การเปรียบเทียบการพัฒนาซอฟต์แวร์แบบ Agent-first ด้วย Google Antigravity กับ AI Coding Assistant แบบ Copilot

Comparative Study of Agent-first Software Development Using Google Antigravity and Copilot-style AI Coding Assistants

พระมหาประสพสุข ปิยภาณี (สุขล้วน)[1*]

Phramaha Prasopsuk Piyabhani (Sukluan)[1*]

[1]งานเทคโนโลยีสารสนเทศ สำนักงานวิชาการ วิทยาลัยสงฆ์พุทธโสธร มหาวิทยาลัยมหาจุฬาลงกรณราชวิทยาลัย จังหวัดฉะเชิงเทรา 24000

[1]Information Technology Section, Academic Office, Phutthasothorn Buddhist College, Mahachulalongkornrajavidyalaya University, Chachoengsao 24000, Thailand

[*]Corresponding author: e-mail: prasopsuk.suk@mcu.ac.th

บทคัดย่อ

วิวัฒนาการของปัญญาประดิษฐ์เพื่อช่วยพัฒนาซอฟต์แวร์ได้ก้าวข้ามจากยุคตัวช่วยเติมเต็มโค้ด (AI Coding Assistant แบบ Copilot) เข้าสู่ยุคตัวแทนอัตโนมัติแบบกำหนดเป้าหมาย (Agent-first Software Development) อย่าง Google Antigravity บทความวิชาการนี้มีวัตถุประสงค์เพื่อศึกษา วิเคราะห์ และเปรียบเทียบกระบวนทัศน์ สถาปัตยกรรมเชิงเทคนิค ประสิทธิภาพการทำงาน และการกำกับดูแลระหว่างทั้งสองแนวทาง โดยสังเคราะห์ผ่าน 5 มิติหลัก ได้แก่ 1) สถาปัตยกรรมบริบทและการให้เหตุผล 2) ความเป็นอิสระในการปฏิบัติการและเครื่องมือ 3) ผลิตภาพและภาระทางปัญญาของนักพัฒนา 4) คุณภาพซอฟต์แวร์และการเปลี่ยนแปลงข้ามระบบ และ 5) ธรรมาภิบาลและความปลอดภัย จากการวิเคราะห์เปรียบเทียบในบริบทงานพัฒนาระบบสารสนเทศมหาวิทยาลัยชี้ให้เห็นว่า โมเดลแบบ Copilot เหมาะกับงานสร้างโค้ดระดับย่อย (Micro-level Coding) แต่มีข้อจำกัดในงานที่ต้องปรับปรุงสถาปัตยกรรมหลายไฟล์ การรันคำสั่ง และการตรวจสอบระบบ ขณะที่โมเดล Agent-first ด้วย Google Antigravity ช่วยลดภาระทางปัญญาของนักพัฒนา ยกระดับความคล่องตัวในการพัฒนาฟังก์ชันระดับระบบ และสนับสนุนการควบคุมด้วย Skill-based Governance อย่างมีประสิทธิภาพ บทความวิชาการนี้จึงเป็นแนวทางให้สถาบันอุดมศึกษาสามารถวางยุทธศาสตร์การเลือกใช้เทคโนโลยี AI ในการพัฒนาระบบสารสนเทศได้อย่างคุ้มค่าและปลอดภัย

คำสำคัญ: Agent-first, Google Antigravity, GitHub Copilot, การพัฒนาซอฟต์แวร์, ตัวแทนปัญญาประดิษฐ์, ภาระทางปัญญา

ABSTRACT

The evolution of artificial intelligence in software engineering has transitioned from autocomplete code assistants (Copilot-style AI) to goal-directed autonomous agents (Agent-first development paradigms) exemplified by Google Antigravity. This academic article analyzes and compares the paradigms, technical architectures, developer productivity, and governance models of both approaches across five key dimensions: (1) context window and reasoning scope, (2) operational autonomy and tool orchestration, (3) developer productivity and cognitive load, (4) multi-file software quality and regression testing, and (5) security governance and compliance. Comparative architectural analysis within university information system contexts indicates that while Copilot-style tools excel at micro-level autocomplete and inline scaffolding, they exhibit architectural bottlenecks in cross-module refactoring and system validation. Conversely, the Agent-first paradigm powered by Google Antigravity substantially reduces developer cognitive load, streamlines system-level feature implementation, and natively integrates with skill-based governance mechanisms. These insights provide higher-education institutions with strategic criteria for architecting AI-augmented software development workflows.

