การพัฒนาแบบปรับเปลี่ยนและยืดหยุ่นของกรอบโครงร่าง ระบบการบริหารสารสนเทศบัณฑิตวิทยาลัย

โดย

บัวเรียน สูงพล

วิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตาม
หลักสูตรปรัชญาดุษฎีบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ
วิทยาลัยนวัตกรรมดิจิทัลเทคโนโลยี

บัณฑิตวิทยาลัย มหาวิทยาลัยรังสิต
ปีการศึกษา 2567

ADAPTABLE AND FLEXIBLE DEVELOPMENT OF GRADUATE INFORMATION MANAGEMENT SYSTEM FRAMEWORK

BY

BUERIAN SOONGPOL

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR

THE DEGREE OF DOCTOR OF PHILOSOPHY

IN INFORMATION TECHNOLOGY

COLLEGE OF DIGITAL INNOVATION TECHNOLOGY

GRADUATE SCHOOL, RANGSIT UNIVERSITY

ACADEMIC YEAR 2024

ดุษฎีนิพนธ์เรื่อง

การพัฒนาแบบปรับเปลี่ยนและยืดหยุ่นของกรอบโครงร่าง
ระบบการบริหารสารสนเทศบัณฑิตวิทยาลัย

โดย

บัวเรียน สูงพล

ได้รับการพิจารณาให้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร
ปริญญาปรัชญาดุษฎีบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ

มหาวิทยาลัยรังสิต
ปีการศึกษา 2567

รศ. อานนท์ สุขเสถียรวงศ์ ประธานกรรมการสอบ	ผศ. ดร.กวีวัฒน์ อำนาจโชติพันธุ์ กรรมการ
รศ. ดร.สิริพร ศุภราทิตย์ กรรมการ	รศ. ดร.ดวงอาทิตย์ ศรีมูล กรรมการ
ผศ. ดร.มีนนภา รักษ์หิรัญ กรรมการและอาจารย์ที่ปรึกษาร่วม	รศ. ดร.ปณิธิ เนตินันทน์ กรรมการและอาจารย์ที่ปรึกษา

บัณฑิตวิทยาลัยรับรองแล้ว

(ศ. ดร.สื่อจิตต์ เพ็ชร์ประสาน)
คณบดีบัณฑิตวิทยาลัย
14 กุมภาพันธ์ 2568

Dissertation entitled

ADAPTABLE AND FLEXIBLE DEVELOPMENT OF GRADUATE INFORMATION MANAGEMENT SYSTEM FRAMEWORK

by

BUERIAN SOONGPOL

was submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Information Technology

Rangsit University
Academic Year 2024

----------	----------
Assoc. Prof. Anon Sukstrienwong Examination Committee Chairperson	Asst. Prof. Kawiwat Amnatchotiphan, D.Eng Member
----------	----------
Assoc. Prof. Siriporn Supratid, D.Tech.Sci. Member	Assoc. Prof. Duang-arthit Srimoon, D.Eng Member
----------	----------
Asst. Prof. Meennapa Rukhiran, Ph.D. Member and Co-Advisor	Assoc. Prof. Paniti Netinant, Ph.D. Member and Advisor

Approved by Graduate School

(Prof. Suejit Pechprasarn, Ph.D.)
Dean of Graduate School
February 14, 2025

กิตติกรรมประกาศ

ขอกราบขอบพระคุณ นายบุญธรรม สูงพล และ นางวงเดือน สูงพล บิดา-มารดา และครอบครัว ที่คอยส่งเสริมสนับสนุนและเป็นแรงบันดาลใจที่ดีในการเรียนรู้และในการพัฒนาตัวเองอย่างต่อเนื่อง ขอบคุณภรรยาและบุตรชาย ที่คอยเป็นกำลังใจและสนับสนุนมาโดยตลอด

ดุษฎีนิพนธ์ฉบับนี้สำเร็จได้เป็นอย่างดีเนื่องมาจากได้รับการอนุเคราะห์จาก รศ.ดร.ปณิธิ เนตินันทน์ อาจารย์ที่ปรึกษา และ ผศ.ดร.มีนนภา รักษ์หิรัญ ที่ปรึกษาร่วมที่ให้ความเอาใจใส่และค่อยช่วยเหลือเป็นอย่างดี เป็นแบบอย่างที่ดีและคอยให้คำปรึกษาแนะนำเป็นอย่างดีมาโดยตลอด รศ.อานนท์ สุขเสถียรวงศ์ ประธานกรรมการสอบ ผศ.ดร.สิริพร ศุภราทิตย์ ผศ.ดรกวีวัตน์ อำนาจโชติพันธุ์ และ ผศ.ดร.ธรรณพ อารีพรรค กรรมการสอบทุกท่าน ที่กรุณาให้เกียรติมาเป็นกรรมการสอบดุษฎีนิพนธ์ และให้ความคิดเห็นอันเป็นประโยชน์ต่อดุษฎีนิพนธ์ฉบับนี้ให้สำเร็จโดยสมบูรณ์

ขอขอบพระคุณคณาจารย์ที่ให้ความรู้ทางวิชาการ และเจ้าหน้าที่ที่หลักสูตรปรัชญาดุษฎีบัณฑิตสาขาวิชาเทคโนโลยีสารสนเทศมหาวิทยาลัยรังสิตที่ให้ความอนุเคราะห์ในการประสานงานต่าง ๆ

ขอขอบพระคุณผู้บริหารสำนักงานหลักประกันสุขภาพแห่งชาติ (สปสช) และผู้บริหารฝ่ายพัฒนาสารสนเทศดิจิตอลและเจ้าหน้าที่ทุก ๆท่านที่ให้กำลังใจและสนับสนุนมาโดยตลอดการศึกษา

ขอขอบคุณ พี่ ๆ ในหลักสูตรปรัชญาดุษฎีบัณฑิตสาขาวิชาเทคโนโลยีสารสนเทศมหาวิทยาลัยรังสิตทุกท่านที่ช่วยเป็นแรงใจและร่วมสู้ฝ่าฟันด้วยกันทุก ๆ ท่าน

บัวเรียน สูงพล

ผู้วิจัย

6304570	:	บัวเรียน สูงพล
ชื่อดุษฎีนิพนธ์	:	การพัฒนาแบบปรับเปลี่ยนและยืดหยุ่นของกรอบโครงร่างระบบการบริหารสารสนเทศบัณฑิตวิทยาลัย
หลักสูตร	:	ปรัชญาดุษฎีบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ
อาจารย์ที่ปรึกษา	:	รศ. ดร.ปณิธิ เนตินันทน์
อาจารย์ที่ปรึกษาร่วม	:	ผศ. ดร.มีนนภา รักษ์หิรัญ

บทคัดย่อ

การดำเนินงานทางธุรกิจที่เป็นประจำได้เปลี่ยนแปลงไปเป็นบริการดิจิทัล ซึ่งเพิ่มแพลตฟอร์มที่ทันสมัยหลายแห่ง การปรับเข้ากันอย่างรวดเร็ว ความยืดหยุ่น และผลกระทบต่อสิ่งแวดล้อมผ่านการใช้พลังงาน ขยะจากฮาร์ดแวร์ และการลงทุนในเทคโนโลยี การพัฒนาระบบที่ยืดหยุ่นและยั่งยืนซึ่งเน้นประสิทธิภาพการใช้พลังงานสามารถช่วยสร้างสรรค์การพัฒนาซอฟต์แวร์ได้อย่างดี เนื่องจากการเปลี่ยนแปลงของการให้บริการดิจิทัล การวิจัยนี้ได้รับแรงบันดาลใจจากความต้องการในการปรับปรุงการใช้พลังงานในการออกแบบและพัฒนาซอฟต์แวร์ในช่วงแรก เนื่องจากมีความต้องการด้านประสิทธิภาพทางเทคโนโลยีและความยั่งยืนที่เพิ่มขึ้น แม้ว่าวิธีการ Agile แบบดั้งเดิมจะมีประสิทธิภาพในการพัฒนาซอฟต์แวร์แบบ Iterative และการมีส่วนร่วมของผู้มีส่วนได้ส่วนเสีย แต่บ่อยครั้งที่มันประสบปัญหาในเรื่องความยั่งยืนและประสิทธิภาพด้านพลังงานในระยะยาว การใช้ Agile ที่ขยายออกซึ่งรวม Agile เข้ากับสถาปัตยกรรมหลายชั้นและกรอบการมองด้านแง่มุม (ALAI) ผลการศึกษานี้ไม่เพียงแต่เป็นทฤษฎีเท่านั้น แต่ยังเกี่ยวข้องกับการปฏิบัติจริง เนื่องจากมีการสำรวจประสิทธิภาพการใช้พลังงานของวิธีการพัฒนาซอฟต์แวร์ ALAI โดยใช้ระบบบริการข้อมูลการรับสมัครนักเรียนระดับบัณฑิตศึกษา (GAISS) เป็นตัวอย่าง GAISS เป็นระบบที่ซับซ้อนที่จัดการกระบวนการรับสมัครระดับบัณฑิตศึกษาทั้งหมด ตั้งแต่การส่งใบสมัครไปจนถึงการตัดสินใจสุดท้าย การศึกษานี้ได้คำนวณการใช้พลังงานจากหน้าแสดงรายละเอียดนักศึกษาโดยการวิเคราะห์จาก Log Microsoft IIS ตั้งแต่เดือนกุมภาพันธ์ 2022 ถึงพฤษภาคม 2024 ผลลัพธ์ที่นำไปใช้ได้โดยตรงแสดงให้เห็นว่า GAISS ที่อิงกับกรอบ ALAI ลดการใช้พลังงานลง 10.7914% เมื่อเปรียบเทียบกับการพัฒนาซอฟต์แวร์ Agile แบบดั้งเดิม ALAI ใช้พลังงาน 892.80 kWh เทียบกับ 1000.80 kWh ของ Agile ระหว่างการดำเนินการ ซึ่งแสดงให้เห็นถึงการประหยัดพลังงาน ผลการศึกษานี้แสดงให้เห็นถึงประโยชน์ของการรวมกรอบการมองด้านแง่มุมและวิธีการหลายชั้นเข้ากับวิธีการ Agile ซึ่งมีส่วนช่วยในการสนทนาเกี่ยวกับการพัฒนาซอฟต์แวร์ที่ยั่งยืน การศึกษานี้เน้นความสำคัญของกรอบงานที่มีประสิทธิภาพการใช้พลังงานอย่าง ALAI เพื่อลดผลกระทบต่อสิ่งแวดล้อมของระบบซอฟต์แวร์และส่งเสริมความยั่งยืนของการพัฒนาซอฟต์แวร์

(ดุษฎีนิพนธ์มีจำนวนทั้งสิ้น 91 หน้า)

คำสำคัญ:วิธีการทำงานอย่างคล่องแคล่วรวดเร็ว; เชิงลักษณะ; ประสิทธิภาพทางพลังงาน; ความยืดหยุ่น; กรอบการทำงาน; การแบ่งชั้น; การพัฒนาซอฟต์แวร์; ความยั่งยืน

ลายมือชื่อนักศึกษา ..... ลายมือชื่ออาจารย์ที่ปรึกษา .....

ลายมือชื่ออาจารย์ที่ปรึกษาร่วม .....

6304570	:	Buerian Soongpol
Dissertation Title	:	Adaptable and Flexible Development of Graduate Information Management System Framework
Program	:	Doctor of Philosophy in Information Technology
Dissertation Advisor	:	Assoc. Prof. Paniti Netinant, Ph.D.
Dissertation Co-Advisor	:	Asst. Prof. Meennapa Rukhiran, Ph.D

Abstract

Regular business operations transform into digital services, increasing advanced multi-platforms, rapid operational alignment, flexibility, and environmental impact through energy consumption, hardware waste, and technology investments. Flexible and sustainable system development models emphasizing energy efficiency can help innovate software development as digital servicing applications shift. This research is motivated by the need to improve energy consumption in early software design and development due to rising technological efficiency and sustainability demands. Although effective in iterative development and stakeholder engagement, traditional Agile methodologies often struggle with long-term sustainability and energy efficiency. An extended Agile, combining Agile, a software development methodology that promotes adaptive planning, evolutionary development, early delivery, and continuous improvement, with layered architecture, a design principle that separates the concerns of a software application into different layers, and aspect-oriented framework, a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns, (ALAI), promises to improve system modularity, flexibility, maintainability, and sustainability. This study's findings are theoretical and practically relevant, as they explore the energy efficiency of ALAI software development methodologies, using the graduate admission information system services (GAISS) as an example. GAISS is a complex system that handles the entire graduate admissions process, from application submission to final decision. The study quantifies the energy usage of a student-list webpage by analyzing Microsoft IIS server logs from February 2022 to May 2024. Directly applicable findings show that the GAISS based on the ALAI framework reduces energy consumption by 10.7914% compared to traditional Agile software development. ALAI used 892.80 kWh versus Agile's 1000.80 kWh during operations, saving energy. These findings demonstrate the benefits of integrating aspect-oriented frameworks and layering approaches into Agile methodologies, contributing to sustainable software development discourse. The study emphasizes the importance of energy-efficient frameworks like ALAI to reduce software systems' environmental impact and promote software development sustainability. With their practical relevance, this study's findings assist software developers and organizations in choosing software design and development methods that maximize operational efficiency and environmental sustainability.

(Total 91 pages)

Keywords: Agile; aspect-oriented; energy efficiency; flexibility; framework; layering; software development; sustainability.

Student's Signature..... Dissertation Advisor's Signature.....

Dissertation Co-Advisor's Signature.....

สารบัญ

สารบัญ (ต่อ)

สารบัญตาราง

สารบัญรูป

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

เทคโนโลยีดิจิทัลเป็นส่วนสำคัญของสังคมร่วมสมัยส่งผลต่อการดำเนินงานธุรกิจ โดยอาศัยซอฟต์แวร์ทำการควบคุมเทคโนโลยีดิจิทัลเพื่อตอบสนองกระบวนการอุตสาหกรรม การบริการ และการอำนวยความสะดวกในการติดต่อสื่อสารของมนุษย์ การดำเนินธุรกิจต้องการพัฒนาความเป็นผู้นำทางด้านการบริการ การตอบสนองการแข่งขันในสังคมดิจิทัล อีกทั้งการดำเนินการอย่างมีประสิทธิภาพ ความคล่องตัว

นวัตกรรมได้กระตุ้นความต้องการที่สูงขึ้นสำหรับซอฟต์แวร์ประยุกต์ที่ทันสมัยและผสานรวมกัน อย่างไรก็ตาม การเพิ่มขึ้นของความต้องการด้านธุรกิจนี้ก็นำมาซึ่งความท้าทายที่สำคัญด้วย งบประมาณบ่อยครั้งต้องเผชิญกับการกดดัน และความต้องการในการพัฒนาและผสานรวมซอฟต์แวร์ดูเหมือนจะไม่มีที่สิ้นสุด โดยทุกส่วนประกอบต้องการความพิถีพิถันในการพิจารณาความต้องการธุรกิจที่เป็นไปได้ใหม่สามารถครอบคลุมหลายพื้นที่ในการออกแบบและพัฒนาซอฟต์แวร์ ซึ่งรวมถึงความเข้ากันได้ในมิติต่างๆ ตั้งแต่ความซับซ้อนของความเข้ากันได้ระหว่างแพลตฟอร์ม (Platform) ต่างๆ จนถึงความละเอียดของโครงสร้างฐานข้อมูลที่หลากหลาย รวมถึงความซับซ้อนของภาษาโปรแกรมทั้งด้านหน้า (Frontend) และด้านหลัง (Backend) และความซับซ้อนของกระบวนการออกแบบและพัฒนาซอฟต์แวร์ และการยั่งยืน (Camañes, Tobajas, & Fernandez, 2024 ; Oyedeji, Seffah,& Penzenstadler, 2018)

ทุกธุรกิจต้องมีการเปลี่ยนแปลงอย่างหลีกเลี่ยงไม่ได้โดยความต้องการของซอฟต์แวร์ ซึ่งเปลี่ยนแปลงไปตามความการของยุคสมัย ผู้ใช้ และเทคโนโลยี เพราะฉะนั้น การเปลี่ยนแปลงของ

ซอฟต์แวร์จึงจำเป็นต้องมีความสามารถในการปรับตัวและความยืดหยุ่น เพื่อให้สามารถตอบสนองต่อความต้องการและสภาพแวดล้อมที่เปลี่ยนแปลงได้อย่างมีประสิทธิภาพ

ความซับซ้อนที่เพิ่มขึ้นจากผลกระทบที่เกิดจากความต้องการของธุรกิจใหม่ในการพัฒนาซอฟต์แวร์และการผสานรวม ส่งผลกระทบหลายด้านต่อความพยายามของโลกในการพัฒนาอย่างยั่งยืน น่าเสียดายที่ความต้องการในการพัฒนาซอฟต์แวร์ที่เพิ่มขึ้นไม่เพียงแต่ทำให้ฮาร์ดแวร์ใช้พลังงานมากขึ้นและทำให้เกิดการสะสมของขยะอิเล็กทรอนิกส์ (E-waste) เท่านั้น (Reyna, Hanham, & Orlando, 2024) แต่ยังต้องการนักพัฒนาที่มีความเชี่ยวชาญมากขึ้นด้วย (Haputhanthrige, Asghar, Saleem, & Shamim, 2024) การขยายตัวของอุปกรณ์คอมพิวเตอร์ ศูนย์ข้อมูล และเซิร์ฟเวอร์จำเป็นต้องใช้ทรัพยากรพลังงานเพิ่มเติม (Bein, 2018) ซึ่งขัดขวางการบรรลุความเป็นกลางทางคาร์บอนและทำให้การปล่อยก๊าซเรือนกระจกเพิ่มขึ้น การแพร่หลายของคอมพิวเตอร์และอุปกรณ์ดิจิทัลได้นำไปสู่การเพิ่มขึ้นของการใช้พลังงาน โดยเฉพาะอย่างยิ่งการหมดลงของแบตเตอรี่ (Kuaban et al., 2023) และการใช้งาน CPU (Rukhiran, Boonsong, & Netinant, 2024) ซึ่งส่งผลกระทบในทางลบต่อผู้ใช้และระบบโครงสร้างพื้นฐาน การปรับเปลี่ยนและทำซ้ำโซลูชันซอฟต์แวร์อย่างต่อเนื่องต้องใช้ทรัพยากรการคำนวณจำนวนมาก ซึ่งทำให้การใช้พลังงานเพิ่มขึ้นตลอดวงจรชีวิตการพัฒนาซอฟต์แวร์ แนวโน้มนี้เป็นเรื่องที่น่ากังวลอย่างยิ่งและต้องการการดำเนินการทันทีท่ามกลางแนวโน้มนี้ ทุกคนสามารถปรับปรุงความยั่งยืนในการใช้พลังงานได้อย่างมาก และรับประกันประสิทธิภาพการใช้พลังงานที่สูงขึ้น (Sriraman, & Raghunathan, 2023; Fagarasan, Cristea, Popa, & Pisla, 2023) โดยมุ่งเน้นไปที่การพัฒนารูปแบบแนวคิดและนวัตกรรมการออกแบบของโซลูชันการพัฒนาซอฟต์แวร์ใหม่ ๆ ที่ยืดหยุ่นต่อความต้องการทางธุรกิจที่ขยายตัวและสามารถปรับขนาดได้ตามความจำเป็น ถึงเวลาแล้วที่จะต้องมีการเปลี่ยนแปลง

งานวิจัยก่อนหน้านี้ได้สำรวจแนวทางแก้ไขด้านประสิทธิภาพการใช้พลังงานสำหรับการออกแบบฮาร์ดแวร์และซอฟต์แวร์ในหลายภาคส่วนอย่างกว้างขวาง รวมถึงทรัพยากรคำนวณที่ใช้ในซอฟต์แวร์ (Ciancarini et al., 2020; García-Berná et al., 2021) โมเดลซอฟต์แวร์ (Fagarasan et al., 2023; Alrabaiah, & Medina-Medina, 2021) อินเตอร์เฟสผู้ใช้กราฟิก (GUI) (García-Berná et al., 2021) ภาษาโปรแกรม (Manimegalai, Sandhanam, Nandhini, & Pandia, 2023) การเข้ารหัส (Vračar, Stojanović, Stanimirović, & Prijić, 2019) ความสามารถในการคำนวณแบบคลาวด์ (Donca, Stan,

Misaros, Gota, & Miclea, 2022) และความปลอดภัย (Rukhiran, Boonsong, & Netinant, 2024) โดยการแก้ไขข้อบกพร่องเหล่านี้และการสร้างกลยุทธ์ในการปรับใช้ทรัพยากรอย่างมีประสิทธิภาพและส่งเสริมการผสานของแหล่งพลังงานที่ได้โดยสามารถเปิดทางสู่อนาคตที่ยั่งยืนมากขึ้นได้ การกระทำเช่นนี้ไม่เพียงแต่รักษาความเป็นผู้นำในด้านนวัตกรรมในโดเมนดิจิทัล แต่ยังส่งเสริมความหวังและความมั่นใจในการสร้างโลกที่เป็นมิตรกับสิ่งแวดล้อมมากขึ้นอีกด้วย

การศึกษานี้นำเสนอโครงร่าง Agile-Layering-Aspect-Oriented (ALAI) เป็นวิธีการที่โดดเด่นและไม่เคยถูกออกแบบมาก่อน โครงร่างนี้ที่เป็นนวัตกรรมมีเป้าหมายเพื่อเปลี่ยนแปลงการผสมผสานของวิธีการ Agile กับความแข็งแกร่งของการจัดชั้นและความสามารถในปรับเปลี่ยนของโครงร่างแบบมุมมอง (AOF) อย่างยืดหยุ่นและสามารถปรับได้ โครงร่าง ALAI มีศักยภาพที่เพิ่มประสิทธิภาพของระบบซอฟต์แวร์อย่างมีนัยสำคัญและยืดหยุ่นทางสถาปัตยกรรม ซึ่งสนับสนุนในการสร้างโซลูชันที่มีประสิทธิภาพทางพลังงานและยืดหยุ่น และการพัฒนาซอฟต์แวร์ที่เป็นมิตรต่อสิ่งแวดล้อม เหมาะอย่างยิ่งกับความต้องการที่เปลี่ยนแปลงและหลากหลายขององค์กรอิเล็กทรอนิกส์ที่ทันสมัย ซึ่งต้องการโซลูชันซอฟต์แวร์ที่สามารถขยายตัวได้ แข็งแรง และยั่งยืน

การศึกษานี้จึงไม่เพียงแต่เติมเต็มช่องว่างในความรู้ที่มีอยู่ แต่ยังให้ข้อมูลที่มีค่าซึ่งสามารถนำไปใช้ประโยชน์จริงในทางปฏิบัติ โดยช่วยให้ผู้ที่มีส่วนเกี่ยวข้องสามารถทำการตัดสินใจที่ดียิ่งขึ้นเกี่ยวกับการพัฒนาและการใช้ซอฟต์แวร์ในอนาคต เพื่อให้สอดคล้องกับเป้าหมายด้านความยั่งยืนและการจัดการทรัพยากรอย่างมีประสิทธิภาพเช่น ALAI สามารถสนับสนุนการสร้างสถาปัตยกรรมซอฟต์แวร์ที่มีประสิทธิภาพทางพลังงานและยืดหยุ่นได้อย่างไร โดยนำเสนอการวิเคราะห์อย่างละเอียดเกี่ยวกับประสิทธิภาพทางพลังงานของโครงร่าง ALAI ที่ถูกขยายโดย Agile ซึ่งเป็นการก้าวหน้าของการพัฒนาซอฟต์แวร์ที่ยั่งยืน การสาธิตประโยชน์ทางปฏิบัติของการรวมความยั่งยืนและความยืดหยุ่นในกระบวนการพัฒนาซอฟต์แวร์ (SDLC) ให้เห็นถึงความสำคัญของการเปลี่ยนแปลงแนวทางการดำเนินงานของสถาบันการศึกษาให้เป็นไปในทิศทางที่มีความยั่งยืนมากขึ้น และให้แบบจำลองที่สามารถปรับใช้ได้สำหรับการประยุกต์ใช้ในอุตสาหกรรมทั่วไปได้

งานวิจัยนี้มุ่งหวังเพื่อศึกษาข้อมูลเชิงประจักษ์เกี่ยวกับประสิทธิภาพของโครงร่าง ALAI ในการส่งเสริมการปฏิบัติการในการพัฒนาซอฟต์แวร์ที่ยั่งยืนและยืดหยุ่นมากขึ้น โดยมุ่งเน้นไปที่

ประสิทธิภาพทางพลังงานและความยืดหยุ่นทางสถาปัตยกรรม วิธีการและขั้นตอนการทำงานภายในโครงร่าง ALAI เช่น การรวมระบบและการส่งมอบอย่างต่อเนื่อง เสริมความสามารถในการพัฒนาซอฟต์แวร์แบบ Agile เพื่อประสบความสำเร็จในการยั่งยืนทางด้านพลังงานให้เกิดประสิทธิภาพทางพลังงาน เช่นโครงร่าง ALAI ส่งเสริมการใช้เครื่องมือการทดสอบอัตโนมัติเพื่อตรวจจับและแก้ไขปัญหาความไม่มีประสิทธิภาพทางพลังงานในรหัสขณะกำลังพัฒนาโปรแกรม

1.2 วัตถุประสงค์ของการวิจัย

ในการทำวิจัยครั้งนี้ได้กำหนดหัวข้อวัตถุประสงค์การวิจัยที่เหมาะสมเพื่อให้งานวิจัยเป็นไปตามเป้าหมายที่ผู้วิจัยได้กำหนด ทั้งสิ้น 3 ข้อ ดังนี้

1.1.1 เพื่อออกแบบสถาปัตยกรรมซอฟต์แวร์ที่ยั่งยืน ยืดขยาย ปรับเปลี่ยนและสนับสนุนการประหยัดพลังงาน

1.1.2 เพื่อพัฒนาสถาปัตยกรรมซอฟต์แวร์ที่ยั่งยืนและสามารถนำไปใช้ในเชิงปฏิบัติการได้อย่างมีประสิทธิภาพ มีการออกแบบที่มีความยืดหยุ่นและปรับตัวได้ดีต่อการเปลี่ยนแปลง รวมถึงสามารถรองรับการขยายตัวของการให้บริการ โดยเฉพาะเมื่อใช้กระบวนการพัฒนาแบบ Agile ซึ่งเป็นแนวทางที่เน้นการปรับตัวอย่างรวดเร็วและการทำงานร่วมกันอย่างใกล้ชิด

1.1.3 เพื่อทดสอบของกรอบโครงร่างบนพื้นฐานการผสมผสานการพัฒนาแบบ Agile เข้ากับการทำงานแบบลำดับชั้น (Layers) และการออกแบบพัฒนากรอบโครงร่างเชิงลักษณะ (Aspect-Oriented Framework ใช้คำย่อว่า ALAI) ที่มีผลต่อความยั่งยืนเชิงปฏิบัติทางสถาปัตยกรรมเพื่อการประหยัดพลังงาน

1.3 ขอบเขตการดำเนินการ

ผู้วิจัยได้ทำการกำหนดขอบเขตของการออกแบบสถาปัตยกรรมซอฟต์แวร์ ซึ่งประกอบไปด้วยรายละเอียด ดังนี้

1.3.1 ศึกษางานวิจัยต่าง ๆ ที่เกี่ยวข้องในหัวข้อ

1.3.1.1 แนวความคิดของ DevOps

1.3.1.2 Layered Software Architecture โครงสร้างซอฟต์แวร์แบบชั้น

1.3.1.3 Aspect-Oriented Framework (AOF) and Flexibility กรอบโครงร่าง คือการพัฒนาแบบใหม่ถูกนำเสนอเมื่อปี 2022 (Netinant, Rukhiran, & Soongpol, 2022)

1.3.2 ทำการออกแบบกรอบโครงร่างสถาปัตยกรรมซอฟต์แวร์ที่ยั่งยืนสามารถนำไปใช้เชิงปฏิบัติการมีความสามารถยืดหยุ่นและปรับตัวทางสถาปัตยกรรมเพื่อตอบสนองความต่อการเปลี่ยนแปลงและเพิ่มเติมการให้การบริการ

1.3.3 การพัฒนาสถาปัตยกรรมซอฟต์แวร์ที่ยั่งยืนสามารถนำไปใช้เชิงปฏิบัติการมีความสามารถยืดหยุ่นและปรับตัวทางสถาปัตยกรรมเพื่อตอบสนองความต่อการเปลี่ยนแปลงและเพิ่มเติมการให้การบริการมาใช้ในการพัฒนาระบบข้อมูลการรับสมัครเข้าศึกษาต่อระดับบัณฑิตศึกษา Graduate Admission Information System Services (GAISS)

1.3.4 ทำการพัฒนาการขยายกระบวนการการทำงานกรอบโครงร่างบนพื้นฐานการผสมผสานการพัฒนาแบบ Agile เข้ากับการทำงานแบบลำดับชั้น (Layers) และการออกแบบพัฒนากรอบโครงร่างเชิงลักษณะ (Aspect-Oriented Framework ใช้คำย่อว่า ALAI) ที่มีผลต่อความยั่งยืนเชิงปฏิบัติทางสถาปัตยกรรมเพื่อการประหยัดพลังงาน

1.3.5 การพัฒนาระบบซอฟต์แวร์ที่ยั่งยืนสามารถนำไปใช้เชิงปฏิบัติการมีความสามารถยืดหยุ่นและปรับตัวทางสถาปัตยกรรมเพื่อตอบสนองความต่อการเปลี่ยนแปลงและเพิ่มเติมการให้การบริการมาใช้ในการพัฒนาระบบข้อมูลการรับสมัครเข้าศึกษาต่อระดับบัณฑิตศึกษา Graduate Admission Information System Services (GAISS) โดยใช้งานพัฒนาแบบ Agile เปรียบเทียบกับ ALAI

1.3.6 สรุปผลการเปรียบเทียบการพัฒนาระบบรับสมัครเข้าศึกษาต่อระดับบัณฑิตศึกษา Graduate Admission Information System Services (GAISS) ด้านการยืดหยุ่น การปรับตัวและการ

ประหยัดพลังงาน ของสถาปัตยกรรมที่มีผลต่อการใช้รูปแบบพัฒนาแบบ Agile เปรียบเทียบกับ ALAI

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1.4.1 เกิดการลดภาระงานทางด้านการบริหารงานบุคลากรและลดปัญหาข้อผิดพลาดที่เกิดขึ้นบ่อยครั้งจากการประมวลผลข้อมูลด้วยวิธีการดำเนินการด้วยมือในช่วงแรก

1.4.2 สามารถเพิ่มประสิทธิภาพของการดำเนินงานและความสามารถในการรับสมัครนักศึกษา

1.4.3 ได้สถาปัตยกรรมซอฟต์แวร์ที่ยั่งยืน ยืดขยาย ปรับเปลี่ยนและสนับสนุนการประหยัดพลังงาน

1.4.4 สถาปัตยกรรมซอฟต์แวร์ที่ยั่งยืนสามารถนำไปใช้เชิงปฏิบัติการมีความสามารถยืดหยุ่นและปรับตัวทางสถาปัตยกรรมเพื่อตอบสนองความต่อการเปลี่ยนแปลงและเพิ่มเติมการให้การบริการ

1.4.5 กระบวนการพัฒนาซอฟต์แวร์โดยใช้กรอบโครงร่างบนพื้นฐานการผสมผสานการพัฒนาแบบ Agile เข้ากับการทำงานแบบลำดับชั้น (Layers) และการออกแบบพัฒนากรอบโครงร่างเชิงลักษณะ ALAI

1.5 นิยามศัพท์เฉพาะ

สถาปัตยกรรมซอฟต์แวร์ที่ยั่งยืน (Sustainable Software Architecture) คือแนวทางในการออกแบบและพัฒนาซอฟต์แวร์โดยคำนึงถึงความยั่งยืนในหลายด้าน ไม่ว่าจะเป็นความสามารถในการบำรุงรักษา, ความสามารถในการปรับขยาย, ประสิทธิภาพการทำงาน, และการลดผลกระทบต่อสิ่งแวดล้อม

การออกแบบพัฒนากรอบโครงร่างเชิงลักษณะ (Aspect-Oriented Framework ใช้คำย่อว่า ALAI) (Aspect-Oriented Framework Design and Development) เป็นแนวทางการพัฒนาซอฟต์แวร์ที่เน้นการแยกความกังวล (Concerns) ที่ข้ามส่วนของระบบออกมาเป็นลักษณะ (Aspects) เพื่อทำให้การจัดความซับซ้อนได้ง่ายขึ้น และเพิ่มความสามารถในการบำรุงรักษาและขยายระบบได้อย่างมีประสิทธิภาพ ทำให้สามารถตอบสนองต่อความต้องการของผู้ใช้และการเปลี่ยนแปลงในอนาคตได้อย่างมีประสิทธิภาพ

การพัฒนาแบบ Agile เป็นแนวทางการพัฒนาซอฟต์แวร์ที่เน้นการตอบสนองอย่างรวดเร็วต่อการเปลี่ยนแปลงและการทำงานร่วมกันระหว่างทีมพัฒนาและลูกค้า เป้าหมายหลักของ Agile คือ การส่งมอบซอฟต์แวร์ที่มีคุณค่าและมีคุณภาพสูงในระยะเวลาที่สั้นที่สุด และปรับตัวตามความต้องการที่เปลี่ยนแปลงไป

โครงสร้างซอฟต์แวร์แบบชั้น (Layered Software Architecture) เป็นหนึ่งในรูปแบบสถาปัตยกรรมซอฟต์แวร์ที่นิยมใช้มากที่สุด โดยแบ่งส่วนของซอฟต์แวร์ออกเป็นชั้นต่างๆ ซึ่งแต่ละชั้นมีหน้าที่เฉพาะและทำงานร่วมกันอย่างเป็นลำดับชั้น โครงสร้างนี้ช่วยให้การพัฒนา, บำรุงรักษา, และการขยายระบบมีความง่ายขึ้น

บทที่ 2

ทบทวนวรรณกรรมที่เกี่ยวข้อง / ทฤษฎีที่เกี่ยวข้อง

2.1 แนวโน้มปัจจุบันในการพัฒนาซอฟต์แวร์เทคโนโลยี

การพัฒนาซอฟต์แวร์เป็นอุตสาหกรรมที่กำลังเปลี่ยนแปลงอย่างรวดเร็วเนื่องจากความก้าวหน้าทางเทคโนโลยี การเปลี่ยนแปลงของแนวโน้มตลาด และความคาดหวังของผู้บริโภคที่เพิ่มมากขึ้น (Huang et al., 2022) ทีมพัฒนาได้นำเอาวิธีการ Agile มาใช้มากขึ้น โดยเน้นความยืดหยุ่น การปรับปรุงอย่างต่อเนื่อง และการส่งมอบอย่างรวดเร็ว แนวทางนี้ช่วยให้องค์กรสามารถตอบสนองต่อการเปลี่ยนแปลงในตลาดและข้อมูลจากลูกค้าได้โดยมีประสิทธิภาพมากขึ้น (Ozdenizci Kose, 2021) วิธีการ Agile ถูกนำไปใช้ในหลายด้านของวิศวกรรมซอฟต์แวร์ รวมถึงการสร้างกระบวนการสำหรับการรวมระบบอย่างต่อเนื่องและการส่งมอบอย่างต่อเนื่อง (CI/CD) (Donca et al., 2022 ; Erdenebat, Bud, Batsuren, & Kozsik, 2023) วิธีการเหล่านี้มีประโยชน์ในการปรับปรุงกระบวนการและสร้างความมั่นใจให้แก่นักพัฒนา โดยกระตุ้นความจินตนาการของพวกเขาด้วยความเป็นไปได้ในการประสบความสำเร็จ แนวความคิดในการออกแบบด้วยวิธี Agile เน้นไปที่การพัฒนาซอฟต์แวร์อย่างรวดเร็วและยืดหยุ่น เพื่อให้สามารถปรับตัวได้ตามความเปลี่ยนแปลงที่เกิดขึ้นในระหว่างการพัฒนา วิธีการนี้มุ่งเน้นไปที่การทำงานเป็นทีม เช่น การทำงานร่วมกันระหว่างนักพัฒนาและลูกค้า การสร้างซอฟต์แวร์แต่ละส่วนให้สมบูรณ์และพร้อมใช้งานตามลำดับ การส่งมอบซอฟต์แวร์ที่ใช้งานได้จริงในระยะเวลาสั้น และการรับฟังและปรับปรุงซอฟต์แวร์ตามความต้องการของลูกค้าและการตอบสนองต่อการเปลี่ยนแปลงในข้อกำหนดการทำงาน วิธีการ Agile ช่วยให้การพัฒนาซอฟต์แวร์เป็นไปอย่างรวดเร็ว ยืดหยุ่น และมีความเป็นมาตรฐานสูง โดยที่ยังสามารถตอบสนองต่อความต้องการและข้อกำหนดต่างๆ ของลูกค้าได้อย่างมีประสิทธิภาพ แต่อย่างไรก็ตาม Agile ก็ยังมีข้อจำกัดอยู่ 1) ความซับซ้อนของโครงการ: Agile มักเหมาะกับโครงการที่มีขอบเขตไม่

