ข้อเสนอโครงงานมหาบัณฑิต

(MASTER PROJECT PROPOSAL)

ชื่อเรื่อง (ภาษาไทย) การวัดความคล่องตัวต่อการเปลี่ยนแปลงสำหรับระบบอิงบริการ

ชื่อเรื่อง (ภาษาอังกฤษ) Measurement of Agility to Change for Service-Based Systems

เสนอโดย นายคงสวัสดิ์ โพธิพนม

**เลขประจำตัวนิสิต** 537 14042 21

**สาขาวิชา** วิศวกรรมซอฟต์แวร์

**ภาควิชา** วิศวกรรมคอมพิวเตอร์

**คณะ** วิศวกรรมศาสตร์

**สถานที่ติดต่อ** 789/23 ซอยอุดมสุข 27 ถนนสุขุมวิท 103

 แขวงบางนา เขตบางนา กรุงเทพฯ 10260

**โทรศัพท์** 087-672-1579

**อีเมล์** Kongsawas.P@Student.chula.ac.th

**อาจารย์ที่ปรึกษา** รศ.ดร.ทวิตีย์ เสนีวงศ์ ณ อยุธยา

**อาจารย์ที่ปรึกษาร่วม** -

**หน่วยงานที่ร่วมในโครงงาน -**

**ตัวแทนหน่วยงาน -**

คำสำคัญ **(ภาษาไทย)** ระบบอิงบริการ ตัววัดความคล่องตัว

**คำสำคัญ (ภาษาอังกฤษ)** Service-Based System, Agility Metric

ข้อเสนอโครงงานมหาบัณฑิต

ชื่อหัวเรื่อง

ภาษาไทย การวัดความคล่องตัวต่อการเปลี่ยนแปลงสำหรับระบบอิงบริการ

ภาษาอังกฤษ Measurement of Agility to Change for Service-Based Systems

1. **ที่มาและความสำคัญของปัญหา**

แนวคิดสถาปัตยกรรมเชิงบริการ (Service-Oriented Architecture) เป็นแนวคิดในการออกแบบระบบซอฟต์แวร์ ที่มีส่วนประกอบพื้นฐานของระบบคือ บริการหรือเซอร์วิซ ซึ่งเป็นหน่วยของซอฟต์แวร์ (Software Unit) ที่ทำฟังก์ชันงานบางอย่าง และให้บริการแก่ส่วนอื่นของระบบผ่านส่วนต่อประสาน (Interface) ที่ประกาศไว้ การพัฒนาซอฟต์แวร์ทำได้โดยเรียกใช้เซอร์วิซที่มีอยู่แล้ว หรือนำเซอร์วิซต่าง ๆ มาประกอบกันเป็นเซอร์วิซใหม่ หรือเรียกใช้เซอร์วิซต่าง ๆ เพื่อตอบสนองกระบวนการธุรกิจ การพัฒนาจึงทำได้สะดวกรวดเร็ว เพราะสามารถนำเซอร์วิซที่มีอยู่กลับมาใช้ใหม่ได้ และสามารถเปลี่ยนหรือถอดประกอบเซอร์วิซเป็นตัวใหม่ได้ บทความและวรรณกรรมที่เกี่ยวกับการพัฒนาระบบด้วยแนวคิดสถาปัตยกรรมเชิงบริการ ได้กล่าวถึงประโยชน์ของการพัฒนาระบบด้วยแนวคิดนี้ไว้หลายประการด้วยกัน เช่น ช่วยเพิ่มความสามารถในการทำงานร่วมกันกับระบบต่าง ๆ ภายในองค์กร เพิ่มผลตอบแทนจากการลงทุน เพิ่มความคล่องตัว และลดต้นทุนที่ใช้ในการพัฒนาระบบ เป็นต้น ซึ่งเป็นสิ่งจูงใจให้องค์กรต่าง ๆ ในปัจจุบันสนใจที่จะนำแนวคิดสถาปัตยกรรมเชิงบริการไปปรับเปลี่ยน หรือพัฒนาระบบของตนให้สอดคล้องกับเป้าหมายทางธุรกิจขององค์กร

ผู้วิจัยเห็นว่า แม้ว่าบทความ และวรรณกรรมต่าง ๆ จะกล่าวถึงประโยชน์ของสถาปัตยกรรมเชิงบริการไว้มากมาย แต่ประโยชน์นั้นน่าจะเห็นผลหลังจากที่ได้มีการพัฒนาระบบไปแล้ว คือเมื่อมีความต้องการสร้างระบบใหม่อิงเซอร์วิซเดิมหรือต้องการปรับเปลี่ยนระบบเดิม แต่ประโยชน์อาจจะไม่ปรากฏในทันที องค์กรอาจประสบปัญหาในการนำแนวคิดสถาปัตยกรรมเชิงบริการมาใช้เช่นกัน เนื่องด้วยต้องใช้เวลา และความพยายามไปกับการศึกษาทำความเข้าใจแนวคิดวิธีการในการพัฒนา การศึกษาถึงผลกระทบ รวมถึงการเปลี่ยนแปลงต่าง ๆ ภายในองค์กรกว่าจะพัฒนาระบบด้วยแนวคิดสถาปัตยกรรมเชิงบริการขึ้นมาได้

ดังนั้นผู้วิจัยจึงมีความสนใจที่จะทำการทดลองเพื่อทดสอบประโยชน์ของการพัฒนาระบบด้วยแนวคิดสถาปัตยกรรมเชิงบริการว่ามีความเหมาะสมที่จะเลือกนำมาใช้ในการพัฒนาระบบ โดยเฉพาะในช่วงของการบำรุงรักษาระบบหลังจากที่ระบบได้ถูกส่งมอบไปแล้วนั้น มักจะพบกับการเปลี่ยนแปลงระบบอยู่บ่อยครั้ง เนื่องจากต้องการปรับให้เหมาะสมกับการทำงาน หรือให้สอดคล้องกับการเปลี่ยนแปลงทางธุรกิจ ด้วยเหตุนี้ผู้วิจัยจึงเลือกที่จะพิจารณาถึงประโยชน์ของการพัฒนาระบบด้วยแนวคิดสถาปัตยกรรมเชิงบริการในแง่ความคล่องตัว (Agility) ซึ่งในที่นี้จะหมายถึงความสามารถของระบบในการตอบสนองต่อการเปลี่ยนแปลง การทดลองจะทำการเปรียบเทียบความคล่องตัวของระบบอิงบริการ (Service-Based System) กับระบบที่ไม่ได้อิงบริการ (Non Service-Based System) ในการดำเนินการกับการเปลี่ยนแปลง การเปรียบเทียบจะใช้แบบจำลองตัววัดความคล่องตัว (Agility Metric) ซึ่งพิจารณาจากความซับซ้อนของการเปลี่ยนแปลง (Change Complexity) กับระยะเวลาที่ใช้ในการดำเนินการกับการเปลี่ยนแปลง (Change Duration) โดยในส่วนของความซับซ้อนของการเปลี่ยนแปลงนั้นจะพิจารณาในสามมิติด้วยกัน ได้แก่ มิติของระดับชั้นในเชิงสถาปัตยกรรมของซอฟต์แวร์ที่การเปลี่ยนแปลงนั้นไปกระทบ (Architectural Depth) มิติความกว้างของกระบวนการทำงาน (Process Breadth) และมิติความซับซ้อนของโค้ด (Code Complexity) ที่การเปลี่ยนแปลงนั้นไปกระทบ