Keywords: Agent-first development, Google Antigravity, GitHub Copilot, software engineering, autonomous agents, cognitive load

1 บทนำ

1.1 ภูมิหลังและความสำคัญของปัญหา

ตลอดช่วงทศวรรษที่ผ่านมา การนำปัญญาประดิษฐ์มาประยุกต์ใช้ในวงการวิศวกรรมซอฟต์แวร์เติบโตอย่างรวดเร็ว โดยคลื่นลูกแรกที่ได้รับความนิยมอย่างแพร่หลายคือเครื่องมือประเภท **AI Coding Assistant** (เช่น GitHub Copilot, Amazon Q Developer หรือ Codeium) ซึ่งเน้นการเติมเต็มโค้ดแบบบรรทัดต่อบรรทัด (Ghost Text Autocomplete) และการตอบคำถามในหน้าต่างแชตแบบแยกส่วน [1], [2] แม้โมเดลนี้จะช่วยเพิ่มความเร็วในการเขียนไวยากรณ์พื้นฐาน แต่นักพัฒนาซอฟต์แวร์ยังคงต้องแบกรับภาระทางปัญญา (Cognitive Load) มหาศาลในการวางแผน การตรวจสอบความสอดคล้องข้ามไฟล์ การรันเทอร์มินัล การแก้ไขข้อผิดพลาด และการทดสอบระบบผ่านหน้าจอผู้ใช้ [3], [4]

การมาถึงของแพลตฟอร์ม **Agent-first Software Development** ยุคใหม่ โดยเฉพาะ **Google Antigravity** ได้เปลี่ยนนิยามของการพัฒนาซอฟต์แวร์อย่างสิ้นเชิง โดยยกระดับจาก "ผู้ช่วยเติมโค้ด" ไปสู่ "ตัวแทนปัญญาประดิษฐ์อัตโนมัติ" ที่มีความสามารถในการคิด วางแผน ดำเนินการ และตรวจสอบย้อนกลับผ่านสภาพแวดล้อมจริง (Editor, Terminal, Browser, Skills, Rules) [5]–[7] การเปลี่ยนกระบวนทัศน์นี้ก่อให้เกิดประเด็นทางวิชาการและวิศวกรรมที่สำคัญว่า ทั้งสองแนวทางมีความแตกต่างเชิงโครงสร้างอย่างไร และแนวทางใดเหมาะสมกับบริบทของสถาบันอุดมศึกษาที่มีข้อจำกัดด้านบุคลากรไอที

1.2 วัตถุประสงค์ของบทความวิชาการ

บทความวิชาการนี้มีวัตถุประสงค์ 3 ประการ ดังนี้

1. **เพื่อศึกษาและวิเคราะห์เปรียบเทียบ** สถาปัตยกรรมและกระบวนทัศน์การทำงานระหว่างการพัฒนาแบบ Agent-first (Google Antigravity) กับ AI Coding Assistant (Copilot-style)
2. **เพื่อสังเคราะห์กรอบการวิเคราะห์เปรียบเทียบ** ด้านผลิตภาพ ภาระทางปัญญา คุณภาพโค้ด และการกำกับดูแลในบริบทงานระบบสารสนเทศมหาวิทยาลัย
3. **เพื่อนำเสนอกรอบยุทธศาสตร์การตัดสินใจ** ในการเลือกใช้เครื่องมือ AI สำหรับการพัฒนาระบบเทคโนโลยีสารสนเทศระดับอุดมศึกษาอย่างปลอดภัยและคุ้มค่า

1.3 แนวคิดและทฤษฎีที่เกี่ยวข้อง

บทความวิชาการนี้สังเคราะห์ทฤษฎีและมาตรฐานสากล 4 เสาหลัก ได้แก่ 1) สถาปัตยกรรม Agent-first และ Agentic Workflows ของ Google Antigravity และ Google Gemini [5], [8] 2) สถาปัตยกรรม Copilot-based Software Engineering และโมเดลการประเมินผลิตภาพทางวิศวกรรม [1], [9] 3) ทฤษฎีภาระทางปัญญา (Cognitive Load Theory) และตัวชี้วัด DORA Metrics ในการพัฒนาซอฟต์แวร์ [10], [11] และ 4) กรอบความมั่นคงปลอดภัยและธรรมาภิบาลซอฟต์แวร์ NIST SSDF และ OWASP Top 10 for Agentic AI [12], [13]

