิเคราะห์และออกแบบระบบเชิงวัตถุ รวมถึง เทคโนโลยีที่ใช้สำหรับการพัฒนา ส่วนที่สามกล่าวถึง ผลลัพธ์ที่ได้จากการศึกษา ออกแบบโครงสร้าง และโปรแกรมต้นแบบของระบบสนับสนุนฐานข้อมูลสำหรับการออกรายงานปริมาณข้อมูลฝุ่นละอองบนเว็บ ส่วนที่สี่ อภิปรายและวิจารณ์ผลที่ได้จากการออกแบบ และส่วนสุดท้ายเป็นบทสรุปและงานในอนาคตที่สามารถต่อยอดจากงานที่กล่าวมาข้างต้น

อุปกรณ์และวิธีการ

รายละเอียดของข้อมูล เครื่องมือ รวมถึงกระบวนการและขั้นตอนสำหรับการวิเคราะห์และออกแบบระบบสนับสนุนฐานข้อมูลสำหรับการพัฒนาต้นแบบการออกรายงานปริมาณข้อมูลฝุ่นละอองบนเว็บประกอบด้วย

1. ศึกษากระบวนการจัดเก็บข้อมูลและออกรายงานสำหรับระบบเดิม

1.1 กระบวนการทำงานภายในห้องคลีนรูม

กระบวนการประกอบชิ้นส่วนในอุตสาหกรรมการผลิตฮาร์ดดิสก์ (Hard Disk Drive Manufacturing) มนุษย์และเครื่องจักรจะทำงานร่วมกันภายในห้องที่มีการควบคุมปริมาณฝุ่นละอองหรือ ห้องคลีนรูม (Clean room) ซึ่งเป็นห้องที่มีการควบคุมสภาพแวดล้อม (Environment) โดยส่วนใหญ่จะ ใช้สำหรับงานวิจัยทางด้านวิทยาศาสตร์หรืองานทางด้านอุตสาหกรรมการผลิตที่ต้องมีการควบคุมปัจจัยต่างๆ โดยปัจจัยสำคัญสำหรับใช้ในการควบคุมในกระบวนการผลิตสำหรับอุตสาหกรรมการประกอบชิ้นส่วนฮาร์ดดิสก์ ได้แก่ ปริมาณฝุ่นละอองที่ผ่านมาทางอากาศภายในห้องที่มีการประกอบชิ้นส่วนการผลิต

โดยห้องคลีนรูมมีการแบ่งระดับไว้เป็นคลาส (Class) แต่ละคลาส จะแตกต่างกันตามตามจำนวนขนาดของอนุภาคที่ยอมให้มีได้ในอากาศ ต่อหนึ่งหน่วยปริมาตรลูกบาศก์ฟุต (number of particles per cubic meter)[5] สำหรับมาตรฐานที่ใช้สำหรับการกำหนดปริมาณฝุ่นละอองในแต่ละคลาสนั้นมีหลายมาตรฐาน ได้แก่ มาตรฐานยูเอส (US) มาตรฐานไอเอสโอ (ISO) และมาตรฐานบีเอส (BS) โดยแต่ละมาตรฐานจะมีการกำหนดระดับของคลาสและปริมาณฝุ่นที่มีได้ในแต่ละคลาสไว้แตกต่างกัน สำหรับการแบ่งระดับของห้องคลีนรูม (Clean room)ในอุตสาหกรรมการผลิตฮาร์ดดิสก์ที่ทำการวิจัยนั้นได้มีการแบ่งระดับของคลาสออกเป็น 3 ระดับตามมาตรฐานของไอเอสโอ ได้แก่ คลีนรูมคลาสสิบ (Clean room Class 10) คลีนรูมคลาสร้อย (Clean room class 100) และคลีนรูมคลาสหมื่น (Clean room Class 10000: 10K)[6] โดยแต่ละระดับนั้นจะกำหนดจำนวนฝุ่นละอองที่มีได้ภายในห้องตามขนาดของปริมาณฝุ่นละออง (ขนาดของฝุ่นละอองวัดจากเส้นผ่านศูนย์กลางของอนุภาค) เช่น ปริมาณฝุ่นละอองขนาด 0.5 ไมครอน หมายถึง ปริมาณฝุ่นละอองที่มีขนาดเส้นผ่านศูนย์กลาง 0.5 ไมครอน จะมีขนาดใหญ่กว่าปริมาณฝุ่นละอองที่มีขนาด 0.3 ไมครอน แสดงการแบ่งระดับปริมาณฝุ่นละอองของห้องคลีนรูมตามมาตรฐานของไอเอสโอ ดังนี้

5

ตารางที่ 1 การแบ่งระดับปริมาณฝุ่นละอองเป็นคลาส (Class) ตามมาตรฐานของ ISO ในอุตสาหกรรมการผลิตฮาร์ดดิสก์

Clean room Classes	Process Area	Cleanliness Specification
10K	HAS & Cleaning	10,000 particles/ft^3 @ 0.5 μm 30,000 particles/ft^3 @ 0.3 μm
100	HDE	100 particles/ft^3 @ 0.5 μm 300 particles/ft^3 @ 0.3 μm
10	HDE Assembly station	10 particles/ft^3 @ 0.5 μm 30 particles/ft^3 @ 0.3 μm

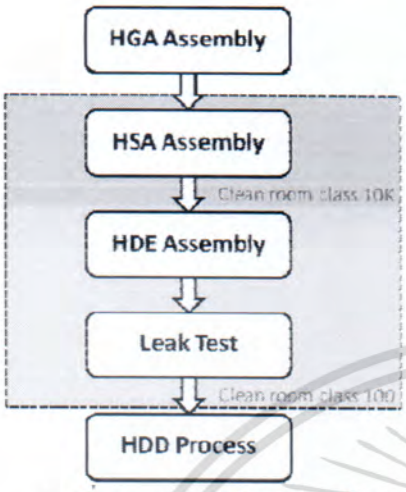
จากตารางที่ 1 [6] แสดงการแบ่งระดับคลาสของปริมาณฝุ่นละอองในอุตสาหกรรมการประกอบชิ้นส่วนฮาร์ดดิสก์ตามมาตรฐาน ISO โดยแบ่งคลีนรูมออกเป็น 3 ระดับ ได้แก่

1) Clean room Class 10 (คลาสสิบ) กำหนดให้มีปริมาณฝุ่นละอองไม่เกิน 10 particles ที่ขนาด 0.5 ไมครอน ไม่เกิน 30 particles ที่ขนาด 0.3 ไมครอน ต่อหนึ่งลูกบาศก์ฟุต

2) Clean room Class 100 (คลาสร้อย) มีปริมาณฝุ่นได้ไม่เกิน 100 particles ที่ขนาด 0.5 ไมครอน และ ไม่เกิน 300 particles ที่ขนาด 0.3 ไมครอน ต่อหนึ่งลูกบาศก์ฟุต

3) Clean room Class 10K (คลาสหมื่น) กำหนดให้มีปริมาณฝุ่นละอองได้ไม่เกิน 10,000 particles ต่อหนึ่งลูกบาศก์ฟุต ที่ขนาดอนุภาค 0.5 ไมครอน และได้ไม่เกิน 30,000 particles ที่ขนาดอนุภาค 0.3 ไมครอน ต่อหนึ่งลูกบาศก์ฟุต

สำหรับการแบ่งพื้นที่การผลิตนั้นจะพิจารณาจากการทำงาน โดยแต่ละพื้นที่การผลิต (Process Area) จะมีหน้าที่การทำงานที่แตกต่างกัน บางพื้นที่อาจต้องประกอบชิ้นส่วนภายในห้องที่มีการควบคุมปริมาณฝุ่นละออง (ห้อง clean room) เช่น กระบวนการประกอบหัวอ่าน ขณะเดียวกันบางกระบวนการ ไม่จำเป็นต้องมีการควบคุมปริมาณฝุ่นละออง เช่น กระบวนการบรรจุเพื่อเตรียมส่งมอบให้ลูกค้า (packaging) เป็นต้น กระบวนการประกอบชิ้นส่วนในอุตสาหกรรมการผลิตฮาร์ดดิสก์และตัวอย่างบางกระบวนการผลิต [7] แสดงดังภาพที่ 1



ภาพที่ 1 กระบวนการในสายการผลิต (Production line) ของโรงงานอุตสาหกรรมการผลิตฮาร์ดดิสก์ (HDD Manufacturing)

1) HSA (Head Stack Assembly) Area เป็นพื้นที่สำหรับกระบวนการประกอบระบบของหัวอ่าน เช่นการนำ HGA (Head Gimbals Assembly เป็นการเชื่อม slider เข้ากับ suspension) มาเรียงซ้อนกันและเชื่อมต่อกับมอเตอร์สำหรับหมุนหัวอ่าน ทำงานในห้อง Clean room Class 10K

2) HDE (Hard Disk Enclosure) Area เป็นพื้นที่สำหรับกระบวนการประกอบ HSA เข้ากับชิ้นส่วนต่างๆ (ยกเว้น CARD) เช่น VCM ลงบน Base แล้วผ่านกระบวนการสร้าง Servo Pattern (การเขียนสัญญาณ) ลงบนวนจร เป็นต้น ทำงานในห้อง Clean room Class 100

3) HDD (Hard Disk Drive) Area เป็นพื้นที่สำหรับการประกอบฮาร์ดดิสก์ไดร์ฟ ซึ่งเป็นขั้นตอนสุดท้ายสำหรับการประกอบฮาร์ดดิสก์ จะรวมถึงการทดสอบภายใต้สภาวะต่างๆ และการติดฉลากของสินค้าเพื่อเตรียมสำหรับการส่งมอบให้ลูกค้า ซึ่งกระบวนการนี้จะทำงานในห้องทั่วไปที่ไม่ได้ควบคุมปริมาณฝุ่นละออง

จากภาพที่ 1 แสดงพื้นที่การผลิตในกระบวนการประกอบชิ้นส่วนฮาร์ดดิสก์ ซึ่งประกอบด้วยหลายพื้นที่ และการทำงานของแต่ละพื้นที่การผลิตจะมีการกำหนดคลาสของพื้นที่ดังตัวอย่าง HSA Assembly กำหนดอยู่ในคลีนรูมคลาสหมื่น (Clean room class 10K) หมายถึง ในกระบวนการที่มีการประกอบ HSA จะมีการควบคุมปริมาณฝุ่นละอองภายในอากาศให้มีได้ไม่เกิน 10000 Particle ต่อหนึ่งลูกบาศก์ฟุต ส่วนพื้นที่ HDE Assembly และ Leak Test จะทำงานอยู่ในคลีนรูมคลาสร้อย (Clean room class 100) หมายถึง ในกระบวนการ HDE และ Leak Test จะมีการควบคุมปริมาณฝุ่นละอองภายในห้องที่มีการประกอบชิ้นส่วนให้มีปริมาณฝุ่นละอองในอากาศได้ไม่เกิน 100 particle ต่อหนึ่งลูกบาศก์ฟุต ขณะที่พื้นที่ HDD Process นั้นเป็นพื้นที่ที่ไม่ต้องมีการควบคุมปริมาณฝุ่นละออง ดังนั้นจะทำงานอยู่ในพื้นที่ปกติ [7]