1. **ทฤษฎีที่เกี่ยวข้อง**
	1. **สถาปัตยกรรมเชิงบริการ**

สถาปัตยกรรมเชิงบริการ (Service-Oriented Architecture: SOA) [1] เป็นแนวคิดที่ใช้สร้างแบบจำลองสถาปัตยกรรมสำหรับการวางโครงสร้างพื้นฐานของระบบซอฟต์แวร์ขององค์กร ที่มุ่งเน้นในการเพิ่มประสิทธิภาพการทำงาน ความคล่องตัว และผลิตภาพ บนพื้นฐานของเซอร์วิซที่สามารถใช้ร่วมกันได้และนำกลับมาใช้ใหม่ได้ ทั้งนี้เนื่องจากระบบเทคโนโลยีสารสนเทศขององค์กรต่าง ๆ มีการพัฒนาระบบด้วยเทคโนโลยีที่หลากหลาย ซึ่งทำให้ยากในการจะใช้งานระบบเหล่านั้นร่วมกัน ยากต่อการการบำรุงรักษา มีค่าใช้จ่ายที่สูง และมีความคล่องตัวน้อยในการที่จะปรับเปลี่ยนระบบให้สอดรับกับการเปลี่ยนแปลงทางธุรกิจ แนวคิดสถาปัตยกรรมเชิงบริการจึงได้ถูกนำมาใช้เพื่อแก้ปัญหาดังกล่าว

การออกแบบระบบด้วยแนวคิดสถาปัตยกรรมเชิงบริการนั้น สามารถแบ่งออกได้เป็น 4 ระดับชั้นดังนี้ (รูปที่ 1)

* ระดับชั้นกระบวนการทางธุรกิจ (Business Processes Layer) เป็นระดับชั้นที่ประกอบด้วยส่วนของเซอร์วิซตั้งแต่หนึ่งเซอร์วิซขึ้นไป ที่ถูกนำมาใช้ในการจัดการกระบวนทางธุรกิจ ในการแก้ไขปัญหา หรือดำเนินการอย่างใดอย่างหนึ่ง
* ระดับชั้นส่วนของเซอร์วิซ (Services Layer) เป็นระดับชั้นที่ประกอบด้วยเซอร์วิซต่าง ๆ ที่สามารถนำกลับมาใช้ใหม่ได้ โดยเซอร์วิซเหล่านั้นถูกสร้างมาจากมอดูลต่าง ๆ ของโปรแกรมที่อยู่ระดับชั้นทรัพยากรการดำเนินงาน โดยมีส่วนต่อประสาน (Interface) ที่ประกาศไว้สำหรับให้ซอฟต์แวร์ส่วนอื่นเรียกใช้งาน
* ระดับชั้นส่วนประกอบวิสาหกิจ (Enterprise Components Layer) เป็นระดับชั้นของส่วนประกอบที่มีหน้าที่ในการจัดการตรรกะทางธุรกิจ และทำหน้าที่ต่อประสานกับระดับชั้นทรัพยากรการดำเนินงานในการดำเนินการของตรรกะนั้น ๆ
* ระดับชั้นทรัพยากรการดำเนินงาน (Operational Resources Layer) เป็นระดับชั้นที่ประกอบไปด้วยระบบสารสนเทศ โปรแกรม และฐานข้อมูล ที่ใช้ในองค์กร ซึ่งเซอร์วิซต่าง ๆ ได้ถูกสร้างจากส่วนประกอบเหล่านี้



รูปที่ 1 ระดับชั้นของสถาปัตยกรรมเชิงบริการ[1]

* 1. **ความคล่องตัว**

ความคล่องตัว (Agility) [2] เป็นประโยชน์อย่างหนึ่งที่องค์กรหรือหน่วยงานต่าง ๆ คาดหวังว่าจะได้รับจากการพัฒนาระบบด้วยแนวคิดสถาปัตยกรรมเชิงบริการ โดยความคล่องตัวนี้จะเกี่ยวกับความสามารถขององค์กรในการตอบสนองต่อการเปลี่ยนแปลงว่ามีประสิทธิภาพมากน้อยเพียงใด เนื่องด้วยการพัฒนาระบบแบบสถาปัตยกรรมเชิงบริการนี้เป็นการพัฒนาโดยการนำเซอร์วิซต่าง ๆ ที่องค์กรมีอยู่เดิมนำกลับมาใช้ใหม่ ซึ่งจะช่วยลดระยะเวลา และความพยายามที่ใช้ในการพัฒนา โดยไม่ต้องเริ่มทำใหม่ทั้งหมด รูปที่ 2 แสดงตัวอย่างการพัฒนาระบบการลงเวลา (Timesheet) จะเห็นว่าการพัฒนาระบบแบบสถาปัตยกรรมเชิงบริการนั้นสามารถลดในส่วนที่ต้องพัฒนาใหม่ลงเหลือเพียง 35 เปอร์เซ็นต์ และนำเซอร์วิซที่มีอยู่เดิมกลับมาใช้ใหม่ถึง 65 เปอร์เซ็นต์ อีกทั้งยังช่วยลดระยะเวลาในการส่งมอบลงประมาณ 50 เปอร์เซ็นต์เมื่อเทียบกับการพัฒนาระบบด้วยวิธีเดิม



รูปที่ 2 ตัวอย่างการพัฒนาด้วยสถาปัตยกรรมเชิงบริการเทียบกับการพัฒนาด้วยวิธีเดิม [2]

* 1. [**ค่าความซับซ้อน**](http://th.wikipedia.org/w/index.php?title=%E0%B8%84%E0%B8%A7%E0%B8%B2%E0%B8%A1%E0%B8%8B%E0%B8%B1%E0%B8%9A%E0%B8%8B%E0%B9%89%E0%B8%AD%E0%B8%99%E0%B8%82%E0%B8%AD%E0%B8%87%E0%B8%A7%E0%B8%87%E0%B8%88%E0%B8%A3%E0%B8%82%E0%B8%AD%E0%B8%87%E0%B9%82%E0%B8%84%E0%B9%89%E0%B8%94%E0%B9%82%E0%B8%9B%E0%B8%A3%E0%B9%81%E0%B8%81%E0%B8%A3%E0%B8%A1&action=edit&redlink=1)**ไซโคลมาติก**

ค่าความซับซ้อนไซโคลมาติก (Cyclomatic Complexity) [3] ถูกพัฒนาในปี คศ.1976 โดย Thomas J. McCabe ใช้ในการวัดความซับซ้อนของโปรแกรม ซึ่งเป็นตัวบ่งชี้ว่าโปรแกรมนั้นมีความยากต่อการบำรุงรักษา และการทดสอบมากน้อยเพียงใด โดยสามารถคำนวณได้จากสองวิธี