ใหญ่มาก หากโครงการมีความซับซ้อนมากเกินไปอาจทำให้การจัดการและการสื่อสารกับทีมมีความยากลำบากมากขึ้น 2) การเปลี่ยนแปลงความต้องการ: Agile สามารถให้ความยืดหยุ่นในการปรับเปลี่ยนความต้องการได้ แต่หากมีการเปลี่ยนแปลงความต้องการบ่อยเกินไปอาจทำให้เกิดความไม่เสถียรในการพัฒนา 3) การสื่อสาร: การสื่อสารระหว่างทีมพัฒนาและผู้ใช้เป็นสิ่งสำคัญใน Agile หากมีปัญหาในการสื่อสารอาจทำให้เกิดความเข้าใจผิดพลาดและผลลัพธ์ที่ไม่ถูกต้อง 4) การฝึกฝนและปรับเปลี่ยน: การทำ Agile ต้องการความเปลี่ยนแปลงในวิธีการทำงานและวัฒนธรรมองค์กร การที่ทีมและองค์กรยอมรับและปรับตัวตามวิธีการใหม่อาจเป็นอุปสรรคสำหรับบางท 5) ความต้องการในทีมที่มีความสามารถ: Agile ต้องการทีมที่มีความสามารถในการทำงานร่วมกันอย่างราบรื่น หากมีสมาชิกในทีมที่ไม่เข้าใจหรือไม่ยอมรับวิธีการนี้อาจทำให้โครงการไม่ประสบความสำเร็จ แล้วในส่วนแนวความคิดสำคัญของ Agile ประกอบด้วย

1) ความยืดหยุ่น (Flexibility): Agile ให้ความสำคัญกับการปรับเปลี่ยนแผนการทำงานและผลิตภัณฑ์ตามความต้องการของลูกค้าหรือการเปลี่ยนแปลงที่เกิดขึ้นระหว่างโครงการ ทำให้สามารถทำงานได้อย่างมีประสิทธิภาพในสภาวะที่มีการเปลี่ยนแปลงบ่อย ๆ

2) ความยืดหยุ่น (Flexibility): Agile ให้ความสำคัญกับการปรับเปลี่ยนแผนการทำงานและผลิตภัณฑ์ตามความต้องการของลูกค้าหรือการเปลี่ยนแปลงที่เกิดขึ้นระหว่างโครงการ ทำให้สามารถทำงานได้อย่างมีประสิทธิภาพในสภาวะที่มีการเปลี่ยนแปลงบ่อย ๆ

3) การส่งมอบและตอบสนองลูกค้า (Customer Collaboration and Responsiveness): Agile ส่งเสริมการทำงานร่วมกับลูกค้าอย่างใกล้ชิด โดยการเป็นพร้อมที่จะปรับตัวตามความต้องการของลูกค้าและการตอบสนองต่อข้อเสนอแนะของลูกค้า

4) การพัฒนาผ่านทีม (Team-based Development): Agile ให้ความสำคัญกับการทำงานร่วมกันของทีม การสนับสนุนและกระตุ้นการทำงานร่วมกันเพื่อให้สามารถสร้างผลิตภัณฑ์ที่มีคุณภาพได้

5) การสร้างคุณค่า (Value-driven): Agile ให้ความสำคัญกับการสร้างผลิตภัณฑ์ที่มีคุณค่าสูงสุดให้แก่ลูกค้า โดยการให้ความสำคัญกับการพัฒนาฟีเจอร์ที่สำคัญและมีประสิทธิภาพ

6) การปรับปรุงต่อเนื่อง (Continuous Improvement): Agile สนับสนุนกระบวนการทำงานที่สามารถปรับปรุงต่อเนื่องได้ โดยการทำการประเมินและปรับปรุงวิธีการทำงานเพื่อสร้างผลิตภัณฑ์ที่มีคุณภาพและเป็นไปตามความต้องการของลูกค้าได้มากขึ้น

อีกแนวโน้มที่สำคัญในการพัฒนาซอฟต์แวร์คือการเคลื่อนไหวไปสู่โครงสร้างที่ซับซ้อนมากขึ้น เช่น ไมโครเซอร์วิส (Microservices) (Waseem, Liang, & Shahin, 2020) และการคำนึงถึงเซิร์ฟเวอร์ที่ไม่ต้องมีเซิร์ฟเวอร์ (Serverless Computing) (Hassan, Barakat, & Sarhan, 2021) ซึ่งเป็นโครงสร้างที่ช่วยให้สามารถพัฒนาและปรับปรุงซอฟต์แวร์ได้อย่างยืดหยุ่นและมีประสิทธิภาพสูงขึ้น การออกแบบที่ซับซ้อนเช่นนี้ด้วยความยืดหยุ่นและปรับปรุงได้สูงกว่าการออกแบบแบบโมโนลิธิก ไม่ได้เป็นแค่แนวโน้มที่ผ่านมา หลักการนี้เป็นเรื่องที่มีความสำคัญและกำลังเจริญเติบโตอย่างต่อเนื่องในอุตสาหกรรมซอฟต์แวร์ การใช้งานอย่างแพร่หลายของโครงสร้างแบบไมโครเซอร์วิสและคอมพิวติ้งแบบไม่ต้องมีเซิร์ฟเวอร์ได้แสดงให้เห็นถึงประสิทธิภาพและความยืดหยุ่นที่ยิ่งใหญ่ของมันอย่างชัดเจน การใช้งานโครงสร้างเหล่านี้เป็นเครื่องมือที่ทำให้นักพัฒนามีความสามารถและเสรีในการทำงาน ทำให้พวกเขาสามารถมุ่งเน้นไปที่ด้านนวัตกรรมและด้านธุรกิจที่สำคัญของการพัฒนาแอปพลิเคชันได้อย่างอิสระ การเปลี่ยนแปลงนี้กำลังสร้างความตื่นเต้นให้กับโอกาสที่อยู่ในผลงานของพวกเขา

การพัฒนาซอฟต์แวร์กำลังเปลี่ยนแปลงอย่างต่อเนื่อง ทำให้จำเป็นต้องทำการประเมินใหม่เกี่ยวกับกระบวนการสร้างและส่งมอบซอฟต์แวร์ (Waseem et al., 2020 ; Sadowski, & Zimmermann, 2019) หนึ่งในตัวอย่างคือ DevOps ซึ่งรวมการดำเนินงานและการพัฒนาซอฟต์แวร์เข้าด้วยกันเพื่อลดรอบชีวิตการพัฒนาในขณะที่ยังรักษาคุณภาพและความเชื่อถือได้ (Erdenebat et al., 2023 ; Subramanya, Sierla, & Vyatkin, 2022) การพัฒนาที่สำคัญอีกอย่างคือการรวมการประยุกต์ใช้ปัญญาประดิษฐ์ (AI) และเรียนรู้ของเครื่อง (ML) เข้ากับการพัฒนาซอฟต์แวร์ การรวมกันนี้กำลังเป็นทางลัดสู่แอปพลิเคชันที่มีความฉลาดมากขึ้น (Karamitsos, Albarhami, & Apostolopoulos, 2020) ที่สามารถทำให้การตัดสินใจอัตโนมัติและการคาดการณ์พฤติกรรมของผู้ใช้ได้ โดยทำให้ประสิทธิภาพในการดำเนินงานและประสบการณ์ของผู้ใช้ดียิ่งขึ้น แนวความคิดของ DevOps เกิดจากความต้องการให้การพัฒนาและการดำเนินงานซอฟต์แวร์เป็นไปอย่างรวดเร็ว มีประสิทธิภาพและมีคุณภาพสูง โดย DevOps ได้มุ่งเน้นไปที่การรวมกลุ่มระหว่างทีมการพัฒนาซอฟต์แวร์ (Development) และทีมดำเนินงานระบบ (Operations) เข้าด้วยกัน ซึ่งส่งผลให้มีการพัฒนาและการส่งมอบซอฟต์แวร์ที่เร็วขึ้น มีความน่าเชื่อถือสูงขึ้น และสามารถปรับปรุงและปรับแต่งได้อย่างต่อเนื่องตามความต้องการของผู้ใช้ โดยมีหลักการและวิธีการหลัก ๆ ดังต่อไปนี้

1) Automation (อัตโนมัติ): การใช้เครื่องมือและกระบวนการอัตโนมัติเพื่อลดเวลาและความผิดพลาดในการสร้าง ทดสอบ และส่งมอบซอฟต์แวร์

2) Collaboration (การร่วมมือ): การสร้างการทำงานร่วมกันระหว่างทีมพัฒนาและทีมดำเนินงานระบบ เพื่อให้ทราบถึงความต้องการและปัญหาที่เกิดขึ้นในขณะที่พัฒนา

3) Continuous Integration (การรวมระบบแบบต่อเนื่อง): การรวมการเปลี่ยนแปลงของโค้ดใหม่ในระบบอย่างสม่ำเสมอ เพื่อให้ทราบถึงข้อผิดพลาดและปัญหาที่เกิดขึ้นได้ทันที

4) Continuous Delivery (การส่งมอบแบบต่อเนื่อง): การสร้างกระบวนการที่ทำให้สามารถส่งมอบซอฟต์แวร์ได้ทันทีที่มีความพร้อม โดยลดระยะเวลาในการพัฒนาและการส่งมอบ

5) Monitoring and Feedback (การตรวจสอบและการตอบรับ): การติดตามและวิเคราะห์การใช้งานของซอฟต์แวร์ เพื่อทราบถึงปัญหาและข้อผิดพลาด และนำความรู้นี้ไปปรับปรุงสิ่งที่มีปัญหาต่อไป

ในกระบวนการ DevOps แต่ละขั้นตอนมีความสำคัญและมีบทบาทเฉพาะตัวในการเพิ่มประสิทธิภาพการพัฒนาและส่งมอบซอฟต์แวร์ นี่คือรายละเอียดของแต่ละขั้นตอน

1) Plan (วางแผน):

1.1) กำหนดเป้าหมายและวัตถุประสงค์ของโครงการ

1.2) สร้างแผนงานและตารางเวลาสำหรับการพัฒนา

1.3) ประชุมทีมเพื่อให้เข้าใจถึงบทบาทและหน้าที่ของแต่ละคน

2) Create (สร้าง):

2.1) เขียนโค้ดตามที่วางแผนไว้

2.2) ใช้เครื่องมือจัดการเวอร์ชัน (เช่น Git) เพื่อจัดเก็บและควบคุมโค้ด

2.3) พัฒนาโค้ดร่วมกันระหว่างทีมพัฒนา โดยเน้นการสื่อสารและการทำงานร่วมกัน

3) Verify (ตรวจสอบ):

3.1) ทดสอบโค้ดที่พัฒนาเพื่อให้แน่ใจว่าไม่มีข้อผิดพลาด

3.2) ใช้การทดสอบอัตโนมัติ (Automated Testing) เพื่อตรวจสอบความถูกต้องและคุณภาพของโค้ด

3.3) ทดสอบแบบต่อเนื่อง (Continuous Integration) เพื่อตรวจสอบการรวมระบบอย่างสม่ำเสมอ

4) Package (แพ็กเกจ):

4.1) สร้างแพ็กเกจซอฟต์แวร์ที่พร้อมสำหรับการส่งมอบ

4.2) ใช้เครื่องมือในการสร้างและแพ็กเกจซอฟต์แวร์ (เช่น Docker) เพื่อให้การปรับใช้ (Deployment) ทำได้ง่ายขึ้น

4.3) ตรวจสอบแพ็กเกจเพื่อให้แน่ใจว่าพร้อมสำหรับการใช้งานในสภาพแวดล้อมการผลิต (Production)

5) Configure (กำหนดค่า):

5.1) กำหนดค่าและปรับแต่งสภาพแวดล้อมการทำงานให้ตรงกับความต้องการของซอฟต์แวร์

5.2) ใช้เครื่องมือจัดการการกำหนดค่า (เช่น Ansible, Chef, Puppet) เพื่อให้แน่ใจว่าสภาพแวดล้อมที่ใช้ตรงกับการตั้งค่าที่ต้องการ

6) Release (เผยแพร่):

6.1) ส่งมอบซอฟต์แวร์ไปยังสภาพแวดล้อมการผลิต

6.2) ใช้การส่งมอบแบบต่อเนื่อง (Continuous Delivery/Continuous Deployment) เพื่อให้การปล่อยซอฟต์แวร์ทำได้อย่างรวดเร็วและปลอดภัย

6.3) ติดตามและตรวจสอบการเผยแพร่เพื่อให้แน่ใจว่าไม่มีปัญหาหรือข้อผิดพลาด

7) Monitor (ตรวจสอบ):

7.1) ติดตามและตรวจสอบการทำงานของซอฟต์แวร์ในสภาพแวดล้อมการผลิต

7.2) ใช้เครื่องมือตรวจสอบ (เช่น Prometheus, Grafana, ELK Stack) เพื่อวิเคราะห์และรายงานประสิทธิภาพและความเสถียรของระบบ

7.3) รวบรวมข้อมูลการใช้งานและข้อผิดพลาดเพื่อปรับปรุงและปรับแต่งซอฟต์แวร์ในรอบถัดไป

ขั้นตอนเหล่านี้ช่วยให้ทีมพัฒนาและทีมดำเนินงานสามารถทำงานร่วมกันอย่างมีประสิทธิภาพและส่งมอบซอฟต์แวร์ที่มีคุณภาพสูงสุดไปยังผู้ใช้อย่างรวดเร็วและมีเสถียรภาพ

อย่างไรก็ตาม การพัฒนาเหล่านี้ก็นำมาซึ่งความซับซ้อนที่เพิ่มขึ้น เช่น การแบ่งส่วนและการกระจายของสถาปัตยกรรมระบบ (เช่น ไมโครเซอร์วิส (Waseem et al., 2020) ความต้องการในการเรียนรู้และการปรับตัวอย่างต่อเนื่อง (Rukhiran, & Netinant, 2020) และประสิทธิภาพในการ

สื่อสารและการทำงานร่วมกันของทีม (Dustdar, & Gall, 2003). แม้ว่าจะมีอุปสรรคที่จับต้องได้เหล่านี้ แต่ก็ไม่ได้หมายความว่าจะไม่มีความหวัง การยึดมั่นในหลักการเหล่านี้ในขณะที่รักษาคุณภาพ ความพึงพอใจของผู้ใช้ และการสอดคล้องกับกลยุทธ์เป็นสิ่งจำเป็นสำหรับการบรรลุความสำเร็จในความสามารถด้านซอฟต์แวร์ที่เปลี่ยนแปลงตลอดเวลา (Pashutan, Abdolvand, & Harandi, 2022) ความมั่นใจนี้ช่วยเพิ่มความสามารถในการจัดการกับความซับซ้อนเหล่านี้อย่างมีประสิทธิภาพ.

ตารางที่ 2.1 แสดงรายละเอียดเกี่ยวกับความก้าวหน้าของเทคโนโลยีการออกแบบและพัฒนาซอฟต์แวร์ โดยเน้นถึงนวัตกรรมแต่ละอย่างที่มีส่วนทำให้ระบบสารสนเทศในปัจจุบันอยู่ในสภาพเช่นนี้ นี่เป็นทรัพยากรที่มีคุณค่า ซึ่งช่วยอธิบายภูมิหลังทางประวัติศาสตร์ของสาขาวิชาของโดยการนำเสนอภาพรวมที่ครอบคลุมเกี่ยวกับเทคโนโลยีสำคัญ ๆ และผลกระทบของเทคโนโลยีเหล่านั้นต่อการพัฒนาซอฟต์แวร์

ตารางที่ 2.1 รายละเอียดเกี่ยวกับความก้าวหน้าของเทคโนโลยีการออกแบบและพัฒนาซอฟต์แวร์

ความก้าวหน้า	คำอธิบาย	ผลกระทบ
Modular programming	แนวทางการพัฒนาระบบในยุคแรก ๆ มุ่งเน้นไปที่การแบ่งซอฟต์แวร์ออกเป็นโมดูลแยกต่างหากที่สามารถสลับเปลี่ยนได้ โดยแต่ละโมดูลจะจัดการกับงานย่อยเฉพาะอย่างหนึ่ง	การแยกส่วนฟังก์ชันที่สามารถแก้ไขได้และเพิ่มประสิทธิภาพในการแก้บั๊กได้รับการปรับปรุงโดยการแยกแยะส่วนที่สำคัญของฟังก์ชัน ซึ่งเป็นการเตรียมพื้นฐานสำหรับสถาปัตยกรรมระบบที่ซับซ้อนมากขึ้น

ตารางที่ 2.1 รายละเอียดเกี่ยวกับความก้าวหน้าของเทคโนโลยีการออกแบบและพัฒนาซอฟต์แวร์ (ต่อ)

ความก้าวหน้า	คำอธิบาย	ผลกระทบ
Object-oriented Programming (OOP) (Lavy, & Rami, 2018)	การนำเสนอแนวคิดของวัตถุในการเขียนโปรแกรม ทำให้สามารถซ่อนข้อมูลและฟังก์ชันไว้ในรูปแบบของส่วนประกอบที่ใช้ซ้ำได้	การเพิ่มความสามารถในการใช้โค้ดซ้ำและการเปลี่ยนขนาด (Scalability) ทำให้เกิดระบบที่ซับซ้อนและสามารถจัดการได้ง่ายขึ้น ซึ่งทำให้สามารถพัฒนาระบบได้ด้วยประสิทธิภาพมากขึ้นและมีประสิทธิภาพมากยิ่งขึ้น
Service-oriented Architecture (SOA) (Zein, 2024)	รูปแบบสถาปัตยกรรมที่อนุญาติให้บริการสื่อสารกันผ่านเครือข่ายผ่านโปรโตคอลเพื่อดำเนินการฟังก์ชันสำหรับกันและกัน	การส่งเสริมให้เกิดการพัฒนาระบบแบบกระจายได้ง่ายขึ้น ทำให้สามารถสานหรือแทนที่ส่วนประกอบได้โดยง่าย ซึ่งทำให้ความยืดหยุ่นและความสามารถในการทำงานร่วมกันของระบบเพิ่มขึ้น
Cloud computing and SaaS (Gupta, Fernandez-Crehuet, & Gupta, 2022)	แพลตฟอร์มคลาวด์และโมเดลซอฟต์แวร์เป็นบริการ (SaaS) ปรากฏขึ้น ซึ่งมอบทรัพยากรและแอปพลิเคชันที่สามารถขยายได้ผ่านอินเทอร์เน็ต	ช่วยให้ระบบการศึกษาสามารถขยายทรัพยากรตามความต้องการได้ เพิ่มประสิทธิภาพในการใช้งานพื้นฐาน และเพิ่มความสามารถในการทำงานร่วมกันและการเข้าถึงข้อมูลจากทุกที่ได้อย่างสะดวก

ตารางที่ 2.1 รายละเอียดเกี่ยวกับความก้าวหน้าของเทคโนโลยีการออกแบบและพัฒนาซอฟต์แวร์ (ต่อ)

ความก้าวหน้า	คำอธิบาย	ผลกระทบ
Microservices architecture (Söylemez, Tekinerdogan, & Kolukısa Tarhan, 2022)	สไตล์สถาปัตยกรรมที่จัดโครงสร้างแอปพลิเคชันให้เป็นชุดของบริการที่แยกตัวออกมาอย่างเหลือเชื่อมต่อให้สัมพันธ์กันน้อยลง ซึ่งเพิ่มความยืดหยุ่นในการพัฒนาแบบโมดูล	ส่งเสริมให้เกิดการพัฒนาระบบที่ง่ายต่อการเข้าใจ การพัฒนา การทดสอบ และมีความคงทนต่อการกัดกร่อนของสถาปัตยกรรมมากขึ้น รองรับการปฏิบัติการนำเข้าและการใช้งานต่อเนื่องได้
Containerization (e.g., Docker) (Kithulwatta, Jayasena, Kumara, & Rathnayaka, 2022)	เทคโนโลยีการทำคอนเทนเนอร์ช่วยในการห่อหุ้มซอฟต์แวร์ในระบบไฟล์ที่สมบูรณ์ซึ่งมีทุกอย่างที่ต้องการในการเรียกใช้โค้ด เช่น runtime, เครื่องมือระบบ และไลบรารีระบบ	ช่วยให้สภาพแวดล้อมการพัฒนาที่สม่ำเสมอ เจริญขึ้นในกระบวนการใช้งาน และเพิ่มประสิทธิภาพในการขยายขนาดและการเคลื่อนย้ายของแอปพลิเคชันในสภาพแวดล้อมที่แตกต่างกัน
DevOps practices (Mishra, & Otaiwi, 2020)	วิธีการที่รวมการพัฒนาซอฟต์แวร์ (Dev) และการดำเนินงานด้านเทคโนโลยีสารสนเทศ (Ops) มีจุดมุ่งหมายเพื่อลดรอยต์ชนิดการพัฒนาและให้การส่งมอบอย่างต่อเนื่อง	ส่งเสริมวัฒนธรรมการร่วมมือระหว่างทีมการพัฒนาและทีมดำเนินงาน เพิ่มความเร็ว คุณภาพ และความน่าเชื่อถือของกระบวนการพัฒนาและการใช้งาน

ตารางที่ 2.1 รายละเอียดเกี่ยวกับความก้าวหน้าของเทคโนโลยีการออกแบบและพัฒนาซอฟต์แวร์ (ต่อ)

ความก้าวหน้า	คำอธิบาย	ผลกระทบ
CI/CD (Donca et al., 2022 ; Erdenebat et al., 2023)	CI/CD เป็นวิธีการที่ใช้ในการส่งมอบแอปพลิเคชันไปยังลูกค้าอย่างสม่ำเสมอโดยการนำเอาการอัตโนมัติมาใช้ในขั้นตอนต่าง ๆ ของการพัฒนาแอปพลิเคชัน	มันได้เสริมความสามารถในการทำซ้ำและปล่อยฟีเจอร์ใหม่หรือแก้ไขได้อย่างรวดเร็ว โดยให้แน่ใจว่าซอฟต์แวร์ทางการศึกษาสามารถพัฒนาได้อย่างรวดเร็วตามข้อเสนอแนะจากผู้ใช้หรือความต้องการที่เปลี่ยนแปลง
Aspect-oriented program-ming (AOP) (Netinant, Elrad, & Fayad, 2001)	การเขียนโปรแกรมที่มุ่งเน้นการเพิ่มโมดูลลงไปในระบบโดยการแยกปัญหาที่ตัดกันออกจากกัน	การใช้งานแบบนี้ช่วยเพิ่มความแม่นยำและการจัดการโค้ดแบบโมดูลได้ดีขึ้นโดยเฉพาะสำหรับส่วนของระบบที่ต้องการใช้ร่วมกันข้ามโมดูลหลายๆ ส่วน เช่น การบันทึกข้อมูลและความปลอดภัย ซึ่งทำให้ระบบมีความสามารถในการปรับปรุงและการบำรุงรักษาได้ดียิ่งขึ้น

ที่มา: ผู้วิจัย, 2567

การพัฒนาซอฟต์แวร์ได้เดินทางไปในทิศทางที่ไม่ยอมรับข้อตกลงไปยังสถาปัตยกรรมที่ซับซ้อนมากขึ้น เป็นโมดูล และมีความยืดหยุ่นในการขยายขนาดที่มีประสิทธิภาพเพื่อตอบสนองต่อความต้องการที่เปลี่ยนแปลงอย่างต่อเนื่องของระบบที่เป็นปัจจุบัน โดยมีการเน้นเฉพาะในแพลตฟอร์มการศึกษา การเดินทางจากการเขียนโปรแกรมแบบโมดูลาร์ไปจนถึง OOP และวิธีการ

ล่าสุดเช่น ไมโครเซอร์วิสและ DevOps เน้นให้การเปลี่ยนแปลงมากขึ้นสู่ระบบที่เพิ่มประสิทธิภาพในด้านการดำเนินงานและส่งเสริมความยืดหยุ่นและความสามารถในการปรับตัวขึ้นให้มากขึ้น พัฒนาการเหล่านี้ช่วยให้สามารถผนวกเทคโนโลยีที่ทันสมัย เช่น AI และ ML ได้อย่างราบรื่น ทำให้มีแอปพลิเคชันที่มีความสามารถในการตอบสนองและตอบโต้ที่ฉลาดมากขึ้น อย่างไรก็ตาม เพื่อให้ประโยชน์จากความสามารถของระบบเหล่านี้อย่างเต็มที่ จำเป็นต้องมีการให้ความสำคัญต่อคุณภาพความพอใจของผู้ใช้ และการจัดเตรียมก่อนหน้าเพราะความซับซ้อนกำลังเพิ่มขึ้นเรื่อย ๆ การจัดเตรียมก่อนหน้านี้ไม่เพียงเพิ่มพลังให้กับผู้ชมเท่านั้น แต่ยังทำให้พวกเขารู้สึกมีความรับผิดชอบต่อความสำเร็จของระบบซอฟต์แวร์ด้วย วิธีการที่เสนอนี้มีความสำคัญมากในงานวิจัยนี้ เนื่องจากมันช่วยให้สามารถพัฒนาระบบซอฟต์แวร์ที่ทนทาน มีประสิทธิภาพ และยืดหยุ่นในโดเมนเทคโนโลยีที่เปลี่ยนแปลงอย่างต่อเนื่องได้

2.2 คอมพิวเตอร์และการพัฒนาที่ยั่งยืน

การคำนวณและการพัฒนาที่ยั่งยืนแทนจุดที่สำคัญที่ที่นวัตกรรมเทคโนโลยีตรงกับความรับผิดชอบทางสิ่งแวดล้อม ศึกษาหลายๆ แหล่งได้สำรวจพื้นที่ที่กำลังเปลี่ยนไปของการคำนวณที่ยั่งยืนโดยเน้นความจำเป็นของการปฏิบัติที่มีสำนึกต่อสิ่งแวดล้อมในการพัฒนาซอฟต์แวร์ การออกแบบฮาร์ดแวร์ และโครงสร้างพื้นฐานดิจิทัล การศึกษาของ (García-Berná et al., 2021)ให้ความสำคัญกับการเพิ่มขึ้นของการต้องการทรัพยากรคอมพิวเตอร์ในการใช้ซอฟต์แวร์ การดำเนินการเกี่ยวกับประสิทธิภาพของระบบด้านการดูแลสุขภาพโดยใช้เครื่องมือฮาร์ดแวร์ได้ถูกใช้และทดสอบโดยผู้ใช้กับงานหลายรายการ ข้อมูลที่รวบรวมเกี่ยวกับประสิทธิภาพในการใช้พลังงานถูกวิเคราะห์และพบว่ามีลักษณะของ GUI ที่มีส่วนสำคัญต่อความยั่งยืน (Alharbi, Alyoubi, Altuwairiqi, & Ellatif, 2021) ศึกษาการบริหารความเสี่ยงของการพัฒนาซอฟต์แวร์ในสภาพแวดล้อมหลายโครงการ วิธีการทดลองถูกใช้เพื่อหาคุณลักษณะที่สำคัญมากขึ้นในการประเมินความเสี่ยงวิธีการต่าง ๆ ได้เห็นด้วยกันว่ามีความสำคัญของคุณลักษณะสามประการ (ความสำคัญ, ความน่าจะเป็น, และขนาดความเสี่ยง) คุณลักษณะหลักสำหรับการจำแนกระดับความเสี่ยงคือ ความสำคัญ, ความน่าจะเป็น, และขนาดของความเสี่ยง ที่รับรองด้วยวิธีการลดลักษณะ (Manimegalai, Sandhanam, Nandhini, & Pandia, 2023) ได้ศึกษาการเพิ่มประสิทธิภาพในการใช้พลังงานโดยเปรียบเทียบแอปพลิเคชันต่าง ๆ และสำรวจโครงสร้างศูนย์ข้อมูลเขียวปัจจุบัน การศึกษานี้ประเมิน

วิธีการเขียนโค้ดที่ประหยัดพลังงานในคลาวด์คอมพิวติ้งและระบุกระบวนการในการลดการใช้พลังงานในสภาพแวดล้อมคลาวด์อย่างมาก โดยเน้นความแตกต่างสำคัญระหว่างแอปพลิเคชันภาษา Java และ JavaScript และให้การใช้ทรัพยากรที่ประสิทธิภาพสูงสุดและเพิ่มประสิทธิภาพในการใช้พลังงาน ผ่านการประเมินโค้ดและการวิเคราะห์ เส้นโค้ดเฉพาะ อัลกอริทึม และหลักการออกแบบที่ทำให้แอปพลิเคชันมีประสิทธิภาพทางพลังงานมากขึ้นถูกระบุ โดยสามารถประหยัดพลังงานได้สูงสุดถึง 17% ผ่านการปฏิบัติการเขียนโค้ดที่ประหยัดพลังงาน (Akinyele, Amole, Olabode, Olusesi, & Ajewole, 2021) นำเสนอ Agile Beeswax วิธีการใหม่ที่มีรากฐานบน Agile สำหรับการพัฒนาแอปพลิเคชัน ซึ่งจะทำหน้าที่ในการแก้ไขความท้าทายและความต้องการของกระบวนการพัฒนาโมบายไลล์ Agile Beeswax ประกอบด้วยวงรอบที่เกิดซ้ำซ้อนสองส่วนหลัก ได้แก่การออกแบบและพัฒนาแบบส่วนตัว โครงสร้างอยู่ในระหว่างหกขั้นตอน วิธีการวิจัยถูกพัฒนาผ่านกระบวนการวิจัยที่รวมถึงการทบทวนวรรณกรรมอย่างละเอียด การสัมภาษณ์กับนักพัฒนา การสำรวจแบบสำรวจ และการจัดทำข้อเสนอ การศึกษาพบว่าผู้เข้าร่วมมักนำเสนอวิธีการแบบอุ่นใจโดยเป็นส่วนใหญ่ โดยมี 84% ใช้วิธีการแบบอุ่นใจในการพัฒนาแอปพลิเคชันบนโทรศัพท์มือถือ ในบรรดานั้น 56% ต้องการ Scrum methodology, ประมาณ 13% แต่ละรายชอบ XP และ Kanban, และ 61% พิจารณาว่าวิธีการแบบอุ่นใจเป็นวิธีที่มีประสิทธิภาพที่สุดสำหรับการพัฒนาแอปพลิเคชัน (Ozdenizci Kose, 2021) ศึกษาวิธีการจำลองและวิเคราะห์เพื่อการวางแผน การออกแบบ และการพัฒนาไมโครกริดโดยใช้แหล่งพลังงานที่สะอาด กรอบการจำลองสำหรับการวางแผนพลังงานที่ยั่งยืนนี้นำเสนอวิธีการแบบเบ็ดเสร็จที่ครอบคลุมทุกมิติ โดยรวมถึงมิติทางสังคม เทคนิค เศรษฐกิจ สิ่งแวดล้อม และนโยบาย ผลการวิจัยแสดงให้เห็นว่าซอฟต์แวร์ระบบพลังงานที่มีอยู่บ่อยครั้งขาดการแสดงสิ่งที่เกี่ยวข้องกับด้านสังคม ซึ่งเน้นให้ความสำคัญกับความจำเป็นในการรวมมิติทางสังคมร่วมกับพารามิเตอร์ทางเทคนิค เศรษฐกิจ และสิ่งแวดล้อมในการจำลองระบบพลังงาน โดยการซับซ้อนรีวิวที่ผ่านมาเกี่ยวกับการพัฒนาซอฟต์แวร์ที่ยั่งยืน การศึกษานี้จึงระบุช่องว่างในการวิจัยในวิธีการพัฒนาที่ใช้ Agile เพื่อให้เข้ากับความต้องการธุรกิจที่เพิ่มขึ้นขณะขึ้นระดับการออกแบบและพัฒนาซอฟต์แวร์ทีละขั้น

ตารางที่ 2.2 รายละเอียดของวิธีการพัฒนาซอฟต์แวร์ที่เหมาะสมกับความยั่งยืนตามลักษณะของมันหรือตรงตามความต้องการที่เป็นฟังก์ชัน ประสิทธิภาพ การลดขยะ และการตอบสนองเป็นปัจจัยที่สำคัญเมื่อพัฒนาระบบซอฟต์แวร์ที่ยั่งยืน มีช่วงหลากหลายของวิธีการพัฒนา

ซอฟต์แวร์ที่สามารถออกแบบและใช้ในการก้าวหน้าด้านความยั่งยืนในอุตสาหกรรมการพัฒนาซอฟต์แวร์ได้

ตารางที่ 2.2 วิธีการพัฒนาซอฟต์แวร์สำหรับการพัฒนาซอฟต์แวร์ที่ยั่งยืน

Agile development	วิธีการแบบ Agile เน้นการพัฒนาแบบวนซ้ำ ความยืดหยุ่น และการมีส่วนร่วมของผู้มีส่วนได้ส่วนเสีย วิธีการนี้สามารถผนวกรวมคุณลักษณะและแนวปฏิบัติที่เน้นความยั่งยืนเข้ากับกระบวนการพัฒนาซอฟต์แวร์ได้อย่างรวดเร็ว ปรับตัวต่อการเปลี่ยนแปลงในมาตรฐานสิ่งแวดล้อมได้อย่างรวดเร็ว (Alhammad, & Moreno, 2024)
Lean software development	หลักการแบบ Lean ที่เน้นการลดของเสียโดยตรงนั้นช่วยสนับสนุนความยั่งยืนโดยการลดการใช้ทรัพยากร ซึ่งแปลว่ากระบวนการออกแบบซอฟต์แวร์ในขั้นต้นมีประสิทธิภาพมากขึ้นและช่วยระบุฟังก์ชันการทำงานที่ไม่จำเป็นซึ่งใช้ทรัพยากรเพิ่มเติม จึงสนับสนุนความยั่งยืน (Waheed, Khodeir, & Fathy, 2024)
DevOps	การผสานการพัฒนาและการดำเนินงานไม่เพียงช่วยปรับปรุงกระบวนการทำงานให้ราบรื่นและลดความซ้ำซ้อนเท่านั้น แต่ยังเพิ่มประสิทธิภาพอย่างมาก การเพิ่มประสิทธิภาพเหล่านี้มีส่วนโดยตรงต่อความยั่งยืนในการพัฒนาซอฟต์แวร์ การใช้ท่ออัตโนมัติใน DevOps ยังส่งเสริมการปรับใช้ที่สม่ำเสมอ มีประสิทธิภาพ และปราศจากข้อผิดพลาด ซึ่งช่วยลดของเสีย (Atadoga, Umoga, Lottu, & Sodiya, 2024)
Green coding	วิธีการนี้เกี่ยวข้องกับการเขียนโค้ดที่มีประสิทธิภาพสูงสุด ลดการใช้ทรัพยากรการคำนวณให้น้อยที่สุด และลดรอยเท้าพลังงานของการทำงานซอฟต์แวร์ แนวปฏิบัติการเขียนโค้ดสีเขียวรวมถึงการปรับแต่งอัลกอริทึมและลดความซับซ้อน (Manimegalai et al., 2023 ; Rathee, & Chhabra, 2022)

ตารางที่ 2.2 วิธีการพัฒนาซอฟต์แวร์สำหรับการพัฒนาซอฟต์แวร์ที่ยั่งยืน (ต่อ)

AOP	โดยการแยกข้อกังวลอย่างมีประสิทธิภาพ การเขียนโปรแกรมเชิงมุมมองช่วยให้โค้ดมีความเป็นโมดูลและนำกลับมาใช้ใหม่ได้ดีขึ้น ซึ่งจะช่วยเพิ่มความสามารถในการบำรุงรักษาและลดความจำเป็นในการออกแบบใหม่บ่อยครั้งหรือการอัปเดตที่ใช้ทรัพยากรมาก (Rukhiran, & Netinant, 2020)
SOA	SOA แบ่งฟังก์ชันการทำงานออกเป็นบริการที่แยกจากกัน ซึ่งสามารถจัดการและขยายได้อย่างอิสระ ความเป็นโมดูลนี้สนับสนุนประสิทธิภาพด้านพลังงานโดยการอนุญาตให้สามารถปรับขนาดและจัดการทรัพยากรได้อย่างแม่นยำตามความต้องการ (Rathee, & Chhabra, 2022)
Open source software	การใช้ซอฟต์แวร์โอเพ่นซอร์สสามารถส่งเสริมความยั่งยืนโดยการสนับสนุนการพัฒนาร่วมกันและการแบ่งปันโซลูชันซอฟต์แวร์ ลดความพยายามซ้ำซ้อนในอุตสาหกรรม และปรับปรุงคุณภาพและประสิทธิภาพของซอฟต์แวร์ผ่านการมีส่วนร่วมของชุมชน (Ye et al., 2021)
CI/CD	แนวปฏิบัติ CI/CD ช่วยให้การอัปเดตและการปรับปรุงถูกรวมเข้าด้วยกันและปรับใช้อย่างต่อเนื่อง ซึ่งสามารถรวมถึงการเพิ่มความยั่งยืน วงจรต่อเนื่องนี้สนับสนุนการปรับปรุงทีละขั้นในด้านการเพิ่มประสิทธิภาพของซอฟต์แวร์และการตอบสนองต่อเป้าหมายด้านความยั่งยืน (Kruglov, Succi, & Vasuez, 2021)