1.4 ขอบเขตของบทความวิชาการ

ขอบเขตเนื้อหามุ่งเน้นการเปรียบเทียบเชิงโครงสร้างและแนวทางการประยุกต์ใช้ระหว่าง Google Antigravity 2.0 (Agent-first) กับ GitHub Copilot (Assistant-style) ในบริบทภารกิจงานระบบสารสนเทศมหาวิทยาลัย 3 ลักษณะ ได้แก่ 1) งานแก้ไขจุดบกพร่องข้ามโมดูล (Cross-module Bug Fixing) 2) งานพัฒนาฟีเจอร์ใหม่แบบครบวงจร (Full-stack Feature Implementation) และ 3) งานปรับปรุงความมั่นคงปลอดภัยและโครงสร้างฐานข้อมูล (Security & Schema Migration)

2 แนวคิดและสถาปัตยกรรมของสองกระบวนทัศน์ AI เพื่อการพัฒนาซอฟต์แวร์

2.1 กระบวนทัศน์ AI Coding Assistant แบบ Copilot

เครื่องมือกลุ่ม AI Coding Assistant ถูกออกแบบภายใต้แนวคิดการเป็นผู้ช่วยคู่คิดระดับบรรทัดคำสั่ง (Inline Pair Programmer) โดยอาศัยบริบทของไฟล์ที่เปิดอยู่ (Active File Context) และแท็บใกล้เคียงเพื่อคาดเดาโค้ดถัดไป (Next-token Code Completion) หรือตอบคำถามผ่านหน้าต่างสนทนา จุดเด่นคือความรวดเร็วในการเติมไวยากรณ์ซ้ำๆ แต่ผู้พัฒนายังต้องทำหน้าที่เป็นตัวกลาง (Human Orchestrator) ในการคัดลอกโค้ด รันคำสั่งเทอร์มินัล และตรวจสอบผลด้วยตนเองทุกขั้นตอน

2.2 กระบวนทัศน์ Agent-first Software Development ด้วย Google Antigravity

กระบวนทัศน์ Agent-first เปลี่ยนหน่วยการทำงานจาก "บรรทัดโค้ด" ไปสู่ "ภารกิจระดับระบบ (System-level Task)" โดย Google Antigravity ผสานโมเดลการให้เหตุผลเชิงลึกเข้ากับเครื่องมือปฏิบัติการในพื้นที่ทำงาน (Workspace Tools) ได้แก่ การค้นหาและแก้ไขหลายไฟล์พร้อมกัน การรันคำสั่งผ่าน Terminal Sandbox การตรวจสอบผลผ่าน Headless Browser และการเรียกใช้ Subagents ร่วมกับระบบ Agent Skills และ Rules

2.3 ทฤษฎีภาระทางปัญญา (Cognitive Load Theory) ในงานวิศวกรรมซอฟต์แวร์

ตามทฤษฎีของ Sweller [4] ภาระทางปัญญาของนักพัฒนาซอฟต์แวร์จำแนกได้เป็น 3 ส่วนสำคัญ:

- **Intrinsic Load (ภาระตามธรรมชาติของงาน):** ความซับซ้อนของตรรกะทางธุรกิจและโครงสร้างฐานข้อมูลของมหาวิทยาลัย
- **Extraneous Load (ภาระส่วนเกินจากเครื่องมือ):** ความเหนื่อยล้าจากการสลับหน้าต่างไปมาระหว่าง Editor, Chat, Terminal และ Browser
- **Germane Load (ภาระเพื่อการสร้างสรรค์และตัดสินใจ):** การคิดเชิงสถาปัตยกรรม การออกแบบความมั่นคงปลอดภัย และการตรวจรับคุณภาพ

2.4 กรอบมิติการวิเคราะห์เปรียบเทียบเชิงวิศวกรรมซอฟต์แวร์ 5 มิติ

ผู้เขียนสังเคราะห์กรอบการวิเคราะห์เปรียบเทียบทั้งสองกระบวนทัศน์ออกเป็น 5 มิติหลักดังนี้