ค่ามาตรฐานที่จะต้องกำหนดไว้ในแต่ละพื้นที่ ซึ่งแตกต่างกันออกไปนั้น ประกอบด้วยค่ามาตรฐานสองค่า ได้แก่

1) Control Limit ระดับปริมาณฝุ่นมาตรฐานในกระบวนการผลิตภายในห้องคลีนรูมโดยทั่วไปจะได้จากการคำนวณของวิศวกรตามอุตสาหกรรมการผลิตนั้นๆ

2) Specification Limit ปริมาณฝุ่นละอองอยู่ภายใต้ข้อกำหนดของลูกค้าหรือตามมาตรฐานโรงงานในกระบวนการผลิตภายในห้องคลีนรูม

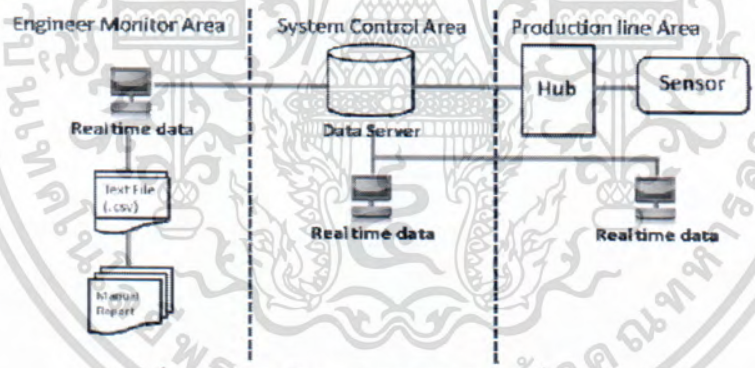
7

ซึ่งโดยทั่วไปแล้วการกำหนดค่า Specification Limit จะมีค่าน้อยกว่าการกำหนดค่า Control Limit เสมอ โดยหากปริมาณฝุ่นในเซ็นเซอร์ ณ จุดใด มีค่าปริมาณฝุ่นละอองมากกว่าค่า Control Limit ตามที่กำหนดไว้ หมายความว่า ณ จุดนั้นๆ มีการเกิดปริมาณฝุ่นเกินมาตรฐาน (out of control) ผู้ที่รับผิดชอบจะต้องหาสาเหตุการเกิดปริมาณฝุ่นละอองที่เกินการกำหนด ซึ่งอาจเกิดได้จากหลายสาเหตุ เช่น เกิดจากมีการปรับตั้งค่าอุปกรณ์เซ็นเซอร์ หรือ ปริมาณฝุ่นละอองอาจเกิดมาจากมนุษย์ที่ทำงานอยู่ ณ กระบวนการผลิตนั้นๆ เป็นต้น

1.2 กระบวนการจัดเก็บข้อมูลสำหรับระบบ PMS

การเก็บข้อมูลปริมาณฝุ่นละออง เริ่มจากการเก็บข้อมูลในสายการผลิตจากพื้นที่ต่างๆ โดยหนึ่งพื้นที่การผลิต (Process area) จะประกอบด้วยหลายสายการผลิต (Production Line) ในหนึ่งสายการผลิตจะประกอบด้วยหลายสเตชั่น (station) โดยสเตชั่น หมายถึง หนึ่งกระบวนการผลิต เช่น station ที่ทำหน้าที่ใส่ screw ของ Top Clamp เป็นต้น ซึ่งบางสเตชั่นจะมีการติดตั้งอุปกรณ์เซ็นเซอร์เพื่อตรวจจับปริมาณฝุ่นละอองภายในห้องคลีนรูม การติดตั้งเซ็นเซอร์ในแต่ละสเตชั่น นั้นขึ้นอยู่กับการออกแบบของวิศวกรของแต่ละประเภทอุตสาหกรรมการผลิต

จากนั้นในแต่ละสเตชั่น (Station) การทำงาน จะมีการเชื่อมต่อระหว่างซอฟต์แวร์และอุปกรณ์ฮาร์ดแวร์เพื่อเก็บค่าปริมาณฝุ่นละอองในแต่ละวันภายในห้องคลีนรูม ซึ่งจะเก็บข้อมูลปริมาณฝุ่นละอองดังกล่าวจากอุปกรณ์เซ็นเซอร์ แสดงดังภาพที่ 2



ภาพที่ 2 การจัดเก็บข้อมูลปริมาณฝุ่นละอองในพื้นที่ ที่เกี่ยวข้อง

จากภาพที่ 2 แสดงการจัดเก็บข้อมูลปริมาณฝุ่นละอองในพื้นที่ ที่เกี่ยวข้องจากสายการผลิต (Production Line) โดยการเก็บข้อมูลจะเริ่มจาก การเก็บข้อมูลจากอากาศโดยอุปกรณ์เซ็นเซอร์ (sensor) ในสายการผลิต ผ่านแสงเลเซอร์ หากแสงที่ได้เป็นแสงทึบจะนับปริมาณฝุ่นละอองเป็นหนึ่งจุด ตรงกันข้ามหากแสงที่ได้เป็นแสงใสแสดงว่าไม่มีฝุ่นละออง จากนั้นเซ็นเซอร์ จะส่งข้อมูลผ่านทางพอร์ต TCP/IP เก็บไว้ที่

ศูนย์กลางการเก็บข้อมูล (Data center) ในส่วนพื้นที่การควบคุม (System Control Area) ซึ่งเป็นพื้นที่กลางสำหรับเก็บข้อมูลและจัดเก็บเครื่องเซิร์ฟเวอร์หลัก จากนั้นวิศวกรที่ทำงานในพื้นที่มอนิเตอร์ข้อมูล (Engineer Monitor Area) จะนำข้อมูลที่เก็บในเซิร์ฟเวอร์ ในลักษณะของเท็กซ์ไฟล์ (.CSV) มาจัดเรียงและสร้างรายงานโดยมนุษย์

ซึ่งหลังจากที่ได้ข้อมูลปริมาณฝุ่นละอองจากเท็กซ์ไฟล์ แล้วนั้นวิศวกรที่ทำหน้าที่หลักสำหรับมอนิเตอร์ข้อมูลปริมาณฝุ่นละออง จะนำข้อมูลดังกล่าวมาจัดเรียงหรือปรับเปลี่ยนให้อยู่ในรูปแบบรายงานเชิงกราฟ (Graph-based Report) ด้วยโปรแกรมไมโครซอฟท์เอ็กซ์เซล แสดงดังภาพที่ 3

ภาพที่ 3 การจัดเรียงข้อมูลและสร้างกราฟปริมาณฝุ่นละอองจากเท็กซ์ไฟล์ด้วยโปรแกรมสำเร็จรูป

จากภาพที่ 3 แสดงการจัดเรียงข้อมูลปริมาณฝุ่นละอองที่ได้จากเท็กซ์ไฟล์ ในระบบ PMS เดิม โดยลักษณะของข้อมูลเท็กซ์ไฟล์ ในที่ 3 เป็นไฟล์ที่สามารถเปิดได้ด้วยโปรแกรมสำเร็จรูป เช่น เอ็กซ์เซล เป็นต้น โดยข้อมูลสำคัญที่จะนำมาสร้างกราฟ ประกอบด้วยภาพ

1) วันที่ที่จัดเก็บปริมาณฝุ่นละออง (Sample Date)

2) เวลาเก็บที่จัดเก็บปริมาณฝุ่นละออง (Sample Time)

3) ชื่อเซ็นเซอร์ที่จัดเก็บ (Sensor Name)

4) ปริมาณฝุ่นละออง (Sample Value)

โดยจะมีการเก็บข้อมูลจะเก็บข้อมูลทุกๆ 1 เดือน ดังนั้นใน 1 เดือนจะมีไฟล์ 1 ไฟล์ ขนาดไฟล์มีขนาดใหญ่ประมาณ 40-60 เมกะไบต์ (MB) จำนวนข้อมูลในแต่ละเดือนมีประมาณ 600,000 ข้อมูล (record) ทำให้การค้นหาข้อมูลในแต่ละวันหรือช่วงเวลา หรือแม้กระทั่งการตรวจสอบปริมาณฝุ่นละอองทำได้ยาก และเสียเวลาหรับการจัดเรียงข้อมูลใหม่ดังที่ได้กล่าวข้างต้น

จากภาพที่ 3 แสดงตัวอย่างการสร้างรายงาน เปรียบเทียบปริมาณฝุ่นละอองในเดือน มีนาคม (March) และเดือนเมษายน (April) เป็นการตรวจสอบปริมาณฝุ่นละอองที่เกินค่าควบคุม (Control Limit) ในพื้นการการผลิต HSA โดยขั้นตอนการจัดเรียงจะต้องนำข้อมูลที่อยู่ในไฟล์ จำนวนประมาณ 600,000 ข้อมูลต่อหนึ่งเดือน มาสรุปว่ามีเซ็นเซอร์ใดบ้างที่มีค่าเกินค่าควบคุม ใน HSA Process Area

ซึ่งจากตัวอย่างที่กล่าวมานั้นเป็นเพียงการออกรายงานเพื่อเปรียบเทียบข้อมูลปริมาณฝุ่นละอองในพื้นที่เดียวและเปรียบเทียบจากข้อมูลสองเดือนเท่านั้น หากต้องการเปรียบเทียบในทุกๆ พื้นที่การผลิต (Process Area) จะทำให้เกิดปัญหาทางด้านเวลาและระยะเวลาที่ใช้สำหรับจัดเรียงข้อมูลเพื่อสร้างรายงานแบบโดยมนุษย์ (Manual Report) ดังนั้นจึงมีการต่อยอดการพัฒนาระบบฐานข้อมูลเพื่องานตรวจจับฝุ่นละอองจากระบบ PMS เดิม เป็นระบบสนับสนุนฐานข้อมูลสำหรับการออกรายงานปริมาณข้อมูลฝุ่นละอองบนเว็บ (Web-based database support system for particle monitoring reports) เพื่อให้การจัดเก็บและออกรายงานมีประสิทธิภาพเพิ่มมากขึ้น

2. วิเคราะห์และออกแบบกระบวนการออกรายงานที่ต่อยอดจากระบบ PMS

2.1 ความต้องการของระบบใหม่

จากการวิเคราะห์ปัญหาและความต้องการของกระบวนการออกรายงานปริมาณฝุ่นละอองแบบ Manual Report ความต้องการหลัก (Functional Requirement) ของการพัฒนาระบบสนับสนุนฐานข้อมูลสำหรับการออกรายงานปริมาณข้อมูลฝุ่นละอองบนเว็บ มีความต้องการที่เป็นหน้าที่หลัก (Functional Requirement) ดังนี้