วิธีที่หนึ่ง คำนวณเส้นทางของกราฟการไหลของโปรแกรม (Flow Graph) คิดจากจำนวนเส้นเชื่อมต่อระหว่างโหนด (Edge) และจำนวนโหนด (Node) ของกราฟ ดังสมการ

 $υ\left(G\right)= e-n+2$ (1)

* ʋ(G) คือค่าความซับซ้อนไซโคลมาติก
* e คือจำนวนเส้นเชื่อมต่อระหว่างโหนด
* n คือจำนวนโหนด

วิธีที่สอง คำนวณจากจำนวนโหนดตัดสินใจ (Decision Node) ดังสมการ

 $υ\left(G\right)= P+1$ (2)

* เมื่อ P คือจำนวนโหนดตัดสินใจ

รูปที่ 3 แสดงตัวอย่างกราฟการไหลของโปรแกรม และการคำนวณค่าความซับซ้อนของโปรแกรมโดยวิธีไซโคลมาติก



รูปที่ 3 ตัวอย่างกราฟการไหลของโปรแกรม และการคำนวณค่าความซับซ้อนไซโคลมาติก

1. **งานวิจัยที่เกี่ยวข้อง**

จากการศึกษางานวิจัยที่เกี่ยวข้องกับการวัดความคล่องตัว สามารถแบ่งกลุ่มของงานวิจัยออกได้เป็น 2 กลุ่ม กลุ่มแรกเป็นกลุ่มงานวิจัยที่เกี่ยวข้องกับการวัดความคล่องตัวของซอฟต์แวร์ (Software Agility) โดยมีงานวิจัยที่ชื่อ “Measuring the agility of a SOA approach”[4] ได้เสนอวิธีการวัดความคล่องตัวของระบบที่ออกแบบด้วยสถาปัตยกรรมเชิงบริการ ซึ่งมีการเสนอตัววัดความคล่องตัว (Agility Metric) ที่พิจารณาจากความสัมพันธ์ของความซับซ้อนของการเปลี่ยนแปลงระบบ (Change Complexity) เทียบกับเวลาที่ใช้ในการดำเนินการเปลี่ยนแปลงระบบ (Change Duration) โดยมองความซับซ้อนของการเปลี่ยนแปลงระบบเป็น 2 มิติในลักษณะของพื้นที่สี่เหลี่ยมของการเปลี่ยนแปลง ได้แก่ มิติของจำนวนระดับชั้นในเชิงสถาปัตยกรรมของซอฟต์แวร์ที่การเปลี่ยนแปลงนั้นไปกระทบ (Architectural Depth) และมิติความกว้างของกระบวนการทำงาน (Process Breadth) ที่ใช้ในการดำเนินการกับการเปลี่ยนแปลง ดังนั้นความซับซ้อนของการเปลี่ยนแปลงระบบจะเป็นผลคูณของค่าจากสองมิติดังกล่าว จากนั้นทำการวัดข้อมูลในการดำเนินการกับการเปลี่ยนแปลงของสองระบบ ระบบแรกเป็นระบบที่ออกแบบด้วยสถาปัตยกรรมเชิงบริการ ส่วนระบบที่สองเป็นระบบที่ไม่ได้ออกแบบด้วยสถาปัตยกรรมเชิงบริการ และทำการเปรียบเทียบค่าที่ได้จากการวัด โดยคาดหวังว่าค่าความคล่องตัวของระบบที่ออกแบบด้วยสถาปัตยกรรมเชิงบริการจะสูงกว่าระบบที่ไม่ได้ออกแบบด้วยสถาปัตยกรรมเชิงบริการ อีกหนึ่งงานวิจัยในกลุ่มนี้เป็นงานวิจัยที่ชื่อว่า “A Practical Measurement for the Agility of Software Development Processes”[5] ได้เสนอวิธีการวัดความคล่องตัวของซอฟต์แวร์ด้วยการเทียบเคียงความคล่องตัวกับอัตราการเปลี่ยนแปลงของค่าเอ็นโทรปี โดยเอ็นโทรปีในงานวิจัยนี้จะหมายถึงความไม่แน่นอนของซอฟต์แวร์ ซึ่งพิจารณาจากล็อกการเปลี่ยนแปลง (Change Logs) ของซอฟต์แวร์นั้น ณ ช่วงเวลาหนึ่ง หากค่าอัตราการเปลี่ยนแปลงของค่าเอ็นโทรปีออกมาสูง จะถือว่าซอฟต์แวร์นั้นรับการเปลี่ยนแปลงได้บ่อยจึงมีความคล่องตัวสูงด้วยเช่นกัน งานวิจัยดังกล่าวได้ทำการทดสอบกับโอเพนซอร์ซ (Open Source) ซอฟต์แวร์ 2 ระบบ ระบบแรกเป็นระบบที่ใช้วิธีอะไจล์ในการพัฒนา (Agile System) และระบบที่สองเป็นระบบที่ไม่ได้ใช้วิธีอะไจล์ในการพัฒนา (Non-Agile System)