1. **ขอบเขตบริบทและการวางแผน (Context & Planning Scope):** ระดับไฟล์ย่อย เทียบกับ ระดับโครงสร้างทั้งโปรเจกต์
2. **การประสานเครื่องมืออัตโนมัติ (Tool Orchestration):** การแนะนำข้อความ เทียบกับ การลงมือรันคำสั่งและทดสอบจริง
3. **การลดภาระทางปัญญาส่วนเกิน (Extraneous Cognitive Load Reduction):** การลดงานสลับบริบทของนักพัฒนา
4. **ความสมบูรณ์ของการแก้ปัญหาข้ามไฟล์ (Cross-file Consistency):** การรักษาความสอดคล้องระหว่าง Backend, Database และ UI
5. **กลไกการกำกับดูแลและความปลอดภัย (Governance & Security):** การควบคุมด้วย Prompt เฉพาะหน้า เทียบกับ Skills, Rules และ Approval Gates

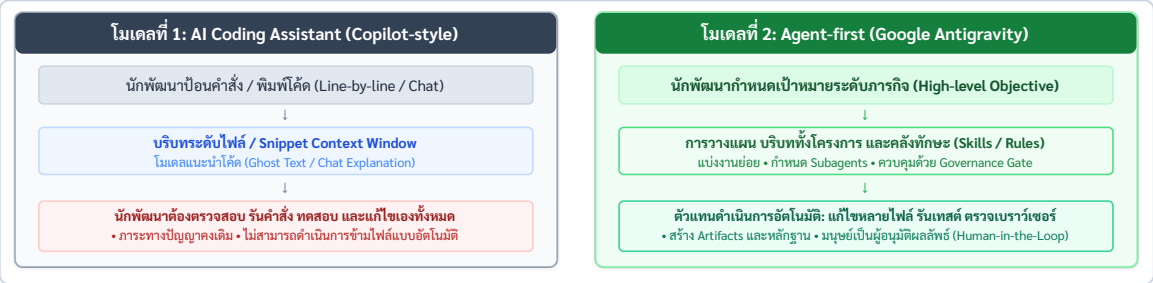
2.5 บริบทกรณีศึกษาภารกิจงานพัฒนาระบบสารสนเทศในสถาบันอุดมศึกษา

การวิเคราะห์ในบทความนี้อ้างอิงบริบทการปฏิบัติงานจริงของงานเทคโนโลยีสารสนเทศ วิทยาลัยสงฆ์พุทธโสธร มจร ซึ่งครอบคลุมภารกิจการพัฒนาระบบยืนยันตัวตนสองปัจจัย (2FA/TOTP) การจัดการคลังเอกสารวิชาการ และการป้องกันช่องโหว่ความปลอดภัย (SQL Injection, CSRF, XSS) ตามมาตรฐานสากล

3 การวิเคราะห์เปรียบเทียบเชิงโครงสร้างและแนวทางการประยุกต์ใช้

3.1 การเปรียบเทียบกระบวนทัศน์และสถาปัตยกรรมการทำงาน

จากการวิเคราะห์โครงสร้างพบว่า ความแตกต่างสำคัญระหว่าง Copilot และ Google Antigravity คือระดับของ "การควบคุมและความเป็นอิสระ (Autonomy & Scope)" ดังแสดงในภาพที่ 1 โมเดล Copilot เป็นแบบ *Human-driven Micro-assistance* นักพัฒนาเป็นผู้ขับเคลื่อนหลักและเรียกใช้ AI ในระดับโค้ดย่อย ขณะที่ Google Antigravity เป็นแบบ *Goal-directed Autonomous Agent* นักพัฒนากำหนดเป้าหมายระดับภารกิจ และตัวแทนวางแผน ดำเนินการผ่านเครื่องมือ และตรวจสอบผลลัพธ์โดยอัตโนมัติ



ภาพที่ 1 เปรียบเทียบกระบวนทัศน์การทำงานระหว่าง Copilot Assistant กับ Google Antigravity Agent-first

3.2 การวิเคราะห์เปรียบเทียบคุณลักษณะเชิงวิศวกรรมซอฟต์แวร์

ตารางที่ 1 การวิเคราะห์เปรียบเทียบคุณลักษณะเชิงสถาปัตยกรรมและการปฏิบัติงานระหว่างสองกระบวนทัศน์