ที่มา: ผู้วิจัย, 2567

ในสาขาการพัฒนาซอฟต์แวร์ที่เปลี่ยนแปลงตลอดเวลา ความยั่งยืนได้รับการยอมรับมากขึ้นว่าเป็นวัตถุประสงค์ที่สำคัญ โดยมีแรงผลักดันมาจากความจำเป็นในการลดผลกระทบต่อสิ่งแวดล้อม การพัฒนาซอฟต์แวร์ที่มีประสิทธิภาพและยั่งยืนต้องใช้อิทธิพลของแนวคิดที่ครอบคลุมทั้งหมด ซึ่งรวมถึงมิติหลายมิติ เช่น การออกแบบ กระบวนการ การดำเนินการ การปรับใช้ การบำรุงรักษา และการปฏิบัติตามกฎระเบียบ ตามที่ระบุไว้ใน (Rukhiran, & Netinant, 2020) องค์ประกอบสำคัญเช่นการทำโมดูลให้เหมาะสม การใช้วิธีการทำงานที่รวดเร็ว ความสามารถในการใช้ส่วนประกอบอีกครั้ง และการปฏิบัติการเขียนโค้ดสีเขียวมีบทบาทสำคัญในกรอบการทำงานนี้ นอกจากนี้ยังมีการใช้เทคโนโลยีขั้นสูงเช่นคลาวด์คอมพิวติง (Manimegalai et al., 2023) และการเสมือนเซิร์ฟเวอร์ ร่วมกับการปฏิบัติตามกฎระเบียบด้านสิ่งแวดล้อมอย่างเคร่งครัด เพื่อเพิ่ม

ประสิทธิภาพในการใช้ทรัพยากรและจัดเตรียมการพัฒนาซอฟต์แวร์ให้สอดคล้องกับเป้าหมายด้านสิ่งแวดล้อมทั่วไป การยึดมั่นต่อกฎหมายสิ่งแวดล้อมและการใช้กลยุทธ์รวมร่วม ทำให้กระบวนการพัฒนาซอฟต์แวร์ได้รับการยกระดับในการดูแลรักษาสิ่งแวดล้อม และยังเป็นผู้ร่วมกิจกรรมที่มีส่วนร่วมในนวัตกรรมที่ยั่งยืน ตารางที่ 2.3 แสดงถึงความสำคัญของกลยุทธ์นี้ในการพัฒนาซอฟต์แวร์

ตารางที่ 2.3 ปัจจัยที่มีผลต่อความยั่งยืนในวงจรชีวิตการพัฒนาซอฟต์แวร์

Dimension	**Factor**	**Description**
Design	Modularity	การออกแบบแบบโมดูลช่วยให้การบำรุงรักษา การอัปเดต และการขยายขนาดเป็นเรื่องง่ายขึ้น ลดการใช้ทรัพยากรโดยรวม และเสริมการใช้งานของซอฟต์แวร์ให้ยาวนานขึ้น (Lavy, & Rami, 2018)
	Energy efficiency	การออกแบบซอฟต์แวร์ให้ใช้ทรัพยากรคอมพิวเตอร์และพลังงานน้อยลง ช่วยเสริมสร้างความยั่งยืนโดยลดการใช้พลังงาน (Pazienza, Baselli, Vinci, & Trussoni, 2024)
Process	Agile methodologies	การปฏิบัติตามหลัก Agile เช่น การพัฒนาแบบก้าวเป็นก้าวและการให้ความคืบหน้าอย่างต่อเนื่อง สามารถเสริมความสามารถในการปรับตัวและประสิทธิภาพของการพัฒนาซอฟต์แวร์ สอดคล้องกับการปฏิบัติที่ยั่งยืน (Ghimire, & Charters, 2022 ; Lee, & Chen, 2023)
	Code reusability	การส่งเสริมการใช้โค้ดซ้ำช่วยเร่งความเร็วในการพัฒนาและลดพลังงานและทรัพยากรที่ต้องใช้ในการสร้างโค้ดใหม่ตั้งแต่ต้น (Şanlıalp, Öztürk, & Yiğit, 2022)

ตารางที่ 2.3 ปัจจัยที่มีผลต่อความยั่งยืนในวงจรชีวิตการพัฒนาซอฟต์แวร์ (ต่อ)

Dimension	Factor	Description
Implementation	Green coding practices	การปฏิบัติการเขียนโค้ดที่ปรับปรุงประสิทธิภาพของอัลกอริทึมและการใช้ทรัพยากรลดลง ลดผลกระทบต่อสิ่งแวดล้อมของซอฟต์แวร์อย่างมีนัยสำคัญ (Yuan, Gao, & Xiang, 2023)
	Use of sustainable technologies	การผสมผสานเทคโนโลยีที่มีประสิทธิภาพในการใช้พลังงานมากขึ้นหรือที่สนับสนุนหลักการคอมพิวติงสีเขียว มีส่วนสำคัญต่อความยั่งยืนของผลิตซอฟต์แวร์ (Ahmad, Yahaya, & Sallehudin, 2022)
Deployment Wizard	Cloud computing	การใช้บริการคลาวด์สามารถเพิ่มประสิทธิภาพในการใช้พลังงานผ่านการบริหารจัดการรวมกันและการจัดสรรทรัพยากรอย่างชัดเจน ลดผลกระทบต่อสิ่งแวดล้อมจากการปล่อยก๊าซคาร์บอนหรือการใช้ทรัพยากรในการใช้งานซอฟต์แวร์ (Lis, Sudolska, Pietryka, & Kozakiewicz, 2020)
	Server virtualization	เทคโนโลยีเสมือนจำลองช่วยให้ใช้ทรัพยากรของเซิร์ฟเวอร์ได้อย่างมีประสิทธิภาพมากขึ้น ลดความต้องการใช้ฮาร์ดแวร์ทางกายภาพและลดการใช้พลังงาน (Carabaş, & Popescu, 2017)

ตารางที่ 2.3 ปัจจัยที่มีผลต่อความยั่งยืนในวงจรชีวิตการพัฒนาซอฟต์แวร์ (ต่อ)

Dimension	Factor	Description
Maintenance	Continuous optimization	การอัปเดตและปรับปรุงซอฟต์แวร์เพื่อเพิ่มประสิทธิภาพและความเป็นมาตรฐาน สามารถทำให้อายุการใช้งานและความยั่งยืนของผลิตภัณฑ์ซอฟต์แวร์เกิดขึ้นได้ (Khalifeh et al., 2023)
	Lifecycle management	การบริหารจัดการวงจรชีวิตของซอฟต์แวร์ที่มีประสิทธิภาพรวมถึงยุติการใช้งานที่คาดการณ์ไว้ล่วงหน้าและกลยุทธ์ในการยุติการใช้งาน ช่วยให้การใช้งานและการกำจัดผลิตภัณฑ์ซอฟต์แวร์เป็นไปอย่างยั่งยืน (Valmohammadi, & Mortaz Hejri, 2023)
Policy and compliance	Regulatory compliance	การปฏิบัติตามข้อบังคับและบทบาทเกี่ยวกับสิ่งแวดล้อมและความยั่งยืนสามารถนำการพัฒนาซอฟต์แวร์ไปสู่การปฏิบัติที่ยั่งยืนมากขึ้น (Mishra, 2021)
	Sustainability metrics and KPIs	การนำเสนอตัวชี้วัดประสิทธิภาพสำคัญสำหรับความยั่งยืนช่วยให้สามารถวัดและส่งเสริมการปรับปรุงผลกระทบทางสิ่งแวดล้อมของการพัฒนาซอฟต์แวร์ได้ (Fagarasan et al., 2023)

ที่มา: ผู้วิจัย, 2567

วิธีการพัฒนาซอฟต์แวร์นั้นเป็นอุปสรรคที่มีน้ำหนักมากต่อความยั่งยืน เรียกใช้การจัดการขยะและการใช้พลังงานอย่างมากๆ การสร้างอนาคตที่ยั่งยืนเป็นเรื่องสำคัญโดยการนำมาใช้ปฏิบัติการที่ยั่งยืน เช่น การปรับปรุงอัลกอริทึมและการออกแบบระบบและซอฟต์แวร์ ยังควรใช้ประโยชน์จากพลังงานทดแทนเพื่อลดผลกระทบต่อสิ่งแวดล้อมด้วย

2.3 วิธีการ Agile และความยั่งยืน

การพัฒนาซอฟต์แวร์แบบ Agile โดยเฉพาะการใช้โครงร่าง Rapid Application Development (RAD) มีข้อได้เปรียบมากมายสำหรับโครงการที่ต้องการความยืดหยุ่นและการเสร็จสิ้นอย่างรวดเร็ว (Naz, & Khan, 2015) โดยการเน้นความตอบสนองและความเร็ว RAD ช่วยให้ทีมพัฒนาซอฟต์แวร์ทำงานได้อย่างมีประสิทธิภาพและรวดเร็วมากขึ้น โดยใช้การปล่อยรุ่นแบบระหว่างการทำและการให้คำติชมจากผู้ใช้อย่างต่อเนื่อง (Beynon-Davies, Carne, Mackay, & Tudhope, 1999) ความพึงพอใจของลูกค้าถูกเพิ่มขึ้น และเวลาการเปิดตลาดสำหรับแอปพลิเคชันซอฟต์แวร์ใหม่ลดลง ผ่านการมีส่วนร่วมของผู้ส่งมอบและผู้ใช้งานสิ้นสุดระหว่างกระบวนการพัฒนาทั้งหมด RAD ยังช่วยลดความเสี่ยงในโครงการผ่านการตรวจจับและแก้ไขปัญหาในระหว่างวงจรการพัฒนาตั้งแต่ต้นได้อีกด้วย RAD ยังมีลักษณะที่เน้นการปรับปรุงคุณภาพ ที่ได้รับผลดีผ่านการประเมินประสิทธิภาพและประสบการณ์ของผู้ใช้อย่างต่อเนื่อง (Behutiye et al., 2020) RAD ส่งเสริมการสื่อสารและการทำงานร่วมกันที่ดีขึ้นระหว่างผู้ส่งมอบ, นักพัฒนา, และสมาชิกในทีม ซึ่งทำให้การดำเนินโครงการเป็นไปอย่างเรียบง่ายและมีประสิทธิภาพมากขึ้น

อย่างไรก็ตาม การนำเสนอวิธีการ Agile สำหรับการบำรุงรักษาซอฟต์แวร์ สร้างอุปสรรคให้เกิดขึ้น เช่น การเพิ่มขอบเขตโปรเจค (Scope Creep), ข้อจำกัดทรัพยากร และการปรับตัวให้เข้ากับวัฒนธรรม (Marnada, Raharjo, Hardian, & Prasetyo, 2022) Agile ต้องการการทำซ้ำบ่อย และการตอบสนองอย่างรวดเร็ว ซึ่งอาจเป็นภาระต่อทรัพยากร ทำให้เกิดอาการช้ำใจ และในที่สุดจะลดผลิตภัณฑ์ลง นอกจากนี้ ในสภาพแวดล้อมที่มาตรฐานและความเป็นเดียวกันมีความสำคัญ ความสม่ำเสมอของผลลัพธ์ของโครงการที่เกิดจากความยืดหยุ่นของ Agile อาจเป็นปัญหา วิธีการพัฒนาซอฟต์แวร์แบบ Agile มีลักษณะเด่นด้านความยืดหยุ่น ประสิทธิภาพ และการให้บริการที่ใส่ใจถึงผู้ใช้ มันรับประกันการส่งมอบซอฟต์แวร์ที่ดีกว่าและที่น่าพอใจลูกค้าและเกินความคาดหวังของลูกค้า (Mishra, & Alzoubi, 2023) ผู้เขียนมีการวิเคราะห์อย่างกระชับเกี่ยวกับวิธีการพัฒนาซอฟต์แวร์แบบ Agile และ RAD โดยให้ความชัดเจนในองค์ประกอบพื้นฐานของแต่ละวิธีในตาราง 2.4 การเปรียบเทียบโดยกระชับนำเสนอความแตกต่างระหว่าง RAD และ Agile โดยเน้นที่วิธีการจัดการโปรเจค การเกี่ยวข้องกับลูกค้า การพัฒนาซอฟต์แวร์ และการโต้ตอบของทีมทำงาน การเข้าใจนี้สามารถ

ช่วยให้องค์กรเลือกวิธีการพัฒนาที่เหมาะสมที่สุดตามความต้องการและข้อจำกัดของโปรเจคของพวกเขา

ตารางที่ 2.4 การเปรียบเทียบระหว่าง Agile และ RAD แยกแยะคุณลักษณะหลัก

Feature	Agile Development	RAD
Focus	เน้นการพัฒนาแบบรอบเรียบร้อย ความยืดหยุ่น และข้อเสนอแนะจากลูกค้าตลอดรอบการพัฒนา	เน้นการสร้างพร็อตติปไปให้เร็วและข้อเสนอแนะเพื่อพัฒนาซอฟต์แวร์ได้อย่างรวดเร็ว
Development process	รวมการดำเนินการแบบพุ่ง (Sprints) หรือการทำซ้ำที่ส่วนเล็กๆ ของซอฟต์แวร์ถูกสร้างและตรวจสอบอย่างต่อเนื่อง	ใช้การสร้างพร็อตติ้ต่อเนื่องเพื่อปรับปรุงความต้องการและการแก้ไขปัญหาอย่างรวดเร็วโดยมักมีการวางเป้าหมายน้อยลงในการวางแผน
Team collaboration	มีการร่วมมืออย่างสูงระหว่างผู้เกี่ยวข้องทั้งหมด รวมถึงการสื่อสารอย่างต่อเนื่องและการอัปเดตเป็นประจำ	การร่วมมือเน้นไปที่คำแนะนำเกี่ยวกับพร็อตติ้มากขึ้น โดยมีการเน้นลดลงในการสื่อสารระหว่างผู้เกี่ยวข้องอย่างต่อเนื่องตลอดทุกเฟสการพัฒนา
Customer involvement	ต้องการการมีส่วนร่วมของลูกค้าอย่างเต็มที่ตลอดโปรเจคสำหรับการให้คำแนะนำและการตัดสินใจ	คำแนะนำจากลูกค้าเป็นสิ่งสำคัญโดยส่วนใหญ่ในขั้นตอนการสร้างพร็อตติ้เพื่อสรุปความต้องการ
Flexibility	มีความยืดหยุ่นสูง ทำให้สามารถทำการเปลี่ยนแปลงขอบเขตและความต้องการของโปรเจคตามคำแนะนำและการทดสอบได้	ถึงแม้จะยืดหยุ่น แต่มักจะถูกจำกัดไว้ในช่วงเริ่มต้นของการพัฒนาในช่วงของการสร้างพร็อตติ้
Documentation	การให้ความสำคัญต่ำลงในการเอกสาร มากขึ้นกับซอฟต์แวร์ที่ทำงานได้	มีการเอกสารเพียงน้อย เนื่องจากให้ความสำคัญกับการสร้างพร็อตติ้อย่างรวดเร็วและความเร็ว

ตารางที่ 2.4 การเปรียบเทียบระหว่าง Agile และ RAD แยกแยะคุณลักษณะหลัก (ต่อ)

Feature	Agile Development	RAD
End product	ผลิตภัณฑ์ที่พร้อมจะส่งมอบได้ในท้ายทุกการรอบ โดยมีการปรับปรุงในการรอบต่อไป	ระบบที่สมบูรณ์ถูกสร้างขึ้นในช่วงเวลาที่สั้นลง โดยมีการให้ความสำคัญน้อยลงในการปรับปรุงที่เกิดขึ้นระหว่างการใช้งานเริ่มแรก
Suitability	เหมาะสำหรับโครงการที่คาดหวังว่าความต้องการจะเปลี่ยนแปลงไปตามเวลา	เหมาะสำหรับโครงการที่ต้องการการผลิตได้รวดเร็วและที่ความต้องการสามารถจำลองผ่านโปรโตไทป์ได้
Project size	สามารถขยายขนาดและปรับปรุงสำหรับโครงการที่เล็กและใหญ่ได้	โดยทั่วไปมักเหมาะสำหรับโครงการขนาดเล็กถึงขนาดกลางเนื่องจากการจำลองเป็นเรื่องที่ต้องใช้ทรัพยากรมาก
Risk management	การบริหารจัดการความเสี่ยงอย่างต่อเนื่องผ่านการทบทวนและการทำซ้ำอย่างเป็นประจำ	การจัดการความเสี่ยงเป็นไปตามการทดสอบโดยผู้ใช้และการให้คำแนะนำกลับเกี่ยวกับโปรโตไทป์ในช่วงแรก

ที่มา: ผู้วิจัย, 2567

2.4 โครงสร้างซอฟต์แวร์แบบชั้น

Layered Software Architecture หมายถึง การออกแบบโครงสร้างของซอฟต์แวร์ที่แบ่งส่วนของระบบออกเป็นชั้นๆ โดยแต่ละชั้นมีหน้าที่และความสำคัญที่แตกต่างกัน และส่งผลให้การพัฒนา การทดสอบ และการบำรุงรักษาง่ายขึ้น ซึ่งชั้นบนสุดจะใช้บริการจากชั้นด้านล่างเพื่อสร้างความสมดุลในการแบ่งงานและการทำงานของระบบ โครงสร้างชั้นลอยฟ้า (Layered Architecture) นี้มักถูกนำมาใช้ในการพัฒนาระบบซอฟต์แวร์ที่มีขนาดใหญ่และซับซ้อน เช่น ระบบฐานข้อมูล ระบบเซิร์ฟเวอร์ หรือแอปพลิเคชันขนาดใหญ่ วิธีการแบ่งแยกชั้นเป็นกลยุทธ์โครงสร้างที่ใช้ในการพัฒนาซอฟต์แวร์ โดยที่ซอฟต์แวร์ถูกแบ่งออกเป็นชั้นเฉพาะแต่ละชั้นมีหน้าที่แตกต่างกันโดยมีความสัมพันธ์เชิงลำดับ และแต่ละชั้นจะมีหน้าที่ที่ได้รับมอบหมายให้ดำเนินการตามความ

รับผิดชอบที่เฉพาะเจาะจง (Rukhiran, Buaroong, & Netinant, 2022) วิธีการนี้มีประสิทธิภาพที่สูงขึ้นอย่างมากในการเสริมความยั่งยืนของระบบซอฟต์แวร์ ความคงทนทาน และความยืดหยุ่น โดยการแบ่งซอฟต์แวร์เป็นชั้นหรือเลเยอร์ที่แตกต่างกัน และกำหนดหน้าที่ของแต่ละเลเยอร์อย่างชัดเจน การแบ่งระบบที่ซับซ้อนออกเป็นส่วนประกอบที่จัดการได้ง่ายขึ้น ทำให้การใช้วิธีการเลเยอร์ช่วยปรับปรุงความสามารถในการขยายตัวและการบำรุงรักษาของแอปพลิเคชันได้ดีขึ้น (Tu, 2023) แต่ละเลเยอร์ทำหน้าที่เฉพาะตัวที่แตกต่างกัน (Rukhiran, & Netinant, 2020) ซึ่งรวมถึงแต่ไม่จำกัดเฉพาะการเข้าถึงข้อมูล, ธุรกิจลอจิก, หรือการแสดงผล และจะสื่อสารเฉพาะกับเลเยอร์ที่อยู่ติดกันเท่านั้น ภายใต้การแบ่งแยกความกังวลนี้ นักพัฒนาสามารถอัปเกรดหรือปรับเปลี่ยนเลเยอร์ใดเลเยอร์หนึ่งได้โดยมีผลกระทบต่อน้อยที่สุดต่อเลเยอร์อื่น ๆ ซึ่งช่วยให้การบำรุงรักษาและการอัปเดตเป็นไปได้ง่ายขึ้น

องค์ประกอบสำคัญเพิ่มเติมของวิธีการ Layering คือความสามารถในการปรับตัว (Netinant et al., 2001) ระบบซอฟต์แวร์จะต้องสามารถปรับตัวได้อย่างรวดเร็วต่อสภาพแวดล้อมทางธุรกิจที่เปลี่ยนแปลงและข้อกำหนดใหม่ ๆ โดยไม่ต้องหยุดทำงานหรือพัฒนาขึ้นมาใหม่อย่างมาก วิธีการแบ่งชั้นช่วยให้สามารถรวมฟังก์ชันใหม่หรือแก้ไขฟังก์ชันที่มีอยู่โดยมุ่งเน้นเฉพาะที่ชั้นที่เกี่ยวข้อง ซึ่งช่วยรักษาความต่อเนื่องและประสิทธิภาพของระบบ โดยทั่วไปแล้ว การนำวิธีการแบ่งชั้นไปใช้จะช่วยเพิ่มความยั่งยืนในการพัฒนาซอฟต์แวร์ได้อย่างมาก

การแบ่งซอฟต์แวร์ออกเป็นส่วนๆ แต่ละส่วนมีหน้าที่เฉพาะที่กำหนดไว้ วิธีการแบ่งชั้นในการพัฒนาซอฟต์แวร์นี้ช่วยเพิ่มความสามารถในการคงอยู่ในระยะยาวของระบบซอฟต์แวร์อย่างมาก (Hocaoglu, 2017) โครงสร้างการจัดระเบียบนี้ช่วยให้การบำรุงรักษาและการอัปเดตเป็นเรื่องง่ายขึ้นและลดความเสี่ยงและความซับซ้อนที่เกี่ยวข้องกับการแก้ไขข้อบกพร่องและการอัปเดต (Rukhiran, & Netinant, 2020) นอกจากจะส่งเสริมความยั่งยืนทางสิ่งแวดล้อมแล้ว การแบ่งชั้นยังช่วยให้สามารถขยายระบบได้โดยการอนุญาตให้แต่ละชั้นขยายตัวได้อย่างอิสระเพื่อตอบสนองต่อความต้องการที่เปลี่ยนแปลงไป การแบ่งชั้นนี้เป็นส่วนของการใช้ทรัพยากรอย่างมีประสิทธิภาพมากขึ้นและลดผลกระทบต่อสิ่งแวดล้อมที่เกี่ยวข้องกับการดำเนินการของระบบซอฟต์แวร์ ซึ่งทำให้งานของคุณมีความรับผิดชอบต่อสิ่งแวดล้อมมากขึ้น

การใช้วิธีการแบ่งชั้นนี้ยังมีประโยชน์ที่เกี่ยวข้องกับความยั่งยืนผ่านการเพิ่มประสิทธิภาพในการใช้ทรัพยากร (Chinenyeze, Liu, & Al-Dubai, 2014) ขอบเขตที่ชัดเจนระหว่างชั้นให้การรับรองว่าทุกชั้นถูกปรับแต่งให้เหมาะสมกับวัตถุประสงค์ที่กำหนดไว้ ทำให้ประสิทธิภาพรวมของระบบเพิ่มขึ้นและปฏิบัติตามหลักการของการคำนึงถึงสิ่งแวดล้อมในการคำนวณให้เห็นได้ชัด โครงสร้างแบบโมดูลของวิธีการแบ่งชั้นช่วยให้สามารถนำส่วนประกอบมาใช้ซ้ำได้ ซึ่งลดเวลาที่ใช้ในการพัฒนาซอฟต์แวร์และทรัพยากรที่ใช้ไป รวมถึงการลดรอยระเทยของมันด้วย การแบ่งชั้นช่วยให้สามารถนำชั้นที่มีอยู่ก่อนหน้ามาใช้ในแอปพลิเคชันต่าง ๆ ทำให้ลดความจำเป็นในการดำเนินการใหม่และการใช้ทรัพยากรที่เกี่ยวข้องกับการพัฒนาซอฟต์แวร์จากมุมมองด้านความยั่งยืนลง แนะนำว่าการนำเสนอชั้นการเข้าถึงข้อมูลในแอปพลิเคชันธุรกิจหลาย ๆ รายการ สามารถลดความจำเป็นในการเขียนโค้ดที่ซ้ำซ้อนและพลังงานและวัสดุที่ใช้ในการทดสอบลงได้ การใช้ทรัพยากรไม่เพียงแค่ถูกอนุรักษ์โดยการใช้ส่วนประกอบที่ใช้ซ้ำ แต่รอบการพัฒนายังถูกย่อส่วน ทำให้การประมวลผลเร็วขึ้นและลดรอยพรางทางนิเวศ

ในสภาวะเทคโนโลยีที่เปลี่ยนแปลงอย่างต่อเนื่องในปัจจุบัน ความยืดหยุ่นและความสามารถในการปรับเปลี่ยนของวิธีการเลเยอร์นั้นเป็นประโยชน์ที่สำคัญโดยเฉพาะ โดยเฉพาะเมื่อพิจารณาถึงความต้องการธุรกิจและเทคโนโลยีที่เปลี่ยนแปลงอยู่ตลอดเวลา ผู้วิจัยพบว่าระบบซอฟต์แวร์ต้องมีความสามารถในการปรับเปลี่ยนได้อย่างยืดหยุ่นเพื่อรับการปรับปรุงใหม่ๆ โดยที่ไม่ต้องทำการเปลี่ยนแปลงโครงสร้างใหญ่โต การแบ่งแยกชั้นชั้นเองช่วยให้สามารถปรับปรุงส่วนที่เป็นอิสระต่างๆ ตอบสนองต่อการเปลี่ยนแปลงในเทคโนโลยีหรือกลยุทธ์ธุรกิจได้อย่างมีประสิทธิภาพ นอกจากนี้การใช้โครงสร้างแบบโมดูลาร์ช่วยให้สามารถบูรณาการกับเทคโนโลยีที่กำลังพัฒนาและบริการภายนอกได้อย่างมีประสิทธิภาพ เพื่อให้องค์กรสามารถปรับปรุงระบบซอฟต์แวร์หรือขั้นตอนการพัฒนาเทคโนโลยีได้อย่างรวดเร็ว การนำมาใช้ของการแบ่งชั้นในการพัฒนาซอฟต์แวร์ (Rana, & Saleh, 2022) รวมถึงการใช้ในแอปพลิเคชันที่มีระบบเลเยอร์หลายระดับ (Abderlrahman, Zhan, & Chong, 2020) และสถาปัตยกรรมของไมโครเซอร์วิสที่ทันสมัยมากขึ้น (Söylemez, Tekinerdogan, & Tarhan, 2024) ได้แสดงให้เห็นถึงนวัตกรรมและประโยชน์ของวิธีการนี้ การจัดหมวดหมู่ที่ชัดเจนของความรับผิดชอบในแต่ละเลเยอร์ทำให้ระบบง่ายต่อการทดสอบ การแก้ไขข้อผิดพลาด และการบำรุงรักษาทั้งในกรณีทั้งสอง การองค์กรยังส่งเสริมความยืดหยุ่นได้โดย

การอนุญาตให้เลเยอร์แต่ละชั้นขยายตัวอิสระตามความต้องการ ซึ่งจะเพิ่มประสิทธิภาพการใช้ทรัพยากรและประสิทธิภาพของระบบทั่วไปในโครงสร้างต่อไป

โครงสร้างของซอฟต์แวร์แบบเลเยอร์ (Layered Software Architecture) เป็นรูปแบบการออกแบบซอฟต์แวร์ที่แบ่งแยกการทำงานออกเป็นชั้นๆ หรือเลเยอร์ต่างๆ โดยแต่ละเลเยอร์จะรับผิดชอบในการปฏิบัติงานที่มีลักษณะเฉพาะของมัน และสามารถสื่อสารกับเลเยอร์ที่อยู่ใกล้เคียงได้ โดยทั่วไปแล้ว เลเยอร์จะถูกจัดออกเป็นลำดับหรือชั้นขั้นตามความสำคัญและระดับการทำงาน เช่น

1) Presentation Layer (ชั้นนำบริการหน้าจอ) เป็นเลเยอร์ที่มีหน้าที่ในการแสดงข้อมูลและอินเตอร์แอคทีฟกับผู้ใช้ โดยไม่ต้องคำนึงถึงว่าข้อมูลเหล่านั้นมาจากที่ไหน มักจะเป็นส่วนของเว็บเบราว์เซอร์หรืออินเตอร์เฟซผู้ใช้งาน

2) Business Logic Layer (ชั้นตัวเลขดำเนินการหลัก) เป็นเลเยอร์ที่รับผิดชอบในการประมวลผลข้อมูลและการดำเนินการตามกฎเกณฑ์ทางธุรกิจ โดยไม่คำนึงถึงว่าข้อมูลมาจากที่ไหนหรือกลับไปที่ไหน มักจะเป็นส่วนของ Application Server หรือซอฟต์แวร์บริการ

3) Data Access Layer (ชั้นเข้าถึงข้อมูล) เป็นเลเยอร์ที่รับผิดชอบในการเชื่อมต่อกับแหล่งข้อมูล เช่น ฐานข้อมูล และดำเนินการเกี่ยวกับข้อมูล เช่น การคิวรี (Query) และการปรับปรุงข้อมูล

การใช้เลเยอร์ช่วยในการแบ่งแยกหน้าที่และความรับผิดชอบของซอฟต์แวร์ ช่วยให้ง่ายต่อการแก้ไขปรับปรุง และเพิ่มขีดความสามารถของระบบได้อย่างยืดหยุ่น และมีประสิทธิภาพมากขึ้นในการพัฒนาและบำรุงรักษาระบบในระยะยาว

2.5 Aspect-Oriented Framework (AOF) and Flexibility

Aspect-Oriented Framework (AOF) เป็นโครงสร้างที่ใช้ในการพัฒนาซอฟต์แวร์เพื่อช่วยในการจัดการกับความยากลำบากของการเขียนโค้ด โดยทำให้โปรแกรมเมอร์สามารถแยกปัญหาที่ซับซ้อนออกจากโค้ดหลัก ซึ่งช่วยในการเพิ่มความยืดหยุ่นและความสามารถในการบำรุงรักษาโปรแกรมได้ง่ายขึ้น โครงสร้างนี้ใช้ในการแยกปัญหาที่เกี่ยวข้องกับด้านที่เรียกว่า "Aspect" ซึ่งเป็นเหตุการณ์หรือความสัมพันธ์ที่เกิดขึ้นในระหว่างการทำงานของโปรแกรม เช่น การจัดการกับการ

บันทึกข้อผิดพลาด การตรวจสอบสิทธิ์การเข้าถึงข้อมูล เป็นต้น โดย Aspect-Oriented Framework จะช่วยในการแยกปัญหาเหล่านี้ออกจากโค้ดหลักและจัดการกับมันในรูปแบบที่เรียกว่า "Aspect" ซึ่งทำให้โปรแกรมเมอร์สามารถใช้งานและบำรุงรักษาโค้ดได้อย่างมีประสิทธิภาพและเป็นระเบียบโดยทั่วไปแล้ว การใช้งาน AOF ช่วยในการลดความซับซ้อนของโค้ด การซ่อนรายละเอียดที่ซับซ้อนและเพิ่มความเข้าใจในโค้ดรวมถึงช่วยในการทำให้ระบบเป็นระเบียบมากขึ้นด้วยการแยกปัญหาออกเป็นส่วนๆ ทำให้ง่ายต่อการบำรุงรักษาและพัฒนาโปรแกรมในอนาคต

AOF เป็นแนวคิดที่ปรับปรุงความยืดหยุ่นของการพัฒนาซอฟต์แวร์อย่างมหาศาล (Netinant et al., 2001) และความยั่งยืน (Boeing, Leon, Nesbeth, Finkelstein, & Barnes, 2018) การแยกคอมโพเนนต์จากตัวต้นแบบของตรรกะธุรกิจหลักทำให้เกิดการพัฒนาโค้ดเบสที่มีโครงสร้างและเป็นไปได้ที่จะจัดการได้ง่ายขึ้น วิธีการที่มุ่งเน้นทิศทาง (AOAs) ลดความต้องการในการบำรุงรักษาสำหรับโครงการซอฟต์แวร์ เช่น การบันทึกข้อมูลเข้า-ออก การรักษาความปลอดภัย และการดำเนินการทรานแซคชัน ซึ่งมักมีผลต่อส่วนต่าง ๆ ของแอปพลิเคชัน (Davis, 2020) ซึ่งเป็นการมีส่วนร่วมในการสร้างความยั่งยืนสำหรับโครงการเช่นนี้ การลดเวลาและทรัพยากรที่จำเป็นในการบำรุงรักษาโค้ดที่มีอยู่และความเสี่ยงของข้อบกพร่องระหว่างการอัปเดต การแยกนี้จะทำให้ยืดอายุการใช้งานของระบบซอฟต์แวร์ได้อย่างมีประสิทธิภาพ

AOF ช่วยให้องค์กรสามารถนำฟังก์ชันใหม่เข้าสู่รหัสที่มีอยู่แล้วได้โดยไม่ต้องมีการปรับเปลี่ยน ความยืดหยุ่นนี้มีประโยชน์โดยเฉพาะเมื่อความต้องการทางธุรกิจมีการปรับเปลี่ยนบ่อยหรือแอปพลิเคชันต้องผสานกับระบบภายนอกที่กำลังเปลี่ยนแปลง; โดยการรวม AOA กับรูปแบบการเขียนโปรแกรมอื่น ๆ เช่น การเขียนโปรแกรมแบบฟังก์ชันหรือแบบออบเจกต์ จะทำให้สามารถใช้จุดเด่นของแต่ละรูปแบบเพื่อสร้างระบบที่ทนทานมากขึ้น สามารถขยายขนาดได้มากขึ้นและง่ายต่อการบำรุงรักษาได้มากขึ้น (Rukhiran, & Netinant, 2020) อย่างไรก็ตาม การนำ AOF มาใช้งานมีความซับซ้อนในการเข้าใจและการจัดการความสัมพันธ์ระหว่างส่วนประสมและแอปพลิเคชันหลักน่าจะทำให้กระบวนการเรียนรู้เพิ่มขึ้นและอาจทำให้กระบวนการดีบักและการทดสอบมีความซับซ้อนขึ้นได้ ไม่ว่าจะมีอุปสรรคเหล่านี้ แต่ AOF ได้มีบทบาทสำคัญในการปรับปรุงคุณภาพของซอฟต์แวร์โดยเพิ่มความยืดหยุ่นและความทนทานของมุมมองของการพัฒนาซอฟต์แวร์ โดยส่งเสริมสถาปัตยกรรมซอฟต์แวร์ที่มีโมดูลเพิ่มเติมและยืดหยุ่นมากขึ้น และช่วยองค์กรในการ

ตอบสนองต่อพัฒนาการเทคโนโลยีและความต้องการของตลาดได้อย่างรวดเร็วมากขึ้น (Netinant et al., 2001 ; Kumar, Grover, & Kumar, 2009 ; Park, Kang, & Lee, 2006)