กลุ่มที่สองเป็นกลุ่มงานวิจัยที่เกี่ยวข้องกับการวัดความคล่องตัวของกระบวนการทำงานและการดำเนินงาน (Process and Operation Agility) โดยงานวิจัยที่ชื่อ “A measure of agility as the complexity of the enterprise system” [6] ได้เสนอแบบจำลองในการวัดความคล่องตัวของระบบวิสาหกิจ (Enterprise System) ด้วยแนวคิดที่ว่าหากระบบวิสาหกิจมีความซับซ้อนของกระบวนการทำงานต่ำ ระบบวิสาหกิจนั้นจะมีความคล่องตัวในการปรับเปลี่ยนกระบวนการทำงานที่สูงกว่าระบบวิสาหกิจที่มีความซับซ้อนของกระบวนการทำงานสูง งานวิจัยได้ทำการเปรียบเทียบความซับซ้อนของกระบวนการทำงานของระบบการให้บริการด้านโทรคมนาคมที่ใช้อยู่ ณ ปัจจุบัน (As Is Process) กับกระบวนการทำงานที่จะทำการปรับรื้อใหม่ (Reengineering Process) การวัดความซับซ้อนจะพิจารณาจากอัตราการเปลี่ยนแปลงสถานะของกระบวนการทำงาน หากค่าอัตราการเปลี่ยนแปลงสถานะของกระบวนการทำงานของระบบออกมาสูงจะกล่าวได้ว่าระบบมีความซับซ้อนสูง ส่วนการวัดความคล่องตัวของกระบวนการทำงานจะพิจารณาจากต้นทุน กับเวลาที่ใช้ทั้งหมดในการเปลี่ยนสถานะของกระบวนการทำงานตั้งแต่เริ่มกระบวนการจนจบกระบวนการทำงาน หากต้นทุนและเวลาที่ใช้ทั้งหมดออกมาสูงแสดงว่าระบบนั้นมีความคล่องตัวต่ำ แนวคิดวิธีการของงานวิจัยนี้เป็นการวัดความคล่องตัวของกระบวนการทำงานของระบบซึ่งไม่สามารถนำมาใช้ในการวัดความคล่องตัวของซอฟต์แวร์ได้ แต่สำหรับแนวคิดในการพิจารณาความซับซ้อนเป็นแนวคิดที่เหมาะสมที่จะนำมาสะท้อนถึงความคล่องตัวได้ดี งานวิจัยต่อมาชื่อ “Agility Measurement Index – A Metric for the Crossroads of Software Development Methodologies” [7] ได้เสนอตัววัดที่เป็นดัชนีการวัดความคล่องตัว (Agility Measurement Index) เพื่อช่วยให้องค์กรสามารถเลือกแบบจำลองกระบวนการ (Process Model) ที่ดีที่สุด และเหมาะกับแต่ละโครงการพัฒนาซอฟต์แวร์ จากผลของงานวิจัยพบว่าหากค่าที่ได้จากการวัด มีค่าไม่สูงมากจะบ่งบอกว่าระบบที่จะทำการพัฒนานั้นค่อนข้างนิ่ง และมีความซับซ้อนในกระบวนการพัฒนาน้อย ซึ่งจะเหมาะกับแบบจำลองน้ำตก (Waterfall Model) แต่ถ้าหากค่าที่ได้ออกมาสูงจะเหมาะกับแบบยูนิไฟด์ (Unified Process) หรือแบบเอกซ์พี (Extreme Programming: XP) มากกว่า และจะต้องนำค่าน้ำหนักที่ผู้ทำการวัดกำหนดให้กับแต่ละมุมมองต่าง ๆ มาประกอบการพิจารณาด้วย ในกรณีที่ค่าน้ำหนักในมุมมองของการติดต่อกับผู้ใช้มีค่าที่สูง จะเหมาะกับแบบเอกซ์พี แต่หากค่าในมุมมองของระยะเวลาที่ใช้ และค่าของความเสี่ยงมีค่าที่สูง จะเหมาะกับแบบยูนิไฟด์มากกว่า ข้อสังเกตสำหรับงานวิจัยดังกล่าวคือ ค่าที่ใช้ในสูตรการคำนวณของตัววัด จะมาจากการให้ค่าน้ำหนักจากตัวผู้วัด ซึ่งต้องอาศัยประสบการณ์ในการพัฒนาซอฟต์แวร์มาพอสมควร การวัดทำในช่วงก่อนเริ่มโครงการ และไม่สามารถนำมาวัดความคล่องตัวของซอฟต์แวร์ได้ เนื่องจากเป็นการวัดความคล่องตัวของแบบจำลองกระบวนการทำงาน อีกงานวิจัยหนึ่งในกลุ่มนี้เป็นงานวิจัยที่ชื่อว่า “Evaluating Agility in Extended Enterprise Systems: A Transportation Network case”[8] ซึ่งได้เสนอตัววัดที่เป็นค่าดัชนีความคล่องตัวรวม (Total Agility Index) เพื่อใช้ในการประเมินความสามารถของระบบวิสาหกิจในการปรับตัวเข้ากับการเปลี่ยนแปลงที่ไม่คาดคิดของสิ่งแวดล้อม โดยนำตัววัดไปใช้กับข้อมูลระบบการขนส่งมวลชนของเมืองนิวยอร์กในสถานการณ์ต่าง ๆ เช่น วันที่มีการแข่งขันกีฬา วันที่มีการเดินขบวนพาเหรด เป็นต้น เมื่อผลที่ได้จากการวัดออกมามีค่าต่ำ แสดงว่าระบบมีความคล่องตัวสูงในการปรับเข้ากับสถานการณ์ แต่หากผลที่ได้จากการวัดมีค่าสูงจะกล่าวได้ว่าระบบมีความคล่องตัวน้อยในการที่จะปรับตัวเข้ากับสถานการณ์นั้น ๆ ความคล่องตัวของงานวิจัยนี้จะเป็นลักษณะความคล่องตัวของการดำเนินงาน (Operation Agility) ดังนั้นจึงไม่สามารถนำมาใช้ในการวัดความคล่องตัวของซอฟต์แวร์ได้

จากงานวิจัยที่ได้กล่าวมาข้างต้น ผู้วิจัยได้เลือกให้ความสนใจกับงานวิจัยในกลุ่มที่เกี่ยวข้องกับการวัดความคล่องตัวของซอฟต์แวร์ และได้เลือกบทความ [4] ในการอ้างอิงสำหรับโครงงานนี้ เนื่องจากมีการเสนอวิธีการ และตัววัดที่สามารถทำความเข้าใจ และนำไปประยุกต์ใช้ได้ง่าย ซึ่งตัววัดที่เสนอมาสามารถประยุกต์ใช้ได้กับทุกระบบเทคโนโลยีสารสนเทศ ส่วนอีกงานวิจัยหนึ่งในกลุ่มที่ได้ทำการวัดความคล่องตัวของซอฟต์แวร์จากอัตราการเปลี่ยนแปลงของค่าเอนโทรปี มีลักษณะเป็นการพิจารณาความถี่ของการเกิดล็อกการเปลี่ยนแปลง ณ ช่วงเวลาหนึ่ง ผู้วิจัยเห็นว่าการเกิดล็อกการเปลี่ยนแปลงจำนวนมาก ไม่สามารถนำไปเป็นตัวบ่งชี้ว่าซอฟต์แวร์มีความคล่องตัวสูง เพราะการเปลี่ยนแปลงอาจจะเกิดจากการที่ซอฟต์แวร์ พัฒนาไม่ตรงตามความต้องการของผู้ใช้ หรือมีข้อบกพร่องสูง จึงจำเป็นต้องมีการร้องขอให้ทำการแก้ไข และเปลี่ยนแปลงระบบ

1. **แนวคิดและวิธีการดำเนินงาน**

จากที่มาและความสำคัญของปัญหา ผู้วิจัยได้ให้ความสนใจที่จะทำการทดลองเพื่อทดสอบถึงประโยชน์ของการพัฒนาด้วยแนวคิดสถาปัตยกรรมเชิงบริการ โดยจะพิจารณาในแง่ความคล่องตัวของซอฟต์แวร์ ในการตอบสนองต่อการเปลี่ยนแปลง ซึ่งการทดลองจะทำการเปรียบเทียบความคล่องตัวของระบบอิงบริการ กับระบบที่ไม่ได้อิงบริการ ในการดำเนินการกับการเปลี่ยนแปลง โดยมีรายละเอียดของการดำเนินงานดังนี้

* 1. **ศึกษาและเลือกตัววัดความคล่องตัว (Agility Metric)**

จากการศึกษางานวิจัยที่เกี่ยวข้อง ผู้วิจัยได้เลือกที่จะนำตัววัดความคล่องตัวของซอฟต์แวร์ที่เสนอในบทความ [4] มาใช้โดยพิจารณาจากความสัมพันธ์ของความซับซ้อนของการเปลี่ยนแปลงระบบ เทียบกับเวลาที่ใช้ในการดำเนินการเปลี่ยนแปลงระบบ ดังสมการ

$Agility = Change Complexity / Change Duration $(3)