1) สามารถบันทึกและตั้งค่า Control Limit และ Specification Limit ได้

2) สามารถแสดงกราฟของปริมาณฝุ่นเป็นกราฟเส้นโดยสามารถเปรียบเทียบจำนวนฝุ่นแต่ละสเตชั่นโดยข้อมูลปริมาณฝุ่นละอองของกราฟ 1 เส้นต่อข้อมูลปริมาณฝุ่นของ 1 เซ็นเซอร์

3) สามารถออกรายงานเชิงกราฟของจำนวนฝุ่นละอองและเปรียบเทียบจำนวนฝุ่นในแต่ละสเตชั่นในแต่ละวันได้

4) สามารถออกรายงานเชิงกราฟของจำนวนฝุ่นละอองและเปรียบเทียบจำนวนฝุ่นในแต่ละสเตชั่นในแต่ละช่วงเวลาได้

5) ณ เวลาที่เกิดปริมาณฝุ่นเกินมาตรฐานที่กำหนดสามารถดู Action ได้ว่าสาเหตุการเกิดปริมาณฝุ่นเกินมาตรฐาน เกิดจากสาเหตุใดและมีผู้ดำเนินการแก้ไขให้ปริมาณฝุ่นอยู่ในระดับที่ควบคุมไว้ (Control) หรือไม่

6) สามารถออกรายงานเปรียบเทียบปริมาณฝุ่นที่เกินปริมาณมาตรฐานที่กำหนดได้

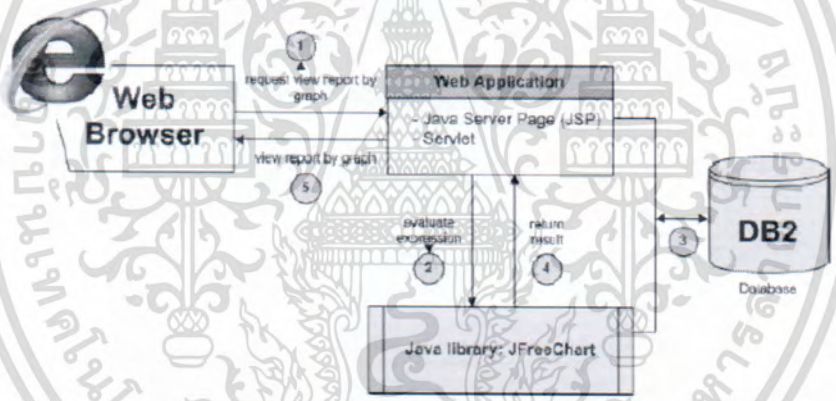
ทั้งนี้ระบบการออกรายงานดังกล่าวสามารถรองรับความต้องการที่ไม่ใช่หน้าที่หลัก (Non-functional Requirement) ประกอบด้วย

1) ระบบที่พัฒนาขึ้นมีส่วนติดต่อกับผู้ใช้งาน (user interface) ที่ง่ายต่อการใช้งาน

2) มีระบบการกำหนดสิทธิของผู้ใช้ (Authentication) สำหรับการกำหนดสิทธิ์ผู้เข้าใช้งาน

2.2 สถาปัตยกรรมของระบบใหม่โดยรวม

สถาปัตยกรรมการทำงานส่วนของการออกรายงานปริมาณฝุ่นละอองบนเว็บจะมีการเชื่อมต่อกับระบบฐานข้อมูล (Database) เพื่อแสดงผลข้อมูลปริมาณฝุ่นละอองตามความต้องการของผู้ใช้งานในลักษณะรายงานเชิงกราฟ (Graph-based Report) บนเว็บ แสดงดังภาพที่ 4

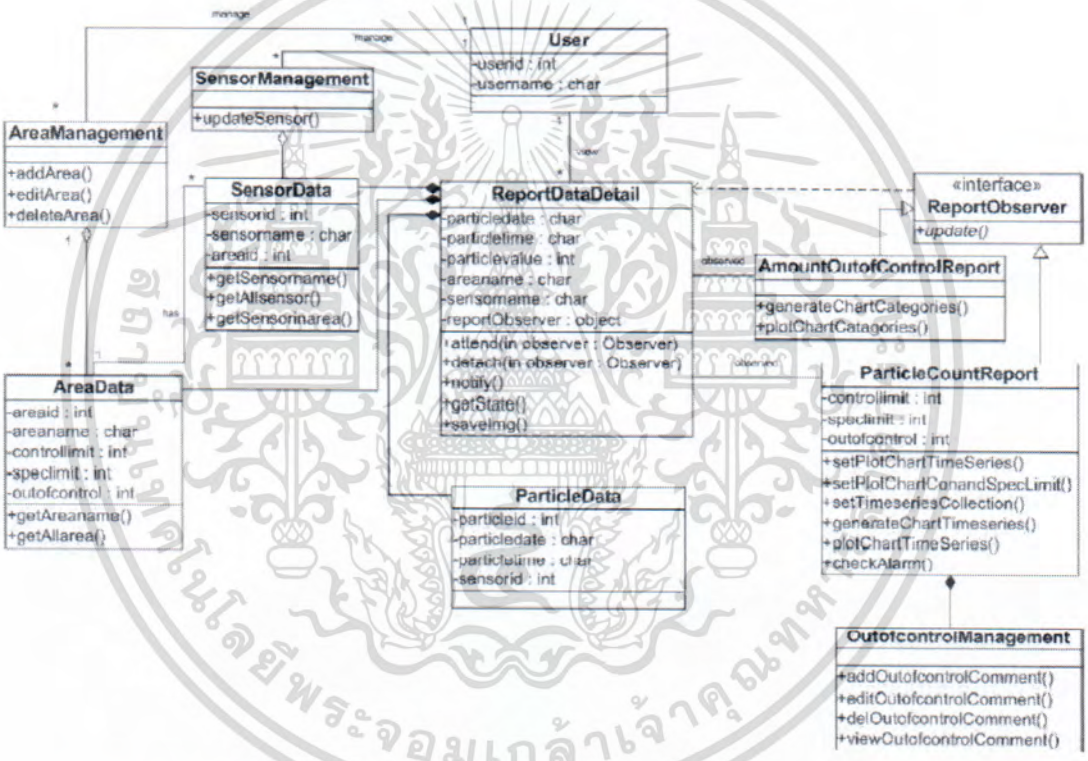


ภาพที่ 4 สถาปัตยกรรมส่วนการออกรายงาน

จากภาพที่ 4 แสดงการทำงานภาพรวมของระบบย่อยในส่วนของการออกรายงานเชิงกราฟ โดยเริ่มจากผู้ใช้งาน (user) จะส่งคำร้องขอรายงานของข้อมูลปริมาณฝุ่นละอองผ่านทางเว็บบราวเซอร์ (Web browser) โดยผ่านเว็บแอปพลิเคชั่น (Web Application) ที่พัฒนาขึ้น โดยใช้เทคโนโลยี J2EE ได้แก่ ภาษาเจเอสพี (Java Server Page: JSP) และเซิร์ฟเลท (Servlet) จากนั้นเซิร์ฟเวอร์จะนำข้อมูลปริมาณฝุ่นละอองที่ต้องการจากระบบฐานข้อมูลที่จัดเก็บอยู่ในระบบฐานข้อมูล DB2 มาแสดงรายงานชนิดกราฟซึ่งจะสร้างกราฟจากการเรียกใช้ไลบรารี สำหรับการสร้างราย ได้แก่ JFreeChart

2.3 การวิเคราะห์และออกแบบกระบวนการออกแบบรายงานบนเว็บ

การทำงานของระบบสนับสนุนฐานข้อมูลสำหรับการออกรายงานปริมาณข้อมูลฝุ่นละอองบนเว็บจะเริ่มจากการเชื่อมต่อกับระบบฐานข้อมูล (Database) ซึ่งมีการออกแบบการจัดเก็บข้อมูลปริมาณฝุ่นละอองตามวันและเวลาของแต่ละพื้นที่ที่ติดตั้งอุปกรณ์เซ็นเซอร์ (sensor) เพื่อนำข้อมูลปริมาณฝุ่นละอองจากฐานข้อมูลมาใช้สำหรับออกรายงานบนเว็บ สำหรับการทำงานในส่วนของการออกรายงานจะนำอ็อปเซิร์ฟเวอร์แพทเทิร์น (Observer Pattern) มาใช้สำหรับตรวจสอบการทำงานตามเหตุการณ์ที่จะเกิดขึ้น โดยจะทำงานตามเหตุการณ์ที่ผู้ใช้งานเลือกรายงานตามความต้องการของผู้ใช้ ซึ่งจะมีตัวตรวจสอบ (Observer) เพื่อตรวจสอบประเภทของรายงานชื่อ ReportType โดยจะมีการแจ้งการตรวจสอบประเภทรายงานอยู่ตลอดเวลา โดยแนวคิดของการออกแบบ แสดงคลาสไดอะแกรม (Class Diagram) ของการออกรายงานดังภาพที่ 5



ภาพที่ 5 แสดงคลาสไดอะแกรม (Class Diagram) การออกรายงานสำหรับงานตรวจจับฝุ่นละอองส่วนของการออกรายงาน

จากภาพที่ 5 แสดง คลาสไดอะแกรม และความสัมพันธ์ระหว่างคลาสของระบบสนับสนุนฐานข้อมูลสำหรับการออกรายงานปริมาณข้อมูลฝุ่นละอองบนเว็บ โดยส่วนประกอบของคลาสในส่วนของ

การออกรายงานเริ่มจาก คลาส ReportObserver จะมี Object ที่ทำหน้าที่สังเกตการเปลี่ยนแปลงความต้องการรายงานจากผู้ใช้งานจาก คลาส ReportDataDetail โดยเมื่อใดที่วัตถุ (Object) เรียกใช้ Method Notify() ของ ReportDataDetail การทำงานของเมธอด update จะถูกเรียกใช้งานทันที โดยเมธอด update จะทำหน้าที่เก็บสถานะของ ReportDataDetail ไว้ใน Attribute ที่เก็บสถานะของ observer โดยแสดงการทำงานของแต่ละคลาสเกี่ยวกับการสร้างรายงาน ดังนี้

1) คลาส ReportDataDetail เป็นคลาสแม่ (Super Class) สำหรับ Object ที่มีสถานะเปลี่ยนแปลงได้ ReportDataDetail จะประกาศเมธอดต่างๆ ที่จำเป็นในการกำหนดว่าจะให้ Object ใดทำหน้าที่สังเกตการเปลี่ยนแปลงของสถานะตนเอง โดยใช้เมธอด attach เพื่อเพิ่ม และใช้เมธอด detach เพื่อลด Observer ลง โดยจะมีการแจ้งให้กับ ReportObserver ทราบเมื่อเกิดการเปลี่ยนแปลงในสถานะของตนเอง