2.6 การประหยัดพลังงานในการพัฒนาซอฟต์แวร์

การนำหลักการประหยัดพลังงานมาใช้ในกระบวนการพัฒนาซอฟต์แวร์โดยมีการรวมกับกลยุทธ์ชั้นสูงจะมีประโยชน์มากมาย คุณลักษณะนี้สอดคล้องกับโครงสร้างแบบเลเยอร์ของซอฟต์แวร์ซึ่งช่วยลดการใช้พลังงานและเพิ่มความยั่งยืนของระบบ การประหยัดพลังงานถูกนำเข้าในการตัดสินใจเกี่ยวกับโครงสร้างตั้งแต่ขั้นตอนการออกแบบแนวคิด เน้นความมีความยืดหยุ่นและการปรับขนาดได้ ด้วยการตัดสินใจเหล่านี้ จะช่วยให้การขยายของอนาคตและการปรับปรุงสามารถทำได้โดยใช้ทรัพยากรเพียงน้อยเท่าที่จำเป็น และรับรองความสามารถในการทำงานอย่างราบรื่นในสภาพแวดล้อมของฮาร์ดแวร์ที่หลากหลายที่มีระบบที่ซับซ้อน การใช้ข้อมูลและส่งเสริมความร่วมมือของทีมในช่วงต้นของการพัฒนาจะสร้างกรอบพื้นฐานที่สนับสนุนความยั่งยืนตามธรรมชาติ เมื่อซอฟต์แวร์ก้าวผ่านขั้นตอนการพัฒนาและการประยุกต์ใช้ ความสนใจจะถูกนำกลับมาให้สนับสนุนการรวมวิธีการเขียนโค้ดที่สูงสุดในเรื่องของประสิทธิภาพทางพลังงาน การแก้ไขโค้ดที่ใช้กลยุทธ์เพื่อลดการใช้พลังงานถูกนำเข้าไปในวิธีการแล้ว ระหว่างช่วงเวลาเหล่านี้ การนำเทคโนโลยีที่เป็นมิตรกับสิ่งแวดล้อม เช่น เวอร์ชวลิเซชันเซิร์ฟเวอร์และการทำคอนเทนเนอร์ เข้าสู่การใช้งานช่วยเพิ่มประสิทธิภาพในการใช้พลังงานด้วยการจัดการทรัพยากรอย่างมีประสิทธิภาพและลดความต้องการในโครงสร้างพื้นฐานที่ใช้พื้นที่จริง ๆ ลง

การนำแนวทางเช่นการคอมพิวติ้งในรูปแบบคลาวด์และการใช้ศูนย์ข้อมูลที่มีประสิทธิภาพทางพลังงาน ถูกนำมาใช้งานในช่วงขั้นตอนการนำไปใช้งาน (Manimegalai et al., 2023 ; Katal, Dahiya, & Choudhury, 2023) การใช้การบริหารจัดการแบบกลางที่มีการแบ่งปันทรัพยากร ช่วยลดผลกระทบต่อสิ่งแวดล้อมที่เกี่ยวข้องกับการดำเนินการของแอปพลิเคชันซอฟต์แวร์ได้อย่างมีประสิทธิภาพ กลยุทธ์การใช้งานแอปพลิเคชันซอฟต์แวร์อย่างมีประสิทธิภาพรับรองความสามารถในการส่งมอบและการดำเนินการของซอฟต์แวร์ตามวัตถุประสงค์ที่เกี่ยวข้องกับการยัดเข้าใช้งานที่มีประสิทธิภาพ ระยะการบำรุงรักษาของการพัฒนาซอฟต์แวร์ ที่สรุปเสร็จสิ้นรักษาการให้ความสำคัญกับประสิทธิภาพทางพลังงานโดยการนำเสนอการปรับปรุงและอัปเดตอย่าง

เป็นประจำ (Ahmad et al., 2022) การปรับปรุงต่อเนื่องของซอฟต์แวร์ รับรองว่าจะสามารถใช้งานได้อย่างต่อเนื่องในการปรับปรุงข้อกำหนดของผู้ใช้ที่กำลังเปลี่ยนแปลงและความก้าวหน้าทางเทคโนโลยี นอกจากนี้ การนำมาใช้วิธีการบริหารจัดการรอบการดำเนินงาน (Lifecycle Management) ในการรับรองว่าซอฟต์แวร์สามารถถูกเพิกถอนและถูกแทนที่ในลักษณะที่เป็นมิตรกับสิ่งแวดล้อมเมื่ออายุการใช้งานสิ้นสุดลงด้วยความเหมาะสม

โดยการใช้มาตรการเพื่อเพิ่มประสิทธิภาพทางพลังงานอย่างเป็นระบบตลอดวงจรการพัฒนาซอฟต์แวร์ องค์กรสามารถให้ความชุกในเรื่องฟังก์ชันและความยั่งยืนทางสิ่งแวดล้อมได้ ตารางที่ 2.5 นำเสนอภาพรวมของความพยายามต่อเนื่องในการเพิ่มประสิทธิภาพทางพลังงานของการพัฒนาซอฟต์แวร์ โดยมีการให้ความสำคัญกับกระบวนการพัฒนาซอฟต์แวร์ของแต่ละองค์กรในทุกๆด้าน

ตารางที่ 2.5 องค์กรที่ให้ความสำคัญกับประสิทธิภาพทางพลังงานในการพัฒนาซอฟต์แวร์ ชั้นแบ่ง

Area	Topics to Discuss
Foundational concepts	ความหมายและความสำคัญของประสิทธิภาพการใช้พลังงานในการพัฒนาซอฟต์แวร์ ความสัมพันธ์ระหว่างประสิทธิภาพการใช้พลังงาน การลดต้นทุน ผลกระทบต่อสิ่งแวดล้อม และประสิทธิภาพของระบบ (Ciancarini et al., 2020)
Energy-efficient coding practices	เทคนิคสำหรับการเพิ่มประสิทธิภาพของโค้ด เช่น การใช้วิธีการแบบมีประสิทธิภาพ, การลดความซับซ้อนในการคำนวณ, และการลดการใช้หน่วยความจำ; วิธีการเขียนโค้ดที่มีประสิทภาพทรัพยากรที่ดีที่สุด (Şanlıalp et al., 2022 ; Kravets, & Egunov, 2022)

ตารางที่ 2.5 องค์กรที่ให้ความสำคัญกับประสิทธิภาพทางพลังงานในการพัฒนาซอฟต์แวร์ ชั้นแบ่ง (ต่อ)

Area	Topics to Discuss
Design and architecture	ผลกระทบที่สัมผัสได้จากการออกแบบและสถาปัตยกรรมซอฟต์แวร์ต่อการใช้พลังงาน, ประโยชน์ของการมีโมดูลและการขยายขนาด, และแพทเทิร์นสถาปัตยกรรมที่สนับสนุนความพิถีพิถันของพลังงาน (เช่น สถาปัตยกรรมที่ใช้เหตุการณ์เป็นหลัก, บริการขนาดเล็ก); เหล่านี้ไม่ใช่เพียงการอภิปรายทฤษฎีเท่านั้น แต่เป็นปัจจัยในโลกของความเป็นจริงที่สามารถลดรอยเท้าพลังงานของซอฟต์แวร์ของคุณอย่างมีนัยสำคัญ (Schaarschmidt, Uelschen, Pulvermüller, & Westerkamp, 2020 ; Paradis, Kazman, & Tamburri, 2021)
Tools and technologies	เครื่องมือและเทคโนโลยีที่มีประสิทธิภาพในการตรวจสอบและเพิ่มประสิทภาพของพลังงานรวมถึงปลั๊กอิน IDE, โปรไฟล์เลอร์, และเครื่องมือวิเคราะห์ซอฟต์แวร์; เหล่านี้ไม่ใช่เพียงเพิ่มความซับซ้อนเพิ่มเติม แต่เป็นเครื่องมือที่มีประสิทธิภาพที่สามารถช่วยให้นักพัฒนาสามารถเพิ่มประสิทภาพของพลังงานได้อย่างมีประสิทธิภาพ (Avotins et al., 2022 ; Khan, Abbas, Lee, & Abbas, 2021)
Green technologies and practices	การใช้เทคโนโลยีเสมือนเซิร์ฟเวอร์ (Server virtualization), การใช้คอนเทนเนอร์ (containerization), และบริการคลาวด์ (cloud services); กระบวนการเลือกใช้ฮาร์ดแวร์ที่มีประสิทธิภาพในการใช้พลังงานอย่างละเอียดโดยพิจารณาปัจจัยเช่น การบริโภคพลังงาน, ความต้องการในการทำความเย็น, และประสิทธิภาพ; ปฏิบัติการในการติดตั้งโดยให้ความสำคัญกับการใช้พลังงาน; ตัวอย่างเทคโนโลยีสีเขียวที่ช่วยลดการใช้พลังงานในการดำเนินการซอฟต์แวร์ (Chaurasia, Kumar, Chaudhry, & Verma 2021 ; Ravikumar et al., 2022)

ตารางที่ 2.5 องค์กรที่ให้ความสำคัญกับประสิทธิภาพทางพลังงานในการพัฒนาซอฟต์แวร์ ชั้นแบ่ง (ต่อ)

Area	Topics to Discuss
Agile and CI/CD optimization	การวิเคราะห์ถึงวิธีการที่วิธีการ Agile และ CI/CD สามารถปรับใช้ให้เหมาะสมกับประสิทธิภาพในการใช้พลังงาน เช่น โดยการรวมความสำคัญของประสิทธิภาพในการใช้พลังงานเป็นตัววัดหลักในการวางแผนสปรินต์ และโดยการตรวจสอบการใช้พลังงานอย่างต่อเนื่องขณะที่ดำเนินการ CI/CD; ผลกระทบของการพัฒนาแบบรอบการและการให้คำแนะนำอย่างต่อเนื่องต่อการลดการสูญเสียและการใช้ทรัพยากรในการผลิตและการใช้งานซอฟต์แวร์ (Kruglov et al., 2021 ; Ournani, Rouvoy, Rust, & Penhoat 2021)
Case studies and practical examples	ตัวอย่างและการศึกษาเชิงลึกในโลกของความเป็นจริงที่ชัดเจนแสดงให้เห็นถึงการปฏิบัติและประโยชน์ของวิธีการใช้พลังงานอย่างมีประสิทธิภาพในการพัฒนาซอฟต์แวร์; เหล่านี้ไม่ใช่แค่ไอเดียที่นำมาสร้างขึ้นแต่เป็นการแก้ปัญหาที่เป็นประโยชน์ที่ได้รับการนำมาใช้จริงในกระบวนการพัฒนาของบริษัท (Dorokhova et al., 2021 ; Testasecca, Lazzaro, Sarmas, & Stamatopoulos, 2023)

ที่มา: ผู้วิจัย, 2567

บทที่ 3

ระเบียบวิธีการวิจัย

3.1 ภาพรวมของกรอบการวิจัยและวิธีการ

การใช้แบบจำลองและวิธีการวิจัยที่ผู้เขียนใช้ในการสำรวจประสิทธิภาพของกรอบ ALAI ในการก้าวหน้าของการปฏิบัติการพัฒนาซอฟต์แวร์ที่ยั่งยืน - โดยเฉพาะความมีประสิทธิภาพทางพลังงานและความยืดหยุ่นทางสถาปัตยกรรม – ได้รับการอธิบายไว้ในงานวิจัยนี้ เป็นการวิจัยที่เป็นปฏิบัติการจริง การสำรวจของผู้วิจัยเน้นไปที่การออกแบบและการประเมินกรอบโครงร่างในการพัฒนา GAISS (Netinant, Rukhiran, & Soongpol, 2022) อ้างอิงหลักการรักษาความปลอดภัยของข้อมูลตามมาตรฐาน ISO27001:2013 (Arponpong, & Soongpol, 2023) วิธีการนี้ได้พัฒนาสถาปัตยกรรมซอฟต์แวร์ที่ยั่งยืนและแข็งแกร่งโดยการผสมผสานเทคนิคของ Agile แบบส่วนขยาย, AOP, และเทคนิคการแบ่งชั้นอย่างละเอียดอย่างมาก เพื่อเพิ่มประสิทธิภาพในการปรับตัวของระบบและลดการใช้พลังงาน ผู้เขียนของ GAISS ได้ผสานวิธีการที่เน้นความยั่งยืนในการลงทุนเข้ากับทุกขั้นตอนของวงจรชีวิตการพัฒนาซอฟต์แวร์ โดยใช้กลยุทธ์ที่คิดอย่างรอบคอบและคำนึงถึงทรัพยากรอย่างประหยัด เพื่อประเมินผลของมาตรการเหล่านี้ ผู้เขียนได้ใช้เทคนิคการเก็บข้อมูลที่พิถีพิถัน ซึ่งรวมถึงการวัดการใช้พลังงานโดยตรงและการสำรวจไฟล์บันทึกของเซิร์ฟเวอร์ Microsoft IIS ซึ่งทำให้ได้ข้อคิดที่สำคัญเกี่ยวกับประสิทธิภาพทางพลังงานของระบบ นอกจากนี้ โดยใช้หลายเกณฑ์ที่กำหนดไว้ล่วงหน้าและวิธีการวิเคราะห์ทางคณิตศาสตร์ต่างๆ ผู้เขียนได้ดำเนินการวิเคราะห์เปรียบเทียบเพื่อประเมินประสิทธิภาพของวิธีการ Agile แบบปกติเมื่อเทียบกับวิธีการ Agile ที่ยั่งยืน วิธีการวิจัยยังอธิบายเกี่ยวกับอุปสรรคที่พบและการปรับเปลี่ยนต่อมาที่ทำขึ้นขณะดำเนินการใช้เทคนิคที่เป็นมิตรกับสิ่งแวดล้อมและหลักการที่ยั่งยืนในขั้นตอนการพัฒนาซอฟต์แวร์

การใช้ Proof of Concept (POC) จากการวิจัยนี้ วิธีการวิจัยได้สนับสนุนและพิสูจน์ถูกต้องถึงความเป็นไปได้และประสิทธิภาพของกรอบ ALAI ซึ่งมีความสามารถในการใช้งานจริงใน

สภาพแวดล้อมที่เป็นไปได้ GAISS ที่ออกแบบในตอนแรกสำหรับงานเปิดรับสมัครระดับบัณฑิตของมหาวิทยาลัยปี 2020 ถูกเลือกเป็นกรณีศึกษาเพื่อตรวจสอบนี้ การรวมเข้าด้วยกันของวิธีการที่ยั่งยืนในแต่ละขั้นตอนของวงจรชีวิตการพัฒนาเป็นวัตถุประสงค์หลักเมื่อออกแบบ GAISS ซึ่งมีวัตถุประสงค์ที่จะรวมกระบวนการ RAD แบบปกติและวิธีการ RAD ที่ยั่งยืนเข้าด้วยกัน วิธีการนี้เกี่ยวข้องกับการระบุวัตถุประสงค์ที่ยั่งยืนอย่างแม่นยำ เช่น การใช้ทรัพยากรอย่างมีประสิทธิภาพมากขึ้นและการลดการใช้พลังงาน และการดำเนินการวัตถุประสงค์เหล่านี้ผ่านการใช้วิธีการ Agile ที่ขยายขนาดพร้อมทั้งเพิ่มเติมด้วยโครงร่างที่มีลักษณะและสถาปัตยกรรมชั้น วิธีการนี้ช่วยให้การวิจัยสามารถสำรวจสมมติฐานหลายรายการเกี่ยวกับความยืดหยุ่นทางสถาปัตยกรรมและประสิทธิภาพทางพลังงานได้ในขอบเขตที่ควบคุมอย่างเข้มข้น โดยการเปรียบเทียบข้อมูลเชิงปฏิบัติการจริงกับผลลัพธ์ที่คาดหวัง วิธีการ POC ได้ให้ข้อมูลที่เป็นเชิงลูกศรที่เกี่ยวข้องกับข้อได้เปรียบและความยากลำบากในการใช้งานกรอบ ALAI วิธีการนี้ยืนยันแบบจำลองการวิจัยและนำเสนอข้อมูลที่เชื่อถือได้เกี่ยวกับความยืดหยุ่นและการขยายขนาดของกรอบในบริบทการพัฒนาซอฟต์แวร์ที่กว้างขึ้น

รูปที่ 3.1 ภาพรวมของขั้นตอนวิธีการวิจัย

ที่มา: ผู้วิจัย, 2567

แสดงโครงร่างและวิธีการวิจัยอย่างเป็นระบบสำหรับการพัฒนาซอฟต์แวร์ที่ยั่งยืนในภาพที่ 1 หลังจากที่ระบุปัญหาและวัตถุประสงค์ของการวิจัยไว้แล้ว กระบวนการที่เป็นการทำซ้ำนี้จะดำเนินไปยังการสร้างโครงร่าง ALAI ซึ่งเป็นการขยายขนาดของ Agile พร้อมกับการรวมกันของเทคนิคการแบ่งชั้นและเทคนิค AOF ในขั้นตอนที่สามนี้ เลือกใช้กรณีศึกษาระบบการศึกษาเพื่อแสดงตัวอย่างการปฏิบัติของโครงร่าง ALAI ในการทำงานในปฏิบัติการแบบจริง หลังจากที่เสร็จสิ้น

การออกแบบและการสร้างระบบ จะดำเนินการเก็บข้อมูลและการวิเคราะห์ข้อมูลโดยใช้ไฟล์บันทึกจากเซิร์ฟเวอร์ Microsoft IIS วิธีการ Agile แบบปกติถูกเปรียบเทียบกับวิธีการ Agile ที่ยั่งยืนที่ถูกนำมาใช้ใน GAISS ผ่านการวิเคราะห์เปรียบเทียบ การแก้ไขปัญหาและการปรับปรุงแสดงถึงลักษณะการทำซ้ำของวิธีการ Agile วงจรการวิจัยสรุปด้วยการตรวจสอบและประเมินโมเดลวิจัย

3.2 บริการระบบข้อมูลการรับสมัครเข้าศึกษาต่อระดับบัณฑิตศึกษา Graduate Admission Information System Services (GAISS)

GAISS ไม่ใช่เพียงโครงสร้างพื้นฐานดิจิทัลสำหรับสถาบันการศึกษาเท่านั้น แต่ยังเป็นการเดินหน้าที่สำคัญในการบริหารจัดการการรับสมัครเข้าศึกษาต่อระดับบัณฑิตศึกษาด้วย การสร้าง GAISS มีการเคลื่อนไหวมาจากความต้องการในการปรับปรุงและเพิ่มประสิทธิภาพในกระบวนการดำเนินงานทางบริหารที่เกี่ยวข้องกับการสมัครเข้าศึกษาต่อระดับบัณฑิตศึกษา ตั้งแต่เริ่มต้นมา GAISS ได้แสดงความสำคัญเป็นเครื่องมือที่สำคัญในการจัดการการรับสมัครเข้าศึกษาต่อระดับบัณฑิตศึกษา โดยการปรับตัวให้เข้ากับเทคโนโลยีที่เปลี่ยนแปลงอย่างต่อเนื่องและความต้องการที่เปลี่ยนแปลงของสถาบันการศึกษาอย่างต่อเนื่อง

ในปี 2019 การระบาดของโรคติดเชื้อโควิด-19 ส่งผลกระทบอย่างมีนัยสำคัญต่อ GAISS (ซึ่งอาจเป็นองค์กรหรือระบบใดๆ ที่มีชื่อเรียก GAISS) ซึ่งต้องปรับตัวเพื่อรับมือกับสถานการณ์ที่ไม่เคยเกิดขึ้นมาก่อน การระบาดครั้งนี้ทำให้การดำเนินการต่างๆ หยุดชะงัก หรือถูกปรับเปลี่ยนรูปแบบอย่างเร่งด่วนเพื่อให้สอดคล้องกับมาตรการควบคุมโรค ดังนั้นในการตอบสนองสถานการณ์ที่เกิดขึ้น ดังนั้น การรับสมัครเข้าศึกษาต่อระดับบัณฑิตศึกษาต้องสามารถเข้าถึงได้ตลอดเวลา และบริการต้องดำเนินการต่อไป แม้ว่ามหาวิทยาลัยจะปิดทำการก็ตาม ก่อนปี 2020 การสร้าง GAISS เริ่มต้นจากเป้าหมายในการแทนที่กระบวนการการรับสมัครที่ใช้แรงงานมากและใช้กระดาษด้วยวิธีทางดิจิทัลที่เชื่อถือได้และมีประสิทธิภาพมากขึ้น ในขั้นตอนแรกของการสร้างระบบนี้ ระบบได้จัดลำดับความสำคัญให้กับความสามารถในการประมวลผลข้อมูลและการเก็บรักษาข้อมูลพื้นฐานเพื่อจัดการเอกสารการสมัครเข้าศึกษาต่อระดับบัณฑิตศึกษา บันทึกข้อมูลนักศึกษา และหน้าที่ทางการบริหาร วัตถุประสงค์หลักของการปรับปรุงเร็ว ๆ นี้คือเพื่อลดภาระงานทางด้านการบริหารงานบุคลากรและลดปัญหาข้อผิดพลาดที่เกิดขึ้นบ่อยครั้งจากการประมวลผลข้อมูลด้วยวิธีการ

ดำเนินการด้วยมือในช่วงแรก ในเดือนมีนาคม พ.ศ. 2563 ด้วยการบูรณาการเทคโนโลยีดิจิทัลและอินเทอร์เน็ตเข้าสู่การบริหารการศึกษาอย่างก้าวหน้า GAISS ได้ขยายความสามารถของตนเพื่อรวมระบบการใช้งานเว็บเบสที่เรียกว่า "ระบบการสมัครก่อน" ไว้ในเครือข่ายของมัน การเปลี่ยนแปลงนี้เพิ่มความสะดวกในการเข้าถึงข้อมูลสำหรับผู้ดูแลระบบและผู้สมัคร และนำเสนอแพลตฟอร์ม E-Services หลักเพื่อช่วยในการก้าวหน้าของเป้าหมายการศึกษาของหลักสูตรปริญญาโทโดยเฉพาะในการรับสมัครนักศึกษาระดับบัณฑิต โดยเฉพาะ เหตุการณ์ที่กล่าวถึงก่อนหน้านี้ เป็นสิ่งที่สำคัญในการเสริมความคล่องตัวของระบบ ความสามารถในการปรับเปลี่ยนและความเข้ากันได้กับผู้ใช้ การขยายขีดความสามารถของระบบให้เหนือกว่าการประมวลผลข้อมูลทั่วไป เพื่อเพิ่มประสิทธิภาพและรองรับการทำงานที่ซับซ้อนยิ่งขึ้น

การพัฒนา GAISS ได้ทวีความซับซ้อนมากขึ้นตามกาลเวลา ด้วยการผสานฟังก์ชันและแพลตฟอร์มหลากหลายเข้าด้วยกัน ซึ่งทำให้ต้องพิจารณาปัญหาด้านข้อมูลและการจัดการข้อมูลจำนวนมากอย่างรอบคอบ ความต้องการที่เพิ่มขึ้นของระบบการสมัครก่อนในระบบข้อมูลนักศึกษาสร้างความท้าทายในการออกแบบเลย์เอาท์สำหรับข้อต่อการทำงานที่ซ้ำซ้อนและอินเทอร์เฟซของผู้ใช้หลายอย่าง นอกจากนี้ GAISS ยังได้ปรับปรุงบริการเสริมเพิ่มเติมล่าสุด เช่น แชทบอท และขั้นตอนการตัดสินใจ นอกจากนี้ยังมีการใช้เทคโนโลยีคลาวด์คอมพิวติ้งและเทคโนโลยีมือถือ ซึ่งเป็นตัวอย่างของเทคโนโลยีขั้นสูงที่ถูกนำมาใช้งานด้วย ทุกครั้งที่มีการพัฒนาเทคโนโลยีต่อไป จะมีผลให้ฟังก์ชันของ GAISS ขยายออกไปอย่างมาก การนำเข้าใช้เทคโนโลยีคลาวด์คอมพิวติ้งได้เพิ่มประสิทธิภาพของระบบได้ด้วยการเพิ่มความยืดหยุ่นและความเข้าถึง ทำให้ผู้ดูแลระบบและนักศึกษาสามารถเข้าถึงระบบจากทุกที่ และสร้างการผสานงานร่วมกับแพลตฟอร์มดิจิทัลอื่น ๆ ได้อย่างราบรื่นโดยไม่ยุ่งยาก การรวมบริการเสริมเพิ่มเติมไปยัง GAISS ได้เพิ่มประสิทธิภาพของระบบโดยช่วยให้สามารถทำการวิเคราะห์ข้อมูลและบริการระบบข้อมูลที่แข็งแกร่งมากขึ้นสำหรับคณาจารย์และนักศึกษา การปรับปรุงเหล่านี้เป็นสิ่งสำคัญในการเพิ่มประสิทธิภาพของกระบวนการตัดสินใจที่เกี่ยวข้องกับการรับสมัครเข้าศึกษาต่อระดับบัณฑิตศึกษา ผู้พัฒนาได้ทำการออกแบบใหม่ GAISS เพื่อให้ง่ายต่อการทำงานซับซ้อนโดยเฉพาะการลดช่วงเวลาในการออกแบบและพัฒนาซอฟต์แวร์ลง นอกจากนี้ โครงร่าง ALAI ยังทำหน้าที่แก้ไขปัญหาเรื่องการยั่งยืนใน GAISS ผ่านโครงสร้างขั้นตอนแบบชั้น ที่ช่วยให้ระบบมีการจัดการทรัพยากรที่มีประสิทธิภาพและยั่งยืนมากขึ้น โครงสร้างชั้นของระบบช่วยให้เกิดความยืดหยุ่นและความเข้ากันได้ในการตอบสนองต่อการพัฒนา

เทคโนโลยีที่เปลี่ยนแปลง ความต้องการของนักศึกษา และนโยบายการศึกษาที่เปลี่ยนแปลงอยู่ตลอดเวลา วิธีการนี้เป็นตัวอย่างที่ชัดเจนในความจำเป็นของการมีโซลูชันที่สามารถขยายได้ เนื่องจากมันสามารถจัดการข้อมูลจำนวนมากและคำถามจากผู้ใช้ได้อย่างมีประสิทธิภาพในช่วงเวลาที่มีความต้องการสูง GAISS ยังต้องการความยั่งยืนและการบำรุงรักษาระบบที่ดีขึ้น เช่น การปรับปรุงและการบำรุงรักษาที่ง่ายขึ้น การลดค่าใช้จ่ายในการบำรุงรักษา และการเพิ่มความยั่งยืนของซอฟต์แวร์

GAISS, ซึ่งพัฒนาขึ้นโดยใช้ซอฟต์แวร์ ALAI ในปัจจุบันเป็นแพลตฟอร์มแบบครอบคลุมที่ใช้สำหรับการบริหารและการศึกษาทั้งหมด ระบบนี้รองรับฟังก์ชันต่าง ๆ เช่น การจัดโปรไฟล์ของคณาจารย์และนักศึกษา ความสะดวกในการลงทะเบียนเรียน และการตรวจสอบผลการเรียน การพัฒนาระบบนี้อย่างต่อเนื่องเป็นหลักฐานที่แสดงถึงความมุ่งมั่นในการปรับตัวให้สอดคล้องกับความก้าวหน้าทางเทคโนโลยี และการตอบสนองต่อความต้องการที่เพิ่มขึ้นในสถานการณ์การศึกษาปัจจุบัน โดยระบบ GAISS มีบทบาทสำคัญในฐานะเครื่องมือหลักในการบริหารจัดการหลักสูตรระดับบัณฑิตศึกษา โดยที่มีบทบาทสำคัญในการเพิ่มประสิทธิภาพของการดำเนินงานและความสามารถในการรับสมัครนักศึกษา รวมถึงการดูแลต่อเนื่องของนักศึกษา รูปที่ 3.2 มีบทสรุปของกระบวนการพัฒนาซอฟต์แวร์สำหรับ GAISS โดยครอบคลุมช่วงเวลาต่าง ๆ ของการพัฒนาซอฟต์แวร์ โดยเน้นที่ความสำคัญของฟังก์ชันที่สำคัญ ข้อจำกัด การพัฒนาเทคโนโลยี และผลลัพธ์ของการพัฒนาเหล่านี้ GAISS เป็นตัวอย่างที่ชัดเจนของ ALAI ที่ใช้งานจริง ซึ่งแสดงให้เห็นถึงประโยชน์ของวิธีการนี้ในการดูแลระบบที่ยั่งยืน สามารถขยายขนาดได้ และซับซ้อน

รูปที่ 3.2 ช่วงเวลาการพัฒนาของ GAISS

ที่มา: ผู้วิจัย, 2567

3.3 การนำ GAISS มาใช้ในการพัฒนาซอฟต์แวร์แบบ Agile ดั้งเดิม

การนำ GAISS มาใช้งานโดยใช้วิธีการพัฒนาซอฟต์แวร์แบบ Agile ดั้งเดิมถูกแสดงในรูปที่ 3.3 กระบวนการที่ทำซ้ำอย่างมีประสิทธิภาพได้ปรับตัวอย่างมีประสิทธิภาพตามความต้องการที่เปลี่ยนแปลงของระบบและผู้ใช้ของมัน การนำ GAISS มาใช้และความสำเร็จของมันได้รับอิทธิพลอย่างมากจากการเน้นของ Agile ที่ให้ความสำคัญกับคำติชมจากผู้ใช้โดยเฉพาะในสถานการณ์ที่มีอุปสรรคที่ไม่เคยเกิดมาก่อนและการเจริญเทคโนโลยีอย่างรวดเร็ว เช่น การระบาดของ COVID-19

รูปที่ 3.3 GAISS ใช้งานโดยใช้วิธีการพัฒนาซอฟต์แวร์แบบ Agile ดั้งเดิม

ที่มา: ผู้วิจัย, 2567

รูปที่ 3.3 แสดงการพัฒนาก่อนหน้านี้สำหรับ GAISS ซึ่งใช้วิธีการพัฒนาซอฟต์แวร์แบบ Agile ดั้งเดิมดังต่อไปนี้

3.3.1 การวางแผนและการเก็บความต้องการเบื้องต้น: วิธีการ Agile ซึ่งเป็นการทำงานร่วมกันอย่างมีประสิทธิภาพ เริ่มต้นด้วยช่วงการวางแผนอย่างครอบคลุมผู้มีส่วนได้ส่วนเสียสำคัญ รวมถึงคณาจารย์ ผู้ดูแลระบบ และบุคลากรด้านเทคโนโลยีสารสนเทศ มีบทบาทสำคัญในการพิจารณาเหล่านี้ การมีส่วนร่วมอย่างมากของพวกเขาช่วยในการกำหนดฟังก์ชันพื้นฐานที่จำเป็นสำหรับ GAISS ได้อย่างชัดเจน ฟังก์ชันเหล่านี้ครอบคลุมตั้งแต่การเก็บรักษาข้อมูลและการประมวลผลข้อมูลพื้นฐาน ไปจนถึงความต้องการที่ซับซ้อนมากขึ้นเช่นการเข้าถึงเว็บและ E-

Services สำหรับการรับสมัครนักศึกษา ระหว่างการประชุมเหล่านี้ ทีม Agile ได้ระวังการเก็บรักษาบันทึกเรื่องราวของผู้ใช้อย่างละเอียด และสร้าง Backlog ของคุณลักษณะที่มีลำดับความสำคัญตามความสำคัญและผลกระทบของพวกเขา

3.3.2 การวางแผน Sprint และวงจรการพัฒนา: ทีมพัฒนา GAISS ที่ได้รับคำแนะนำจากหลักการ Agile ได้ตั้งขั้นตอนลำดับความสำคัญอย่างชัดเจน แต่ละสปรินต์เริ่มต้นด้วยการประชุมวางแผน ที่นั่นมีการกำหนดงานจาก Backlog ที่ได้รับการจัดลำดับตามความสำคัญ ทีมดำเนินการในวงจรที่ยาวเพียงสองสัปดาห์ ซึ่งช่วยให้มีความก้าวหน้าอย่างช้า ๆ และการประเมินต่อเนื่องในทิศทางของโปรเจกต์ ความยืดหยุ่นของวิธีการ Agile มีค่ามากโดยเฉพาะในช่วงนี้ โดยเฉพาะเมื่อต้องจัดการกับการเปลี่ยนแปลงที่เกิดขึ้นอย่างกะทันหันในการต้องการการเข้าถึงระยะไกลเนื่องจากวิกฤตโรคระบาดที่เกิดขึ้น

3.3.3 การรวมการทดสอบแบบต่อเนื่องเป็นส่วนสำคัญของวิธีการ Agile เพื่อให้มั่นใจในความมั่นคงของระบบในระหว่างการพัฒนา และเพื่อให้แน่ใจว่าคุณลักษณะที่เพิ่มใหม่เข้าไปสอดคล้องกับความสามารถที่มีอยู่แล้วในระบบ โดยการใช้สคริปต์การทดสอบอัตโนมัติ ทุกโมดูลใหม่หรือการอัพเดตจะถูกทดสอบอย่างเข้มงวด ซึ่งช่วยให้ทีมสามารถแก้ไขปัญหาที่เกิดขึ้นได้ทันทีด้วยการดำเนินการทดสอบอย่างต่อเนื่อง GAISS สามารถรักษามาตรฐานคุณภาพและความสามารถที่เหนือกว่า ซึ่งเป็นการหลีกเลี่ยงข้อเสียที่เกี่ยวข้องกับวิธีการพัฒนาซอฟต์แวร์แบบลำดับน้ำ ที่เลื่อนการทดสอบไปจนกระทั่งโครงการเสร็จสมบูรณ์

3.3.4 การวนซ้ำและวงจรตอบรับ: กรอบการทำงานแบบ Agile มีชื่อเสียงเนื่องจากกระบวนการที่วนซ้ำและการพึ่งพาต่อข้อเสนอแนะจากผู้ใช้ ตลอดการพัฒนาของ GAISS ข้อมูลตอบรับจากผู้ใช้งานสุดยอดและผู้มีส่วนได้ส่วนเสียอยู่เสมอ และถูกนำเข้าไปใช้ในกระบวนการอย่างต่อเนื่อง ข้อมูลตอบรับนี้เป็นปัจจัยสำคัญในการกำหนดรูปแบบของสปรินต์ในอนาคต ทำให้ทีมพัฒนาสามารถปรับปรุงระบบในเวลาที่เป็นเรียลไทม์ได้ ตัวอย่างเช่น ตอบสนองต่อข้อเสนอแนะเบื้องต้นจากผู้ใช้งานเกี่ยวกับความสะดวกในการใช้งานผ่านทางเว็บ ผู้วิจัยพัฒนาสามารถกำหนดลำดับความสำคัญในการปรับปรุงเหล่านี้ในสปรินต์ถัดไปได้อย่างทันท่วงที

3.3.5 การนำเสนอซอฟต์แวร์เริ่มต้นเมื่อ GAISS ได้รับการนำไปใช้โดยผู้ใช้งานเป็นลำดับหลังจากการตรวจสอบและปรับปรุงความสามารถพื้นฐานของมันได้เรียบร้อยแล้ว การเน้นให้ความสำคัญกับการส่งมอบอย่างต่อเนื่องของ Agile ทำให้การนำระบบไปใช้งานได้เป็นไปได้ การปรับปรุงที่จำเป็นถูกนำเข้าใช้ตอบสนองต่อปฏิสัมพันธ์ของผู้ใช้และประสิทธิภาพของระบบการ

บำรุงรักษาต่อเนื่อง: วิธีการ Agile ยังคงมีผลต่อระยะเวลาการบำรุงรักษาของ GAISS หลังการนำระบบไปใช้งานแล้ว การวิวัฒนาการของระบบได้รับการส่งเสริมโดยการรวมอัพเดตและการปรับปรุงตามระยะเวลาในสปรินต์เพิ่มเติม ที่ตอบสนองต่อความต้องการที่เกิดขึ้นและการพัฒนาเทคโนโลยีใหม่ๆ กรอบงาน Agile ช่วยให้โครงการรักษาความยืดหยุ่น การใส่ใจถึงผู้ใช้ และความทนทานในสถานการณ์ภายนอกที่เปลี่ยนแปลงอย่างรวดเร็วได้