จากสมการในส่วนของค่าความซับซ้อนซึ่งมีการพิจารณาพื้นที่ของการเปลี่ยนแปลงในสองมิติ ได้แก่ มิติของจำนวนระดับชั้นในเชิงสถาปัตยกรรมของซอฟต์แวร์ที่การเปลี่ยนแปลงนั้นไปกระทบ และมิติความกว้างของกระบวนการทำงานที่ใช้ในการดำเนินการกับการเปลี่ยนแปลง อาจจะไม่เพียงพอในการอธิบายถึงความซับซ้อนของการเปลี่ยนแปลง ผู้วิจัยเห็นว่าควรจะปรับเป็นการมองในสามมิติในลักษณะของปริมาตรของการเปลี่ยนแปลง ด้วยการเพิ่มมิติในส่วนของการที่คำขอเปลี่ยนแปลงนั้นไปกระทบกับโค้ดของระบบ ซึ่งจะมีความซับซ้อนแตกต่างกันออกไปในแต่ละมอดูลของโปรแกรม และความซับซ้อนของโค้ดนี้มีผลต่อความยากง่ายในการดำเนินการกับการเปลี่ยนแปลงด้วยเช่นกัน ดังเช่นรูปที่ 4 โดยสามารถแสดงการปรับสมการของความซับซ้อนของการเปลี่ยนแปลงได้ดังนี้

 $Change Complexity = Architectural Depth \* Process Breadth \* Code Complexity$ (4)

* ค่าระดับชั้นในเชิงสถาปัตยกรรมของซอฟต์แวร์ที่การเปลี่ยนแปลงนั้นไปกระทบ (Architectural Depth) ได้มาจากการนับจำนวนของระดับชั้นในเชิงสถาปัตยกรรมของซอฟต์แวร์ที่การเปลี่ยนแปลงนั้นไปกระทบ หรือทำให้ต้องมีการเปลี่ยนแปลงแก้ไขในระดับชั้นนั้น ๆ ซึ่งระดับชั้นในที่นี้อาจจะแบ่งเป็น ระดับชั้นประสบการณ์ของผู้ใช้ (User Experience) ระดับชั้นกระบวนการเชิงธุรกิจ (Business Process) และระดับชั้นของการจัดเก็บข้อมูลสารสนเทศ (Information Storage) เป็นต้น
* ค่าความกว้างของกระบวนการทำงาน (Process Breadth) ได้มาจากการนับจำนวนตำแหน่ง (Number of Roles) ในองค์กรที่ได้มีการมอบหมายให้ดำเนินการกับการเปลี่ยนแปลง เช่น ตำแหน่งนักวิเคราะห์ระบบ ตำแหน่งผู้พัฒนาโปรแกรม และตำแหน่งผู้ทดสอบระบบ เป็นต้น รวมกับจำนวนการส่งต่องานระหว่างตำแหน่งงาน (Number of Handoffs between Roles) ยกตัวอย่าง เช่น นักวิเคราะห์ระบบส่งงานเอกสารการออกแบบระบบให้กับผู้พัฒนาโปรแกรมเพื่อใช้ในการพัฒนาระบบ หรือผู้พัฒนาโปรแกรมส่งโปรแกรมที่พัฒนาเสร็จให้กับผู้ทดสอบระบบทำการทดสอบระบบว่าเป็นไปตามข้อกำหนดความต้องการหรือไม่ เป็นต้น ซึ่งแสดงได้ดังสมการ

$Process Breadth = No. of Roles + No. of Handoffs between Roles$ (5)

* ค่าความซับซ้อนของโค้ด (Code Complexity) สามารถคำนวณได้จากค่าความซับซ้อนไซโคลมาติกของโค้ดในแต่ละมอดูล ที่การเปลี่ยนแปลงนั้นไปกระทบ

รูปที่ 4 มิติความซับซ้อนของการเปลี่ยนแปลง

* 1. **เลือกหน่วยทดลอง**

เนื่องจากโครงงานนี้ต้องการทำการทดลองเพื่อเปรียบเทียบความคล่องตัวของระบบอิงบริการ กับระบบที่ไม่อิงบริการ ในการตอบสนองต่อการเปลี่ยนแปลง ดังนั้นจึงจำเป็นที่จะต้องเลือกหน่วยทดลองที่เหมาะสมเพื่อเป็นตัวแทนของทั้งสองระบบดังนี้

* + 1. **ระบบที่ไม่อิงบริการ (Non Service-Based System)**

ผู้วิจัยได้เลือกหน่วยทดลองที่เป็นระบบบันทึกการลา (E-Leave) ของบริษัท แอคเซลเล้นซ์ (ประเทศไทย) เพื่อเป็นตัวแทนของระบบที่ไม่อิงบริการ โดยแสดงคุณลักษณะของระบบบันทึกการลาดังตารางที่ 1 และสถาปัตยกรรมของระบบบันทึกการลา ดังรูปที่ 5

ตารางที่ 1 คุณลักษณะของระบบบันทึกการลา

|  |  |
| --- | --- |
| ประเภทของระบบ | เว็บแอปพลิเคชัน |
| ภาษาที่ใช้พัฒนา | จาวา |
| กรอบงานที่ใช้ | จาวาเซิร์ฟเวอร์เฟซ (Java Server Face: JSF), ไฮเบอร์เนต (Hibernate) |
| ระบบปฏิบัติการ | วินโดวส์ |
| แอปพลิเคชันเซอร์เวอร์ | ทอมแคต (Tomcat) รุ่น 6.0.29 |
| ระบบจัดการฐานข้อมูล | มายซีเควล (MySQL) รุ่น 5.1 |
| เครื่องมือที่ใช้ในการพัฒนา | เน็ตบีนส์ (NetBeans) รุ่น 6.5.1  |



รูปที่ 5 สถาปัตยกรรมของระบบบันทึกการลา

จากรูปที่ 5 ซึ่งแสดงสถาปัตยกรรมของระบบบันทึกการลา แบ่งออกเป็น 4 ระดับชั้นได้แก่

* ระดับชั้นการนำเสนอ (Presentation Layer) เป็นส่วนของการนำเสนอข้อมูลสารสนเทศแก่ผู้ใช้ และทำการแปลผลการตอบสนองของผู้ใช้ องค์ประกอบต่าง ๆ ในส่วนนี้ประกอบด้วย ฟอร์มของหน้าจอต่าง ๆ ที่เป็นเจเอสพี (Java Server Page: JSP) ซึ่งเป็นเทคโนโลยีหนึ่งของภาษาจาวา และคำสั่ง (Script)
* ระดับชั้นแอปพลิเคชัน (Application Layer) เป็นส่วนของโปรแกรมที่ทำหน้าที่ในการควบคุมการทำงานของหน้าจอโปรแกรมต่าง ๆ ในส่วนการนำเสนอ
* ระดับชั้นโดเมน (Domain Layer) เป็นส่วนของโปรแกรมที่เกี่ยวกับเงื่อนไขทางธุรกิจที่เกี่ยวกับการลา เช่น เงื่อนไขการคำนวณวันลาที่ใช้ไปของพนักงาน (Leave Consumed) เงื่อนไขการอนุมัติการลา (Leave Approval) เป็นต้น
* ระดับชั้นการเข้าถึงข้อมูล (Data Access Layer) จะเป็นส่วนติดต่อระหว่างโปรแกรมกับฐานข้อมูลของระบบบันทึกการลา ซึ่งมีฐานข้อมูลชื่อ “EApps” โดยมีกรอบงานไฮเบอร์เนต (Hibernate Framework) ที่ช่วยจัดการโปรแกรมที่อยู่ในรูปของอ็อบเจกต์ ให้สามารถใช้งานกับระบบฐานข้อมูลได้อย่างมีประสิทธิภาพ
	+ 1. **ระบบอิงบริการ (Service-Based System)**