2) คลาส ReportObserver เป็นคลาสอินเทอร์เฟส (interface class) ที่ประกาศเมธอดในการทำให้เกิดการตรวจสอบสถานะของ Object โดยเมธอด update จะถูกเรียกใช้งานหากมีการเปลี่ยนแปลงสถานะเกิดขึ้น

3) คลาส ParticleCountReport เป็น implement class ของ ReportObserver ทำหน้าที่ตรวจสอบและสร้างรายงาน ปริมาณฝุ่นละอองของแต่ละเซ็นเซอร์ (Particle Count Report)

4) คลาส AmountOutofControlReport เป็น implement class ของ ReportObserver ทำหน้าที่ตรวจสอบและสร้างรายงาน จำนวนครั้งของการปริมาณฝุ่นที่เกินมาตรฐาน (out of control) ในพื้นที่การผลิตต่างๆ (Amount out of control Report)

5) คลาส OutofcontrolManagement เป็นคลาสที่ทำหน้าที่จัดการกับข้อมูลต่างๆ ในกรณีที่เกิด out of control (ปริมาณฝุ่นละออง ณ จุดนั้นเกินมาตรฐานที่กำหนดไว้)

นอกจากนี้สามารถอธิบายคลาสที่เกี่ยวข้องกับการทำงานของระบบทั้งหมดได้ดังนี้

6) คลาส User ทำหน้าที่เก็บข้อมูลทั่วไปของผู้ที่มีสิทธิ์ใช้งานระบบ

7) คลาส AreaManagement ทำหน้าที่จัดการพื้นที่การผลิต (Process Area) หรือ พื้นที่ต่างๆ ของสายการผลิต โดยสามารถทำการเพิ่มพื้นที่การผลิต ด้วยเมธอด addArea() ทำการแก้ไขด้วยเมธอด editArea() และทำการลบด้วยเมธอด deleteArea()

8) คลาส AreaData เป็นคลาสแสดงรายละเอียดของพื้นที่การผลิตในสายการผลิต ซึ่งประกอบด้วย หมายเลขพื้นที่ (areaid) ชื่อพื้นที่ (areaname) ค่าปริมาณฝุ่นมาตรฐาน (controllimit) ค่าปริมาณฝุ่นตามที่กำหนด (speclimit) ค่าจำนวนครั้งที่จะเตือนเมื่อปริมาณฝุ่นเกินที่กำหนด (outofcontrol)

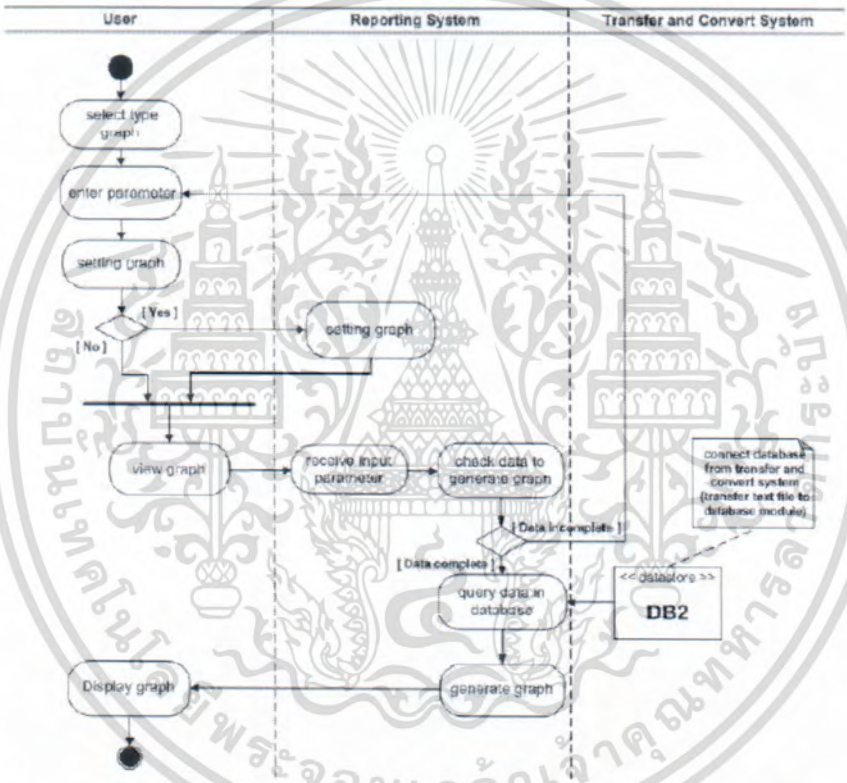
9) คลาส SensorManagement ทำหน้าที่จัดการเซ็นเซอร์ ที่อยู่ในพื้นที่การผลิต โดยผู้ที่มีสิทธิ์การเข้าถึงข้อมูลของพื้นที่การผลิตจะสามารถทำการแก้ไขหรือเปลี่ยนแปลงพื้นที่การผลิตในทุกๆ เซ็นเซอร์ได้ผ่านเมธอด updateSensor()

10) คลาส SensorData เป็นคลาสแสดงรายละเอียดของเซ็นเซอร์ทั้งหมด ประกอบด้วย รหัสเซ็นเซอร์ (sensorid) ชื่อเซ็นเซอร์ (sensorname) area ที่เซ็นเซอร์นั้นทำงานอยู่ (areaid) ดังนั้นคลาส SensorData นี้จะเกี่ยวข้องกับ คลาส AreaData

11) คลาส ParticleData เป็นคลาสจัดเก็บข้อมูลของปริมาณฝุ่นละอองตามวัน (particledate) และเวลา (particletime) ของแต่ละเซ็นเซอร์ (sensorid) ซึ่งจะมีการจัดเก็บข้อมูลปริมาณฝุ่นจากเท็กซ์ไฟล์ (CSV File) ที่มาจากการเก็บข้อมูลภายในสายการผลิต

2.4 กระบวนการออกรายงานระบบใหม่

ลำดับขั้นตอนการสร้างและแสดงผลรายงานปริมาณฝุ่นละอองที่ได้จากการจัดเก็บข้อมูลจากสายการผลิต (Production Line) ในระบบฐานข้อมูล จะแสดงในลักษณะของรายงานเชิงกราฟโดยใช้สถาปัตยกรรมไคลเอนต์/เซิร์ฟเวอร์ และแสดงข้อมูลรายงานจากฐานข้อมูลโดยติดต่อกับระบบย่อยในส่วนของการแปลงไฟล์ แสดงดังภาพที่ 6



ภาพที่ 6 กระบวนการสร้างและแสดงผลรายงานปริมาณฝุ่นละอองจากข้อมูลที่เก็บในระบบฐานข้อมูล

จากภาพที่ 6 แสดงกระบวนการสร้างและแสดงผลรายงานปริมาณฝุ่นละอองที่ผู้ใช้งานต้องการในรูปแบบของรายงานเชิงกราฟ (Graph based report) จากข้อมูลปริมาณฝุ่นละอองที่จัดเก็บอยู่ในฐานข้อมูล DB2 การทำงานจะเริ่มจาก ผู้ใช้งาน (user) จะส่งคำร้อง ประเภทรายงานเชิงกราฟที่ต้องการผ่านทางเครื่องไคลเอนต์ โดยระบุพารามิเตอร์ที่เกี่ยวข้อง เช่น ช่วงเวลาที่ต้องการดูรายงานปริมาณฝุ่นละออง หรือ พื้นที่การ

ผลิต ที่ต้องการเปรียบเทียบปริมาณฝุ่นละออง เป็นต้น โดยเครื่องเซิร์ฟเวอร์ จะรับข้อมูลและทำการประมวลผลข้อมูลที่ต้องการจากระบบฐานข้อมูล จากนั้นจะนำข้อมูลที่ได้จากระบบฐานข้อมูลมาสร้างกราฟและแสดงผลรายงานเชิงกราฟผ่านทางจอภาพให้กับผู้ใช้งานผ่านเครื่องไคลเอนต์ (Client) ซึ่งข้อมูลปริมาณฝุ่นละอองจากฐานข้อมูลดังกล่าวจะมีการเชื่อมต่อกับระบบย่อยในส่วนของการแปลงไฟล์ (Transfer and Convert System)

3. เครื่องมือและเทคโนโลยีที่เกี่ยวข้อง

เทคโนโลยีที่ใช้สำหรับพัฒนาระบบต้นแบบสำหรับระบบฐานข้อมูลเพื่องานตรวจจับฝุ่นละออง ส่วนของรายงานปริมาณฝุ่นภายในห้องคลีนรูมประกอบด้วย

1) เทคโนโลยี J2EE ได้แก่ ภาษา JSP (Java Server Page) และ Servlet

2) Java Library JFreeChart สำหรับสร้างรายงานตามความต้องการของผู้ใช้

3) เชื่อมต่อกับฐานข้อมูล DB2 Universal Database

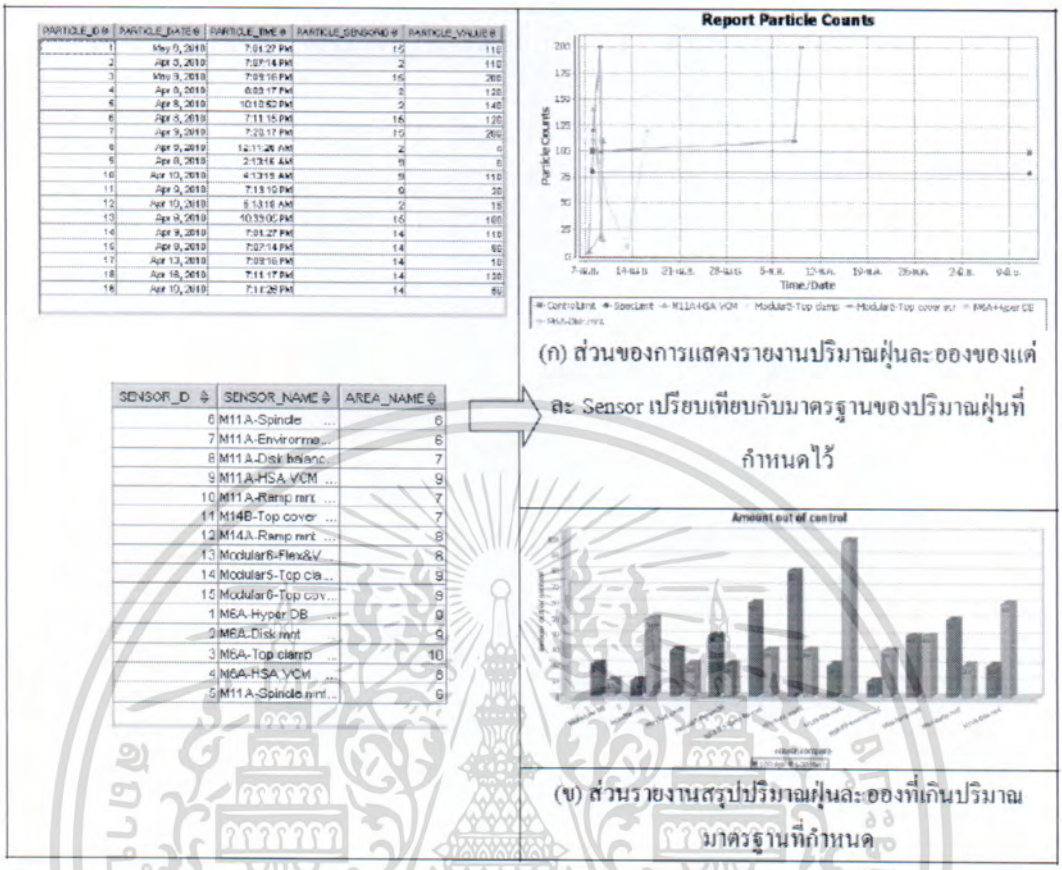
4) Development Tools ได้แก่ Netbeans 6.7