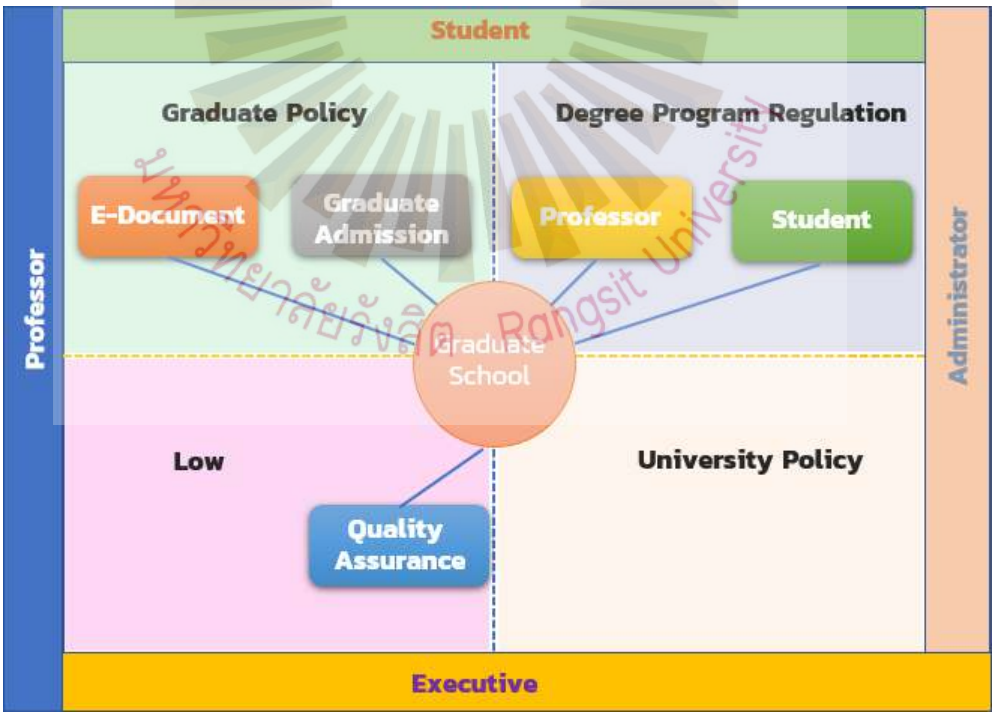
3.4 การเก็บรวบรวมข้อมูล

การวิจัยนี้มีวัตถุประสงค์เพื่อเปรียบเทียบประสิทธิภาพการใช้พลังงานของโมเดล GAISS สองแบบ โดยแบบหนึ่งสร้างขึ้นโดยใช้โมเดล Agile แบบดั้งเดิม และอีกแบบใช้กรอบการทำงาน ALAI ทั้งสองโมเดลถูกนำไปใช้งานบนเว็บเซิร์ฟเวอร์ Microsoft IIS ซึ่งมีความแข็งแกร่งและถูกใช้อย่างแพร่หลายในสภาพแวดล้อมขององค์กร โดยเป็นส่วนหนึ่งของกระบวนการเก็บรวบรวมข้อมูล มีการเก็บรวบรวมข้อมูลเป็นเวลา 825 วัน โดยเริ่มตั้งแต่วันที่ 10 กุมภาพันธ์ 2022 และสิ้นสุดวันที่ 15 พฤษภาคม 2024 ข้อมูลนี้ ซึ่งเก็บรวบรวมจากสถานการณ์จริง ช่วยให้สามารถวิเคราะห์ระบบได้อย่างครอบคลุมผ่านหน้าเว็บที่แสดงรายชื่อนักศึกษาแบบรวม เพิ่มความเป็นปฏิบัติและความเกี่ยวข้องกับการวิจัย การดำเนินงานตามปกติได้แสดงให้เห็นถึงประสิทธิผลของแต่ละแนวทาง ระบบที่เข้าถึงไฟล์บันทึกประกอบด้วยข้อมูลที่จัดเก็บแบบไม่มีความสัมพันธ์ รวมถึง: วันที่, เวลา, ระบบปฏิบัติการ, ตำแหน่งที่ตั้ง, ประเภทเว็บ, ที่อยู่ IP, ความเร็วในการเข้าถึง, และเวลาที่เข้าถึง ข้อมูลสำหรับการวิจัยนี้เน้นที่การวัดพลังงาน ถูกคำนวณโดยใช้เวลาการเข้าถึงต่อวินาทีในการเข้าถึงเว็บสำหรับรายชื่อนักศึกษา มีการใช้เครื่องมือที่ผสานกับเว็บเซิร์ฟเวอร์ MS IIS เพื่อตรวจสอบการใช้พลังงานในระหว่างการพัฒนาและการดำเนินงานของซอฟต์แวร์ ขณะนี้กำลังมีการเตรียมการวิเคราะห์ข้อมูลที่เก็บรวบรวมในช่วงเวลานี้เพื่อเปรียบเทียบประสิทธิภาพการใช้พลังงานของสองโมเดลการพัฒนาซอฟต์แวร์ ระเบียบวิธีการเก็บรวบรวมข้อมูลอย่างเข้มงวดที่ใช้ในงานวิจัยนี้รับประกันการประเมินผลที่แม่นยำถึงผลกระทบของกรอบการทำงาน ALAI เมื่อเปรียบเทียบกับแนวทาง Agile แบบดั้งเดิมต่อการใช้พลังงาน ข้อมูลนี้สร้างพื้นฐานที่มั่นคงสำหรับการระบุว่าแนวทางใดอาจช่วยเพิ่มประสิทธิภาพการดำเนินงานและความยั่งยืนในการพัฒนาซอฟต์แวร์ได้อย่างมีนัยสำคัญ

3.5 การออกแบบเครื่องมือที่ใช้

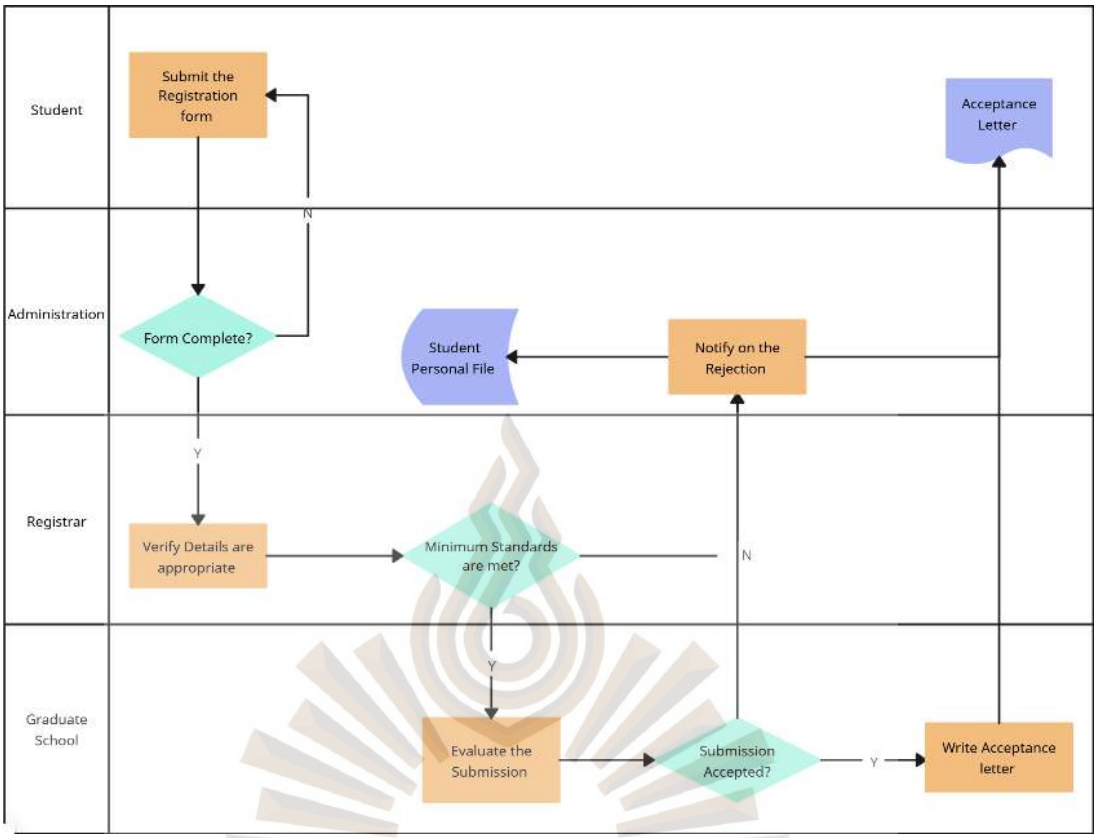
ในการออกแบบซึ่ง ได้ดำเนินการ ตามรายละเอียดดังนี้ การออกแบบ (Design) กระบวนการออกแบบระบบงานที่ใช้ในการวิเคราะห์ เพื่อเน้นการออกแบบระบบให้ตรงกับความต้องการของผู้ใช้งาน ทั้งด้านการนำข้อมูลเข้า รูปแบบของจอภาพ และการแสดงผลการรายงาน

การวิเคราะห์และออกแบบระบบ เป็นขั้นตอนที่สำคัญที่จะทำให้ได้การพัฒนาระบบที่มีประสิทธิภาพ โดยการวิเคราะห์จะเกี่ยวข้องกับงานของการออกแบบผังรายละเอียดต่างๆ ของการดำเนินงานและสร้างผังการทำงานต่างๆ เพื่อให้ง่ายต่อความเข้าใจ เช่น ผังของ สถาปัตยกรรมระบบดังที่แสดงในรูปที่ 3.4 และ การวิเคราะห์และออกแบบระบบ เป็นขั้นตอนที่สำคัญที่จะทำให้ได้การพัฒนาระบบที่มีประสิทธิรูป โดยการวิเคราะห์จะเกี่ยวข้องกับงานของการออกแบบผังรายละเอียดต่างๆ ของการดำเนินงานและสร้างผังการทำงานต่างๆ เพื่อให้ง่ายต่อความเข้าใจ เช่น รายละเอียดรวม(Work Flow Diagram) ดังที่แสดงในรูปที่ 3.5



รูปที่ 3.4 สถาปัตยกรรมระบบ

ที่มา: ผู้วิจัย, 2567



รูปที่ 3.5 รายละเอียดรวม

ที่มา: ผู้วิจัย, 2567

3.6 วิธีการวิเคราะห์เปรียบเทียบ

เนื่องจากบทบาทที่สำคัญของระบบ ALAI และ GAISS ในวงการวิศวกรรมซอฟต์แวร์ จึงเรียกให้ต้องมีการสำรวจอย่างละเอียดเพื่อเข้าใจถึงผลกระทบของระบบเหล่านี้ต่อความยืดหยุ่นและประสิทธิภาพในการใช้พลังงานของการดำเนินงานซอฟต์แวร์อย่างแท้จริง การประเมินนี้จะลึกเข้าไปในการใช้พลังงาน ความยืดหยุ่นทางสถาปัตยกรรม และประสิทธิภาพของระบบ เกณฑ์ประสิทธิภาพในการใช้พลังงานรวมถึงปัจจัยต่าง ๆ เช่น การใช้พลังงานเพิ่มเติม, พลังงานต่อธุรกรรมหรือเซสชันผู้ใช้, และการใช้พลังงานรวมทั้งหมดตลอดช่วงเวลาการพัฒนาและดำเนินการ ระดับความยืดหยุ่นทางสถาปัตยกรรมจะถูกวัดจากความง่ายในการนำมาใช้ให้เกิดการเปลี่ยนแปลง และผลกระทบต่อบริการของระบบ การใช้พลังงานของระบบตลอดเวลาและความสามารถในการปรับเปลี่ยนซอฟต์แวร์ให้เข้ากับความต้องการใหม่ๆ โดยมีการแก้ไขเพียงน้อย ๆ เป็นตัวบ่งชี้สำคัญ

ของความยั่งยืน เพื่อประเมินความสำคัญของความแตกต่างที่สังเกตเห็นระหว่างวิธีการพัฒนาสองแบบ การวิเคราะห์ทางสถิติเป็นสิ่งจำเป็นอย่างยิ่ง การใช้เทคนิคทางสถิติทั้งในการบรรยายและการสรุปได้รับการนำมาใช้เพื่อกำหนดความสำคัญของความแตกต่างและสรุปข้อมูลอย่างกระชับและชัดเจน

ผู้เขียนนำเสนอวิธีการเฉพาะที่เป็นระบบโดยผสมผสานการวิเคราะห์ทฤษฎีและการสำรวจเชิงประจักษ์เพื่อยืนยันความถูกต้องและประสิทธิภาพของวิธีการ ALAI เพื่อเสริมความน่าเชื่อถือของ ALAI นักวิจัยได้ใช้กลยุทธ์การวิจัยอย่างเป็นระบบและครอบคลุมภายในหลายระยะเวลา ซึ่งรวมถึงการเก็บและวิเคราะห์ข้อมูลทั้งแบบคุณภาพและปริมาณ การศึกษานี้มีวัตถุประสงค์เพื่อประเมินผลของ ALAI ต่อการพัฒนาซอฟต์แวร์โดยให้ความสำคัญกับตัวชี้วัดประสิทธิภาพหลัก เช่น การอนุรักษ์พลังงาน และการรวมการพัฒนาบริการเข้ากับระบบ

การวิเคราะห์เปรียบเทียบนี้นำเสนอการสรุปอย่างครอบคลุมของข้อความในการประเมินคุณภาพ ซึ่งให้มุมมองโดยรวมเกี่ยวกับความได้เปรียบของโครงร่าง ALAI เมื่อเปรียบเทียบกับวิธีการ Agile แบบดั้งเดิม การวิเคราะห์นี้นำเสนอการรวมผลการทดสอบทางสถิติกับความเข้าใจที่ได้รับจากข้อมูลคุณภาพ เช่นมุมมองของผู้ใช้และผู้พัฒนาเกี่ยวกับความยืดหยุ่นของระบบและความต้องการในการบำรุงรักษา การวิเคราะห์อย่างครบถ้วนนี้มีเป้าหมายเพื่อการระบุรูปแบบและแนวโน้มที่ช่วยให้เห็นว่าวิธีการหนึ่งสามารถมีประสิทธิภาพมากกว่าอีกวิธีหนึ่งในพื้นที่ที่เฉพาะเจาะจง เช่น ความยืดหยุ่นของระบบหรือความมีประสิทธิภาพในการใช้พลังงาน

ดังที่แสดงในสมการที่ 1 และ 2 ผลกระทบของโครงร่าง ALAI ต่อการใช้พลังงานในระหว่างการออกแบบและพัฒนาซอฟต์แวร์ได้รับการวัดโดยการผสมผสานระหว่างตัววัดและปัจจัยที่หลากหลาย วิธีการนี้มอบการวัดที่ชัดเจนและสามารถแสดงผลเป็นตัวเลขเกี่ยวกับผลกระทบของโครงร่าง ALAI ซึ่งช่วยเสริมสร้างความเข้าใจของผู้อ่านเกี่ยวกับความสัมพันธ์ของโครงร่างนี้กับการวิจัยได้ดียิ่งขึ้น

ผลกระทบทางพลังงานของ ALAI = พลังงานแบบดั้งเดิม - พลังงาน ALAI (3-1)

Where: พลังงานแบบดั้งเดิมคือ พลังงานที่ใช้ในระบบซอฟต์แวร์ที่คล้ายกันที่พัฒนาขึ้นโดยใช้วิธีการโดยไม่มีการใช้กรอบการทำงาน ALAI พลังงาน ALAI คือ พลังงานที่ใช้ในระบบซอฟต์แวร์ที่พัฒนาขึ้นโดยใช้กรอบการทำงาน ALAI

พลังงานต่อการเข้าถึง=เวลาประมวลผลความเข้าถึงเฉลี่ย*การใช้พลังงานเฉลี่ยต่อการเข้าถึง (3-2)

Where: เวลาประมวลผลความเข้าถึงเฉลี่ยคือ รวมเวลาเฉลี่ยในการประมวลผลคำขอเข้าถึงเพียงคำขอเดียวบนแอปพลิเคชันเว็บ ซึ่งเวลานี้มักจะถูกวัดเป็นวินาทีหรือมิลลิวินาทีและแทนระยะเวลาตั้งแต่คำขอถูกเริ่มต้นจนกระบวนการสำเร็จการของแอปพลิเคชัน การใช้พลังงานเฉลี่ยต่อการเข้าถึงคือ ปริมาณพลังงานเฉลี่ยที่ใช้ในการดำเนินการจัดการการเข้าถึงหรือคำขอเพียงคำขอเดียว หน่วยที่ใช้มักจะถูกวัดเป็นวัตต์-วินาที (Watt-seconds) หรือจูล ซึ่งทั้งสองหน่วยนี้แสดงถึงพลังงานที่ใช้งานในระยะเวลาเป็นหน่วยละวินาที

สมการ 1 คำนวณความแตกต่างในการใช้พลังงานระหว่างระบบที่ใช้ ALAI และระบบที่ไม่ใช้ ALAI ค่าบวกแสดงถึงการใช้พลังงานน้อยลงจากโครงร่าง ALAI ในขณะที่ค่าลบแสดงถึงการใช้พลังงานมากขึ้น เพื่อให้การวัดเป็นไปอย่างแม่นยำ ผู้วิจัยใช้ชุดตัววัดที่หลากหลาย ซึ่งรวมถึงการใช้พลังงาน (วัตต์), การใช้งาน CPU (เปอร์เซ็นต์), การใช้งานหน่วยความจำ (เมกะไบต์), การจราจรในเครือข่าย (กิโลไบต์ต่อวินาที), และประสิทธิภาพของระบบ (ธุรกรรมต่อวัตต์) พื้นฐานของข้อมูลที่ผู้วิจัยได้รวบรวมมานั้นเป็นการใช้เครื่องมือตรวจสอบซอฟต์แวร์และฮาร์ดแวร์ การตรวจวัดด้วยเซ็นเซอร์ และมิเตอร์พลังงาน

บทที่ 4

ผลการวิเคราะห์ข้อมูล

4.1 การนำไปใช้ของวิธีการที่ยั่งยืนในการพัฒนาซอฟต์แวร์

มีการเปลี่ยนแปลงที่สำคัญเกิดขึ้นในสิบกว่าปีที่ผ่านมาเกี่ยวกับความเจริญของเทคโนโลยีทางการศึกษาโดยเฉพาะในการพัฒนาระบบด้านการบริหารและการศึกษา การเปลี่ยนแปลงเหล่านี้มากที่สุดถูกสร้างขึ้นโดยความคืบหน้าที่กำลังเกิดขึ้นอย่างต่อเนื่องของเทคโนโลยีและความซับซ้อนที่เพิ่มขึ้นของการดำเนินงานทางการศึกษา ตอนแรกที่เริ่มใช้ระบบการศึกษา มักจะใช้วิธีการ Agile แบบดั้งเดิม ซึ่งถึงแม้จะเป็นการศึกษาที่เป็นนวัตกรรม แต่มักมีความจำเป็นที่จะต้องมีการช่วยเหลือในเรื่องของความยืดหยุ่น ความเปลี่ยนแปลงได้ และความยืดหยุ่น เนื่องจากกระบวนการทำแบบดั้งเดิมมักจะใช้เวลานาน ๆ ในการดำเนินการแบบดูแลด้วยมือ ข้อจำกัดเป็นอย่างมากเนื่องจากสถาบันการศึกษาต้องการฟังก์ชันที่ซับซ้อนมากขึ้นเพื่อจัดการกับการเพิ่มมากขึ้นของรูปแบบการให้บริการและความต้องการด้านการดำเนินงานที่ซับซ้อน วิธี ALAI ซึ่งเน้นไปที่ปรับขนาดได้ ความเปลี่ยนแปลงได้ และความยืดหยุ่น มีโอกาสที่จะเป็นวิธีการที่เหมาะสมเพื่อแก้ไขปัญหาเหล่านี้ได้

เป็นการตอบสนองต่ออุปสรรคเหล่านี้ การพัฒนาระบบซอฟต์แวร์สำหรับระบบการศึกษาได้เริ่มการผสานวิธีการที่ซับซ้อนมากขึ้น เช่นการใช้เทคนิคการเขียนโค้ดแบบปรับให้เข้ากับภาพรวมของระบบและการใช้วิธีการชั้นขั้นต่าง ๆ ในการพัฒนา วิธี ALAI ซึ่งเป็นโครงร่างที่มีศักยภาพสำหรับการสืบค้นนี้ แสดงถึงการเปลี่ยนแปลงแบบจริงจัง ตามที่แสดงในรูปที่ 4.1 โครงร่างนี้ขยายความเข้ากันได้กับวิธีการพัฒนาระบบซอฟต์แวร์ Agile โดยรวมความคำนึงถึงเรื่องความยั่งยืน เรื่องนี้สำคัญอย่างมากสำหรับระบบที่ทันสมัยที่เน้นการพัฒนาอย่างรวดเร็ว มอดูลสูง และประสิทธิภาพในการใช้พลังงาน วิธี ALAI รวมวิธีการ Agile, AOF, และเทคนิคการชั้นขั้นโดยเน้น

ไปที่การพัฒนาอย่างรวดเร็วและความสามารถในการนำกลับมาใช้ซ้ำ แบบโครงร่างนี้ไม่ได้เป็นแค่วิธีการที่มีศักยภาพเท่านั้น แต่ยังเป็นวิธีการที่เข้ากันได้สูงสำหรับความต้องการของผู้เกี่ยวข้องในสภาพแวดล้อมการศึกษา ซึ่งให้ความมั่นใจในความเหมาะสมของมัน

รูปที่ 4.1 วิธีการพัฒนาซอฟต์แวร์ ALAI เป็นการขยายของวิธีการพัฒนาซอฟต์แวร์ Agile

ที่มา: ผู้วิจัย, 2567

AOF ช่วยเพิ่มประสิทธิภาพของการพัฒนาระบบซอฟต์แวร์อีกต่อไป โดยการเพิ่มความสามารถในการแยกปัญหาออกจากกันอีกด้วย ด้วยเหตุนี้ ด้านต่าง ๆ ของซอฟต์แวร์จะถูกจัดการโดยเรียกว่าแยกออกจากกัน ซึ่งรวมถึงการจัดการข้อมูล การพัฒนาอินเตอร์เฟซผู้ใช้ และความปลอดภัยโดยเฉพาะในมาตรฐาน ISO27001:2013 ระบบบริหารจัดการความปลอดภัยข้อมูล โครงสถาปัตยกรรมนี้ช่วยให้เข้าใจและปรับเปลี่ยนระบบได้ง่ายขึ้น ด้วยการแยกส่วนออกมาให้ชัดเจน การทำความเข้าใจและการปรับเปลี่ยนที่ต้องการก็จะมีความสะดวกมากขึ้น ตามที่ระบุใน (Rukhiran, & Netinant, 2020) กรอบโครงร่างนี้ช่วยเสริมสร้างความสามารถในการแยกส่วนได้อย่างมีประสิทธิภาพ โดยการแยกปัญหาที่เกี่ยวข้องกับการตัดสินใจทั่วไปออก ซึ่งทำให้การปรับเปลี่ยนระบบเป็นไปอย่างยืดหยุ่นและจัดการได้ง่ายขึ้น ตอบสนองต่อความต้องการใหม่ ๆ ได้อย่างมีประสิทธิภาพมากยิ่งขึ้น ในความแตกต่างนี้ เทคนิคการกำหนดความรับผิดชอบในระดับที่ชัดเจนในการประยุกต์ใช้โครงสร้างนั้นมีความสำคัญอย่างยิ่ง ทำให้เกิดความเข้าใจขององค์กรที่ดีขึ้น เพิ่มประสิทธิภาพในการขยายขนาดและการรักษาระบบได้อย่างมีประสิทธิภาพ เพื่อให้เป็นตัวอย่าง เมื่อ

มีการเปลี่ยนแปลงในชั้นของอินเตอร์เฟซผู้ใช้ จะไม่มีผลกระทบต่อชั้นการจัดการข้อมูล ซึ่งทำให้ระบบมีความยืดหยุ่นและความง่ายในการบำรุงรักษามากขึ้น วิธีการโครงสร้างนี้มีส่วนทำให้กระบวนการพัฒนามีประสิทธิภาพและมีประสิทธิภาพมากขึ้น และรับประกันความยั่งยืนและต่อเนื่องของบริการระบบได้อย่างมั่นคง

โดยการใช้วิธี ALAI ในการปฏิบัติระบบการสมัครเข้าโรงเรียนล่วงหน้า มหาวิทยาลัยสามารถสร้างระบบสารสนเทศที่มั่นคง ยืดหยุ่น และสามารถปรับเปลี่ยนได้ตามความต้องการได้เป็นอย่างดี การใช้วิธี ALAI อาจต้องการการปรับเปลี่ยนที่สำคัญในกระบวนการพัฒนาซอฟต์แวร์ปัจจุบันและอาจทำให้ทีมพัฒนาต้องเผชิญกับขั้นตอนการเรียนรู้ใหม่ๆด้วย ควรทราบว่ามีความจำเป็นที่จะต้องปรับตัวและปรับตัวกับการเปลี่ยนแปลงที่เกิดขึ้นในระบบการพัฒนาด้วยวิธีนี้ ระบบที่พัฒนาขึ้นด้วยความต้องการของผู้ใช้งานระบบ สามารถปรับได้ตามความต้องการปัจจุบันและมีความทนทานพอที่จะปรับตัวให้เข้ากับการเปลี่ยนแปลงทางเทคโนโลยีในอนาคตและการเปลี่ยนแปลงในความต้องการขององค์กรได้อย่างยอดเยี่ยม ด้วยการนำเสนอกลยุทธ์รวมทั้งหมดนี้ สถาบันการศึกษาสามารถใช้โครงสร้างพื้นฐานดิจิทัลได้อย่างมีประสิทธิภาพเพื่อปรับปรุงการดำเนินงานทางบริหารและการศึกษาในขณะที่ยึดถือหลักการพัฒนาซอฟต์แวร์อย่างยั่งยืน

วิธี ALAI ซึ่งเป็นผลผสมที่เป็นเอกลักษณ์ระหว่างการพัฒนาแบบ Agile, โครงสร้างซอฟต์แวร์ชั้นขั้นและ AOFs ถูกออกแบบอย่างพิถีพิถันเพื่อสร้างระบบซอฟต์แวร์ที่ซับซ้อนและยืดหยุ่นอย่างเหมาะสมและเป็นมิตรกับสิ่งแวดล้อม วิธีการนวัตกรรมนี้ ที่แสดงในรูปที่ 4 เริ่มต้นด้วยขั้นตอนสำคัญของการวางแผนและการวิเคราะห์ความต้องการต่าง ๆ อย่างละเอียด ผู้ส่งเสริมมีการร่วมมือกันเพื่อกำหนดวัตถุประสงค์ ความต้องการของระบบ และเป้าหมายที่เกี่ยวกับความยั่งยืน เช่นความทนทานและประสิทธิภาพในการใช้พลังงาน ขั้นตอนนี้เป็นการเตรียมพื้นฐานสำหรับการสร้างรายการความต้องการอย่างเชี่ยวชาญที่จะให้การพัฒนาระบบซอฟต์แวร์ประสบความสำเร็จโดยการกำหนดลำดับความสำคัญของความต้องการที่เฉพาะเจาะจงและเป็นไปตามแผนที่เกี่ยวข้อง

หลังจากการรวบรวมความต้องการเสร็จสิ้น วิธี ALAI จะเข้าสู่ขั้นตอนของการออกแบบโครงสร้างแบบชั้น ในขั้นตอนนี้ ระบบจะถูกจัดองค์ประกอบให้อยู่ในรูปแบบของชั้นต่าง ๆ ที่มีวัตถุประสงค์เฉพาะ ๆ เช่น การเข้าถึงข้อมูล ตรรกะธุรกิจ หรือการนำเสนอ วิธีการนี้ไม่เพียงแต่เพิ่ม

ความมอดูลาริตี้และทำให้การบำรุงรักษาระบบง่ายขึ้น แต่ยังส่งเสริมความยืดหยุ่นในการขยายขนาดและการปรับเปลี่ยนในอนาคต โดยมีผลทำให้ระบบมีความยั่งยืนได้อย่างมีนัยสำคัญ

ในขั้นตอนการผสาน AOF ซึ่งเกิดขึ้นหลังการออกแบบโครงสร้าง ปัญหาที่ต้องการตัดเชิงทั่วไป เช่นการจัดการธุรกรรม การบันทึกข้อมูล และความปลอดภัย ถูกแยกออกจากตรรกะธุรกิจ การแยกปัญหาเหล่านี้ออกเป็นโมดูลหรือด้านแยกต่างๆ ช่วยป้องกันไม่ให้มีการรบกวนกับฟังก์ชันหลักได้ ด้วยเหตุนี้ ความสามารถในการบำรุงรักษาของระบบถูกเพิ่มขึ้น และโอกาสที่จะพบข้อบกพร่องระหว่างการพัฒนาแบบสับสนลดลง

ต่อมา วิธีการ ALAI ก้าวหน้าไปสู่การพัฒนาแบบสปรินต์และการพัฒนาแบบซ้ำซ้อนตามหลัก Agile ในวิธีการนี้ กระบวนการพัฒนาถูกแบ่งออกเป็นสปรินต์ที่ภายในนั้น จะมีการสร้างคุณสมบัติที่เฉพาะเจาะจง ทดสอบ และประเมินภายใต้ข้อเสนอและความคิดเห็นจากผู้ส่งเสริมและความสำคัญจากสปรินต์ก่อนหน้า องค์ประกอบนี้รับประกันว่าซอฟต์แวร์จะมีการพัฒนาต่อเนื่องและปรับตัวอย่างรวดเร็วเพื่อให้ตอบสนองต่อความต้องการที่เกิดขึ้นอย่างรวดเร็วเพื่อที่จะสามารถตอบสนองต่อความต้องการที่เปลี่ยนแปลงไปได้ตลอดเวลา

ผู้วิจัยใช้วิธีการทดสอบที่เข้มงวด ไม่ว่าจะเป็นการทดสอบด้วยมือหรือการทดสอบอัตโนมัติ ตลอดระหว่างขั้นตอนการผสานและการพัฒนาทดสอบต่อเนื่อง เพื่อให้มั่นใจได้ว่าผู้วิจัยจะสามารถตรวจจับปัญหาในการผสานและคุณภาพของซอฟต์แวร์ได้อย่างต่อเนื่องในระยะเริ่มต้น การทดสอบต่อเนื่องเป็นสิ่งจำเป็นในการรักษาคุณภาพของซอฟต์แวร์และตรวจจับข้อบกพร่องในระยะเริ่มต้น ซึ่งทำให้มั่นใจในความสมบูรณ์ของซอฟต์แวร์ได้อย่างมาก

หลังจากการพัฒนาเสร็จสิ้น วิธี ALAI เข้าสู่ขั้นตอนสำคัญๆ ที่ความมุ่งมั่นของผู้เกี่ยวข้องเป็นสิ่งสำคัญ โดยทำให้มีการตรวจสอบและประเมินซอฟต์แวร์ที่พัฒนาขึ้นให้ตรงตามความต้องการและมีประสิทธิภาพที่คาดหวัง ในขั้นตอนการนำระบบไปใช้งาน ซอฟต์แวร์จะถูกนำไปติดตั้งในสภาพแวดล้อมการทดสอบเพื่อให้ผู้ส่งเสริมสามารถประเมินได้ในสภาพแวดล้อมที่มีความเสถียรในขณะเดียวกัน ขั้นตอนนี้เป็นแพลตฟอร์มที่สำคัญสำหรับการรวบรวมคำติชมจากผู้เกี่ยวข้อง ซึ่งจากนั้นจะถูกนำมาผสานเข้ากับสปรินต์การพัฒนาถัดไป การให้ความสำคัญในการรับฟังความ

คิดเห็นอย่างต่อเนื่องนี้เป็นการเน้นถึงความสำคัญของผู้เกี่ยวข้องในความสำเร็จของซอฟต์แวร์ โดยการรับรู้ความต้องการของผู้ใช้อย่างต่อเนื่องและการปรับให้เข้ากับเป้าหมายที่ยั่งยืนในระยะยาว จะช่วยให้ระบบประสบความสำเร็จได้อย่างต่อเนื่อง

ในที่สุด ความสามารถในการทำงานของซอฟต์แวร์และสถานะปัจจุบันของมันถูกรักษาไว้ผ่านการบำรุงรักษาและการพัฒนาต่อเนื่องไปด้วย โครงสร้างที่เรียบง่ายและการแยกส่วนออกมาอย่างชัดเจนขององค์ประกอบช่วยให้การอัพเดตและการเปลี่ยนแปลงที่ไม่ต้องการการติดตั้งใหม่ใหญ่โตมากขึ้น ซึ่งเสริมความทนทานและความยืดหยุ่นที่ยั่งยืนของระบบได้อย่างมาก

วิธีการนี้ตอบสนองต่อความต้องการที่เปลี่ยนแปลงอย่างต่อเนื่องของการพัฒนาซอฟต์แวร์ในยุคปัจจุบันได้อย่างมีประสิทธิภาพมันรับประกันว่าระบบที่พัฒนาขึ้นมีความทนทาน ยืดหยุ่น และเป็นมิตรต่อสิ่งแวดล้อม ตามวัตถุประสงค์ที่ระบุไว้ในวิธี ALAI

4.2 การออกแบบซอฟต์แวร์โดยมีการคำนึงถึงความยั่งยืนและความยืดหยุ่น

โครงสร้าง GAISS ที่ออกแบบอย่างพิถีพิถันเพื่อครอบคลุมทุกด้านของระบบ ช่วยเสริมความมั่นใจในประสิทธิภาพและความยั่งยืนของมันได้อย่างมาก แต่ละชั้นมีวัตถุประสงค์ที่แตกต่างกัน ทำให้ระบบทำงานได้ในลักษณะที่เป็นมิตรต่อสิ่งแวดล้อมและเรียบง่ายตั้งแต่ต้นแรก การอินเตอร์เฟซผู้ใช้ที่ปฏิสัมพันธ์กับผู้ใช้สุดท้าย และฮาร์ดแวร์ที่เป็นพลังงานของระบบ ทั้งหมดเป็นส่วนหนึ่งของการออกแบบรวมที่แตกต่างนี้ มิติด้านความยั่งยืนของโครงสร้าง GAISS ถูกเน้นในรูปที่ 5 โดยเน้นที่จุดเริ่มต้นเพื่อส่งเสริมประสิทธิภาพ ความยืดหยุ่น และความยั่งยืน

ชั้นฮาร์ดแวร์ของ GAISS ซึ่งเป็นป้อมปราการที่ประกอบด้วยองค์ประกอบทางกายภาพที่จำเป็น เป็นส่วนสำคัญในการส่งเสริมความยั่งยืน หน่วยประมวลผลกราฟิก (GPU) จัดการกับการดำเนินการที่เกี่ยวกับกราฟิกทั้งหมด ในขณะที่หน่วยประมวลผลกลาง (CPU) ทำหน้าที่เป็นสมองของระบบ องค์ประกอบทางเครือข่าย เช่น สวิตช์ มั่นใจว่าข้อมูลถูกส่งผ่านไปได้อย่างราบรื่น สิ่งที่สำคัญที่สุดคือหน่วยจ่ายไฟฟ้าที่ให้พลังงานไฟฟ้าที่จำเป็นสำหรับองค์ประกอบฮาร์ดแวร์ทั้งหมด

การมุ่งมั่นของ GAISS ในการลดผลกระทบจากการดำเนินการและส่งเสริมความยั่งยืนเป็นไปอย่างชัดเจนในการเน้นรุ่นที่มีประสิทธิภาพในการใช้พลังงาน

รูปที่ 4.2 การออกแบบโครงสร้างซอฟต์แวร์เพื่อความยั่งยืนและความยืดหยุ่น

ที่มา: ผู้วิจัย, 2567

ในระบบ GAISS ผู้วิจัยใช้เทคโนโลยีคลาวด์ที่ระดับการสร้างความแตกต่างของฮาร์ดแวร์ (HAL) เพื่อให้ง่ายต่อการปรับขยายของทรัพยากรเช่นพื้นที่จัดเก็บและกำลังประมวลผล ความยืดหยุ่นของระบบช่วยให้ระบบสามารถปรับตัวกับการโหลดที่เปลี่ยนแปลงได้โดยไม่ต้องมีการปรับเปลี่ยนในฮาร์ดแวร์ที่เป็นเอกลักษณ์ เครื่องมือตรวจสอบฮาร์ดแวร์เป็นส่วนสำคัญของชั้นนี้ โดยให้ข้อมูลทันทีเกี่ยวกับสถานะการทำงานและสุขภาพของอุปกรณ์ทางกายภาพ เครื่องมือเหล่านี้เป็นสิ่งสำคัญในการรักษาความเชื่อถือของระบบและปรับปรุงการใช้พลังงานให้เหมาะสม องค์ประกอบที่สำคัญเพิ่มเติมคือการนำเสนอเครื่องเสมือน (VMs) VMs ให้สภาพแวดล้อมที่ยืดหยุ่นโดยอนุญาตให้การดำเนินการของหลายรายการของระบบปฏิบัติการทำงานพร้อมกันบนเครื่องกลางซึ่งเป็นอุปกรณ์ทางกายภาพเดียว คุณสมบัตินี้เพิ่มความยืดหยุ่นและประสิทธิภาพในการใช้ทรัพยากรฮาร์ดแวร์ ซึ่งทำให้ประสิทธิภาพรวมของโครงสร้าง GAISS ดียิ่งขึ้น

ในโครงสร้าง GAISS ชั้นของระบบซอฟต์แวร์ประกอบด้วยระบบปฏิบัติการที่รับผิดชอบในการจัดการทรัพยากรฮาร์ดแวร์และการให้บริการต่อซอฟต์แวร์ที่กำลังทำงาน เป็นตัวอย่าง เช่น ระบบจัดการฐานข้อมูล (Database Management Systems หรือ DBMS) เป็นสิ่งสำคัญในการเก็บรักษา การเรียกข้อมูล การบริหารจัดการ และการปรับเปลี่ยนข้อมูล พวกเขารักษาความมั่นคงและความสมบูรณ์ของข้อมูล ในทางตรงกันข้าม ระบบปฏิบัติการเครือข่าย (Network Operating Systems หรือ NOS) มุ่งเน้นไปที่การจัดการเฉพาะกับการเชื่อมต่อเครือข่าย ไดรเวอร์ช่วยให้การสื่อสารระหว่างอุปกรณ์ฮาร์ดแวร์กับระบบปฏิบัติการ ซอฟต์แวร์โปรแกรมช่วยเสริมชุดฟังก์ชั่นของระบบ รวมถึงการตรวจสอบระบบ การสำรองข้อมูล และบริการด้านความปลอดภัย เหล่านี้เป็นสิ่งจำเป็นสำหรับโครงสร้าง GAISS เพื่อรักษาความสมบูรณ์ของระบบและประสิทธิภาพในการดำเนินการ