ในส่วนของระบบอิงบริการที่จะเลือกมาเป็นหน่วยทดลอง ผู้วิจัยจะจัดทำขึ้นจากระบบบันทึกการลาที่ได้กล่าวมาในข้างต้น โดยจะทำการเพิ่มระดับชั้นในเชิงสถาปัตยกรรมขึ้นมาอีกหนึ่งระดับคือ ระดับชั้นเซอร์วิซ (Service Layer) การพิจารณาถึงเซอร์วิซที่จำเป็นสำหรับระบบบันทึกการลาว่าควรจะมีเซอร์วิซอะไรบ้างนั้น ผู้วิจัยจะใช้แนวทางของ [9] แต่ด้วยเวลาอันจำกัดจึงทำให้การพิจารณาเซอร์วิซที่เป็นไปตามสถาปัตยกรรมเชิงบริการอย่างเต็มรูปแบบนั้นจึงไม่สามารถทำได้ ผู้วิจัยจึงเน้นไปที่การพิจารณาเซอร์วิซเอนทิตี (Entity Services) ซึ่งเป็นเซอร์วิซที่เกี่ยวข้องกับเอนทิตีของข้อมูลในระบบ และเซอร์วิซอรรถประโยชน์ (Utility Services) ที่มีแนวโน้มของการนำกลับมาใช้ใหม่สูง ในส่วนของเซอร์วิซเอนทิตีนั้นจะพิจารณาจากเอนทิตีต่าง ๆ ของระบบบันทึกการลา ดังรูปที่6 โดยจะได้เป็น 2 เซอร์วิซ ได้แก่ เซอร์วิซพนักงาน (EmployeeService) และเซอร์วิซการลา (EleaveService) ในส่วนของเซอร์วิซอรรถประโยชน์นั้น จะเป็นเซอร์วิซสำหรับการแจ้งเตือน (NotificationService) ที่ใช้ในการส่งอีเมลแก่พนักงานผู้ที่ทำการลา และผู้ที่ทำการอนุมัติการลา จากที่กล่าวมาข้างต้นแสดงสถาปัตยกรรมของระบบบันทึกการลาแบบอิงบริการได้ดังรูปที่ 7 โดยเซอร์วิซทั้งสามจะอยู่ในระดับชั้นเซอร์วิซ และถูกเรียกใช้โดยส่วนของโปรแกรมในระดับชั้นโดเมน และมีการเรียกใช้ส่วนของโปรแกรมในระดับชั้นการเข้าถึงข้อมูล



รูปที่ 6แผนภาพความสัมพันธ์ของคลาสของระบบบันทึกการลา



รูปที่ 7สถาปัตยกรรมของระบบบันทึกการลาแบบอิงบริการ

* 1. **ทำการเลือกการเปลี่ยนแปลง**

 การเปลี่ยนแปลงที่เลือกใช้ในการทดลองกับระบบอิงบริการ และไม่อิงบริการนั้น ผู้วิจัยได้เลือกมาสองรายการ ซึ่งจะเป็นรายการที่เกี่ยวข้องในการขอเปลี่ยนแปลงระบบบันทึกการลา โดยมีรายละเอียดของการเปลี่ยนแปลงในแต่ละรายการดังนี้

* รายการที่ 1 เนื่องจากปัจจุบันในช่วงสิ้นปีฝ่ายทรัพยากรบุคคล จะทำการร้องขอให้ผู้ดูแลระบบบันทึกการลา สรุปวันลาคงเหลือของพนักงานแต่ละคนเป็นแฟ้มข้อมูลไมโครซอฟท์ เอกซ์เซล (Microsoft Excel) เพื่อนำไปคำนวณวันลาที่จะยกยอดไปรวมกับวันลาในปีถัดไปของพนักงานแต่ละคน จากนั้นจึงส่งแฟ้มข้อมูลนั้นกลับไปให้กับทางผู้ดูแลระบบช่วยนำข้อมูลการยกยอดวันลาของพนักงานเข้าระบบ ฝ่ายทรัพยากรบุคคลจึงมีความต้องการที่จะให้ระบบช่วยทำการยกยอดวันลาคงเหลือของพนักงานแต่ละคนให้โดยอัตโนมัติ ซึ่งเงื่อนไขในการคิดยกยอดวันลานั้น จะคิดจากวันลาคงเหลือ (Leave Balance) ของพนักงาน เทียบกับสิทธิ์การลาที่พนักงานสามารถลาได้ในปีนั้น (Leave Entitlement) โดยจะสามารถยกยอดวันลาได้สูงสุดคือครึ่งหนึ่งของสิทธิ์การลาในปีนั้น แต่ถ้าหากวันลาคงเหลือของพนักงานน้อยกว่าครึ่งหนึ่งของสิทธิ์การลาก็จะยกยอดไปเท่ากับจำนวนวันลาที่คงเหลือนั้น ยกตัวอย่าง ดังนี้
* กรณีที่ 1 พนักงานชื่อนาย เอ

มีวันลาคงเหลือเท่ากับ 10 วัน

มีสิทธิ์การลาในปีนั้นเท่ากับ 14 วัน

ดังนั้นจำนวนวันลาที่สามารถยกยอดได้เท่ากับ 7 วัน

* กรณีที่ 2 พนักงานชื่อนาย บี

มีวันลาคงเหลือเท่ากับ 5 วัน

มีสิทธิ์การลาในปีนั้นเท่ากับ 14 วัน

ดังนั้นจำนวนวันลาที่สามารถยกยอดได้เท่ากับ 5 วัน

* รายการที่ 2 ระบบบันทึกการลาในปัจจุบันจะรองรับกับพนักงานที่ทำงานวันจันทร์ถึงวันศุกร์เท่านั้น แต่เนื่องจากมีแผนกหนึ่งซึ่งมีลักษณะการทำงานเป็นกะ (Shift) โดยพนักงานในแผนกนี้จะทำงานตามตารางที่หัวหน้าแผนกได้จัดไว้ ซึ่งจะต้องมีการเข้ามาทำงานในวันหยุดของสัปดาห์คือวันเสาร์และอาทิตย์ด้วย และถ้าพนักงานคนใดมีความจำเป็นต้องลางานในวันดังกล่าว จะไม่สามารถทำการบันทึกการลาของตนในระบบได้ เนื่องจากระบบจะไม่นับรวมวันหยุดของสัปดาห์มาคิดเป็นวันลาที่พนักงานได้ลาไป ดั้งนั้นฝ่ายทรัพยากรบุคคลจึงมีความต้องการที่จะให้ระบบทำการรองรับการทำงานของพนักงานที่ทำงานเป็นกะ
	1. **ดำเนินการทดลอง**