5) Web Server ได้แก่ Apache Tomcat6.0 บนระบบปฏิบัติการ Window Server

ผลการศึกษา

การทำงานของส่วนการออกรายงานของระบบฐานข้อมูลเพื่องานตรวจจับฝุ่นละอองจะนำข้อมูลที่จัดเก็บในระบบฐานข้อมูล มาสร้างรายงาน โดยรายงานประกอบด้วย รายงานปริมาณฝุ่นละอองในแต่ละวันหรือช่วงเวลา โดยเปรียบเทียบปริมาณฝุ่นละอองในแต่ละพื้นที่ที่ติดตั้งอุปกรณ์เซ็นเซอร์ (sensor) ตามระดับการควบคุมปริมาณฝุ่นมาตรฐานของห้องคลีนรูมแสดงดังภาพที่ 7 (ก) และรายงานสรุปจำนวนฝุ่นละอองในแต่ละช่วงเวลาหรือแต่ละเดือนเพื่อเปรียบเทียบปริมาณฝุ่นที่เกินมาตรฐานที่กำหนดในแต่ละ พื้นที่ แสดงดังภาพที่ 7 (ข) ดังนี้

15



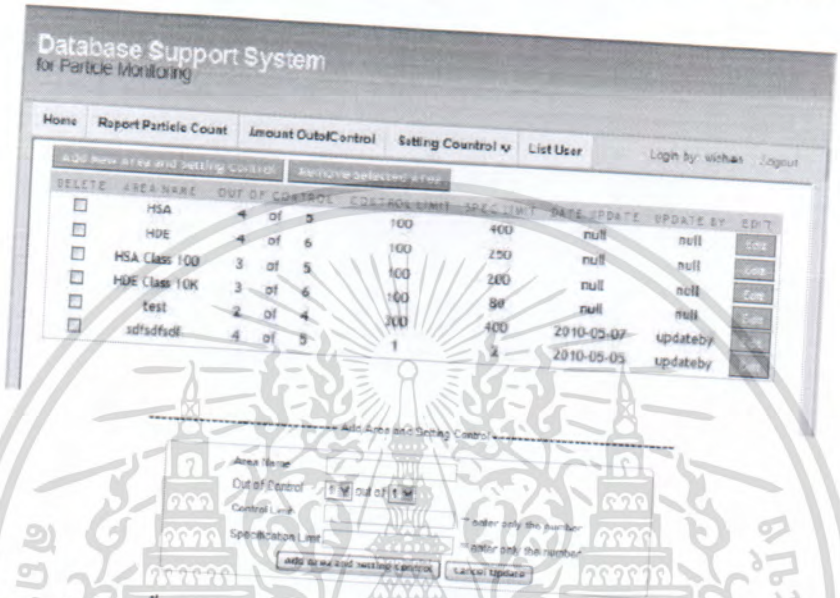
ภาพที่ 7 แสดงรายงานปริมาณฝุ่นละอองภายในห้องคลีนรูม (Clean room)

จากภาพที่ 7 (ก) แสดงรายงานการเปรียบเทียบปริมาณฝุ่นละอองของแต่ละเซ็นเซอร์ เปรียบเทียบกับปริมาณมาตรฐานในแต่ละคลาสหรือ Control Limit และค่าที่โรงงานกำหนด หรือ Specification Limit โดยการทำงานจะนำข้อมูลที่จัดเก็บไว้ในระบบฐานข้อมูล ทำการสืบค้นตามเวลาและเซ็นเซอร์ ที่ต้องการเปรียบเทียบ แสดงในลักษณะของกราฟเส้น โดยกราฟที่ได้จะประกอบด้วย กราฟแสดงค่า Control Limit กราฟแสดงค่า Specification Limit และกราฟแสดงค่าปริมาณฝุ่นละอองของเซ็นเซอร์แต่ละตัวที่ผู้ใช้งานเปรียบเทียบค่าปริมาณฝุ่นละออง

จากภาพที่ 7 (ข) แสดงรายงานสรุปปริมาณฝุ่นละอองที่เกินปริมาณมาตรฐานที่กำหนด หรือ เกินค่า Control Limit โดยการเปรียบเทียบข้อมูลปริมาณฝุ่นละอองของเดือน มีนาคม (March) และเดือนเมษายน (April) ในแต่ละเซ็นเซอร์

16

นอกจากนี้ การตั้งค่า Control Limit และ Specification Limit ดังที่กล่าวไว้ในหัวข้อที่ 2.1 มีความสำคัญต่อการเปรียบเทียบปริมาณฝุ่นละอองเป็นอย่างมาก ดังนั้นระบบจึงสามารถให้พนักงานในแผนกที่เกี่ยวข้อง ได้แก่ วิศวกรที่มอนิเตอร์ข้อมูลปริมาณฝุ่นละออง สามารถ ตั้งค่า/เปลี่ยนแปลง/แก้ไข ค่า Control Limit และ Specification Limit ในแต่ละพื้นที่ได้ โดยแสดงหน้าจอการตั้งค่าในแต่ละพื้นที่ ดังภาพที่ 8

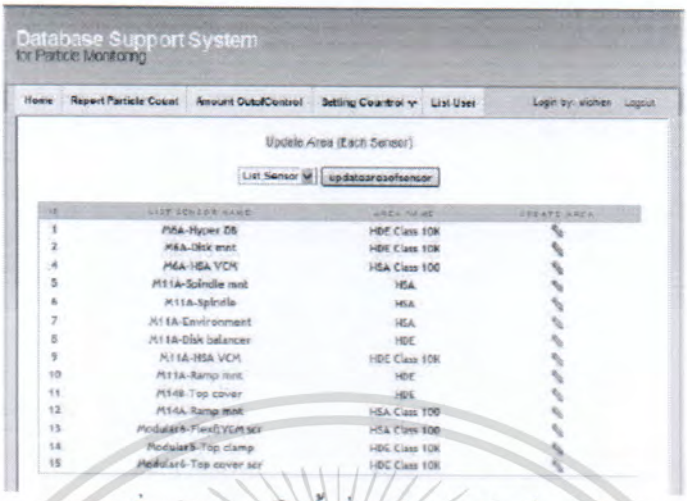


ภาพที่ 8 แสดงส่วนการตั้งค่า (Setup) ค่ามาตรฐานของปริมาณฝุ่นละอองภายในห้องสะอาดแต่ละห้อง

จากภาพที่ 8 แสดงส่วนการตั้งค่าในแต่ละพื้นที่โดยในแต่ละพื้นที่การผลิตจะต้องประกอบด้วย ชื่อพื้นที่ (ส่วนใหญ่นั้นชื่อ พื้นที่ จะแสดงถึงกระบวนการทำงานที่พื้นที่นั้นทำหน้าที่หลัก เช่น พื้นที่ HDE หมายถึง พื้นที่ที่ใช้ประกอบชิ้นส่วนในกระบวนการผลิต HDE เป็นต้น) ค่า Control Limit ค่า Specification Limit และค่า out of control เป็นต้น

ขณะเดียวกัน การทำงานของอุปกรณ์เซ็นเซอร์ที่ทำหน้าที่ตรวจจับปริมาณฝุ่นละออง จะต้องทำงานอยู่ในพื้นที่การผลิตใด พื้นที่หนึ่ง ดังนั้นจึงต้องมีการกำหนดพื้นที่ให้กับอุปกรณ์เซ็นเซอร์แต่ละตัว ว่าจะอยู่ในพื้นที่ใด แสดงการตั้งค่าพื้นที่ให้กับอุปกรณ์เซ็นเซอร์ดังภาพที่ 9

17



ภาพที่ 9 แสดงการแก้ไขพื้นที่การผลิตของแต่ละเซ็นเซอร์

จากภาพที่ 9 แสดงส่วนของการแก้ไขเซ็นเซอร์แต่ละเซ็นเซอร์ ซึ่งจะดึงข้อมูลจากฐานข้อมูลที่ติดต่อกับอุปกรณ์เซ็นเซอร์ในสายการผลิตมา จากนั้นจะมีการกำหนดพื้นที่ (Area) ให้กับอุปกรณ์เซ็นเซอร์แต่ละเซ็นเซอร์ ว่าจะอยู่ในพื้นที่การผลิตใด เช่น เซ็นเซอร์ในสเตชั่นของประกอบ Top cover ในสายการผลิต (Line) ที่ 14B ทำงานอยู่ในพื้นที่ HDE เป็นต้น ซึ่งการจัดการการตั้งค่าของพื้นที่การผลิตและเซ็นเซอร์นั้นจะกำหนดผู้ที่สามารถบันทึกค่าดังกล่าวได้ โดยผู้ที่มีสิทธ์เข้าใช้งานจะเป็นผู้ใช้งานในแผนกของวิศวกรที่มอนิเตอร์ข้อมูลปริมาณฝุ่นละอองเท่านั้น

อภิปรายและวิจารณ์

ระบบสนับสนุนฐานข้อมูลสำหรับการออกรายงานปริมาณข้อมูลฝุ่นละอองบนเว็บ เป็นส่วนหนึ่งของการพัฒนาระบบฐานข้อมูลเพื่องานตรวจจับฝุ่นละออง (Database Support System for Particle Monitoring) เกิดขึ้นจากแนวทางการต่อยอดจากระบบงานเดิมที่มีการใช้จริงในโรงงานอุตสาหกรรมด้านการประกอบชิ้นส่วนฮาร์ดดิสก์ (Hard Disk Drive Manufacturing) โดยได้มีการศึกษาการทำงานและข้อมูลปริมาณฝุ่นละอองที่ได้จากสายการผลิต (Production Line) ภายในห้องคลีนรูม ซึ่งเป็นห้องที่มีการควบคุมสภาพแวดล้อม เพื่อนำมาออกแบบและสร้างต้นแบบของการออกรายงานบนเว็บแทนการออกรายงานแบบเดิมที่ทำโดยมนุษย์ (Manually Report) ทำให้วิศวกรที่ทำหน้าที่หลักสำหรับมอนิเตอร์ข้อมูลปริมาณฝุ่นละอองและจัดสร้างรายงานสำหรับผู้บริหารลดเวลาและขั้นตอนสำหรับการสร้างรายงานดังกล่าว อีกทั้งยังสามารถนำข้อมูลปริมาณฝุ่นละอองที่ได้จากการจัดเก็บในระบบฐานข้อมูลมาวิเคราะห์และต่อยอดเพื่อหาความสัมพันธ์หรือความรู้ที่อยู่ในฐานข้อมูลขนาดใหญ่ ซึ่งอาจเป็นประโยชน์ต่ออุตสาหกรรมการผลิตดังกล่าว