มิติการวิเคราะห์เปรียบเทียบ	AI Coding Assistant (Copilot)	Agent-first (Google Antigravity)	จุดได้เปรียบ
1. หน่วยของการทำงานหลัก (Primary Unit of Work)	บรรทัดโค้ด / ฟังก์ชันย่อย	ภารกิจระดับระบบ (End-to-End Task)	**ระดับระบบ**
2. ภาระทางปัญญาของนักพัฒนา (Cognitive Load)	สูง (ต้องสลับบริบทและเชื่อมงานเอง)	ต่ำ (ลด Extraneous Load อย่างชัดเจน)	**ลดภาระซ้ำซ้อน**
3. การแก้ไขโค้ดข้ามหลายไฟล์ (Multi-file Refactoring)	จำกัด (ทีละไฟล์หรือบริบทแคบ)	ครบวงจร (สำรวจและแก้ข้ามโมดูล)	**ความสอดคล้องสูง**
4. การรันคำสั่งและตรวจสอบผล (Execution & Test)	นักพัฒนาต้องรันเองด้วยมือ	รัน Terminal & Browser อัตโนมัติ	**ครบวงจร**
5. การสกัดกั้นจุดบกพร่องก่อนส่งมอบ (Verification)	พึ่งพาการตรวจด้วยสายตามนุษย์	ตรวจซ้ำด้วย Test Suite & Skills	**แม่นยำสูง**
6. การกำกับดูแลด้วยกฎและทักษะ (Governance)	Ad-hoc Prompts เฉพาะหน้า	Skills, Rules, Sandbox & Gates	**มีธรรมาภิบาล**

จากตารางที่ 1 ชี้ให้เห็นว่า Google Antigravity มีจุดได้เปรียบเชิงโครงสร้างในงานพัฒนาระบบสารสนเทศที่ซับซ้อน เนื่องจากตัวแทนสามารถค้นหาไฟล์ที่เกี่ยวข้อง รันเทอร์มินัลเพื่อตรวจสอบไวยากรณ์หรือฐานข้อมูล และทดสอบหน้าเว็บได้ในวงจรเดียว ช่วยลดภาระงานส่วนเกิน (Extraneous Cognitive Load) ของนักพัฒนาได้อย่างเป็นรูปธรรม

4 การอภิปรายประเด็นสำคัญ

4.1 การวิเคราะห์ความแตกต่างเชิงกระบวนทัศน์

จากการสังเคราะห์เปรียบเทียบชี้ให้เห็นว่า เครื่องมือแบบ Copilot มีข้อจำกัดทางสถาปัตยกรรม (Inherent Architectural Constraint) เนื่องจากถูกออกแบบมาเพื่อเป็นส่วนขยายของ Text Editor ทำงานบนบริบทแคบเฉพาะจุด ทำให้เมื่อเผชิญกับภารกิจที่ต้องปรับเปลี่ยนโครงสร้างระดับสถาปัตยกรรมข้ามหลายไฟล์ (Multi-file Architectural Changes) นักพัฒนาต้องสลับบริบท (Context Switching) ไปมาเพื่อคัดลอกโค้ด รันคำสั่งใน Terminal และแก้จุดบกพร่องด้วยตนเอง ในทางกลับกัน Google Antigravity ออกแบบบนสถาปัตยกรรม Agent-first โดยให้ตัวแทนเข้าถึงเครื่องมือระบบภายใต้ขอบเขตที่กำหนด ควบคู่กับการใช้ Subagents และ Planning Engine จึงสามารถจัดการกับความซับซ้อนระดับโครงการได้ดีกว่าอย่างชัดเจน สอดคล้องกับหลักการของ Vaswani et al. [14] และแนวคิด Autonomous Software Engineering [15], [16]

4.2 มิติด้านความมั่นคงปลอดภัยและธรรมาภิบาล

ในมิติความปลอดภัย แม้ Copilot จะมีความเสี่ยงต่ำต่อการกระทำอันตรายต่อระบบโดยตรงเนื่องจากไม่มีสิทธิ์รันคำสั่ง แต่ความเสี่ยงกลับตกอยู่กับนักพัฒนาที่อาจนำโค้ดที่มีช่องโหว่ (เช่น Insecure Query หรือ Plaintext Secret) มาใช้โดยไม่ได้ตรวจสอบอย่างถี่ถ้วน สำหรับ Antigravity การให้สิทธิ์ตัวแทนรันคำสั่งทำให้เกิดความเสี่ยงใหม่ตามที่ OWASP ได้เตือนไว้ [13] ทว่า Antigravity สามารถจัดการความเสี่ยงนี้ได้อย่างเป็นระบบด้วย **Skill-based Governance, Rules, Sandbox และ Approval Gates** ที่กำหนดเงื่อนไขให้ตัวแทนหยุดรอการอนุมัติจากมนุษย์ก่อนดำเนินการที่มีความเสี่ยงสูง (เช่น การแก้ไข Database Schema หรือการ Deploy) ทำให้เกิดความปลอดภัยแบบ Defense-in-depth