หลังจากที่ได้หน่วยทดลองที่เป็นตัวแทนของระบบอิงบริการ และระบบที่ไม่อิงบริการแล้ว จากนั้นนำการเปลี่ยนแปลงที่เลือกใช้ในการทดลองมาดำเนินการกับแต่ละระบบ กำหนดให้ช่วงของการดำเนินงานแบ่งออกเป็นสามช่วงประกอบด้วย ช่วงการวิเคราะห์ออกแบบ ช่วงการพัฒนาโปรแกรม และช่วงการทดสอบโปรแกรม โดยสมมติให้ผู้วิจัยทำหน้าที่ในสามตำแหน่งได้แก่ นักวิเคราะห์ระบบ ผู้พัฒนาโปรแกรม และผู้ทดสอบระบบ ในทุกขั้นตอนของการดำเนินงานจะมีการบันทึกเวลาการทำงาน (Timesheet) ทุกครั้ง โดยในการดำเนินการของแต่ละระบบกับการเปลี่ยนแปลงแต่ละรายการมีขั้นตอนการดำเนินงาน ดังนี้

* + 1. **ทำการวิเคราะห์และออกแบบระบบจากการเปลี่ยนแปลง**

ขั้นตอนนี้เป็นการนำการเปลี่ยนแปลงมาทำการวิเคราะห์ และออกแบบ ซึ่งจะได้เอกสารการออกแบบที่ประกอบไปด้วยแผนภาพต่าง ๆ ได้แก่ แผนภาพยูสเคส (Use Case Diagram) แผนภาพแสดงความสัมพันธ์ของคลาส (Class Diagram) และแผนภาพลำดับเหตุการณ์ (Sequence Diagram)

* + 1. **บันทึกค่าระดับชั้นในเชิงสถาปัตยกรรมของซอฟต์แวร์และค่าความซับซ้อนของโค้ดที่ถูกกระทบ**

หลังจากขั้นตอนของการวิเคราะห์ และออกแบบ จะสามารถทราบถึงข้อมูลระดับชั้นในเชิงสถาปัตยกรรมของซอฟต์แวร์ที่การเปลี่ยนแปลงนั้นไปกระทบ โดยการพิจารณาจากแผนภาพลำดับเหตุการณ์ จะบอกถึงคลาส หรืออ็อบเจกต์ต่าง ๆ ที่เกี่ยวข้องในลำดับการทำงานของระบบ จากนั้นพิจารณาคลาส หรืออ็อบเจกต์ที่เกี่ยวข้องว่าอยู่ในระดับชั้นเชิงสถาปัตยกรรมใดบ้าง

ในส่วนของค่าความซับซ้อนของโค้ด จะพิจารณาจากคลาสที่เกี่ยวข้องในแผนภาพลำดับเหตุการณ์ โดยวัดค่าความซับซ้อนของโค้ดของคลาสที่เกี่ยวข้องจากค่าไซโคลมาติก และใช้โปรแกรมลอคเมตริกส์ (LocMetrics) ดังเช่นในรูปที่ 8 เป็นเครื่องมือในการวัดความซับซ้อนของโค้ด ซึ่งสามารถบอกถึงจำนวนแฟ้มข้อมูลที่มีอยู่ในระบบ (Source Files) จำนวนสารบบ (Directories) จำนวนบรรทัดของโค้ด (Lines of Code) และจำนวนค่าความซับซ้อนรวมของโค้ด (Total McCabe VG Complexity) เป็นต้น

เมื่อทราบถึงข้อมูลของระดับชั้นในเชิงสถาปัตยกรรม และค่าความซับซ้อนของโค้ดที่ถูกกระทบแล้วจะต้องนำค่าที่ได้บันทึกลงแบบฟอร์มการบันทึกค่าความซับซ้อนของการเปลี่ยนแปลง



รูปที่ 8ตัวอย่างการใช้งานโปรแกรมลอคเมตริกส์

* + 1. **ทำการพัฒนาระบบตามเอกสารการออกแบบ**

ในส่วนนี้จะเป็นขั้นตอนการพัฒนาระบบตามเอกสารการออกแบบ

* + 1. **ทำการทดสอบระบบ**

เป็นขั้นตอนการทดสอบหน้าที่การทำงานของระบบว่าเป็นไปตามเอกสารการออกแบบหรือไม่

* + 1. **ทำการบันทึกค่าความกว้างของกระบวนการทำงาน**

หลังจากที่ดำเนินงานมาจนถึงขั้นตอนของการทดสอบระบบแล้ว ก็จะสามารถทราบถึงค่าความกว้างของกระบวนการทำงาน ซึ่งประกอบด้วยจำนวนตำแหน่งต่าง ๆ ที่ใช้การดำเนินการกับการเปลี่ยนแปลง และจำนวนการส่งต่องานระหว่างตำแหน่งต่าง ๆ ซึ่งจะต้องนำค่าบันทึกลงแบบฟอร์มการบันทึกค่าความซับซ้อนของการเปลี่ยนแปลงด้วยเช่นกัน

* 1. **รวบรวมข้อมูลจากการทดลอง**

ข้อมูลจากการทดลองที่ต้องมีการเก็บรวบรวมได้แก่

* แบบฟอร์มการบันทึกค่าความซับซ้อนของการเปลี่ยนแปลง ซึ่งจะมีการบันทึกข้อมูลของค่าต่าง ๆ ได้แก่
* จำนวนของระดับชั้นในเชิงสถาปัตยกรรมของซอฟต์แวร์ที่ถูกกระทบ
* จำนวนตำแหน่งต่าง ๆ ที่ใช้การดำเนินการกับการเปลี่ยนแปลง
* จำนวนการส่งต่องานระหว่างตำแหน่งต่าง ๆ
* ค่าความซับซ้อนของโค้ดที่การเปลี่ยนแปลงนั้นไปกระทบ
* ข้อมูลการบันทึกเวลาการทำงาน ดังรูปแบบในตารางที่ 2

ตารางที่ 2 ตัวอย่างตารางการบันทึกเวลาการทำงาน

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **No.** | **Project** | **Phase** | **Task** | **Description** | **Role** | **Timesheet Date** | **Working Hour** | **Remark** |
| **1** |  |  |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |  |  |
| **3** |  |  |  |  |  |  |  |  |
| **4** |  |  |  |  |  |  |  |  |
| **5** |  |  |  |  |  |  |  |  |

* 1. **ทำการวิเคราะห์และสรุปผลการทดลอง**

เมื่อทำการเก็บรวบรวมข้อมูลจากการทดลองครบถ้วนแล้ว จะต้องนำข้อมูลเหล่านั้นมาทำการวิเคราะห์ และสรุปผลการทดลองซึ่งมีขั้นตอนดังนี้

* + 1. **คำนวณค่าความคล่องตัว**

นำค่าที่บันทึกได้จากการทดลองของแต่ละระบบที่ดำเนินการกับแต่ละการเปลี่ยนแปลง มาแทนค่าใส่ในสมการที่ (4) และ (3) ในหัวข้อที่ 4.1 เพื่อคำนวณหาค่าความคล่องตัวของระบบในการดำเนินการกับการเปลี่ยนแปลง