ชั้น Middleware/Service ในโครงสร้าง GAISS ประกอบด้วยส่วนสำคัญหลายอย่าง ระบบสำรองข้อมูล เช่น มีความสำคัญในการรับประกันความสมบูรณ์ของข้อมูลและให้การช่วยเหลือในกระบวนการกู้คืน เซิร์ฟเวอร์ MS IIS ในอีกทางหนึ่ง รับผิดชอบในการเป็นโฮสต์ของแอปพลิเคชันเว็บและประมวลผลคำขอ ซึ่งมีส่วนสำคัญต่อความยืดหยุ่นและความปลอดภัยของเว็บเซอร์วิส คอนเทนเนอร์ ในฐานะทางเลือกแทนเครื่องเสมือน มีการประกอบแอปพลิเคชันและความขึ้นอยู่กับมันในสภาพแวดล้อมการเรียกใช้พกพา เป็นการทำให้ง่ายต่อการปรับขนาดและการเรียกใช้งานที่เกี่ยวข้อง คอมโพเนนต์ของ Service-Oriented Architecture (SOA) ในชั้นนี้ช่วยในการแยกแอปพลิเคชันออกเป็นบริการที่ย่อยลง ซึ่งสามารถผสมผสานกันได้อย่างไม่มีช่องว่าง ทำให้มีความยืดหยุ่นและสามารถใช้ซ้ำได้ในโครงสร้าง GAISS ได้

ชั้นความยั่งยืนในโครงสร้าง GAISS เป็นการยืนยันถึงความมุ่งมั่นในการส่งเสริมความมีประสิทธิภาพทางพลังงานที่ยั่งยืน ที่ตั้งในตำแหน่งกลางสำคัญระหว่างชั้นการทำงานและชั้นด้านเอสเป็กต์และชั้นบริการกลาง ชั้นนี้รวมหลักการที่ยั่งยืนเข้าร่วมกับบริการกลางที่ใช้งานและโลจิกธุรกิจพื้นฐานของแอปพลิเคชัน โดยการปฏิบัติตามกฎหมายที่มีประสิทธิภาพทางพลังงานและการวัดค่าสมรรถภาพที่ยั่งยืนตลอดชั้นด้านบนของโครงสร้าง ชั้นความยั่งยืนรับประกันการรวมหลักการที่เชื่อมโยงกับความยั่งยืนเข้ากับการดำเนินงานประจำวันของระบบ มันไม่เพียงแค่การปรับปรุงการใช้พลังงาน เองแต่มันยังใช้วิธีการขั้นสูงและการวิเคราะห์ทำนายเพื่อปรับการจัดสรรทรัพยากรอย่างเคลื่อนไหวและการคาดการณ์โหลดที่จะเกิดขึ้น ซึ่งจะทำให้ประสิทธิภาพและความพร้อมของระบบ

ดีขึ้น มันทำงานร่วมกับชั้นการสร้างขั้นตอนการใช้ทรัพยากรคลาวด์และเสมือนจริงเพื่อรับประกันการใช้ประโยชน์ที่มีประสิทธิภาพที่สุด ซึ่งจะลดการสูญเสียและส่งเสริมให้นำเอาแหล่งพลังงานที่เป็นพลังงานทดแทนเมื่อเป็นไปได้ นอกจากนี้ มันยังดูแลการปฏิบัติตามมาตรฐานกฎระเบียบและวัตถุประสงค์ทางสิ่งแวดล้อมของระบบ โดยพิจารณาการรายงานความยั่งยืนและส่งเสริมการใช้ปฏิบัติการที่เขียวระดับไอทีเช่นการรีไซเคิลของขยะอิเล็กทรอนิกส์ ด้วยการใช้กลยุทธ์ที่ครอบคลุมทั้งหมดเหล่านี้ ชั้นความยั่งยืนส่งเสริมให้ปฏิบัติตามวัตถุประสงค์ด้านความรับผิดชอบต่อสังคม (CSR) และช่วยให้ควบคุมค่าใช้จ่ายในการดำเนินงานได้อย่างมีประสิทธิภาพ พร้อมทั้งปฏิบัติตามนโยบายสิ่งแวดล้อมที่เข้มงวดมากขึ้น ดังนั้น จึงเสริมแนวคิดในการรักษาความยั่งยืนในธุรกิจอย่างต่อเนื่อง

ชั้นการทำงานและด้านเอสเป็กต์ในโครงสร้าง GAISS มีวัตถุประสงค์ที่แตกต่างกัน "Functional Components" หมายถึงส่วนประกอบที่จัดการกับตรรกะธุรกิจพื้นฐานที่เป็นเอกลักษณ์ของ GAISS นั่นเอง ชั้นนี้รวมถึงการบริหารจัดการแอปพลิเคชัน, การประมวลผลการชำระเงิน, และการจัดการข้อมูลผู้ใช้งาน ในทางกลับกัน "Aspectual Components" ซึ่งใช้เทคโนโลยีการเขียนโปรแกรมด้านมุมมอง (Aspect-Oriented Programming Methodologies) จะเน้นการแก้ไขปัญหาที่เกี่ยวข้องกับด้านต่าง ๆ ของแอปพลิเคชันต่าง ๆ ที่เกินกว่าโดเมนแอปพลิเคชันเดียว ๆ ประเด็นเหล่านี้รวมถึงการบันทึกเหตุการณ์ (Logging), การจัดการข้อผิดพลาด (Error Handling), ความปลอดภัย (Security), และการจัดการธุรกรรม (Transaction Management) ส่วนประกอบด้านมุมมอง (Aspectual Components) มีความสามารถในการเพิ่มประสิทธิภาพในการบำรุงรักษาตรรกะองค์กรและประสิทธิภาพรวมของโครงสร้างของ GAISS โดยการให้ความสำคัญต่อประเด็นเหล่านี้โดยตลอดโดยไม่เสียความเรียบร้อย โดยไม่ก่อให้เกิดความสับสนใด ๆ ในโครงสร้าง

ชั้นการเข้าถึงข้อมูลในโครงสร้างของ GAISS เป็นส่วนสำคัญในการเชื่อมต่อแอปพลิเคชันธุรกิจกับฐานข้อมูลทางกายภาพ หน้าที่หลักของมันคือการให้ความมั่นใจในการดึงข้อมูลและจัดเก็บข้อมูลอย่างมีประสิทธิภาพ ชั้นข้อมูลการเข้าถึงทำสิ่งนี้โดยการปรับปรุงการจัดการข้อมูลเพื่อให้มีประสิทธิภาพทางพลังงานและประสิทธิภาพในการทำงาน โดยการลดภาระของระบบอย่างมีประสิทธิภาพและการปรับปรุงความตอบสนอง ชั้นข้อมูลการเข้าถึงมีส่วนสำคัญต่อประสิทธิภาพรวมของโครงสร้าง GAISS อย่างมีนัยสำคัญ

สรุปได้ว่า ชั้นอินเตอร์เฟซเป็นเหมือนประตูเข้าใช้งานที่ผู้ใช้สื่อสารกับ GAISS ผ่านทางนี้ได้ ด้วยการออกแบบที่ ใช้งานง่ายและ ตอบสนอง ชั้นนี้แสดงถึงการทุ่มเทในการออกแบบโดยให้ผู้ใช้เป็นศูนย์กลางอย่างต่อเนื่อง โดยการปรับปรุงประสบการณ์และการโต้ตอบของผู้ใช้ในหลายแพลตฟอร์มและอุปกรณ์ ผู้วิจัยมั่นใจว่าผู้ใช้จะรู้สึกถึงความคุ้มค่าและได้รับการให้บริการอย่างเหมาะสม เพิ่มเติมเสริมที่นี่ ชั้นนี้มีบทบาทสำคัญในการลดการใช้พลังงานที่เกี่ยวข้องกับการประมวลผลที่ตรงข้ามของไคลเอ็นต์ โดยการนำเสนอเทคนิคการเขียนโค้ดที่เรียบง่ายเพื่อลดการถ่ายโอนข้อมูลและระยะเวลาการโหลด โดยการให้ความสำคัญกับความเป็นมิตรกับผู้ใช้ ผู้วิจัยไม่เพียงเพิ่มประสบการณ์การใช้งานที่ดีขึ้นโดยรวมเท่านั้น แต่ยังแสดงให้เห็นถึงการเคารพต่อเวลาและความพยายามของผู้ใช้ด้วย

4.3 ความท้าทายของความยืดหยุ่นและการปรับตัว

วิธีการ ALAI เป็นวิธีการที่มีประสิทธิภาพที่ช่วยให้นักพัฒนาซอฟต์แวร์สามารถเพิ่มความยืดหยุ่นและความสามารถในการปรับตัวของระบบของพวกเขาได้ ซึ่งเกินไปกับการเป็นแค่แนวคิดทฤษฎีเท่านั้น โดยการให้ความสำคัญกับพื้นฐานเหล่านี้ ทฤษฎี ALAI นำเสนอโครงสร้างที่เชิงปฏิบัติสำหรับการพัฒนาซอฟต์แวร์ที่ยืดหยุ่นสำหรับเทคโนโลยีและความต้องการที่กำลังเปลี่ยนแปลงไปอย่างต่อเนื่อง ในการวิเคราะห์นี้ ผู้วิจัยจะสำรวจวิธีการที่เฉพาะเจาะจงของ ALAI ในการจัดการกับแนวคิดเกี่ยวกับความยืดหยุ่นและความสามารถในการปรับตัวได้อย่างเฉพาะตัว

หนึ่งในประโยชน์สำคัญของ ALAI คือวิธีการที่ใช้งานง่ายในการแยกปัญหาออกจากกัน การใช้ ALAI AOF ช่วยเสริมให้นักพัฒนาสามารถปรับเปลี่ยนหรือเพิ่มเติมส่วนประกอบของระบบที่เฉพาะเจาะจง เช่น การเข้าถึงข้อมูล การค้นหา การอัพเดต การลบ การเรียงลำดับ การเก็บข้อมูลล็อก และการจัดการข้อผิดพลาด โดยไม่มีผลกระทบต่อตรรกะในแอปพลิเคชันหลัก การแยกส่วนของคอมโพเนนต์นี้ไม่เพียงทำให้การจัดการรหัสเป็นเรื่องง่ายขึ้นเท่านั้น แต่ยังเพิ่มความยืดหยุ่นของระบบด้วย นักพัฒนาสามารถทำการเปลี่ยนแปลงที่ส่วนหนึ่งๆ ของระบบโดยไม่ต้องกังวลเกี่ยวกับผลกระทบที่เกิดขึ้นกับส่วนอื่น ๆ ทำให้ลดความกังวลเกี่ยวกับการทำให้ระบบทั้งหมดเกิดการขัดขวางได้

ความยืดหยุ่นไม่ใช่เพียงเป็นเป็นเป็นไปได้เท่านั้น แต่เป็นความเป็นจริงกับวิธีการที่มีโครงสร้างแบบชั้นของ ALAI วิธีการนี้แบ่งแอปพลิเคชันเป็นชั้นย่อย (เช่น การนำเสนอ ตรรกะธุรกิจ การเข้าถึงข้อมูล) แต่ละชั้นมีหน้าที่เฉพาะของตน การแยกชั้นนี้ช่วยให้สามารถนำเข้าภาพหรือการอัปเดตได้ในชั้นเดียวโดยไม่มีผลต่อการทำงานของชั้นที่เหลือ เพื่ออธิบายเช่นนี้ สามารถนำเข้าภาพการเปลี่ยนแปลงในชั้นการนำเสนอได้ โดยไม่ทำให้เกิดความขัดขวางในชั้นตรรกะธุรกิจหรือการเข้าถึงข้อมูล ซึ่งจะไม่เพียงเพิ่มประสิทธิภาพแต่ยังเป็นการปฏิวัติในกระบวนการพัฒนาโดยเรียบง่าย ความยืดหยุ่นเป็นด้านสำคัญของ ALAI และมันถูกบรรลุผ่านการใช้เทคโนโลยี Agile Methodologies วิธีการเหล่านี้ซึ่งรวมถึงการพัฒนาแบบอิเตอร์เรทีฟ การให้คำแนะนำต่อไป และการกำหนดลำดับของงาน อยู่ในสามัญของ ALAI พวกเขาช่วยให้เป็นไปได้อย่างรวดเร็วที่จะทำการปรับเปลี่ยนขอบเขตของโปรเจ็กต์ คุณสมบัติ และลำดับความสำคัญ ในโปรเจ็กต์ล่าสุด ผู้วิจัยสามารถปรับเปลี่ยนความต้องการจากผู้มีส่วนได้เป็นอย่างมาก โดยยังรักษาการพัฒนาโดยไม่ขัดขวาง ความสามารถในการปรับตัวให้เหมาะกับความต้องการที่เปลี่ยนแปลงของผู้มีส่วนได้และสถานการณ์ในตลาด เป็นสิ่งสำคัญเพราะมันช่วยให้กระบวนการพัฒนาสามารถปรับตัวตามความจำเป็นได้โดยไม่เผชิญกับอุปสรรคที่สำคัญ

วงจรการพัฒนาที่สามารถปรับตัวได้สามารถส่งเสริมความยืดหยุ่นได้ ส่วนประกอบของ ALAI รับประกันกระบวนการพัฒนาที่ ตอบสนองและ ปรับตัวได้โดยการนำเสนอการตรวจสอบและปรับปรุงเป็นระยะๆ หลังจากทุกรอบการพัฒนา ความสามารถในการปรับตัวเป็นสิ่งสำคัญในการพัฒนาผลิตภัณฑ์ในขณะที่มีแนวโน้มใหม่ๆและความคิดเห็นจากผู้ใช้ที่เกิดขึ้น การออกแบบแบบโมดูลเป็นไปอย่างง่ายดายด้วยการใช้ AOF และหลักการ การแบ่งชั้นซึ่งเป็นการห่อหุ้มคอมโพเนนต์หรือฟังก์ชันในโมดูลหรือด้านที่แยกออกอย่างชัดเจน เพื่อให้เข้าใจง่าย ผู้วิจัยสามารถใช้ AOF ในการห่อหุ้มความกังวลที่ตัดกันได้ ในทางตรงกันข้าม หลักการ การแบ่งชั้นกำหนดหน้าที่ที่แตกต่างกันให้แต่ละชั้น เพิ่มความยืดหยุ่นของระบบต่อการเปลี่ยนแปลงของเทคโนโลยีหรือความต้องการที่กำลังเกิดขึ้น

ขั้นตอนการทำงานที่แสดงในรูปที่ 4.3 แสดงให้เห็นว่าวิธีการ Agile และ ALAI ในพอร์ทัลบริการช่วยเพิ่มความยืดหยุ่นและความสามารถในการปรับตัวอย่างมาก

รูปที่ 4.3 ภาพประกอบเกี่ยวกับ Portal ของ GAISS

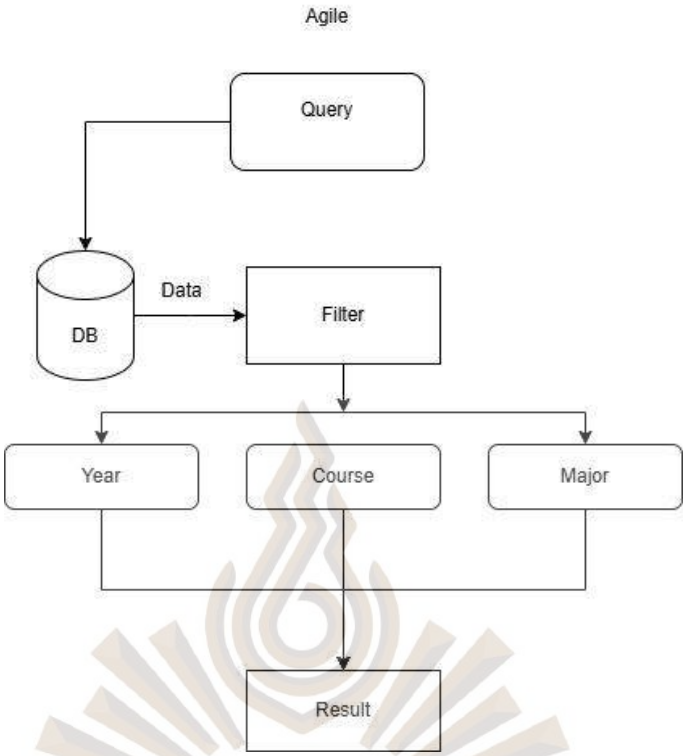
ที่มา: ผู้วิจัย, 2567

ขั้นตอนการทำงานที่แสดงในภาพ จากคำอธิบายของรูปที่ 4.4 ซึ่งแสดงขั้นตอนการทำงานในลักษณะของการจัดการข้อมูลรายชื่อนักศึกษา โดยเปรียบเทียบระหว่าง Agile และ ALAI สามารถสรุปได้ดังนี้: ส่วนของ Agile ลักษณะการทำงาน: การแสดงข้อมูลเป็นแบบแสดง "ข้อมูลรายชื่อนักศึกษาทั้งหมด" ในหน้าเว็บโดยตรง ,ไม่มีการกรองหรือจัดหมวดหมู่ล่วงหน้า ,เหมาะสำหรับการดูข้อมูลทั้งหมดอย่างรวดเร็วหรือสำหรับผู้ใช้งานที่ต้องการดูข้อมูลในภาพรวม ข้อดี:ง่ายต่อการพัฒนาและใช้งาน , ข้อมูลแสดงผลทันทีโดยไม่ต้องตั้งค่าก่อน ข้อเสีย:อาจไม่เหมาะสำหรับผู้ใช้งานที่ต้องการข้อมูลเฉพาะเจาะจง , การโหลดข้อมูลทั้งหมดอาจทำให้ระบบทำงานช้าหรือโหลดช้าในกรณีที่มีข้อมูลจำนวนมาก ส่วนของ ALAI ลักษณะการทำงาน:เพิ่มตัวเลือก "การกรองข้อมูล" ที่ผู้ใช้สามารถเลือก Aspect Concerns เช่น Year (ปีการศึกษา), Course (หลักสูตร), Major (สาขาวิชา), Status (สถานะ) เมื่อผู้ใช้เลือกตัวกรองที่ต้องการแล้ว ระบบจะส่งคำขอ (Get Request) เพื่อดึงข้อมูลที่กรองแล้วมาแสดงผล ข้อดี: ลดปริมาณข้อมูลที่แสดงผล โดยแสดงเฉพาะข้อมูลที่ตรงตามเงื่อนไขของผู้ใช้ ช่วยเพิ่มความรวดเร็วในการค้นหาและแสดงผลข้อมูล เหมาะสำหรับการใช้งานที่ต้องการข้อมูลเฉพาะหรือเจาะจง การเปรียบเทียบ Agile กับ ALAI Agile: เหมาะสำหรับการใช้งานที่เน้นความง่ายและการเข้าถึงข้อมูลแบบรวดเร็ว ALAI: เหมาะสำหรับการใช้งานที่ต้องการความแม่นยำในการเลือกข้อมูลและลดภาระของระบบในการประมวลผลข้อมูลจำนวนมาก

รูปที่ 4.4 หน้าเว็บรายชื่อนักศึกษาโดยใช้ Agile ทางซ้าย และ ALAI ทางขวา

ที่มา: ผู้วิจัย, 2567

ขั้นตอนการทำงานที่แสดงในรูปที่ 4.5 คือการอธิบายขั้นตอนการทำงานโดยใช้ Agile โดยการทำงานเริ่มต้นที่ Query ซึ่งอาจเป็นการร้องขอข้อมูลที่ผู้ใช้งานหรือระบบภายในส่งมา Query ถูกใช้เพื่อเรียกข้อมูลจากฐานข้อมูล (Database) จากนั้นข้อมูลจาก DB เป็นแหล่งข้อมูลหลักที่เก็บข้อมูลทั้งหมดที่เกี่ยวข้องเช่น ข้อมูลเกี่ยวกับปี ,หลักสูตร, และสาขาวิชา เมื่อ Query ส่งคำร้องมาข้อมูลที่ตรงกับคำร้องจะถูกส่งออกมาในรูปแบบ Data ขบวนการถัดไปเป็นการ Filter ข้อมูลที่ได้จากฐานข้อมูลจะถูกส่งไปคัดกรอง การคัดกรองมีหน้าที่จัดการข้อมูล เช่น การตัดข้อมูลที่ไม่เกี่ยวข้องหรือการแยกข้อมูลตามเกณฑ์ที่กำหนด ซึ่งสามารถแยกการคัดกรองข้อมูลออกเป็น 3 หมวดหมู่หลัก คือ Year(ปี การศึกษา) จัดหมวดหมู่ข้อมูลตามช่วงปี Course(หลักสูตร) จัดการกลุ่มข้อมูลตามประเภทของหลักสูตร Major(สาขา) กำหนดหมวดหมู่สาขาวิชา เมื่อข้อมูลที่ถูกแยกและประมวลผลจากทั้ง 3 หมวดหมู่ จะถูกนำมารวบรวมเพื่อสร้าง ผลลัพธ์(Result) ผลลัพธ์นี้สามารถเป็นข้อมูลที่พร้อมใช้งานหรือส่งกลับไปยังผู้ใช้งานตามคำร้อง กระบวนการนี้ออกแบบมาเพื่อให้จัดการข้อมูลมีประสิทธิภาพและช่วยตอบสนองความต้องการของผู้ใช้งานหรือระบบอย่างรวดเร็วและเป็นระบบในการออกแบบ Agile Framework



รูปที่ 4.5 ขั้นตอนการทำงานโดยใช้ Agile

ที่มา: ผู้วิจัย, 2567

ขั้นตอนการทำงานที่แสดงในรูปที่ 4.6 ภาพนี้แสดงถึงกระบวนการจัดการข้อมูลที่มีลำดับขั้นตอน ตั้งแต่การกรองข้อมูล การจัดหมวดหมู่ข้อมูล การวิเคราะห์ประเด็นหลัก และการส่งคำสั่งค้นหาไปยังฐานข้อมูล (DB) โดยมีองค์ประกอบสำคัญดังนี้

4.6.1 Filter: กรองข้อมูลดิบให้เหลือเฉพาะส่วนที่เกี่ยวข้อง

4.6.2 Aspect: จัดข้อมูลที่กรองมาให้อยู่ในลักษณะหรือหมวดหมู่เฉพาะ ทำหน้าที่แบ่งประเภทหรือจัดกลุ่มข้อมูลที่ผ่านการกรองจาก Filter เพื่อจัดระเบียบข้อมูลก่อนนำไปวิเคราะห์ ลักษณะการทำงาน รับข้อมูลที่กรองแล้ว ระบุลักษณะหรือหมวดหมู่ของข้อมูล เช่น ประเภทของคำถามหรือประเด็นที่ผู้ใช้งานสนใจ ส่งข้อมูลที่จัดหมวดหมู่แล้วไปยัง Query เพื่อสร้างคำสั่งค้นหา

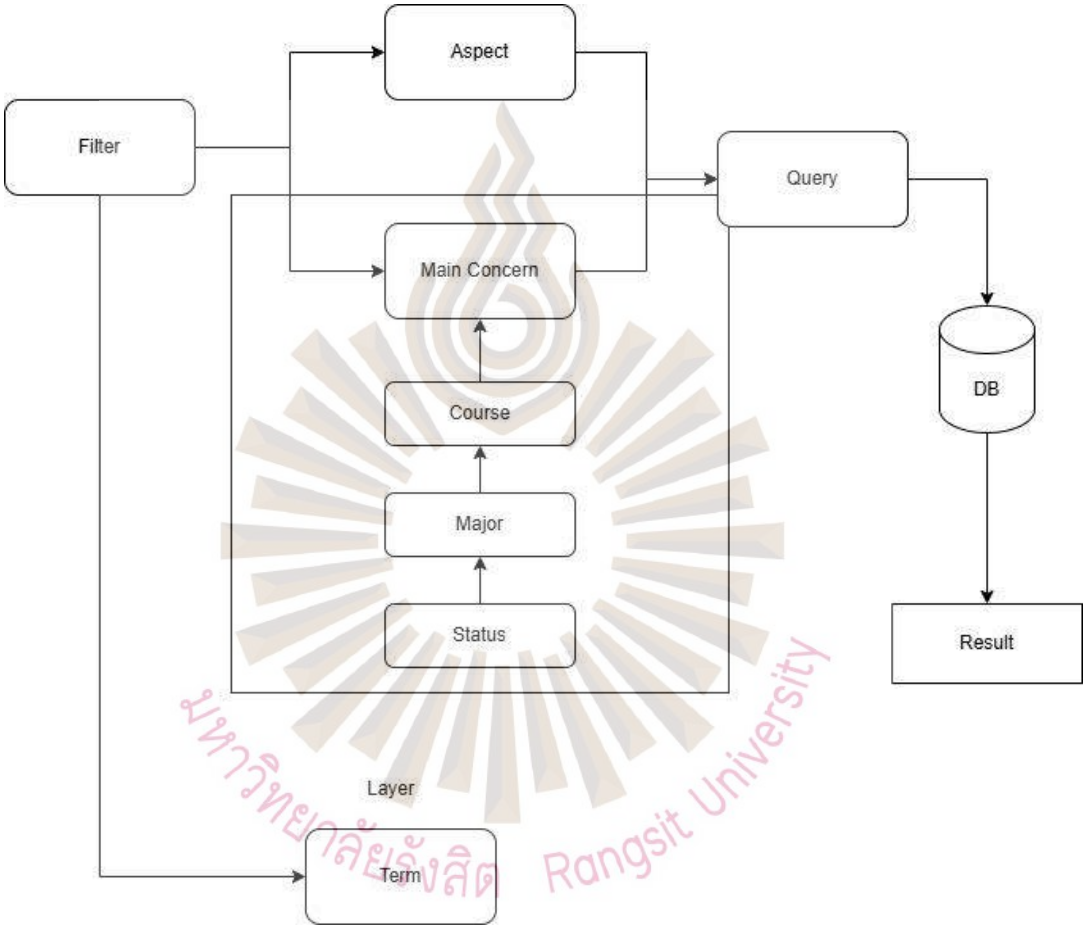
4.6.3 Main Concern: โฟกัสไปที่ข้อมูลหลักที่ต้องการ โดยมีโครงสร้างย่อย (Status ⟶ Major ⟶ Course) โดยถูกนำมาวิเคราะห์เพื่อสรุปและส่งต่อข้อมูลนี้ไปยัง Query เพื่อใช้ในการค้นหาข้อมูลเป็นลำดับต่อไป

4.6.4 Query: ส่งคำสั่งค้นหาที่กำหนดเงื่อนไขไปยังฐานข้อมูลโดยการรวบรวมการส่งข้อมูลได้มาจากในส่วน Main Concern

4.6.5 DB: ฐานข้อมูลที่เก็บข้อมูลทั้งหมดและทำหน้าที่ตอบกลับคำสั่ง

4.6.6 Result: ผลลัพธ์จากการค้นหาข้อมูล ผลลัพธ์นี้สามารถเป็นข้อมูลที่พร้อมใช้งานหรือส่งกลับไปยังผู้ใช้งานตามคำร้อง

4.6.7 Layer และ Term: ใช้เสริมการค้นหาโดยกำหนดคำหรือเงื่อนไขที่เฉพาะเจาะจง



รูปที่ 4.6 ขั้นตอนการทำงานโดยใช้ ALAI

ที่มา: ผู้วิจัย, 2567

นี่เป็นตัวอย่างของระบบ การเตรียมความพร้อมก่อนที่ใช้ Agile ซึ่งสนับสนุนการปรับเปลี่ยนอย่างรวดเร็ว คุณลักษณะที่สำคัญสำหรับระบบที่จำเป็นต้องปรับปรุงอยู่เสมอตามนโยบายของมหาวิทยาลัยที่กำลังเปลี่ยนแปลง หรือคำแนะนำจากผู้ใช้ที่กำลังเปลี่ยนแปลงอยู่ตลอดเวลา โดยใช้วิธีการ Agile แบบดั้งเดิม ระบบ Preadmission สามารถปรับตัวอย่างรวดเร็วเพื่อเข้ากับความต้องการที่กำลังเปลี่ยนแปลงอย่างรวดเร็วโดยไม่จำเป็นต้องทำการพัฒนาใหม่ในระดับที่

มีการเปลี่ยนแปลงมากมาย อย่างตรงข้ามกับนั้น บริการทางเลือกเช่น GradAdmission และ GE Document ได้รับประโยชน์จากวิธีการ ALAI ซึ่งเป็นเครื่องมือที่แข็งแกร่งที่รวมวิธีการ Agile ที่ขยายตัวแล้วด้วยความยืดหยุ่นในสถาปัตยกรรมที่ดีขึ้นผ่าน AOF และการแบ่งชั้นการทำงาน เพื่อให้เข้ากับความต้องการและเปลี่ยนแปลงที่เกิดขึ้นได้อย่างมีประสิทธิภาพ การให้นักพัฒนาระบบสามารถปรับแต่งหรือขยายชั้นต่างๆได้อย่างอิสระ เช่น ตรรกะธุรกิจ การนำเสนอข้อมูล และการเข้าถึงข้อมูล วิธีการนี้ช่วยเสริมความยืดหยุ่นและปรับขนาดของระบบได้อย่างมีประสิทธิภาพ

AOF, ซึ่งเป็นหลักการสำคัญของวิธี ALAI, เป็นสิ่งจำเป็นในการรักษาประสบการณ์การใช้งานที่ถูกผสมผสานและเป็นอันเป็นเดียวกันของผู้ใช้ในพอร์ทัลทั้งหมดสถาปัตยกรรมนี้ช่วยให้สามารถสร้างและบริหารจัดการคุณลักษณะที่ตัดกันข้ามเช่นการบันทึกข้อมูลและความปลอดภัยได้อย่างอิสระ ซึ่งจะป้องกันการขัดข้องใด ๆ ต่อการดำเนินการหลักของแอปพลิเคชัน กรอบการออกแบบนี้ช่วยให้สามารถพัฒนาแบบรอบรู้ของทุกบริการได้ รวมถึงแอปพลิเคชัน Open House และระบบข้อมูลนักศึกษา โดยการทำตามความก้าวหน้าทางเทคโนโลยีและข้อเสนอแนะจากผู้ใช้ด้วยเหตุนี้ มันก็สร้างพื้นฐานที่มั่นคงสำหรับการปรับปรุงอย่างต่อเนื่อง วิธีการนี้จะให้ความสามารถที่กำหนดเองเพื่อตอบสนองความต้องการที่เฉพาะเจาะจง เช่น การแพร่ข้อมูลความรู้หรือการจัดการเอกสารที่ซับซ้อน นอกจากนี้ วิธี ALAI ยังมีส่วนในการปรับปรุงความพร้อมใช้งานของระบบ ความพึงพอใจของผู้ใช้ และการจัดส่งบริการในองค์กร

สถาปัตยกรรมของ GAISS แสดงให้เห็นถึงความสามารถของวิธี ALAI และ Agile โดยส่วนประกอบพื้นฐานของระบบ ซึ่งเป็นฮาร์ดแวร์และชั้นการแยกฮาร์ดแวร์ รับรองความยืดหยุ่นในการขยายขนาดและความอิสระในการดำเนินการจากการกำหนดค่าฮาร์ดแวร์เฉพาะโดยระบบคุณลักษณะ คุณลักษณะนี้มีความสำคัญสำหรับการใช้งานการขยายขนาดและการก้าวหน้าทางเทคโนโลยีโดยไม่ต้องการการออกแบบระบบใหม่ทั้งระบบอย่างละเอียด ชั้นโปรแกรมประยุกต์และชั้นกลางซอฟต์แวร์เป็นส่วนสำคัญในการให้บริการแบบกระจายของระบบโดยการรับรองความปลอดภัยและประสิทธิภาพในการสื่อสารข้อมูลและการจัดการทรัพยากรที่มีอยู่ ผ่านการออกแบบแบบโมดูลและการแบ่งส่วนของความสนใจ โครงสร้างแอปพลิเคชันและชั้นตรรกะธุรกิจช่วยให้การพัฒนาเร็วขึ้นและการปรับเปลี่ยนตามความต้องการที่กำลังเปลี่ยนแปลงไปทางที่ง่ายดาย AOF

ใช้ในการจัดการกับปัญหาที่ตัดกันกัน เช่น การบันทึกข้อมูลและความปลอดภัยโดยอิสระจากตัวควบคุมหลักของตรรกะธุรกิจ

สถาปัตยกรรมของ GAISS ถูกออกแบบมาเพื่อเพิ่มประสิทธิภาพของระบบในสภาวะที่มีความต้องการสูง ชั้นการเข้าถึงข้อมูล เป็นส่วนสำคัญที่รับผิดชอบในการปรับปรุงปฏิสัมพันธ์กับฐานข้อมูลเพื่อเสริมประสิทธิภาพของระบบ ในทางกลับกัน ชั้นของอินเตอร์เฟซผู้ใช้ หรือ UI มุ่งเน้นการให้ประสบการณ์การใช้งานที่เข้าใจง่ายและตอบสนองได้รวดเร็ว ซึ่งได้รับการปรับแต่งให้เหมาะสมกับความต้องการของกลุ่มผู้ใช้ต่าง ๆ โดยการจัดโครงสร้างชั้นอย่างละเอียด ฟังก์ชันที่มีอยู่ในปัจจุบันจะได้รับการสนับสนุน และระบบจะมีการเตรียมพร้อมสำหรับการก้าวหน้าทางเทคโนโลยีในอนาคตและการเปลี่ยนแปลงในความต้องการของผู้ใช้ เช่น การสนับสนุนหลายภาษาอย่างไดนามิก โดยการรวมวิธีการ Agile และ ALAI เข้าไว้ในบริการต่าง ๆ เช่น Preadmission, GradAdmission, และ KM Knowledge, ระบบสามารถเป็นยืดหยุ่นได้ตามความเปลี่ยนแปลงของนโยบายการศึกษาและข้อมูลจากผู้ใช้ได้อย่างเป็นไปตามธรรมชาติ วิธีการเชิงกลยุทธ์นี้รับรองถึงความยืดหยุ่นในการขยายขนาดและความทนทานของระบบ ความยืดหยุ่นและความสามารถในการปรับตัวของพอร์ทัลถูกรับรองว่าสามารถให้บริการแต่ละส่วนให้สอดคล้องกับความต้องการเฉพาะโดยมีประสบการณ์การใช้งานที่เป็นไปตามหลักและสม่ำเสมอได้ถูกเก็บไว้ในระบบ ALAI เป็นอุปกรณ์ป้องกันระบบให้พร้อมกับการพัฒนาเทคโนโลยีที่กำลังจะเกิดขึ้นและการเปลี่ยนแปลงของความต้องการของผู้ใช้ในอนาคต โดยทำให้เกิดการแยกความกังวลอย่างชัดเจนและสถาปัตยกรรมแบบโมดูลได้ การออกแบบสถาปัตยกรรมพิจารณาความต้องการในอนาคตและสนับสนุนการปรับตัวอย่างง่ายดาย ทำให้ซอฟต์แวร์มีความคงทนและยังคงมีความสามารถในการใช้งานต่อไปได้อย่างต่อเนื่อง

4.4 ALAI เปรียบเทียบกับการพัฒนาซอฟต์แวร์แบบ Agile

ในส่วนนี้จะนำเสนอผลลัพธ์ทางปริมาณครับ ในโดเมนของการพัฒนาซอฟต์แวร์ที่เปลี่ยนแปลงอย่างรวดเร็ว การเลือกวิธีการอาจมีผลกระทบอย่างมีนัยสำคัญต่อความยืดหยุ่น ประสิทธิภาพ และความยาวนานของโครงการซอฟต์แวร์ ทั้งวิธีการ ALAI และวิธีการ Agile แบบดั้งเดิม มีประโยชน์และอุปสรรคที่เฉพาะตัวครับ ดังนั้น นักพัฒนาควรเข้าใจเรื่องแตกต่างเหล่านี้