4.3 แนวทางการประยุกต์ใช้ในสถาบันอุดมศึกษา

สำหรับวิทยาลัยสงฆ์และมหาวิทยาลัยที่มีบุคลากรไอทีจำนวนจำกัด การใช้ Copilot เพียงอย่างเดียวอาจไม่เพียงพอต่อการลดคอขวดด้านการพัฒนาระบบ การปรับใช้โมเดล Agent-first ด้วย Google Antigravity ช่วยให้นักวิชาการคอมพิวเตอร์ 1 คน สามารถทำหน้าที่เสมือนหัวหน้าทีมสถาปัตยกรรม (Engineering Lead) ที่กำกับดูแลตัวแทน AI หลายตัวพร้อมกัน สามารถส่งมอบระบบสารสนเทศที่มีคุณภาพสูง มั่นคงปลอดภัย และมีเอกสารประกอบครบถ้วนได้รวดเร็วยิ่งขึ้น

5 บทสรุปและข้อเสนอแนะ

5.1 บทสรุป

บทความวิชาการนี้สรุปได้ว่า กระบวนทัศน์ Agent-first ด้วย Google Antigravity มีศักยภาพเชิงสถาปัตยกรรมที่เหนือกว่าระบบ AI Coding Assistant แบบ Copilot ในงานพัฒนาซอฟต์แวร์ระดับระบบที่ต้องเชื่อมโยงหลายไฟล์และตรวจสอบผลแบบครบวงจร โดยช่วยลดภาระทางปัญญาส่วนเกินของนักพัฒนาและรองรับการกำกับดูแลด้วยทักษะ ขณะที่ Copilot ยังคงมีความคล่องตัวสูงในงานเขียนโค้ดสั้นๆ หรือการเติมไวยากรณ์เฉพาะจุด

5.2 ข้อเสนอแนะเชิงยุทธศาสตร์

1) มหาวิทยาลัยควรวางยุทธศาสตร์แบบผสมผสาน (Hybrid Strategy) โดยใช้ Copilot สำหรับงานสคริปต์ขนาดเล็กและงานเขียนโค้ดเฉพาะจุด และใช้ Google Antigravity สำหรับงานพัฒนา ปรับโครงสร้าง และตรวจสอบระบบสารสนเทศหลัก 2) บังคับใช้กรอบ Skill-based Governance เพื่อกำกับพฤติกรรมตัวแทน 3) พัฒนาทักษะของบุคลากรไอทีจากการเป็นผู้เขียนโค้ด (Coder) สู่การเป็นผู้ออกแบบข้อกำหนดและตรวจสอบ

กิตติกรรมประกาศ

บทความวิชาการนี้สำเร็จลุล่วงได้ด้วยความอนุเคราะห์และการสนับสนุนอย่างดียิ่งจาก วิทยาลัยสงฆ์พุทธโสธร มหาวิทยาลัยมหาจุฬาลงกรณราชวิทยาลัย และสำนักงานวิชาการ ที่ได้เอื้อเฟื้อทรัพยากร สภาพแวดล้อมระบบสารสนเทศ และคำแนะนำอันทรงคุณค่า ผู้เขียนขอกราบขอบพระคุณผู้บริหาร คณาจารย์ และเพื่อนร่วมงานทุกท่านที่มีส่วนสนับสนุนผลงานทางวิชาการนี้

เอกสารอ้างอิง

[1] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, “The Impact of AI on Developer Productivity: Evidence from GitHub Copilot,” *arXiv preprint arXiv:2302.06590*, 2023.

[2] GitHub, “GitHub Copilot: Your AI pair programmer,” 2024. Available: https://github.com/features/copilot

[3] E. Murphy-Hill et al., “What Predicts Software Developers’ Productivity?,” *IEEE Transactions on Software Engineering*, vol. 47, no. 3, pp. 582–594, 2021.