* + 1. **วิเคราะห์ และสรุปผล**

นำค่าต่าง ๆ ที่ได้ของสองระบบมาทำการเปรียบเทียบความเหมือน และความแตกต่างในการดำเนินการกับแต่ละการเปลี่ยนแปลง เพื่อพิจารณาถึงปัจจัยที่มีผลต่อความคล่องตัวของระบบในการดำเนินการกับการเปลี่ยนแปลง ทำการเปรียบเทียบค่าเมื่อใช้สมการการคำนวณของ [4] ว่าสอดคล้องกันหรือไม่ จากนั้นทำการสรุปผลการทดลองว่าระบบตัวอย่างที่พัฒนาแบบอิงบริการนั้น มีความคล่องตัวมากกว่าระบบที่ไม่ได้อิงบริการหรือไม่

1. **วัตถุประสงค์**
	1. เพื่อเสนอตัววัดที่สามารถใช้ในการวัดความคล่องตัวต่อการเปลี่ยนแปลงของระบบอิงบริการ
2. **ขอบเขตการดำเนินงาน**
	1. เสนอตัววัดความคล่องตัวที่พิจารณาความสัมพันธ์ของความซับซ้อนของคำขอเปลี่ยนแปลง เทียบกับเวลาที่ใช้ดำเนินการต่อคำขอเปลี่ยนแปลง และมองความซับซ้อนของคำขอเปลี่ยนในลักษณะของปริมาตรของการเปลี่ยนแปลง ซึ่งพิจารณาจาก ระดับชั้นในเชิงสถาปัตยกรรมของซอฟต์แวร์ที่คำขอเปลี่ยนแปลงนั้นไปกระทบ ความกว้างของกระบวนการทำงาน และความซับซ้อนของรหัสต้นทางที่คำขอเปลี่ยนแปลงนั้นไปกระทบ
	2. เพื่อทำการทดลองดูค่าความคล่องตัวของระบบที่พัฒนาแบบอิงบริการ เทียบกับระบบที่พัฒนาแบบไม่อิงบริการในการตอบสนองต่อการเปลี่ยนแปลงในลักษณะต่าง ๆ
	3. ใช้ระบบบันทึกการลาของบริษัทพัฒนาซอฟต์แวร์แห่งหนึ่ง เป็นหน่วยทดลองโดยใช้เป็นตัวแทนของระบบที่ไม่อิงบริการ ส่วนตัวแทนของระบบอิงบริการจะจัดทำขึ้นจากระบบบันทึกการลา โดยจะมีการพิจารณาออกแบบเซอร์วิซหรือบริการที่เป็นเซอร์วิซเอนทิตี (Entity Services) ซึ่งเป็นเซอร์วิซที่เกี่ยวข้องกับเอนทิตีของข้อมูลในระบบ และเซอร์วิซอรรถประโยชน์ (Utility Services) ที่มีแนวโน้มของการนำกลับมาใช้ใหม่สูง
	4. กำหนดให้ช่วงของการดำเนินงานแบ่งออกเป็นสามช่วงประกอบด้วย ช่วงการวิเคราะห์ออกแบบ ช่วงการพัฒนาโปรแกรม และช่วงการทดสอบโปรแกรม โดยสมมติให้ผู้วิจัยทำหน้าที่ในสามตำแหน่งได้แก่ นักวิเคราะห์ระบบ ผู้พัฒนาโปรแกรม และผู้ทดสอบระบบ
	5. ระบบที่ใช้เป็นหน่วยทดลองจะทำการพัฒนา และติดตั้งบนเครื่องแสตนด์อะโลนเท่านั้น
3. **ขั้นตอนการดำเนินงาน**
	1. ศึกษาวิธีการที่ใช้ในการวัดความคล่องตัวต่อการเปลี่ยนแปลงของระบบซอฟต์แวร์
	2. ปรับปรุงตัววัดที่ใช้ในการวัดความคล่องตัวต่อการเปลี่ยนแปลงของระบบซอฟต์แวร์
	3. ทำการคัดเลือกหน่วยทดลองสำหรับใช้ในการทดลอง
	4. สร้างหน่วยทดลองที่เป็นระบบอิงบริการ
	5. ทำการคัดเลือกการเปลี่ยนแปลงที่เหมาะสมกับการทดลอง
	6. ดำเนินการทดลอง
	7. สรุปและวิเคราะห์ผลการทดลอง
	8. จัดทำบทความตีพิมพ์
	9. จัดทำรายงาน

****

****

รูปที่ 9 ขั้นตอนการดำเนินงาน

1. **ประโยชน์ของโครงงานที่คาดว่าจะได้รับ**
	1. องค์กรสามารถนำตัววัดความคล่องตัวไปใช้ในการวัดความคล่องตัวของระบบซอฟต์แวร์ที่ตนได้ทำการพัฒนา ไม่ว่าจะเป็นระบบที่เป็นแบบอิงบริการหรือไม่ก็ตาม เพื่อนำไปปรับปรุงกระบวนการพัฒนาซอฟต์แวร์ของตนในลำดับต่อไปได้
	2. ได้ข้อมูลเกี่ยวกับความคล่องตัวของระบบแบบอิงบริการในแง่ของการตอบสนองต่อการเปลี่ยนแปลง เพื่อเป็นประโยชน์ต่อการพิจารณาขององค์กรในการนำสถาปัตยกรรมเชิงบริการมาใช้
2. **รายการอ้างอิง**

[1] Gabhart, K., and Bhattacharaya, B. Service Oriented Architecture Field Guide for Executives. Hoboken, New Jersey: John Wiley & Sons, Inc., 2008.

[2] Erl, T. SOA Principles of Service Design. Prentice Hall, 2008.

[3] McCabe, T.J. A Complexity Measure. Software Engineering, IEEE Transactions, pp. 308-320. IEEE, 1976.

[4] Malik, N. Measuring the agility of a SOA approach [Online]. 2007. Available from: http://blogs.msdn.com/b/nickmalik/archive/2007/12/14/measuring-the-agility-of-a-soa-approach.aspx [2011, September 8].

[5] Shawky, D.M., and Ali A.F. A Practical Measurement for the Agility of Software Development Processes. Computer Technology and Development (ICCTD), 2010 2nd International Conference, pp. 230-234. IEEE, 2010.

[6] Arteta, B.M., and Giachetti, R.E.A measure of agility as the complexity of the enterprise system**.** Robotics and Computer-Integrated Manufacturing 20,pp. 495-503.Science Direct, 2004.

[7] Datta, S. Agility Measurment Index – A Metric for the Crossroads of Software Development Methodologies. ACM-SE 44 Proceedings of the 44th annual Southeast regional conference, pp. 271-273. New York: ACM, 2006.

[8] Mansouri, M., Ganguly, A., and Mostashari, A. Evaluating Agility in Extended Enterprise Systems: A Transportation Network case. American J. of Engineering and Applied Sciences, pp. 142-152. Science Publications, 2011.

[9] Erl, T. Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall, 2005.