อย่างละเอียดก่อนกำหนดมาตรการต่อไปของพวกเขา โดยเพื่อให้ง่ายต่อการเข้าใจ ตารางที่ 6 จะนำเสนอการเปรียบเทียบอย่างละเอียดระหว่าง ALAI และวิธีการ Agile แบบดั้งเดิม โดยครอบคลุมหลายด้าน เช่น แต่ไม่จำกัดเพียงนี้: แนวคิด, ด้านการดำเนินงาน, การพิจารณาถึงความยั่งยืน, ความยืดหยุ่น, ความโมดูลาริตี้, สถาปัตยกรรมซอฟต์แวร์, ข้อจำกัด, ประโยชน์, และความเหมาะสมสำหรับโครงการขนาดต่าง ๆ การเปรียบเทียบอย่างสั้นและเข้าใจง่ายนี้มีวัตถุประสงค์เพื่อเน้นให้เห็นว่าแต่ละวิธีการจัดการกับปัญหาที่ซับซ้อนของการพัฒนาซอฟต์แวร์ในยุคปัจจุบันโดยเฉพาะและสอดคล้องกับข้อกำหนดของโครงการที่เฉพาะเจาะจง การช่วยเหลือผู้เล่นหลักนี้มีความสำคัญในการกำหนดว่าวิธีการใดเหมาะสมที่สุดในการแก้ไขความต้องการการพัฒนาที่เฉพาะเจาะจงของพวกเขา

ตารางที่ 4.1 การเปรียบเทียบระหว่างวิธีการพัฒนาซอฟต์แวร์ ALAI กับ Agile

Criteria	ALAI Methodology	Traditional Agile
Concept	รวมวิธีการ Agile กับสถาปัตยกรรมชั้นและโปรแกรมเชิงประการเพื่อเพิ่มความยั่งยืน ความยืดหยุ่น และความโมดูลาริตี้ในการพัฒนาซอฟต์แวร์	เน้นการพัฒนาแบบทดสอบซ้ำซ้อนและคำแนะนำต่อเนื่องเพื่อปรับตัวอย่างรวดเร็วต่อการเปลี่ยนแปลงและส่งมอบซอฟต์แวร์ที่ใช้งานได้ในเวลาที่รวดเร็ว
Steps	รวมถึงขั้นตอนเช่นการวางแผนเบื้องต้น การออกแบบสถาปัตยกรรมชั้น การรวมเข้าด้วยกันของโปรแกรมเชิงประการ การพัฒนาแบบซ้ำซ้อน การวิ่งการพัฒนาขั้นต่อไป การรวมระบบและการทดสอบอย่างต่อเนื่อง และการวนรอบขอคำแนะนำ	เกี่ยวข้องกับการวนรอบอย่างต่อเนื่องของการวางแผน การเขียนโค้ด การทดสอบ และการตรวจทานในช่วงเวลาที่กำหนดไว้ (สปรินท์) โดยให้ความสำคัญมากกับคำแนะนำจากผู้ใช้และการส่งมอบคุณลักษณะอย่างรวดเร็ว

ตารางที่ 4.1 การเปรียบเทียบระหว่างวิธีการพัฒนาซอฟต์แวร์ ALAI กับ Agile (ต่อ)

Criteria	ALAI Methodology	Traditional Agile
Sustainability Concerns	การเน้นความยั่งยืนโดยการปรับปรุงให้มีประสิทธิภาพในการใช้พลังงานและการจัดการทรัพยากรผ่านการออกแบบและการเลือกสถาปัตยกรรม เช่น การแยกปัญหาที่ตัดกันและการสร้างชั้นที่มีประสิทธิภาพ	ความยั่งยืนไม่ใช่จุดศูนย์หลัก แต่การปฏิบัติตาม Agile ในการลดการสูญเสียและเน้นที่คุณลักษณะที่จำเป็นอาจสนับสนุนการพัฒนาที่ยั่งยืนอย่างอ้อมค้อม
Flexibility	ความยืดหยุ่นสูงที่ได้รับจากการแยกปัญหาโดยใช้โปรแกรมเชิงประการและโดยใช้สถาปัตยกรรมชั้น ทำให้สามารถอัพเดตและขยายขนาดได้อย่างง่ายโดยไม่ต้องทำการปรับเปลี่ยนในระบบโดยรวมใหญ่	วิธี Agile เป็นอย่างมากเนื่องจากลักษณะการทำซ้ำและการตอบสนองต่อการเปลี่ยนแปลง แต่ขาดกลไกโครงสร้างที่เฉพาะเจาะจงสำหรับการจัดการการปรับเปลี่ยนขนาดใหญ่โดยง่าย
Modularity	ส่งเสริมการทำให้เป็นโมดูลได้ผ่านการใช้สถาปัตยกรรมชั้นและโปรแกรมเชิงประการ ซึ่งแยกฟังก์ชันต่าง ๆ และปัญหาที่ตัดกันออกเป็นส่วนที่แยกกันและสามารถจัดการได้แยกต่างหาก	ความโมดูลาริตี้ขึ้นอยู่กับวิธีการในการออกแบบซอฟต์แวร์ของทีม และไม่ได้รับการกำหนดเฉพาะโดยวิธีการ Agile บางครั้งการเป็นโมดูลต้องถูกนำเข้าใจและดำเนินการโดยตั้งใจ

ตารางที่ 4.1 การเปรียบเทียบระหว่างวิธีการพัฒนาซอฟต์แวร์ ALAI กับ Agile (ต่อ)

Criteria	ALAI Methodology	Traditional Agile
Software Architecture	ALAI กำหนดให้มีวิธีการออกแบบซอฟต์แวร์เฉพาะโดยการรวมชั้นและประเด็นเพื่อจัดการกับความซับซ้อนอย่างระบบโดยเป็นระบบและเพิ่มความสามารถในการบำรุงรักษาได้อย่างมีระเบียบ	วิธีการ Agile แบบดั้งเดิมไม่กำหนดสถาปัตยกรรมเฉพาะไว้ มันให้ความยืดหยุ่นให้ทีมเลือกสถาปัตยกรรมที่เหมาะสมที่สุดตามความต้องการของโครงการของพวกเขา
Advantages	ALAI ให้โครงสร้างเชิงโครงสร้างที่สามารถจัดการโครงการที่ซับซ้อนและมีความยืดหยุ่นได้อย่างมีประสิทธิภาพมากขึ้น โดยมีการแยกปัญหาอย่างชัดเจนที่ช่วยเพิ่มความสามารถในการบำรุงรักษาและอนุรักษ์การปรับปรุงเป้าหมายได้เป็นพิเศษ	วิธี Agile มีความยืดหยุ่นสูง รอบการพัฒนาอย่างรวดเร็ว และมีการมีส่วนร่วมอย่างมั่นคงจากผู้ส่วนได้ดี ซึ่งสามารถนำไปสู่ความพึงพอใจของลูกค้าสูงและการแก้ไขปัญหาทันทีได้
Project size	สามารถปรับขนาดและใช้งานได้ทั้งสำหรับโครงการขนาดเล็กและใหญ่	โดยทั่วไป เหมาะสำหรับโครงการขนาดเล็กถึงขนาดกลางมากกว่า เนื่องจากลักษณะการสร้างโปรตอไทป์ที่ต้องใช้แรงงานมาก

ที่มา: ผู้วิจัย, 2567

4.5 การวิเคราะห์อธิบาย

การประเมินอย่างครอบคลุมเกี่ยวกับการใช้พลังงานของหน้าเว็บรายชื่อนักศึกษาเป็นเรื่องสำคัญที่จำเป็นสำหรับการเข้าใจถึงประสิทธิภาพและความมีประสิทธิภาพขององค์ประกอบสำคัญนี้ภายใน GAISS ส่วนนี้ให้การวิเคราะห์อย่างละเอียดของข้อมูลบันทึกไฟล์ที่เก็บรวบรวมจาก

เซิร์ฟเวอร์ MS IIS โดยเน้นที่ตัวชี้วัดที่สำคัญ เช่น ความถี่ของการเข้าถึง ระยะเวลาในการตอบสนอง ปริมาณการถ่ายโอนข้อมูล และการกระจายของผู้ใช้ ผู้เขียนมีเป้าหมายที่จะระบุรูปแบบและแนวโน้มที่มีผลต่อการใช้พลังงานโดยการวิเคราะห์ลักษณะเหล่านี้ นอกจากนี้ การศึกษานี้ยังตรวจสอบผลกระทบของตัวแปรหลายประการ เช่น ระยะเวลาของเซสชัน รหัสสถานะ HTTP และ User Agent ต่อประสิทธิภาพพลังงานโดยรวมของหน้าเว็บรายชื่อนักศึกษา การวิเคราะห์ในปัจจุบันนี้ให้ข้อมูลเชิงลึกที่สำคัญเกี่ยวกับการทำงานของหน้าเว็บ ซึ่งช่วยชี้ให้เห็นถึงพื้นที่ที่สามารถปรับปรุงได้ ซึ่งไม่เพียงแต่จะช่วยเพิ่มประสิทธิภาพพลังงาน แต่ยังปรับปรุงประสิทธิภาพของระบบนำไปสู่การดำเนินงานที่ยั่งยืนและมีประสิทธิภาพมากขึ้น

ตารางที่ 4.2 ข้อมูลเชิงพรรณนาสำหรับการวิเคราะห์เชิงปริมาณ

Metric	Description	Value
Prospective students	2022	355
	2023	1,073
	2024	250
Registered Students	2022	319
	2023	1,054
	2024	247
System Users	Super Users	3
	Department Administrators	74
	Users	9
Access Metrics	Total Number of Accesses	946,103
	Frequency of Access (Daily Average)	1,213
Access Methods	POST	460,929
	GET	424,901
	HEAD	57,344
	OPTIONS	2,815
	PUT	58

ตารางที่ 4.2 ข้อมูลเชิงพรรณนาสำหรับการวิเคราะห์เชิงปริมาณ (ต่อ)

Metric	Description	Value
Performance Metrics	2022 Average Response Time (ms)	699.2
	2023 Average Response Time (ms)	732.5
	2024 Average Response Time (ms)	416.2
URL Path (Top 5)	/GradAdmission/RegisterList.aspx	121,080
	/LabelGen.ashx	110,778
	/admin/ManageScholarship.aspx	73,046
	/Admin/ManageAdmincourse.aspx	72,779
	/Admin/ManagePersonalResearch.aspx	60,741
Accessing User Metrics	Unique Users (IP addresses)	5,019
HTTP Status Codes	Successful Requests (200)	442,013
Mobile Device (Top 6)	Client Errors (4xx)	241,775
	Server Errors (5xx)	166,599
User Agent Browser (Top 5)	Redirection	
	Chrome 109.0.0	
	IE 7.0	
	Null	
OS (Top 5)	Chrome 114.0.0	31,336
	Chrome 108.0.0	28,418
	Null	50,115
	Windows 7	43,346
	Android 10	11,473
Energy Consumption Metrics	Average Energy Consumption per Request (Joules)	278
	Average Energy Consumption per ALAI Request (Joules)	248
	Average Energy Consumption Idle (Joules)	203

ที่มา: ผู้วิจัย, 2567

ตั้งแต่เดือนกุมภาพันธ์ 2022 ถึงเดือนพฤษภาคม 2024 การวิเคราะห์เชิงพรรณนาของหน้าเว็บรายชื่อนักศึกษาให้ข้อมูลสำคัญเกี่ยวกับการใช้พลังงาน เมตริกประสิทธิภาพ และรูปแบบการใช้งาน ข้อมูลบ่งชี้ถึงความแปรปรวนอย่างมากในจำนวนของนักศึกษาที่สนใจสมัครและนักศึกษาที่ลงทะเบียนแล้ว โดยมีการเพิ่มขึ้นอย่างเห็นได้ชัดในปี 2023 และลดลงในปี 2024 ความแปรปรวนที่กล่าวถึงนี้เน้นถึงความสำคัญของหน้าเว็บและอิทธิพลที่มีต่อพฤติกรรมของผู้ใช้ ระบบนี้เอื้อให้เกิดการจัดเรียงผู้ใช้ขั้นสูง ผู้ดูแลระบบแผนก และผู้ใช้ทั่วไปอย่างเป็นระเบียบ โดยเน้นที่ลักษณะการบริหารจัดการของแอปพลิเคชัน ข้อมูลเผยให้เห็นถึงการมีส่วนร่วมของผู้ใช้ในระดับสูง โดยมีการเข้าถึงเฉลี่ยวันละ 1,213 ครั้ง ซึ่งส่วนใหญ่ผ่านคำขอ POST (ใช้ในการส่งข้อมูลไปยังเซิร์ฟเวอร์) และคำขอ GET (ใช้ในการร้องขอข้อมูลจากเซิร์ฟเวอร์) ผลลัพธ์แสดงให้เห็นถึงการปรับปรุงประสิทธิภาพอย่างชัดเจน เนื่องจากเวลาเฉลี่ยในการตอบสนองลดลงอย่างมากจาก 732.5 มิลลิวินาทีในปี 2023 เป็น 416.2 มิลลิวินาทีในปี 2024 เว็บไซต์ที่ได้รับการเข้าชมบ่อยที่สุดรวมถึงเว็บไซต์สำหรับการลงทะเบียนและการบริหารจัดการที่สำคัญที่ผู้ใช้ใช้งาน

ด้วยจำนวน IP ที่ไม่ซ้ำกันมากกว่า 5,000 และการเข้าถึงระดับนานาชาติจากสิงคโปร์ สหรัฐอเมริกา และประเทศไทยอย่างมีนัยสำคัญ การวัดผู้ใช้ระบุถึงฐานผู้ใช้ที่หลากหลาย สถานะ HTTP ระบุถึงคำขอที่ประสบความสำเร็จมากมายนอกเหนือจากข้อผิดพลาดของลูกค้าและเซิร์ฟเวอร์ ซึ่งแสดงให้เห็นถึงพื้นที่ที่ต้องปรับปรุงอย่างชัดเจน จากข้อมูลเบราว์เซอร์และอุปกรณ์ พบว่า Chrome เป็นเบราว์เซอร์ที่ใช้บ่อยที่สุด และการใช้งานอุปกรณ์เคลื่อนที่มีความหลากหลาย จึงต้องการความเข้ากันได้กับหลายแพลตฟอร์ม ผลการวิเคราะห์การใช้พลังงานพบว่าการใช้พลังงานเฉลี่ยต่อคำขอเป็น 248 วัตต์ ในขณะที่ช่วงเวลาที่ไม่ทำงานใช้พลังงานเฉลี่ย 203 วัตต์ การใช้พลังงานนี้ย้ำความสำคัญของการปรับปรุงการใช้พลังงานทั้งในสถานะทำงานและสถานะไม่ทำงาน การปรับปรุงความมีประสิทธิภาพรวมและความยั่งยืนของระบบเป็นเรื่องสำคัญ โดยเน้นความเร่งด่วนและความสำคัญของบทบาท ข้อมูลและความเข้าใจที่รวมกันด้านบนนี้ให้ความเข้าใจอย่างละเอียดเรื่องดินแดนการดำเนินการ และยังสร้างแรงจูงใจให้ผู้วิจัยสำรวจกลยุทธ์การปรับปรุงที่เป็นไปได้สำหรับหน้าเว็บรายชื่อนักศึกษา

4.6 ผลการวิเคราะห์

การวิเคราะห์ที่เป็นจำนวนมากนี้ให้ข้อมูลการวิเคราะห์อย่างละเอียดเกี่ยวกับวิธีการพัฒนาซอฟต์แวร์ Agile และ ALAI ที่ใช้สำหรับ GAISS โดยเน้นที่ตัวชี้วัดสำคัญ เช่น จำนวนการเข้าถึง เวลาเฉลี่ยที่ใช้ รวมถึงเวลารวมที่ใช้ และการใช้พลังงานสำหรับแต่ละเดือนตั้งแต่ปี 2022 จนถึงกลางปี 2024 โดยมีข้อมูลตัวเลขที่แสดงในตารางที่ 8 เป็นหลักการศึกษารายละเอียดของผลการวิเคราะห์ ข้อมูลที่แสดงให้เห็นถึงความแตกต่างมีนัยสำคัญในการใช้งานระบบและประสิทธิภาพ โดยมีจุดสูงในจำนวนการเข้าถึง โดยเฉพาะในเดือนกรกฎาคมและสิงหาคม ปี 2022 และเมษายน ปี 2023 ซึ่งแสดงถึงช่วงเวลาที่มีความต้องการสูง เวลาตอบสนองเฉลี่ยมักมีการปรับปรุงในระหว่างปี โดยปี 2024 แสดงให้เห็นถึงประสิทธิภาพที่ดีกว่าในเปรียบเทียบกับปีก่อนหน้า การปรับปรุงนี้เป็นที่ชัดเจนจากเวลาตอบสนองเฉลี่ยที่ลดลงและเวลารวมที่ใช้ในการโหลดหน้าลดลง ซึ่งชี้ให้เห็นถึงความพยายามในการปรับปรุงที่ประสบความสำเร็จ การเปรียบเทียบการใช้พลังงานระหว่างกรอบการทำงานแบบ Agile และ ALAI ย้ำถึงความมีประสิทธิภาพที่ยอดเยี่ยมของ ALAI โดยมีการใช้พลังงานต่ำที่ต่ำกว่าคงที่ของทุกเดือนที่บันทึกไว้ ตัวอย่างเช่น ในเดือนเมษายน ค.ศ. 2023 วิธีการ Agile ใช้พลังงานประมาณ 122,888,159.03 วัตต์ ในขณะที่ ALAI ใช้พลังงานน้อยลงอย่างมีนัยสำคัญที่ 109,626,846.90 วัตต์ แนวโน้มของการใช้พลังงานต่ำกว่าด้วย ALAI ยังคงอยู่ตลอดข้อมูล ทำให้เป็นการยืนยันถึงความมีประสิทธิภาพของกรอบการทำงานในการเพิ่มประสิทธิภาพของพลังงาน ข้อมูลเหล่านี้รวมกันชี้ให้เห็นว่าการใช้กรอบการทำงาน ALAI ไม่เพียงแต่เพิ่มตัวชี้วัดประสิทธิภาพ แต่ยังมีส่วนสำคัญในการประหยัดพลังงานอย่างมาก ทำให้เป็นทางเลือกที่ยั่งยืนมากขึ้นสำหรับการพัฒนาระบบ

ตารางที่ 4.3 จำนวนการเข้าถึงรายเดือน, เวลาตอบสนองเฉลี่ย, เวลารวมที่ใช้, และการใช้พลังงานสำหรับกรอบการทำงาน Agile และ ALAI ใน GAISS ตั้งแต่เดือนกุมภาพันธ์ ค.ศ. 2022 ถึงเดือนพฤษภาคม ค.ศ. 2024

Year	Month	Accesses (Times)	Url Student-list	Average Time Taken (ms)	Total Time Taken (sec)	Agile Energy (watts)	ALAI Energy (watts)
2022	February	19	-	791.10	15.03	4,178.59	3,727.66
	March	118	-	1,346.50	158.89	44,170.59	39,403.98
	April	124	-	181.70	22.53	6,263.56	5,587.64
	May	6	-	222.50	1.34	371.13	331.08
	June	1,841	297	275.20	506.64	140,846.81	125,647.51
	July	12,861	3,936	2,240.10	28,809.93	8,009,159.46	7,144,861.67
	August	22,325	6,336	440.80	9,840.86	2,735,759.08	2,440,533.28
	September	8,598	2,345	851.60	7,322.06	2,035,531.79	1,815,870.09
	October	4,603	1,113	489.80	2,254.55	626,764.73	559,128.25
	November	18,700	5,773	355.70	6,651.59	1,849,142.02	1,649,594.32
	December	22,108	7,713	373.50	8,257.34	2,295,539.96	2,047,819.82
2023	January	21,935	8,132	332.70	7,297.77	2,028,781.31	1,809,848.08
	February	15,153	3,632	295.60	4,479.23	1,245,225.05	1,110,848.25
	March	30,779	5,398	307.50	9,464.54	2,631,142.82	2,347,206.54
	April	490,125	5,123	901.90	442,043.74	122,888,159.03	109,626,846.90
	May	19,932	5,861	359.30	7,161.57	1,990,915.79	1,776,068.76
	June	33,608	11,615	532.50	17,896.26	4,975,160.28	4,438,272.48
	July	77,859	10,525	261.00	20,321.20	5,649,293.32	5,039,657.35
	August	30,947	10,844	823.90	25,497.23	7,088,230.86	6,323,313.86
	September	18,053	4,774	658.10	11,880.68	3,302,828.85	2,946,408.47
	October	9,478	2,153	276.20	2,617.82	727,754.96	649,220.25
	November	16,927	3,649	1,097.10	18,570.61	5,162,630.05	4,605,511.70
	December	15,076	3,826	271.00	4,085.60	1,135,795.69	1,013,227.81

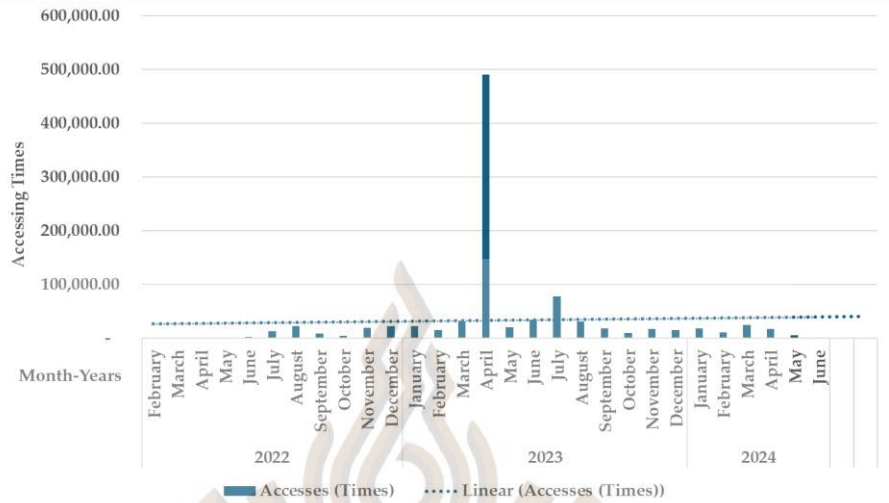
ตารางที่ 4.3 จำนวนการเข้าถึงรายเดือน, เวลาตอบสนองเฉลี่ย, เวลารวมที่ใช้, และการใช้พลังงานสำหรับกรอบการทำงาน Agile และ ALAI ใน GAISS ตั้งแต่เดือนกุมภาพันธ์ ค.ศ. 2022 ถึงเดือนพฤษภาคม ค.ศ. 2024 (ต่อ)

Year	Month	Accesses (Times)	Url Student-list	Average Time Taken (ms)	Total Time Taken (sec)	Agile Energy (watts)	ALAI Energy (watts)
2024	January	17,726	6,222	207.80	3,683.46	1,024,002.66	913,498.77
	February	10,353	3,233	363.50	3,763.32	1,046,201.71	933,302.24
	March	24,730	3,350	621.10	15,359.80	4,270,025.23	3,809,231.14
	April	17,128	3,508	299.80	5,134.97	1,427,522.88	1,273,473.65
	May	4,991	1,722	650.20	3,245.15	902,151.20	804,796.75
	June	-	-	-	-	-	-

ที่มา: ผู้วิจัย, 2567

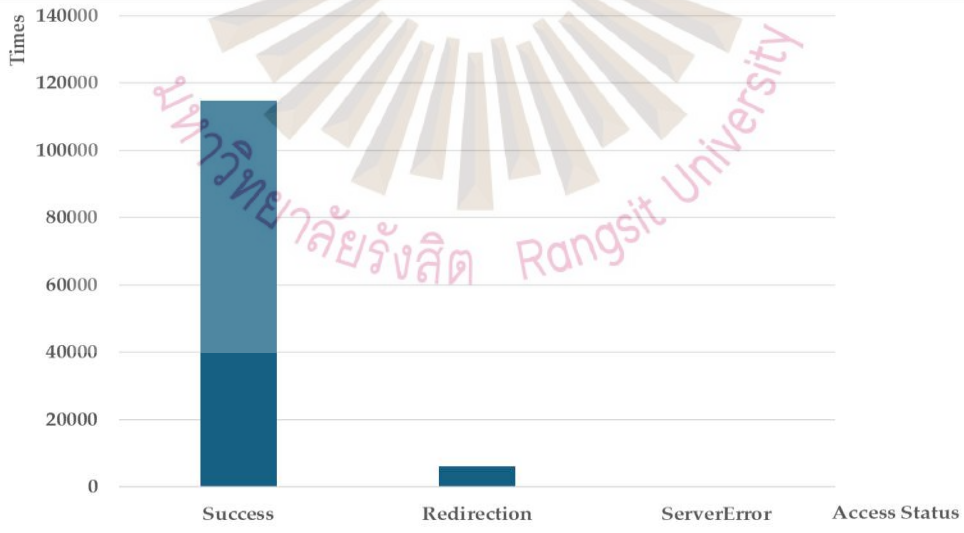
การเข้าถึงรายเดือนของ GAISS ตั้งแต่เดือนกุมภาพันธ์ ค.ศ. 2022 ถึงเดือนพฤษภาคม ค.ศ. 2024 ถูกแสดงในภาพที่ 4.5 ข้อมูลช่วยให้เข้าใจเกี่ยวกับการเปลี่ยนแปลงและแนวโน้มที่สำคัญในการใช้งานของระบบ ข้อมูลชี้ให้เห็นถึงความถี่ในการเข้าถึงที่เรียบที่สุด ยกเว้นจุดสูงที่สำคัญ โดยมีความผันผวนปานกลางในเดือนส่วนใหญ่ อย่างไรก็ตาม เดือนเมษายน ค.ศ. 2023 นั้นเป็นช่วงเวลาที่มีความต้องการสูงอย่างพิเศษ โดยมีการเข้าถึงเกือบ 500,000 ครั้ง ภาพนี้ ที่มีจำนวนการเข้าถึงสูงมากเมื่อเปรียบเทียบกับเดือนอื่น ๆ ย้ำถึงความเร่งด่วนของสถานการณ์และความจำเป็นในการให้ความสนใจทันที จุดสูงนี้ค่อนข้างแตกต่างกับความถี่ในการเข้าถึงที่โดยทั่วไปอยู่ในระดับต่ำกว่า 50,000 ในเดือนที่เหลือ ยังมีจุดสูงที่สำคัญเพิ่มเติมอีกหนึ่งจุด แม้จะน้อยลง ที่ผู้วิจัยพบในเดือนกรกฎาคม ค.ศ. 2022 ซึ่งบ่งบอกถึงการกระตุ้นในกิจกรรมที่สั้นๆ ซึ่งอาจเกี่ยวข้องกับวงจรการเรียนหรือเหตุการณ์พิเศษใดๆ การเพิ่มขึ้นเรื่อย ๆ ในการใช้งานระบบแสดงให้เห็นในการเพิ่มขึ้นเล็กน้อยในเวลาการเข้าถึงโดยรวมในระยะเวลา ซึ่งแสดงโดยเส้นแนวโน้มเชิงเส้นขีดเส้นล้อม ปรากฏการณ์นี้ย้ำถึงการพึ่งพาในการใช้ GAISS เพื่อจัดการการรับเข้าและขั้นตอนที่เกี่ยวข้อง โดยเน้นความจำเป็นในการปรับปรุงประสิทธิภาพอย่างต่อเนื่องเพื่อให้เข้ากับความต้องการที่เพิ่มขึ้นอย่างมีประสิทธิภาพ สำคัญที่ต้องรับรู้ว่าการเปลี่ยนแปลงนี้มีผลต่อการจัดการจำนวนมาก รวมถึงวงจรการรับเข้า วัน

กำหนดในการสมัคร และการอัปเดตระบบ ตัวแปรเหล่านี้ควรถูกพิจารณาอย่างเหมาะสมเมื่อกำหนดกลยุทธ์ในการปรับปรุงระบบ



รูปที่ 4.7 ความถี่ในการเข้าถึงรายเดือนสำหรับหน้าเว็บรายชื่อนักศึกษาตั้งแต่เดือนกุมภาพันธ์ 2022 ถึงเดือนพฤษภาคม 2024

ที่มา: ผู้วิจัย, 2567

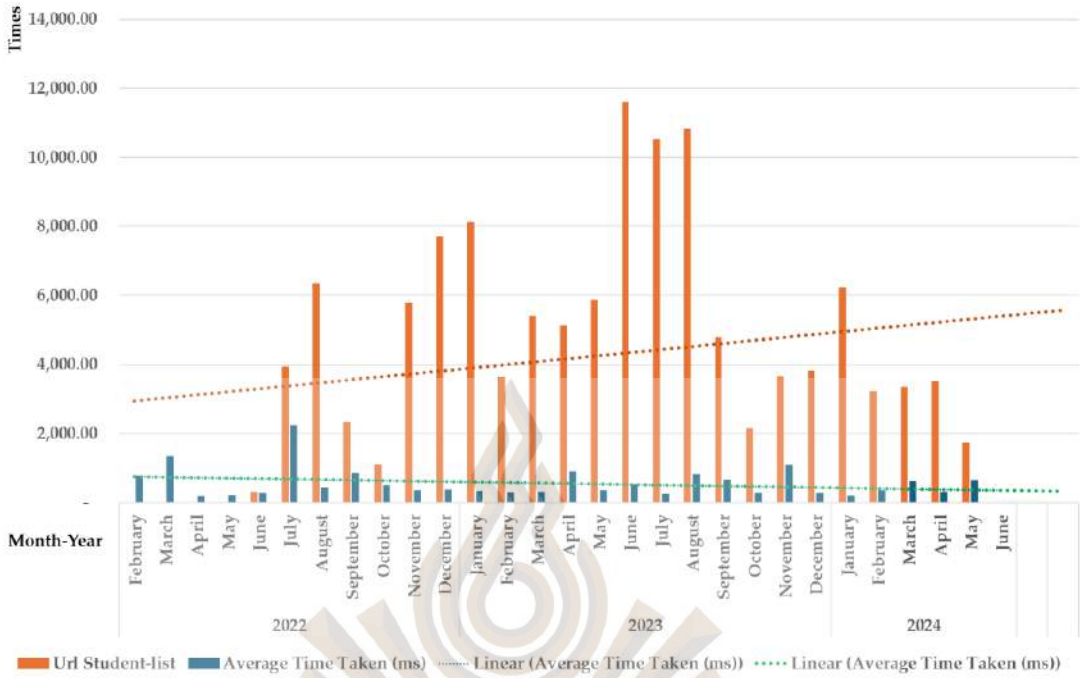


รูปที่ 4.8 หน้าเว็บรายชื่อนักศึกษาโดยใช้ Agile ทางซ้าย และ ALAI ทางขวา

ที่มา: ผู้วิจัย, 2567

การวิเคราะห์ปริมาณที่เป็นตัวเลขของการกระจายของสถานะการเข้าถึงสำหรับหน้าเว็บรายชื่อนักศึกษาใน GAISS นำเสนอในภาพที่ 4.6 การวิเคราะห์นี้ให้ข้อมูลสำคัญเกี่ยวกับความเชื่อถือได้ของระบบและความพึงพอใจของผู้ใช้ ข้อมูลนี้ให้หลักฐานชัดเจนเกี่ยวกับความเชื่อถือได้ที่สุดของระบบและการจัดการคำถามของผู้ใช้อย่างเรียบง่าย การพยากรณ์ว่ามีการพยายามเข้าถึงประมาณหนึ่งแสนครั้งที่ประสบความสำเร็จ แสดงให้เห็นถึงความทนทานของระบบและความสามารถในการตอบสนองต่อความต้องการของผู้ใช้อย่างมีประสิทธิภาพ ข้อมูลนี้ย้ำถึงประสิทธิภาพของ GAISS ในการรักษามาตรฐานการทำงานและความเชื่อถือได้ แน่นอนให้ผู้ใช้ประสบการณ์การเข้าถึงหน้าเว็บรายชื่อนักศึกษาได้อย่างราบรื่นและไม่มีการขัดข้อง

การวิเคราะห์ปริมาณการเข้าถึงรายเดือนและเวลาตอบสนองเฉลี่ยสำหรับหน้าเว็บรายชื่อนักศึกษาของ GAISS ระหว่างกุมภาพันธ์ 2022 และพฤษภาคม 2024 ถูกวิเคราะห์โดยมีในภาพที่ 4.7 ที่มีองค์ประกอบเชิงปริมาณ ข้อมูลที่ได้จากการวิเคราะห์นี้เกี่ยวกับแนวโน้มการใช้งานและประสิทธิภาพของระบบมีความสำคัญอย่างมาก จำนวนการเข้าถึงที่แสดงด้วยแถบสีส้มแสดงให้เห็นถึงแนวโน้มที่เพิ่มขึ้นอย่างต่อเนื่อง โดยมีการกระโดดขึ้นอย่างมีนัยสำคัญในเดือนเมษายน ค.ศ. 2023 และหลายเดือนในปี 2022 โดยเฉพาะเดือนกรกฎาคมและสิงหาคม การกระทำรุนแรงเหล่านี้เป็นไปได้ว่าเกิดขึ้นในช่วงเวลาที่สำคัญทางการศึกษาหรือการบริหารแผนการณ์ เวลาตอบสนองเฉลี่ยที่แสดงโดยเส้นสีน้ำเงินแสดงให้เห็นถึงแนวโน้มที่มั่นคงโดยทั่วไป ถึงแม้จะมีการเปลี่ยนแปลงเล็กน้อย ถึงแม้จะมีปริมาณการเข้าถึงที่เพิ่มขึ้นอย่างต่อเนื่อง แต่เวลาตอบสนองเฉลี่ยยังคงอยู่ในระดับต่ำอย่างต่อเนื่อง ทำให้เน้นถึงความทนทานของระบบในการจัดการภาระงานที่เพิ่มขึ้นโดยไม่มีการทรมานประสิทธิภาพให้ลดลงอย่างมาก เส้นแนวโน้มเชิงเส้นขีดประจำที่แสดงเวลาตอบสนองและจำนวนการเข้าถึงเพิ่มเติมย้ำแนวโน้มอีกครั้ง แนวโน้มการเพิ่มขึ้นของเส้นการเข้าถึงและการลดลงเล็กน้อยของเส้นเวลาตอบสนองย้ำถึงการปรับปรุงประสิทธิภาพและขยายขอบเขตของระบบ การวิเคราะห์นี้เน้นความทนทานและความสามารถของ GAISS ในการจัดการความต้องการที่เพิ่มขึ้นอย่างมีประสิทธิภาพโดยรักษาเวลาตอบสนองที่รวดเร็ว ทำให้มั่นใจได้ในประสบการณ์การใช้งานที่เชื่อถือได้ แม้ในช่วงเวลาที่มีการใช้งานสูง



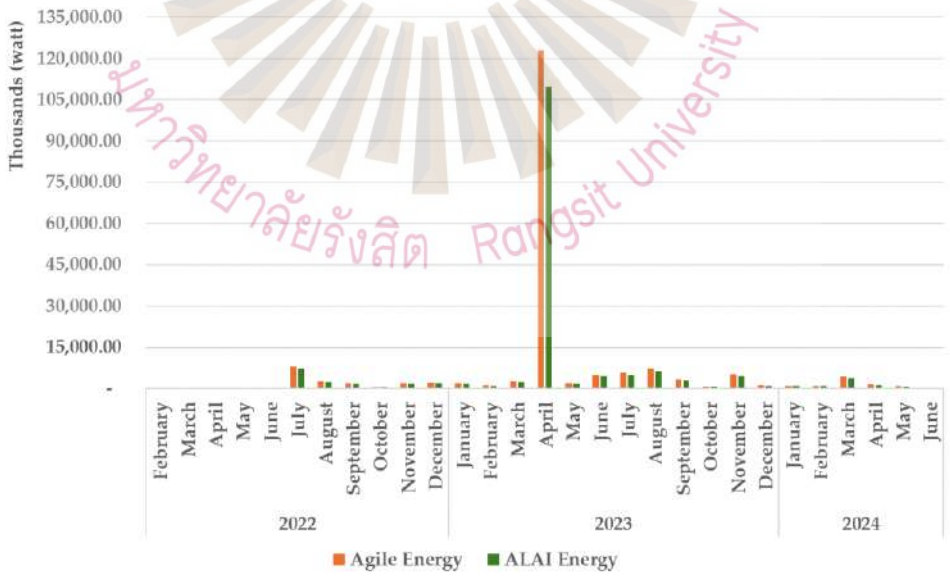
รูปที่ 4.9 จำนวนการเข้าถึงรายเดือนและเวลาตอบสนองเฉลี่ยสำหรับหน้าเว็บรายชื่อนักศึกษาของ GAISS ตั้งแต่กุมภาพันธ์ 2022 ถึงพฤษภาคม 2024