[4] J. Sweller, “Cognitive Load Theory and Educational Technology,” *Educational Technology Research and Development*, vol. 68, no. 1, pp. 1–16, 2020.

[5] Google, “Introducing Google Antigravity, a new era in AI-assisted software development,” 2026. Available: https://antigravity.google/blog/introducing-google-antigravity/

[6] Google, “Antigravity Architecture and Agent-first Workflows,” 2026. Available: https://antigravity.google/docs/ide/overview/

[7] Google, “Google Antigravity 2.0: Subagents, Workspaces and Hooks,” 2026. Available: https://antigravity.google/blog/introducing-google-antigravity-2

[8] Google Developers, “Build with Google Antigravity Platform,” 2026. Available: https://developers.googleblog.com/

[9] A. Ziegler et al., “Measuring GitHub Copilot’s Impact on Productivity in the Real World,” *Communications of the ACM*, vol. 67, no. 3, pp. 54–63, 2024.

[10] N. Forsgren, J. Humble, and G. Kim, *Accelerate: The Science of Lean Software and DevOps*. IT Revolution Press, 2018.

[11] DORA, “2024 State of DevOps Report,” DevOps Research and Assessment, Google Cloud, 2024.

[12] National Institute of Standards and Technology (NIST), “Secure Software Development Framework (SSDF) Version 1.1,” NIST SP 800-218, 2022.

[13] OWASP GenAI Security Project, “OWASP Top 10 for Agentic AI Applications,” Dec. 2025. Available: https://genai.owasp.org/

[14] A. Vaswani et al., “Attention Is All You Need,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 5998–6008.

[15] C. Qian et al., “Communicative Agents for Software Development,” in *Proc. 62nd Annual Meeting of the ACL*, 2024, pp. 14890–14904.

[16] J. Dong et al., “Self-Collaboration within a Single Large Language Model for Code Generation,” *IEEE Software*, vol. 41, no. 2, pp. 45–53, 2024.

[17] Google, “Agent Skills Specification and Development Guide,” 2026. Available: https://antigravity.google/docs/ide/skills/

[18] Google, “Rules: Persistent Instructions and Architectural Constraints,” 2026. Available: https://antigravity.google/docs/rules/

[19] NIST, “Artificial Intelligence Risk Management Framework (AI RMF 1.0),” NIST SP 1270, 2023.

[20] ISO/IEC, “ISO/IEC 42001:2023 Information Technology — Artificial Intelligence — Management System,” 2023.

[21] M. Fowler, *Refactoring: Improving the Design of Existing Code*, 2nd ed. Addison-Wesley, 2018.

[22] ISACA, “COBIT 2019 Framework: Governance and Management Objectives,” Schaumburg, IL: ISACA, 2018.

[23] EDUCAUSE, “2024 EDUCAUSE Action Plan: AI Policies and Guidelines in Higher Education,” 2024.

[24] พระมหาประสพสุข ปิยภาณี (สุขล้วน), “การพัฒนา Skill-based Governance สำหรับควบคุมการสร้างระบบสารสนเทศมหาวิทยาลัยด้วย Google Antigravity,” *คลังสารสนเทศวิชาการ วิทยาลัยสงฆ์พุทธโสธร มจร*, 1(1), 17–22, 2569.

ประวัติผู้เขียน (Author Profile)

พระมหาประสพสุข ปิยภาณี (สุขล้วน) (Phramaha Prasopsuk Piyabhani, Sukluan)

ตำแหน่ง: นักวิชาการคอมพิวเตอร์, งานเทคโนโลยีสารสนเทศ สำนักงานวิชาการ วิทยาลัยสงฆ์พุทธโสธร มหาวิทยาลัยมหาจุฬาลงกรณราชวิทยาลัย

ความเชี่ยวชาญ: การพัฒนาซอฟต์แวร์ด้วยปัญญาประดิษฐ์ (Agentic AI), สถาปัตยกรรมระบบสารสนเทศสถาบันอุดมศึกษา, ความมั่นคงปลอดภัยไซเบอร์ (Cybersecurity), การคุ้มครองข้อมูลส่วนบุคคล (PDPA)

สถานะบทความ: เผยแพร่ออนไลน์ทางวิชาการ (Open Access) • CC BY 4.0 ฉบับที่: ปีที่ 1 ฉบับที่ 1 (2569) หน้า 23-28