สำหรับการทำงานในอนาคตจะมีการนำต้นแบบของการพัฒนาระบบฐานข้อมูลเพื่องานตรวจจับฝุ่นละออง (Database Support System) ไปใช้ในกระบวนการทำงานจริงต่อไป

บทสรุป

บทความนี้ได้นำเสนอ ระบบสนับสนุนฐานข้อมูลสำหรับการออกรายงานปริมาณข้อมูลฝุ่นละอองบนเว็บ ซึ่งเป็นส่วนหนึ่งของระบบฐานข้อมูลเพื่องานตรวจจับฝุ่นละออง (Database Support System for Particle Monitoring) เป็นการแก้ไขปัญหาการออกรายงานโดยมนุษย์ (Manually Report) เป็นการออกรายงานตามความต้องการของผู้ใช้งานผ่านเว็บ โดยพัฒนาและต่อยอดจากระบบงาน PMS ที่มีการออกรายงานแบบ Manual Report ซึ่งมีข้อจำกัดเกี่ยวกับระยะเวลาสำหรับการจัดเรียงข้อมูลเพื่อสร้างกราฟผ่านโปรแกรม Microsoft Excel โดยการพัฒนาระบบต้นแบบดังกล่าวมีการนำเทคโนโลยีทางด้านต่างๆ เข้ามามีส่วนช่วยในการแก้ปัญหาและลดข้อจำกัดการทำงานของระบบ PMS เดิมให้สามารถทำงานได้อย่างมีประสิทธิภาพมากยิ่งขึ้น เช่น กระบวนการวิเคราะห์และออกแบบระบบเชิงวัตถุ (Object Oriented Analysis and Design: OOAD) และสถาปัตยกรรมไคลเอนต์/เซิร์ฟเวอร์

คำขอบคุณ

โครงการพัฒนาระบบงานนี้เป็นส่วนหนึ่งของโครงการพัฒนาทรัพยากรมนุษย์ในอุตสาหกรรมฮาร์ดดิสก์ได้รับทุนจาก I/UCRC (Industry/University Cooperative Research Center) ของเนคเทค (National Electronics and Computer Technology Center: NECTEC) ภายใต้การสนับสนุนของวิทยาลัยร่วมด้านเทคโนโลยีการบันทึกข้อมูลและการประยุกต์ใช้งาน (Data Storage Technology and Applications: DSTAR) สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง (King Mongkut's Institute of Technology Ladkrabang: KMITL) และร่วมกับบริษัทฮิตาชิ โกลบอล สตอเรจ เทคโนโลยีส์ (ประเทศไทย) จำกัด (Hitachi Global Storage Technology: HGST)

เอกสารอ้างอิง

1. Koyamada, K. 1992. Visualization of simulated airflow in a clean room. IEEE Computer Society Technical Committee on Computer Graphics. 156-163.

2. Mrad, F. 1999. The Characterization of a Clean Room Assembly Process. IEEE Transactions on Industry Applications. 35, 399-404.

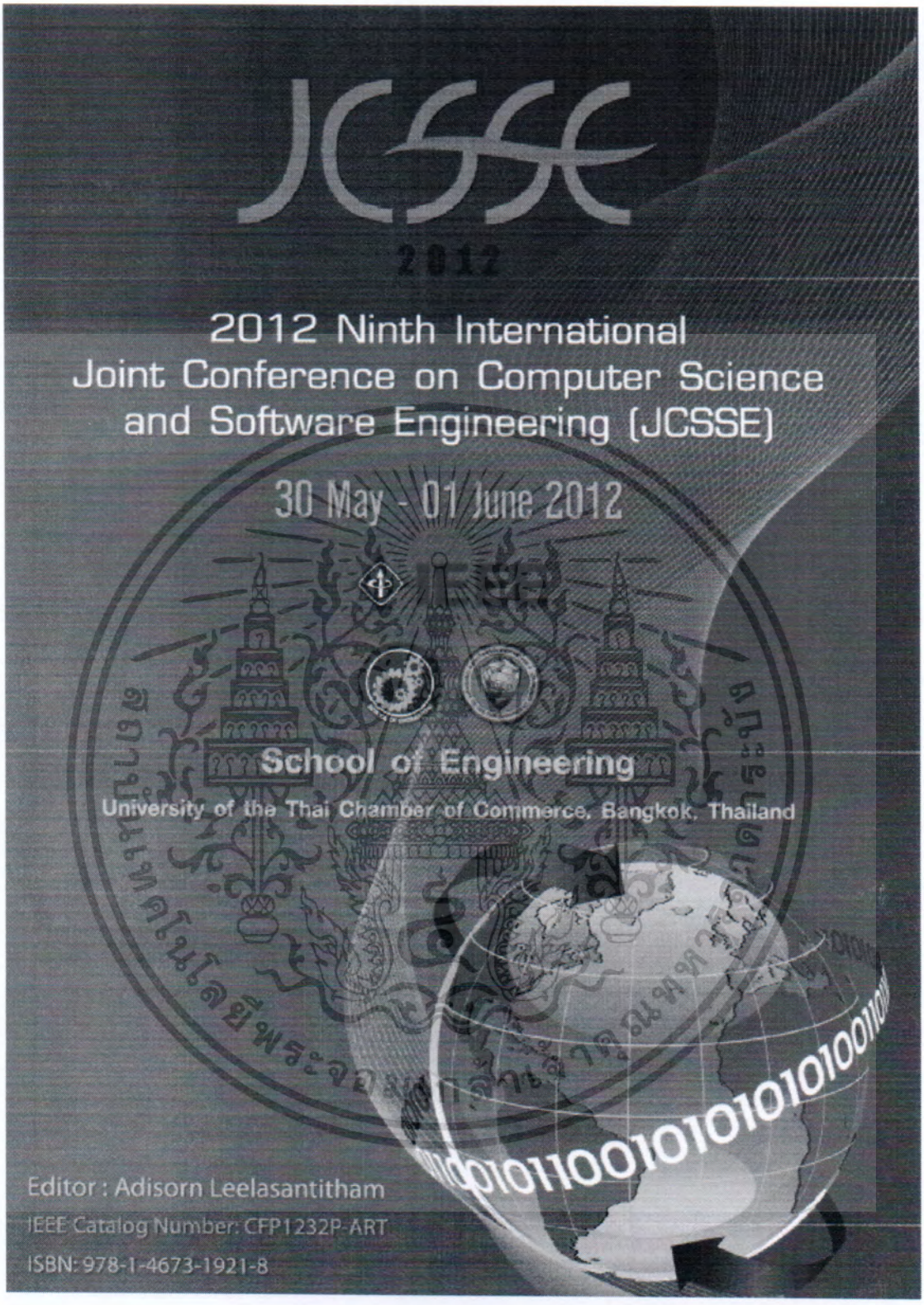
3. Query, C.F. 1999. Continuous Monitoring in Disk Drive Manufacturing. IEEE Transactions on Magnetics. 35, 945-949.

4. Garcia-Arellano, C.M., Lightstone, S.S., Lohman, G.M., Markl, V., Storm, A.J. 2006. Autonomic Features of the IBM DB2 Universal Database for Linux, UNIX and Windows. IEEE Transactions on Systems, Man, and Cybernetics. 36, 365-376.

5. Welker, R.W. 2002. Guideline for Design and Certification of Cleanroom Tooling. Clean room Tooling Document, Revised by Don DeLeo/Tim Karlberg, San Jose Site Contamination Control, August 8.

6. Manufacturing Environment Engineering Department. 2009. ESD and Contamination Control.Hitachi Global Storage Technologies. April.

7. Training & Support Department. 2010. Hard Disk Drive Process Overview. Training Document. Hitachi Global Storage Technologies. April.



Improving Software Maintenance Size Metrics

A Case Study: Automated Report Generation System for Particle Monitoring in Hard Disk Drive Industry

Aphatsara Wirotyakun[1] and Ponrudee Netisopakul[2]
Information Technology Department
King Mongkut's Institute of Technology Ladkrabang
Bangkok, Thailand
[1]s1066447@kmitl.ac.th, [2]ponrudee@it.kmitl.ac.th

Abstract— **A software maintenance size can be used to predict maintenance effort or maintenance time. However, the traditional maintenance size metric only relies on line of source code (LOC), which hardly is suitable for object-oriented software. This research proposed four new maintenance size metrics based on number of classes, number of methods, average number of methods per class, and weighted methods per class. An automated report generation system in HDD industry (ARGS_PMS) was used as a case study to measure the performance of each proposed metric. We found that, for enhanced software maintenance tasks, maintenance size based on an average number of methods per class (MS-MC) gave the best performance in predicting maintenance time. For a refactoring maintenance task, maintenance size based on line of code (MS-LOC), weighted methods per class (MS-WMC) and average methods per class (MS-MC) were the best, better and good estimators for predicting maintenance time, respectively. On average, the maintenance size based on weighted methods per class gave the best performance.**

Keywords-Maintenance Size; Estimated Time to Maintenance; Object-Oriented Software Maintenance Metric

I. INTRODUCTION

In the Hard Disk Drive industry (HDD), the workers and the machines are operating under a controlled environment called "clean rooms", installing with a particle monitoring system (PMS) [1]. The system automatically filters and transfers raw data, such as a number of particles per cubic meter [2][3], from the production line to a database. Then, reports are automatically generated from the system. However, report requirements are often changed. Therefore, the maintainability of the automated report generation system (ARGS_PMS) is important; because the system must be able to support the need for various requirements in the future.

Software maintenance size metric is one of the good indicators for measuring maintenance effort [4][5]. Currently, software maintenance size is calculated based on only line of code, which changed during maintenance process. However, the software is implemented using an object-oriented language; hence, classes and methods should be considered as well. Therefore, this research proposed new software maintenance size metrics based on number of classes and methods. Performance on predicting maintenance time for each metrics was compared using three maintenance versions of the automated report generation system.

The structure of this paper is as followed. Section II reviewed the previous researches on software maintenance size metrics and correlation between maintenance size and maintenance effort. Section III explained our case study. Section IV explained the existing and improved software maintenance size metrics. Section V analyzes and evaluates the results followed by a conclusion in Section VI.