ที่มา: ผู้วิจัย, 2567

การวิเคราะห์อย่างเป็นรายละเอียดเกี่ยวกับการใช้พลังงานรายเดือนของกรอบการทำงาน Agile และ ALAI ใน GAISS ตั้งแต่กุมภาพันธ์ 2022 ถึงพฤษภาคม 2024 นำเสนอในภาพที่ 4.8 การเปรียบเทียบนี้ โดยใช้ข้อมูลที่เชื่อถือได้ ให้ข้อมูลสำคัญเกี่ยวกับประสิทธิภาพทางพลังงานของทั้งสองวิธีการพัฒนา กราฟแท่ง โดยที่การใช้พลังงานของ Agile แสดงด้วยสีส้ม และของ ALAI แสดงด้วยสีเขียว ชัดเจนแสดงให้เห็นว่า ALAI รักษาการใช้พลังงานในระดับที่ต่ำกว่า Agile อย่างสม่ำเสมอ ความเชื่อถือของกรอบการทำงานเหล่านี้เป็นที่ชัดเจนโดยเฉพาะในช่วงเวลาที่มีการใช้งานสูง เช่นในเดือนเมษายน ค.ศ. 2023 เมื่อทั้งสองระบบมีประสบการณ์รวดเร็ว เป็นตัวอย่าง โดยทั่วไปของการใช้พลังงานที่เพิ่มขึ้นเนื่องจากความต้องการเพิ่มขึ้นในระบบ ถึงแม้จะเกิดจุดสูงนี้ แต่ ALAI ยังคงใช้พลังงานอย่างมีเหตุผลเมื่อเปรียบเทียบกับ Agile โดยแสดงให้เห็นถึงความเชื่อถือและรองรับการใช้พลังงานที่น้อยกว่าของ ALAI มากกว่า การลดลงของการใช้พลังงานอย่างต่อเนื่องย้ำถึงความเชื่อถือและประสิทธิภาพของกรอบการทำงาน ALAI ในการส่งเสริมการพัฒนาซอฟต์แวร์อย่างยั่งยืน ผ่านการปรับการจัดสรรทรัพยากรและลดการสูญเสียพลังงานลง ข้อมูลแสดงให้เห็นว่าการนำเสนอ

กรอบการทำงาน ALAI สามารถส่งผลให้เกิดการอนุรักษ์พลังงานที่สำคัญได้ เป็นปัจจัยที่สำคัญในการลดรอยพระจอมแวดล้อมของระบบโดยรวมและเพิ่มประสิทธิภาพในด้านการดำเนินงาน

การประเมินการใช้พลังงานสำหรับหน้าเว็บรายชื่อนักศึกษาใน GAISS ชี้ให้เห็นถึงการปรับปรุงที่สำคัญในประสิทธิภาพเมื่อมีการนำเสนอวิธีการ ALAI เปรียบเทียบกับวิธีการ Agile แบบดั้งเดิม ระบบ Agile ใช้งานรวมเป็นเวลา 185,095 ชั่วโมง ในระหว่างนั้นใช้พลังงานทั้งหมด 185,243.55 กิโลวัตต์หรือ 1,000.80 กิโลวัตต์-ชั่วโมง ในทางกลับกัน กรอบการทำงาน ALAI ใช้พลังงานทั้งหมด 892.80 กิโลวัตต์-ชั่วโมง หรือ 165,253.24 กิโลวัตต์ ซึ่งตัวเลขการใช้พลังงานนี้แสดงให้เห็นถึงการลดลงที่มีนัยสำคัญในการใช้พลังงาน เนื่องจาก ALAI แสดงผลการปรับปรุงในประสิทธิภาพที่ 10.7914% การลดลงของการใช้พลังงานที่สำคัญนี้ไม่เพียงแสดงถึงการปรับปรุงในประสิทธิภาพการดำเนินงานอย่างมีนัยสำคัญเท่านั้น แต่ยังย้ำถึงประโยชน์ทางสิ่งแวดล้อมของกรอบการทำงาน ALAI การส่งเสริมให้นำกรอบการพัฒนาซอฟต์แวร์ที่มีการใช้พลังงานอย่างมีสติ อาทิ เช่น ALAI เป็นการช่วยลดผลกระทบทางสิ่งแวดล้อมที่เกี่ยวข้องกับการใช้ระบบในระยะยาว ซึ่งเป็นขั้นตอนสำคัญสู่การพัฒนาซอฟต์แวร์ที่ยั่งยืน



รูปที่ 4.10 เปรียบเทียบการใช้พลังงานรายเดือนสำหรับกรอบการทำงาน Agile และ ALAI ใน GAISS ตั้งแต่กุมภาพันธ์ 2022 ถึงพฤษภาคม 2024

ที่มา: ผู้วิจัย, 2567

บทที่ 5

สรุปผลอภิปรายและข้อเสนอแนะ

สรุปของการศึกษายืนยันความค้นพบที่มีนัยสำคัญเกี่ยวกับการเลือกใช้กรอบการทำงาน ALAI ของ GAISS โดยเปรียบเทียบกับวิธีการ Agile แบบดั้งเดิม จุดมุ่งหลักของการวิจัยคือการประเมินประสิทธิภาพของกรอบการทำงาน ALAI ในการส่งเสริมความยืดหยุ่นทางสถาปัตยกรรมในการพัฒนาซอฟต์แวร์และเพิ่มประสิทธิภาพในการใช้พลังงาน ข้อมูลที่ได้จากการสะสมข้อมูลระบบตั้งแต่กุมภาพันธ์ 2022 ถึงพฤษภาคม 2024 ที่ได้รับอย่างมีระบบเสร็จสมบูรณ์นำเสนอหลักฐานที่โดดเด่นว่ากรอบการทำงาน ALAI ช่วยลดการใช้พลังงานอย่างมีประสิทธิภาพถึง 10.7914% เมื่อเปรียบเทียบกับวิธีการ Agile แบบดั้งเดิม โดยเฉพาะ, ระบบ GAISS ที่ใช้กรอบการทำงาน ALAI ใช้พลังงานรวมทั้งหมด 892.80 กิโลวัตต์/ชั่วโมง, ในขณะที่ระบบที่ใช้วิธีการ Agile แบบดั้งเดิมใช้พลังงานทั้งหมด 1,000.80 กิโลวัตต์/ชั่วโมง การลดลงขนาดประมาณ 108 กิโลวัตต์ที่มีนัยสำคัญนี้ย้ำถึงความสามารถของกรอบการทำงาน ALAI ในการสูงสุดใช้ประโยชน์จากการใช้ทรัพยากรและลดผลกระทบต่อสิ่งแวดล้อมลง ข้อความยืนยันทฤษฎีการวิจัยว่าความยั่งยืนของระบบถูกเพิ่มขึ้นเมื่อ AOF และโครงสถาปัตยกรรมชั้นสนับสนุนกันถูกผสมรวมอยู่ภายในวิธีการ Agile

5.1 การมีส่วนร่วมทฤษฎี

การศึกษานี้นำเสนอข้อค้นพบที่สำคัญในการพัฒนาซอฟต์แวร์ที่ยั่งยืน กรอบการทำงาน ALAI ซึ่งได้รับการตรวจสอบด้วยการสะสมข้อมูลทางประจักษ์ เสนอวิธีการที่เชิงระเบียบมากขึ้น โดยเน้นไปที่ความได้เปรียบที่ชัดเจนของความยืดหยุ่นและความสามารถในการปรับขนาด AOF และโครงสถาปัตยกรรมชั้นสับสนับสนุนกัน ทั้งคู่ถูกผสมรวมเข้าด้วยกันในกรอบการทำงาน ALAI เพื่อเพิ่มความเป็นระบบและความยืดหยุ่นของระบบได้อีกต่อไป การวิจัยนี้เน้นความสำคัญของส่วนประกอบเหล่านี้ในการเพิ่มประสิทธิภาพและอายุการใช้งานของวงจรชีวิตการพัฒนาซอฟต์แวร์ การวิจัยเพิ่มเติมความรู้ในวรรณกรรมที่มีอยู่อย่างชัดเจนโดยการแสดงให้เห็นว่าสามารถทำการ

ปรับปรุงให้มีประสิทธิภาพในการใช้พลังงานได้โดยมิต้องเสียสมดุลในความยืดหยุ่นและประสิทธิภาพของระบบ

5.2 ผลกระทบทางปฏิบัติ

ผลลัพธ์ของการศึกษานี้มีผลกระทบทางปฏิบัติต่อสถาบันการศึกษาและองค์กรที่เคารพธรรมชาติที่ต้องการลดผลกระทบทางสิ่งแวดล้อมต่อการใช้พลังงานและปรับปรุงประสิทธิภาพการดำเนินงานโดยรวมของพวกเขา หน่วยงานเหล่านี้สามารถบรรลุการอนุรักษ์พลังงานที่สำคัญโดยการนำกรอบการทำงาน ALAI มาใช้งาน ซึ่งจะลดค่าใช้จ่ายในด้านการดำเนินการและเพิ่มวัตถุประสงค์ในการยังคงอยู่อย่างยั่งยืน อย่างสำคัญ การวิจัยแสดงให้เห็นว่าการนำเสนอกรอบการทำงาน ALAI ช่วยให้เกิดระบบที่ยืดหยุ่นและสามารถจัดการได้ง่ายขึ้น และสร้างประหยัดพลังงานทันทีซึ่งเป็นปัจจัยสำคัญในการสร้างความยั่งยืนในระยะยาว วิธีการนี้ย้ำถึงประสิทธิภาพของกรอบการและความสามารถของมันในการให้คุณสมบัติยั่งยืน โดยยืนยันข้อค้นพบเกี่ยวกับประสิทธิภาพในระยะยาวของมัน

5.3 ข้อจำกัดและงานที่จะทำในอนาคต

การค้นพบนี้แม้ว่าจะมีนัยสำคัญ แต่อาจมีข้อจำกัดในการนำไปประยุกต์ใช้กับอุตสาหกรรมหรือระบบอื่นๆ เนื่องจากสถานการณ์เฉพาะของ GAISS การค้นพบนี้แม้จะมีความสำคัญ แต่ก็อาจมีการนำไปประยุกต์ใช้กับอุตสาหกรรมหรือระบบอื่นๆ ได้จำกัด เนื่องจากสถานการณ์เฉพาะของ GAISS ความหลากหลายของระบบซอฟต์แวร์ในกลุ่มภาคส่วนต่างๆ อาจจะแสดงให้เห็นได้เพียงบางส่วนในลักษณะเฉพาะของระบบการรับเข้าศึกษาที่มีการใช้งานและความต้องการในการประมวลผลข้อมูล ดังนั้น บทบาทของการศึกษาในอนาคตในการตรวจสอบความเหมาะสมที่กว้างขวางของโครงร่าง ALAI ในสภาพแวดล้อมที่หลากหลาย เช่น โครงสร้าง และแพลตฟอร์มซอฟต์แวร์ เป็นสิ่งที่สำคัญ การศึกษาเหล่านี้ซึ่งอาจทำให้มีการเสนอผลงานวิจัยที่สำคัญต่อการวิจัยภายหลัง จำเป็นต้องสนับสนุนการแก้ไขปัญหาที่ระบุไว้ เช่น ความซับซ้อนของการผสมผสานหรือความต้องการที่เฉพาะเจาะจงของอุตสาหกรรม เพื่อเสริมสร้างความเหมาะสมและ

ประสิทธิภาพในการใช้โครงร่างนี้ ความสำคัญของการวิจัยในอนาคตในการทำกิจกรรมนี้ ไม่สามารถเน้นไปมากเกินไปได้

การวิจัยนี้ให้ข้อมูลหลักฐานที่มั่นคงที่สนับสนุนการใช้งานโครงร่าง ALAI สำหรับการพัฒนาซอฟต์แวร์อย่างยั่งยืน ศักยภาพของ ALAI ในการเปลี่ยนแปลงวิธีการพัฒนาซอฟต์แวร์เป็นเรื่องชัดเจน ที่แสดงให้เห็นในการประหยัดพลังงานอย่างมากและเพิ่มความยืดหยุ่นในระบบ สิ่งนี้ไม่เพียงเพิ่มความหวังในอนาคตที่ยั่งยืนมากขึ้นเท่านั้น แต่ยังเสริมความมั่นใจในศักยภาพของโครงร่างได้อีกด้วย โดยการผสมผสานของวิธีการขั้นสูงในองค์ประกอบของสถาปัตยกรรมแบบชั้นและ AOF โครงร่าง ALAI สามารถกระตุ้นการพัฒนาระบบซอฟต์แวร์ที่มีความยืดหยุ่นมากขึ้น มีประสิทธิภาพมากขึ้น และมีความยั่งยืนต่อสิ่งแวดล้อม ทำให้ส่งเสริมความเป็นเลิศทางดำเนินงานและด้านสิ่งแวดล้อมได้ การสำรวจและปรับปรุงโครงร่างเหล่านี้ในการศึกษาต่อไป จะเพิ่มประโยชน์และความเหมาะสมของมันในบริบทต่างๆ ซึ่งจะเสริมความมั่นใจในศักยภาพของมันได้อีกด้วย

บรรณานุกรม

Abderlrahman, M. M., Zhan, S., & Chong, A. (2020). A three-tier architecture visual-programming platform for building-lifecycle data management. In *Proceedings of the 11th Annual Symposium on Simulation for Architecture and Urban Design*. doi:10.5555/3465085.3465127

Ahmad, S. R. A., Yahaya, J., & Sallehudin, H. (2022). Green software process factors: A qualitative study. *Sustainability, 14*(18), Article 11180. doi:10.3390/su141811180

Akinyele, D., Amole, A., Olabode, E., Olusesi, A., & Ajewole, T. (2021). Simulation and analysis approaches to microgrid systems design: Emerging trends and sustainability framework application. *Sustainability, 13*(20), Article 11299. doi:10.3390/su132011299

Alhammad, M., & Moreno, A. (2024). Enhancing Agile software development sustainability through the integration of user experience and gamification. In *Agile Processes in Software Engineering and Extreme Programming – Workshops* (pp. 12–20). Springer. doi:10.1007/978-3-031-48550-3_2

Alharbi, I. M., Alyoubi, A. A., Altuwairiqi, M., & Ellatif, M. A. (2021). Enhance risks management of software development projects in concurrent multi-projects environment to optimize resources allocation decisions. *International Journal of Advanced Computer Science and Applications, 12*(6). doi:10.14569/ijacsa.2021.0120626

Alrabaiah, H. A., & Medina-Medina, N. (2021). Agile Beeswax: Mobile app development process and empirical study in real environment. *Sustainability, 13*(4), Article 1909. doi:10.3390/su13041909

Arponpong, C., & Soongpol, B. (2023, March). Guideline Framework of Information Technology Security System by ISO 27001: 2013 Standard of the Securities and Exchange Commission (SEC). *Journal of Digital Business and Social Sciences*, *9*(1),JDB011,1-12.

Atadoga, A., Umoga, U. J., Lottu, O. A., & Sodiya, E. O. (2024). Advancing green computing: practices, strategies, and impact in modern software development for environmental sustainability. *World Journal of Advanced Engineering and Technology Sciences, 11(*1), 220–230. doi:10.30574/wjaets.2024.11.1.0052

บรรณานุกรม (ต่อ)

Avotins, A., Nikitenko, A., Senfelds, A., Kikans, J., Podgornovs, A., & Sazonovs, M. (2022). Development of analysis tools for energy efficiency increase of existing data centres. In *Proceedings of the 2022 IEEE 63th International Scientific Conference on Power and Electrical Engineering of Riga Technical University* (pp. 1-6). Riga, Latvia. doi:10.1109/RTUCON56726.2022.9978876

Bastidas Fuertes, A., Pérez, M., & Meza, J. (2023). Transpiler-based architecture design model for back-end layers in software development. *Applied Sciences, 13*(20), Article 11371. doi:10.3390/app132011371

Behutiye, W., Karhapää, P., López, L., Burgués, X., Martínez-Fernández, S., Vollmer, A. M., Rodríguez, P., Franch, X., & Oivo, M. (2020). Management of quality requirements in agile and rapid software development: A systematic mapping study. *Information and Software Technology, 123*, Article 106225. doi:10.1016/j.infsof.2019.106225

Bein, W. (2018). Energy saving in data centers. *Electronics, 7*(1), Article 5. doi:10.3390/electronics7010005

Beynon-Davies, P., Carne, C., Mackay, H., & Tudhope, D. (1999). Rapid application development (RAD): An empirical review. *European Journal of Information Systems, 8*(3), 211–223. doi:10.1057/palgrave.ejis.3000325

Blanco, J. Z., & Lucrédio, D. (2021). A holistic approach for cross-platform software development. *Journal of Systems and Software, 179*, Article 110985. doi:10.1016/j.jss.2021.110985

Boeing, P., Leon, M., Nesbeth, D., Finkelstein, A., & Barnes, C. (2018). Towards an aspect-oriented design and modelling framework for synthetic biology. *Processes, 6*(9), Article 167. doi:10.3390/pr6090167

Camañes, V., Tobajas, R., & Fernandez, A. (2024). Methodology of eco-design and software development for sustainable product design. *Sustainability, 16*(7), Article 2626. doi:10.3390/su16072626

บรรณานุกรม (ต่อ)

Caraba Ş, M., & Popescu, P. G. (2017). Energy-efficient virtualized clusters. *Future Generation Computer Systems, 74*, 151–157. doi:10.1016/j.future.2015.10.018

Chaurasia, N., Kumar, M., Chaudhry, R., & Verma, O. P. (2021). Comprehensive survey on energy-aware server consolidation techniques in cloud computing. *Journal of Supercomputing, 77*, 11682-11737. doi:10.1007/s11227-021-03760-1

Chinenyeze, S., Liu, X., & Al-Dubai, A. (2014). An aspect-oriented model for software energy efficiency in decentralized servers. In *Proceedings of the 2014 Conference ICT for Sustainability*. doi:10.2991/ict4s-14.2014.14

Ciancarini, P., Ergasheva, S., Kholmatova, Z., Kruglov, A., Succi, G., Vasquez, X., & Zuev, E. (2020). Analysis of energy consumption of software development process entities. *Electronics, 9*(10), Article 1678. doi:10.3390/electronics9101678

Davis, A. L. (2020). Spring AOP. In *Spring Quick Reference Guide* (pp. 33–41). Apress.

Donca, I.-C., Stan, O. P., Misaros, M., Gota, D., & Miclea, L. (2022). Method for continuous integration and deployment using a pipeline generator for agile software projects. *Sensors, 22*(12), Article 4637. doi:10.3390/s22124637

Dorokhova, M., Ribeiro, F., Barbosa, A., Viana, J., Soares, F., & Wyrsch, N. (2021). Real-world implementation of an ICT-based platform to promote energy efficiency. *Energies, 14(*9), Article 2416. doi:10.3390/en14092416

Dustdar, S., & Gall, H. (2003). Architectural Concerns in Distributed and Mobile Collaborative Systems. *Journal of Systems Architecture, 49(*10–11), 457–473. doi:10.1016/s1383-7621(03)00092-4

Erdenebat, B., Bud, B., Batsuren, T., & Kozsik, T. (2023). Multi-project multi-environment approach—An enhancement to existing DevOps and continuous integration and continuous deployment tools. *Computers, 12*(12), Article 254. doi:10.3390/computers12120254

บรรณานุกรม (ต่อ)

Fagarasan, C., Cristea, C., Cristea, M., Popa, O., & Pisla, A. (2023). Integrating sustainability metrics into project and portfolio performance assessment in agile software development: A data-driven scoring model. *Sustainability, 15*(17), Article 13139. doi:10.3390/su151713139

García-Berná, J. A., Fernández-Alemán, J. L., Carrillo de Gea, J. M., Toval, A., Mancebo, J., Calero, C., & García, F. (2021). Energy efficiency in software: A case study on sustainability in personal health records. *Journal of Cleaner Production, 282*, Article 124262. doi:10.1016/j.jclepro.2020.124262

Ghimire, D., & Charters, S. (2022). The impact of agile development practices on project outcomes. *Software, 1(*3), 265–275. doi:10.3390/software1030012

Gupta, C., Fernandez-Crehuet, J. M., & Gupta, V. (2022). Measuring impact of cloud computing and knowledge management in software development and innovation. *Systems, 10*(5), Article 151. doi:10.3390/systems10050151

Haputhanthrige, V., Asghar, I., Saleem, S., & Shamim, S. (2024). The impact of a skill-driven model on scrum teams in software projects: A catalyst for digital transformation. *Systems, 12*(5), Article 149. doi:10.3390/systems12050149

Hassan, H. B., Barakat, S. A., & Sarhan, Q. I. (2021). Survey on serverless computing. *Journal of Cloud Computing: Advances, Systems and Applications, 10*, Article 39. doi:10.1186/s13677-021-00253-7

Hocaoglu, M. F. (2017). Aspect-oriented programming perspective in software agents and simulation. *International Journal of Advanced Technology, 8*(3). doi:10.4172/0976-4860.1000186

Huang, S. M., Yen, D. C., Yan, T. J., & Yang, Y. T. (2022). An intelligent mechanism to automatically discover emerging technology trends: Exploring regulatory technology. *ACM Transactions on Management Information Systems, 13(*2), 1–29. doi:10.1145/3485187

บรรณานุกรม (ต่อ)

Implementation of test-driven development in data access layer within a business system development. Retrieved from https://stis.ac.id/sipadu/pegawai/upload_jurnal/file_1517383759.pdf

Jánki, Z. R., & Bilicki, V. (2023). The impact of the web data access object (WebDAO) design pattern on productivity. *Computers, 12*(8), Article 149. doi:10.3390/computers12080149

Karamitsos, I., Albarhami, S., & Apostolopoulos, C. (2020). Applying DevOps practices of continuous automation for machine learning. *Information, 11*(7), Article 363. doi:10.3390/info11070363

Katal, A., Dahiya, S., & Choudhury, T. (2023). Energy efficiency in cloud computing data centers: A survey on software technologies. *Cluster Computing, 26*, 1845-1875. doi:10.1007/s10586-022-03713-0

Khalifeh, A., Al-Adwan, A. S., Alrousan, M. K., Yaseen, H., Mathani, B., & Wahsheh, F. R. (2023). Exploring the nexus of sustainability and project success: A proposed framework for the software sector. *Sustainability, 15*(22), Article 15957. doi:10.3390/su152215957

Khan, M. U., Abbas, S., Lee, S. U.-J., & Abbas, A. (2021). Measuring power consumption in mobile devices for energy sustainable app development: A comparative study and challenges. *Sustainable Computing: Informatics and Systems, 31*, Article 100589. doi:10.1016/j.suscom.2021.100589

Kithulwatta, W. M. C. J. T., Jayasena, K. P. N., Kumara, B. T. G. S., & Rathnayaka, R. M. K. T. (2022). Integration with Docker container technologies for distributed and microservices applications: A state-of-the-art review. *International Journal of Systems and Service-Oriented Engineering, 12*(1), 1–22. doi:10.4018/ijssoe.297136

Kravets, A. G., & Egunov, V. (2022). The software cache optimization-based method for decreasing energy consumption of computational clusters. *Energies, 15* (20), Article 7509. doi:10.3390/en15207509

บรรณานุกรม (ต่อ)

Kruglov, A., Succi, G., & Vasuez, X. (2021). Incorporating energy efficiency measurement into CI\CD pipeline. In *Proceedings of the 2021 2nd European Symposium on Software Engineering, Larissa, Greece, 19-21 November 2021*. doi:10.1145/3501774.3501777

Kuaban, G. S., Gelenbe, E., Czachórski, T., Czekalski, P., & Tangka, J. K. (2023). Modelling of the energy depletion process and battery depletion attacks for by-powered internet of things (IoT) devices. *Sensors, 23(*13), Article 6183. doi:10.3390/s23136183

Kumar, A., Grover, P. S., & Kumar, R. (2009). A quantitative evaluation of aspect-oriented software quality model (AOSQUAMO). *Software Engineering Notes, 34(*5), 1–9. doi:10.1145/1598732.1598736

Kumar, P. (2012). Aspect-oriented software quality model: The AOSQ model. *Advances in Computer Intelligence, 3*(2), 105–118. doi:10.5121/acij.2012.3212

Kwasek, A., Maciaszczyk, M., Kocot, M., Rzepka, A., Kocot, D., Gąsiński, H., & Prokopowicz, D. (2023). Energy saving practices in the IT area as a factor of sustainable development of the organization: A Case Study of Poland. *Energies, 16* (4), Article 1942. doi:10.3390/en16041942

Lavy, I., & Rami, R. (2018). The circumstances in which modular programming becomes the favor choice by novice programmers. *International Journal of Modern Education and Computer Science, 10*(7), 1–12. doi:10.5815/ijmecs.2018.07.01

Lee, W.-T., & Chen, C.-H. (2023). Agile software development and reuse approach with Scrum and software product line engineering. *Electronics, 12*(15), Article 3291. doi:10.3390/electronics12153291

Lis, A., Sudolska, A., Pietryka, I., & Kozakiewicz, A. (2020). Cloud computing and energy efficiency: Mapping the thematic structure of research. *Energies, 13*(16), Article 4117. doi:10.3390/en13164117

บรรณานุกรม (ต่อ)

Manimegalai, R., Sandhanam, S., Nandhini, A., & Pandia, P. (2023). Energy efficient coding practices for sustainable software development. In *Proceedings of the First International Conference on Science, Engineering and Technology Practices for Sustainable Development*, Coimbatore, Tamilnadu, India. doi:10.4108/eai.17-11-2023.2342635

Marnada, P., Raharjo, T., Hardian, B., & Prasetyo, A. (2022). Agile project management challenge in handling scope and change: A systematic literature review. *Procedia Computer Science, 197*, 290–300. doi:10.1016/j.procs.2021.12.143

Mehdi Ben Hmida, M., Ferraz Tomaz, R., & Monfort, V. (2005). Applying AOP concepts to increase web services flexibility. In *Proceedings of the International Conference on Next Generation Web Services Practices*. doi:10.1109/NWESP.2005.18

Miheliˇc, A., Vrhovec, S., & Hovelja, T. (2023). Agile development of secure software for small and medium-sized enterprises. *Sustainability, 15*(1), Article 801. doi:10.3390/su15010801

Mishra, A., & Alzoubi, Y. I. (2023). Structured software development versus agile software development: A comparative analysis. *International Journal of Systems Assurance Engineering and Management, 14*, 1504–1522. doi:10.1007/s13198-023-01958-5

Mishra, A., & Mishra, D. (2021). Sustainable software engineering: Curriculum development based on ACM/IEEE guidelines. In A. Mishra & Z. Otaiwi (Eds.), *Software Sustainability* (pp. 269–285). Springer. doi:10.1007/978-3-030-69970-3_11

Mishra, A., & Otaiwi, Z. (2020). DevOps and software quality: A systematic mapping. *Computer Science Review, 38*, Article 100308. doi:10.1016/j.cosrev.2020.100308

Naz, R., & Khan, M. N. A. (2015). Rapid applications development techniques: A critical review. *International Journal of Software Engineering and Applications, 9*(11), 163–176. doi:10.14257/ijseia.2015.9.11.15

บรรณานุกรม (ต่อ)

Netinant, P. (2015). Design adaptability for multilingual mobile application software. In *Proceedings of 24th International Conference on Software Engineering and Data Engineering*. San Diego, CA.

Netinant, P., Elrad, T., & Fayad, M. E. (2001). A layered approach to building open aspect-oriented systems: A framework for the design of on-demand system demodularization. *Communications of the ACM, 44*(10), 83–85. doi:10.1145/383845.384200

Netinant, P., Rukhiran, M., & Soongpol, B. (2022). Design Adaptability and Flexibility of Information Management System Framework for Graduate School Services. In *Proceeding of the 6th International Conference on Business and Information Management (ICBIM)*, Guangzhou, China, pp. 156-161. doi: 10.1109/ICBIM57406.2022.00035.

Ournani, Z., Rouvoy, R., Rust, P., & Penhoat, J. (2021). Tales from the code #1: The effective impact of code refactorings on software energy consumption. In *Proceedings of the 16th International Conference on Software Technologies*. Virtual Event.

Oyedeji, S., Seffah, A., & Penzenstadler, B. (2018). A catalogue supporting software sustainability design. *Sustainability, 10*(7), Article 2296. doi:10.3390/su10072296

Ozdenizci Kose, B. (2021). Business process management approach for improving agile software process and agile maturity. *Journal of Software Evolution and Process, 33*(4), Article e2331. doi:10.1002/smr.2331

Paradis, C., Kazman, R., & Tamburri, D. A. (2021). Architectural tactics for energy efficiency: Review of the literature and research roadmap. In *Proceedings of the 54th Hawaii International Conference on System Sciences*. Kauai, Hawaii.

Park, D., Kang, S., & Lee, J. (2006). Design phase analysis of software qualities using aspect-oriented programming. In *Proceedings of the Seventh ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*. doi:10.1109/SNPD-SAWN.2006.34

บรรณานุกรม (ต่อ)

Pashutan, M., Abdolvand, N., & Harandi, S. R. (2022). The impact of IT resources and strategic alignment on organizational performance: The moderating role of environmental uncertainty. *Digital Business, 2*(2), Article 100026. doi:10.1016/j.digbus.2022.100026

Pazienza, A., Baselli, G., Vinci, D. C., & Trussoni, M. V. (2024). A holistic approach to environmentally sustainable computing. *Innovations in Systems and Software Engineering*. doi:10.1007/s11334-023-00548-9

Rana, M. E., & Saleh, O. S. (2022). High Assurance Software Architecture and Design. In *System Assurances* (pp. 271–285). Elsevier. doi:10.1016/B978-0-323-90240-3.00015-1

Rathee, A., & Chhabra, J. K. (2022). Metrics for reusability of Java language components. *Journal of King Saud University - Computer and Information Sciences, 34*(8), 5533–5551. doi:10.1016/j.jksuci.2022.05.010

Ravikumar, G., Begum, Z., Kumar, A. S., Kiranmai, V., Bhavsingh, M., & Kumar, O. K. (2022). Cloud host selection using iterative particle-swarm optimization for dynamic container consolidation. *International Journal of Recent Innovations in Trends in Computer and Communication, 10*(1S), 247-253. doi:10.17762/ijritcc.v10i1s.5846

Reyna, J., Hanham, J., & Orlando, J. (2024). From e-waste to eco-wonder: Resurrecting computers for a sustainable future. *Sustainability, 16*(8), Article 3363. doi:10.3390/su16083363

Rukhiran, M., & Netinant, P. (2020). A practical model from multidimensional layering: Personal finance information framework using mobile software interface operations. *Journal of Information and Communication Technology, 19*(3). doi:10.32890/jict2020.19.3.2

Rukhiran, M., Boonsong, S., & Netinant, P. (2024). Sustainable optimizing performance and energy efficiency in proof of work blockchain: A multilinear regression approach. *Sustainability, 16*(4), Article 1519. doi:10.3390/su16041519

บรรณานุกรม (ต่อ)

Rukhiran, M., Buaroong, S., & Netinant, P. (2022). Software development for educational information services using multilayering semantics adaptation. *International Journal of Services Science, Management, Engineering, and Technology, 13*(1), 1–27. doi:10.4018/ijssmet.307153

Sadowski, C., & Zimmermann, T. (2019). *Rethinking Productivity in Software Engineering*. Apress. Retrieved from doi:10.1007/978-1-4842-4482-1

Şanlıalp, İ., Öztürk, M. M., & Yiğit, T. (2022). Energy efficiency analysis of code refactoring techniques for green and sustainable software in portable devices. *Electronics, 11*(3), Article 442. doi:10.3390/electronics11030442

Schaarschmidt, M., Uelschen, M., Pulvermüller, E., & Westerkamp, C. (2020). Framework of software design patterns for energy-aware embedded systems. In *Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering* (pp. 62-73). Virtual Event. doi:10.5220/0009351000620073 Söylemez, M., Tekinerdogan, B., & Kolukısa Tarhan, A. (2022). Feature-driven characterization of microservice architectures: A survey of the state of the practice. *Applied Sciences, 12*(9), Article 4424. doi:10.3390/app12094424

Söylemez, M., Tekinerdogan, B., & Tarhan, A. K. (2024). Microservice reference architecture design: A multi‐case study. *Software: Practice and Experience, 54*(1), 58–84. doi:10.1002/spe.3241

Sriraman, G., & Raghunathan, S. (2023). A systems thinking approach to improve sustainability in software engineering—A grounded capability maturity framework. *Sustainability, 15*(11), Article 8766. doi:10.3390/su15118766

Strojny, J., Krakowiak-Bal, A., Knaga, J., & Kacorzyk, P. (2023). Energy security: A conceptual overview. *Energies, 16*(13), Article 5042. doi:10.3390/en16135042

Subramanya, R., Sierla, S., & Vyatkin, V. (2022). From DevOps to MLOps: Overview and application to electricity market forecasting. *Applied Sciences, 12*(19), Article 9851. doi:10.3390/app12199851

บรรณานุกรม (ต่อ)

Testasecca, T., Lazzaro, M., Sarmas, E., & Stamatopoulos, S. (2023). Recent advances on data-driven services for smart energy systems optimization and proactive management. In *Proceedings of the 2023 IEEE International Workshop on Metrology for Living Environment*. Milano, Italy. doi:10.1109/MetroLivEnv56897.2023.10164056

Tu, Z. (2023). Research on the application of layered architecture in computer software development. *Journal of Computer, Electronics and Information Management, 11*(3), 34–38. doi:10.54097/jceim.v11i3.08

Understanding Layered Software Architecture. Retrieved from https://systemdesignschool.io/blog/layered-software-architecture

Valmohammadi, C., & Mortaz Hejri, F. (2023). Designing a conceptual green process model in software development: A mixed method approach. *International Journal of Information Management and Data Insights, 3*(2), Article 100204. doi:10.1016/j.ijimdi.2023.100204

Vračar, L. M., Stojanović, M. D., Stanimirović, A. S., & Prijić, Z. D. (2019). Influence of encryption algorithms on power consumption in energy harvesting systems. *Journal of Sensors*, Article 8520562. doi:10.1155/2019/8520562

Waheed, W., Khodeir, L., & Fathy, F. (2024). Integrating Lean and sustainability for waste reduction in construction from the early design phase. *HBRC Journal, 20*(1), 337–364. doi:10.1080/16874048.2024.2318502

Waseem, M., Liang, P., & Shahin, M. (2020). A systematic mapping study on microservices architecture in DevOps. *Journal of Systems and Software, 170*, Article 110798. doi:10.1016/j.jss.2020.110798

Willms, P., & Brandenburg, M. (2019). Emerging trends from advanced planning to integrated business planning. *IFAC-PapersOnLine, 52*(13), 2620–2625. doi:10.1016/j.ifacol.2019.11.602

บรรณานุกรม (ต่อ)

Ye, Y., Barapatre, S., Davis, M. K., Elliston, K. O., Davatzikos, C., Fedorov, A., ... & Becich, M. J. (2021). Open-source software sustainability models: Initial white paper from the informatics technology for cancer research sustainability and industry partnership working group. *Journal of Medical Internet Research, 23*(12), Article e20028. doi:10.2196/20028

Yuan, J., Gao, Z., & Xiang, Y. (2023). Green energy consumption path selection and optimization algorithms in the era of low carbon and environmental protection digital trade. *Sustainability, 15*(15), Article 12080. doi:10.3390/su151512080

Zein, A. (2024). Implementation of service oriented architecture in mobile applications to improve system flexibility, interoperability, and scalability. *Journal of Information Systems and Technology Engineering, 2*(1), 171–174. doi:10.61487/jiste.v2i1.60

Zuluaga, C. A., Aristizábal, L. M., Rúa, S., Franco, D. A., Osorio, D. A., & Vásquez, R. E. (2022). Development of a modular software architecture for underwater vehicles using systems engineering. *Journal of Marine Science and Engineering, 10*(4), Article 464. doi:10.3390/jmse10040464

ประวัติผู้วิจัย

ชื่อ	บัวเรียน สูงพล
วัน เดือน ปีเกิด	29 มกราคม 2523
สถานที่เกิด	จังหวัดมหาสารคาม ประเทศไทย
ประวัติการศึกษา	มหาวิทยาลัยมหาสารคาม ปริญญาบริหารธุรกิจบัณฑิต สาขาวิชาคอมพิวเตอร์ธุรกิจ , 2547 มหาวิทยาลัยรังสิต วิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ, 2562
ที่อยู่ปัจจุบัน	63/340 หมู่ 1 หมู่บ้านพฤกษาวิลล์ 36 ต.หลักหก อ.เมือง จ.ปทุมธานี
สถานที่ทำงาน	สำนักงานหลักประกันสุขภาพแห่งชาติ (สปสช.)
ตำแหน่งปัจจุบัน	ผู้เชี่ยวชาญ