II. BACKGROUND AND RESEARCH METHOD

One of the requirements for an automated report generation system for particle monitoring in HDD industry (ARGS_PMS) is the ability to create new types of reports and adjust existing reports to users' needs. Hence, the system is expected to be flexible and easy to maintain. The modification of a software product after delivery includes both enhancing source code based on additional requirements and refactoring source code to maintain readability. These maintenance tasks are associated with maintenance time and costs [6], which need to be known, planned and allocated resources in advance.

Maintenance size can be used to predict maintenance effort, i.e., maintenance time. However, the current practice for measuring maintenance size is only based on number of line of code, modified and added during maintenance process. This metric is not suitable for predicting maintenance time in advance. In order to estimate maintenance time using design class diagram, new software maintenance size metrics based on number of classes and methods changed are introduced in this paper.

Some previous researches support that maintenance size and maintenance time are correlated. For example, Ye Yang., et al. [4] showed correlation between the maintenance size metrics and an activity of maintenance effort in his empirical study of the software quality management platform (QMP). They found that software maintenance size is correlated to maintenance effort, such as analysis activity and design activity, and most highly correlated to coding effort. However, they calculated maintenance size based on lines of changed and

The research is supported by granted # HDD-01-52-02M.

978-1-4673-1921-8/12/$31.00 ©2012 IEEE

added code of each maintenance version, i.e., classes or methods changed were not considered.

While some previous researches also propose models to predict maintenance effort, generally used lines of code (LOC), which count the line of added, deleted and changed of code. Several empirical researches [5][8] showed correlation between software size and maintenance effort. Jorgensen [8] predicted maintenance effort using maintenance task size, also based on line of code.

However, we can typically [7] classifies types of software metrics into two categories: traditional and object-oriented software metrics. As the line of code measurement is traditional software metrics, the class and method size and structure are object-oriented software metrics.

Wei Li and Sallie Henry [8] investigated object-oriented software metrics and validated the use of metrics to estimate line of code changed per class in maintenance history. The metrics include Depth of Inheritance Tree (DIT), Number of Children (NOC), Message Passing Coupling (MPC), Response for a Class (RFC), Lack of Cohesion in Methods (LCOM), Data Abstraction Coupling (DAC), Weighted Methods per Class (WMC), Number of Methods (NOM), number of line of code (SIZE1), and number of attributes and methods in the classes (SIZE2).

Although the research found that the above metrics have correlation to line of code changed per class in maintenance history, it did not try to use these metrics to estimate maintenance time.

In our research, we proposed the new maintenance size metrics based on number of classes and methods need to be modified. Unlike line of code, classes and methods changes were known during analysis and design, i.e., before implementation. Using a case study of automated report generation system, the actual maintenance time were recorded and compared to the predicted maintenance time calculated using these new maintenance size metrics.

III. Description of A Case Study

This section described process in our case study including the procedure, the case study materials and description of our proposed maintenance size metrics.

A. *Procedure*

This paper compared the effects of using different software maintenance size metrics for predicting maintenance time. In this case study, there were two types of software improvement: 1) functional enhancement tasks: software maintenance accommodating new or changed requirements to the existing automated report generation system and 2) refactoring tasks: software maintenance accommodating refactoring source code to improve modularity and readability in the existing system, without changing its functionality.

The procedure was as followed:

- Deciding which metrics were needed to be recorded or collected during the maintenance tasks.
- Designing the forms to be used to record data during maintenance tasks.
- Modifying classes and methods to accommodating new or changed requirements OR modifying classes and methods to improve software design (refactoring). The result was the modified design class diagram that can be used to calculate values of our proposed maintenance metrics.
- Implementing the maintenance version of the software while collecting data from a case study. For example, number of classes, number of methods and number of line of code added or modified during maintenance process.
- Calculating the software maintenance size metrics based on the previous and newly designed software size metrics. Hence, the totals were five software size metrics.
- Estimating the maintenance time based on each of maintenance size metrics.
- Comparing the predicted maintenance time with actual maintenance time for each metrics.

B. *Case Study Materials*

A case study of an automated report generation system for particle monitoring in Hard Disk Drive industry (ARGS_PMS) was used to test the effects of predicting maintenance time using maintenance size measurement. There were four version of software; the initial version, maintenance version 1, 2 and 3. Software maintenance version 1 and 2 are of type *functional enhancement* maintenance task. Software maintenance version 3 is of type *refactoring* maintenance task.

Functional enhancement maintenance tasks include adding more functions into the existing software. For example, adding one or two new reports into the ARGS_PMS.

Refactoring maintenance tasks include changing class structure or methods in order to improve readability, cohesion and coupling of classes.

Description of initial and each maintenance version of software are shown in Table I.

The five maintenance size metrics of the software maintenance version 1-3 will then be calculated based on these concepts:

- Software maintenance size metric based on line of code (MS-LOC).
- Software maintenance size metric based on number of classes (MS-TC).
- Software maintenance size metric based on number of methods (MS-TM).
- Software maintenance size metric based on average number of methods per class (MS-MC).

- Software maintenance size metric based on weighted methods per class (MS-WMC).

TABLE I. SOFTWARE DESCRIPTION FOR INITIAL AND MAINTENANCE VERSIONS

Table Head	Before	After
Maintenance Version 1	**Initial version:** An automated report generation system for particle monitoring in HDD industry (ARGS_PMS) consisted of the particle count report, an amount out of control report and summary amount trigger report.	ARGS_PMS is improved by adding two new basic reports: (1) a out-of-control particle data exception report for each sensor in process areas and (2) a historical monthly exception report.
Maintenance Version 2	**Maintenance version 1:** An automated report generation system consisted of the basic and simple reports.	ARGS_PMS is improved by adding intermediate reports to the maintenance versions, i.e., historical monthly data report of weekly particle data.
Maintenance Version 3	**Maintenance version 2:** An automated report generation system consisted of the basic, simple and intermediate reports.	ARGS_PMS is improved by refactoring source code to improve readability, cohesion and coupling of classes.

IV. IMPROVING NEW SOFTWARE MAINTENANCE SIZE METRICS

This section explains the detail of existing and proposed maintenance size metrics based on object oriented classes and methods changed.

A. Study of Existing Maintenance Size Metric

A software maintenance size metric based on line of code (MS-LOC) was a preliminary measure for software maintenance size metrics [4]. Basically, MS-LOC is a ratio of added and changed line of code to the total number of line of code in the initial system. The equation is defined as:

$$MS-LOC = \frac{LOC_{Added} + LOC_{Modified}}{\text{Total Line of Code in the Initial System}} \quad (1)$$

However, the modification in OO system not only affects number of line of code but also number of classes and methods. The number of changed classes and methods in our case study is shown in TABLE II.

B. Proposed Maintenance Size Metrics based on number of Classes and Methods Modified

This research proposed four new maintenance size metrics, based on total number of classes (MS-TC), total number of methods (MS-TM), average number of methods per class (MS-MC) and weighted methods per class (MS-WMC) described as follows:

1) Metric1: Maintenance size metric based on total number of classes (MS-TC)

The formula of a maintenance size metric based on total number of classes (MS-TC) is the ratio of numbers of changed classes to the number of total of classes in the initial system as defined below:

$$MS-TC = \frac{|Class_{Added} - Class_{Deleted}| + Class_{Modified}}{\text{Total Class in the Initial System}} \quad (2)$$

2) Metric2: Maintenance size metric based on total number of methods (MS-TM)

A software maintenance size metric based on total number of methods (MS-TM) is the ratio of numbers of changed methods to the total number of methods in the initial system, which is defined below:

$$MS-TM = \frac{|Method_{Added} - Method_{Deleted}| + Method_{Modified}}{\text{Total Method in the Initial System}} \quad (3)$$

3) Metric3: Maintenance size metric based on methods per class (MS-MC):

A software maintenance size metric based on average number of methods per class (MS-MC) is the ratio of average number of changed methods per class to the average number of methods per class in the initial system. The MS-MC equation is defined as:

$$MS-MC = \frac{(Method_{Changed}) / \text{Number of Class Change}}{\frac{\sum_{i=1}^{n} Method(i)}{\text{Total Class in the Initial System}}} \quad (4)$$

4) Metric4: Maintenance size metric based on Weighted Methods per Class (MS-WMC)

A software maintenance size metric based on weighted methods per class (MS-WMC) is improved over the MS-MC using weighted methods per class instead of average number of methods per class. MS-WMC is the ratio of changed weighted methods per class to weighted methods per class in the initial system. The equation MS-WMC is defined as:

TABLE II. COLLECTED DATA DURING SOFTWARE DEVELOPMENT AND MAINTENANCE

	Number of line of code (LOC)				Number of classes				Number of methods				Maintenance Time
	Total	*Added*	*Modified*	*Deleted*	*Total*	*Added*	*Modified*	*Deleted*	*Total*	*Added*	*Modified*	*Deleted*	
Maintenance version 1													
Initial	1,221	-	-	-	13	-	-	-	76	-	-	-	55 hour
Mtn ver. 1	1,609	388	0	0	15	2	1	0	92	12	4	0	9 hour 21 min.
Maintenance version 2													
Mtn ver. 1	1,609	-	-	-	15	-	-	-	92	-	-	-	64 hour 21 min.
Mtn ver. 2	1,778	140	29	0	16	1	1	0	100	8	0	0	3 hour 56 min.
Maintenance version 3													
Mtn ver. 2	1,778	-	-	-	16	-	-	-	100	-	-	-	68 hour 17 min.
Mtn ver. 3	1,776	54	8	66	18	2	6	0	93	3	6	10	1 hour 10 min.

$$MS-WMC=\frac{\left(WeightedMethod_{Changed}\right)/NumberofClassChange}{\frac{\sum_{i\in IS}Complexity(i)}{TotalClassintheInitialSystem}} \quad (5)$$

Where the changed weighted methods per class (WMC) is summation of all changed methods, in a weighted form, divided by number of changed classes.

A weighted method per class (WMC) is a measure of the complexity of the overall decision structure within the methods making up a class [9]. In our case, it is calculated by summarizing all cyclomatic complexity values of all methods in the system, then divided by the number of classes in the system.

Cyclomatic complexity [10] (C_i) applies to an individual method within a class. It directly measures the number of linearly independence paths through a program's source code called flow graph. The nodes (n) of the graph correspond to indivisible groups of command of program and a directed edge (e) connects two nodes. Given a flow graph of each method with n nodes and e edges, the complexity (for each method) is defined as:

$$C_i=e-n+2 \quad (6)$$

V. A Case Study Results

We collected the empirical data during software development and maintenance. The case study consisted of four software version: an initial version and three maintenance versions of ARGS_PMS. The values of collected data are shown in Table II. Table III depicts the values of five software maintenance size metrics, calculated based on previous and newly designed software maintenance size metrics. Then we estimated maintenance time using each metric and compared the actual and estimated maintenance time.

TABLE III VALUES OF FIVE MAINTENANCE SIZE METRICS AND THEIR PREDICTED MAINTENANCE TIME FROM 3 MAINTENANCE VERSIONS

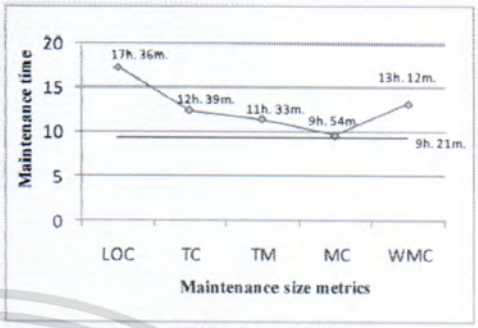
Type	Maintenance Version 1		Maintenance Version 2		Maintenance Version 3	
	MS	*EMT*	*MS*	*EMT*	*MS*	*EMT*
MS-LOC	0.32	17h. 36m.	0.11	7h. 36m.	0.01	41m.
MS-TC	0.23	12h. 39m.	0.13	8h. 21m.	0.50	34h. 54m.
MS-TM	0.21	11h. 33m.	0.09	5h. 46m.	0.13	8h. 52m.
MS-MC	0.18	9h. 54 m.	0.08	5h. 8m.	0.12	8h.11m.
MS-WMC	0.24	13h.12m.	0.09	5h. 46m.	0.03	2h. 30m.

MS is Maintenance Size, EMT is Estimated Maintenance Time. It is hours and min in a minute

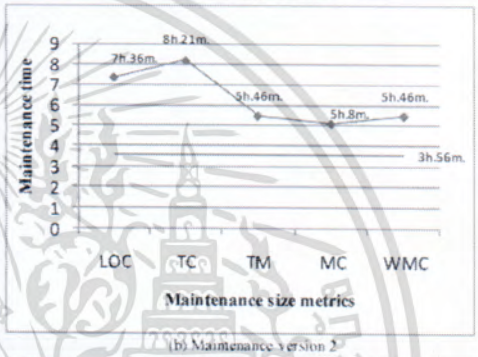
We estimated maintenance time using a simple rule of three proportions. For example, the value 0.32 in MS-LOC indicated that the line of code in maintenance version 1 had grown 32% more than the initial version. Hence, if we need 55 hours to develop 100% of code, then we will need 55 x 0.32 hours to develop an extra 32% more code.

The estimated values of maintenance time for each metrics are shown in table III. Graphs in Figure 1 (a, b and c) compare actual and estimated time for each maintenance version.

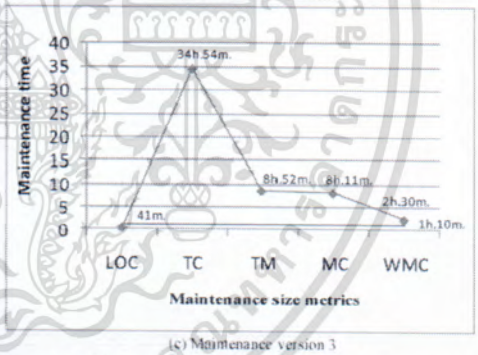
In our study, the actual maintenance time were compared to the predicted maintenance time. The results of comparing the actual and predict maintenance time of each maintenance version appear in Figure1 (a, b and c).



(a) Maintenance version 1



(b) Maintenance version 2



(c) Maintenance version 3

Predict maintenance time — Actual maintenance time

Figure 1 Graphs Comparing Actual and Predicted Maintenance Time for Each Metric

In Figure 1a, the MS-MC metric predicted maintenance time to be 9 hours 54 minutes, closest to the actual maintenance time of 9 hours 21 minutes. The next best metrics were MS-TM and MS-TC, respectively. For maintenance version 2, figure 1b shows that MS-MC also performed the maintenance estimation better than other metrics. The next best metrics were MS-TM and MS-WMC, which tied performance.

For maintenance version 3, figure 1c shows that MS-LOC predicted maintenance time to be 41 minutes while actual time is 1 hours 10 minutes. MS-WMC predicted 2 hours 30 minutes, gave next closest value to the actual maintenance time.

Figure 2 shows an average percentage of error for predicting maintenance time of functional enhancement maintenance tasks, i.e., maintenance version 1 and 2. We found that MS-MC gave an average error of 13.17% comparing to 89.71% error percentage of MS-LOC. In other words, MS-MC gave an improvement of 76.54% over MS-LOC.

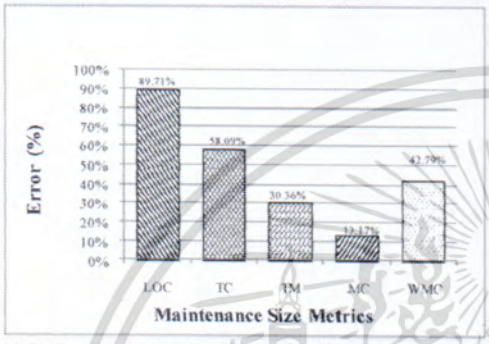


Figure 2. Average Error Percentage for Functional Enhancement Tasks

Figure 3 shows percentage of error for predicting maintenance time of refactoring source code maintenance task, i.e., maintenance version 3. We found that MS-LOC gave an error of 41.43% comparing to 114.29% error percentage of MS-WMC. In other words, MS-LOC gave the best estimate maintenance time in this case.

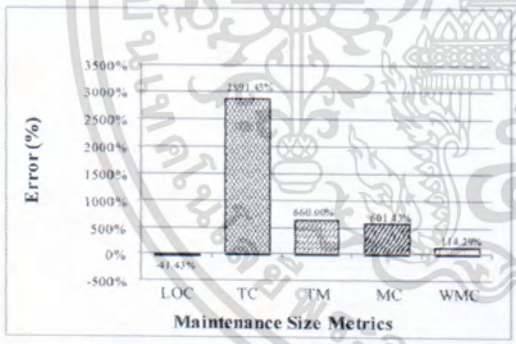


Figure 3. Error Percentage for Refactoring Task

Figure 4 shows an average percentage of error for each metrics for all maintenance versions. We found that MS-WMC gave an average error of 48.56% comparing to 79.12% error percentage of MS-LOC. In other words, MS-WMC gave an improvement of 30.46% over MS-LOC.

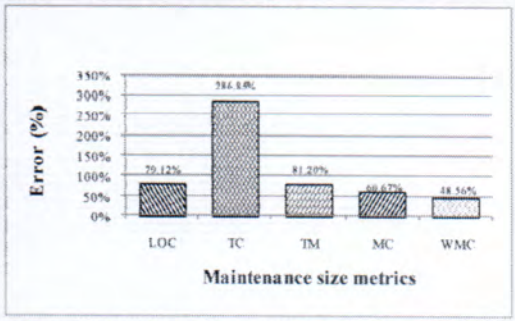


Figure 4. Average Error Percentage for Predicting Maintenance Time

VI. Conclusion And Discussion

The case study demonstrated the effectiveness of using different software maintenance size metrics to predict maintenance time for two types of maintenance tasks.

For functionality enhancement maintenance task, such as adding more reports, the appropriate metrics were software maintenance size based on average number of methods per class (MS-MC), followed by average number of methods (MS-TM) and weighted methods per class (MS-WMC).

For a refactoring maintenance task, minor changes were done to many classes; new classes were added to improve coupling and cohesion. The case study showed that the appropriate metric was maintenance size base on line of code (MS-LOC), followed by weighted methods per class (MS-WMC) and average number of methods per class (MS-MC). While a maintenance size metrics based on number of classes changed, i.e., MS-TC, was not appropriate.

In conclusion, this research proposed four new maintenance size metrics, based on number of classes (MS-TC), number of methods (MS-TM), average methods per class (MS-MC) and weighted methods per class (MS-WMC). The result from a case study demonstrated that, on average, MS-WMC gave the best performance in estimating maintenance time, followed by MS-MC. In other words, MS-WMC gave an average improvement of 30.46% over the traditional maintenance size based on line of source code.

Acknowledgment

This research is partly supported by Industry/University Cooperative Research Center (I/UCRC) at National Electronics and Computer Technology Center (NECTEC). We furthermore acknowledge the support of Data Storage Technology and Applications (DSTAR) at King Mongkut's Institute of Technology Ladkrabang (KMITL). Partners in this project is Hitachi Global Storage Technology (HGST)

REFERENCES

[1] R. W. Welker, "Guideline for Design and Certification of Cleanroom Tooling", Clean room Tooling Document, Revised by Don DeLeo/Tim Karlberg, San Jose Site Contamination Control, August 8 2002.

[2] Manufacturing Environment Engineering Department, "ESD and Contamination Control", Hitachi Global Storage Technologies, April 2009.

[3] Training & Support Department, "Hard Disk Drive Process Overview", Training Document. Hitachi Global Storage Technologies, April 2010.

[4] J. Gaffney et al., "Software Measurement Guidebook", International Thompson Computer Press, Boston, 1995.

[5] M. Jorgensen. "An Empirical Study of Software Maintenance Tasks", Journal of Software Research And Practice, Volume 7, Issue 1, pp.27–48, 1995.

[6] Y. Yang, Q. Li, M. Li and Q. Wang. "An empirical analysis on distribution patterns of software maintenance effort". In 24th IEEE International Conference on Software Maintenance (ICSM 2008), Beijing, China, pp.456-459, September 28 - October 4, 2008.

[7] H. D. Rombach, B. T. Ulery and J. D. Valett. "Toward Full Life Cycle Control: Adding Maintenance Measurement to the SEL". The Journal of Systems and Software, Volume18, Issue 2, pp. 125–138, May 1992.

[8] W. Li and S. Henry, "Maintenance Metrics for the Object Oriented Paradigm", 1st International Software Metrics Symposium. Baltimore, USA, pp. 52-60 , 1993.

[9] S. R. Chidamber and C. F. Kemerer , "A Metrics Suite of Object Oriented Design". IEEE Transactions on Software Engineering, Volume 20 (6), pp 476-493, 1994.

[10] T. J. McCABE, "A Complexity Measure" IEEE Transactions on Software Engineering, Volum 2(4) pp 308-320, 1976.

ประวัติผู้เขียน

นางสาว อภัสรา วิโรจน์ยะกูล เกิดเมื่อวันที่ 9 กันยายน พ.ศ.2528 จบการศึกษา วิทยาศาสตรบัณฑิต สาขาวิทยาการคอมพิวเตอร์ คณะวิทยาศาตร์และเทคโนโลยี มหาวิทยาลัยราชภัฏสวนสุนันทา ในปีการศึกษา 2550 จากนั้นในปี พ.ศ. 2550 – พ.ศ. 2551 ได้เข้าทำงานในตำแหน่งพนักงานจ้างผ่านพัสดุ หน่วยปฏิบัติการวิจัยการสื่อสารเชิงแสงและควอนตัม ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC)