

วิศวกรรมซอฟต์แวร์

วิศวกรรมซอฟต์แวร์ คืออะไร

วิศวกรรมซอฟต์แวร์ หรือ Software Engineering คือศาสตร์ที่ประยุกต์ใช้หลักการทางวิศวกรรมในกระบวนการพัฒนาซอฟต์แวร์ ตั้งแต่การวางแผน ออกแบบ พัฒนาทดสอบ การนำไปใช้งาน และไปจนถึงการบำรุงรักษาซอฟต์แวร์



ถ้ามองให้ลึกลงไป วิศวกรรมซอฟต์แวร์ ก็ถือเป็นศาสตร์แห่งการสร้างสรรคอย่างหนึ่ง เพราะการจะสร้างผลิตภัณฑ์ซอฟต์แวร์ซักตัว นอกจากความรู้กับทักษะด้านพัฒนาซอฟต์แวร์แล้ว นักพัฒนา (Dev.) ก็จำเป็นต้องสามารถคิดนอกกรอบ และสร้างสรรค์นวัตกรรมใหม่ ๆ ขึ้นมาได้ เพื่อให้สอดคล้องกับพฤติกรรมผู้ใช้ ที่เปลี่ยนไปอยู่เสมอ

วิวัฒนาการของวิศวกรรมซอฟต์แวร์

วิศวกรรมซอฟต์แวร์ เป็นสาขาที่เกิดขึ้นจากความต้องการที่จะพัฒนากับจัดการซอฟต์แวร์อย่างมีระบบและเป็นวิทยาศาสตร์, ในช่วงต้นยุค 1940-1950 การพัฒนาซอฟต์แวร์ได้เริ่มต้นขึ้นพร้อมกับการกำเนิดของคอมพิวเตอร์รุ่น

แรก แต่ในช่วงนั้นยังไม่มีการจัดการหรือกระบวนการที่ชัดเจน ส่งผลให้เกิดปัญหาเรื่องคุณภาพและเวลาในการพัฒนาที่ยืดเยื้อ

การเปลี่ยนแปลงสำคัญเกิดขึ้นในช่วงยุค 1960 เมื่อวงการซอฟต์แวร์ได้ตระหนักถึงความสำคัญเรื่องการจัดการกับวิธีการพัฒนาที่มีระบบ ด้วยความพยายามที่จะกำหนดกระบวนการและมาตรฐานที่ดีขึ้น จึงเป็นจุดเริ่มต้นของ "วิศวกรรมซอฟต์แวร์" ในแบบที่เรารู้จักกัน และได้พัฒนาปรับปรุงกระบวนการมาอย่างต่อเนื่อง ตามการเปลี่ยนแปลงของเทคโนโลยีและความต้องการของตลาด

คุณสมบัติและทักษะจำเป็น เพื่อใช้ศึกษาวิศวกรรมซอฟต์แวร์

จำเป็นต้องมีทักษะทั้ง Hard skill และ Soft skill

ทักษะเฉพาะทาง (Hard Skill)

- **Hard Skills** คือ ทักษะความเชี่ยวชาญเชิงเทคนิคหรือความรู้เฉพาะทางที่จำเป็นต่อการทำงาน ซึ่งสามารถเรียนรู้ ฝึกฝน และวัดผลหรือประเมินระดับความสามารถได้อย่างเป็นรูปธรรม เช่น ผ่านใบประกาศนียบัตร วุฒิการศึกษา หรือการทดสอบภาคปฏิบัติ

- **ตัวอย่าง Hard Skill ที่สำคัญ**

ทักษะด้านภาษา: การสื่อสารด้วยภาษาต่างประเทศ

ทักษะด้านคอมพิวเตอร์และเทคโนโลยี: การใช้งานซอฟต์แวร์เฉพาะทาง, การเขียนโปรแกรม (Coding), หรือการวิเคราะห์ข้อมูล (Data Analysis)

ทักษะวิชาชีพเฉพาะด้าน: เช่น งานบัญชี, การแพทย์, กฎหมาย, หรือวิศวกรรม

ทักษะเฉพาะทาง (Soft Skill)

Soft Skill คือ ทักษะด้านอารมณ์ บุคลิกภาพ และมนุษยสัมพันธ์ ที่ใช้ในการสื่อสารและทำงานร่วมกับผู้อื่นได้อย่างมีประสิทธิภาพ ทักษะเหล่านี้วัดผลเป็นตัวเลขได้ยาก แต่มีความสำคัญมากต่อความสำเร็จในการทำงานและการใช้ชีวิต

ตัวอย่าง Soft Skill ที่สำคัญ

- **การสื่อสาร (Communication):** การพูด ฟัง และถ่ายทอดความคิดให้ผู้อื่นเข้าใจได้ง่าย
- **การทำงานเป็นทีม (Teamwork):** การร่วมมือกับคนอื่นเพื่อบรรลุเป้าหมายเดียวกัน
- **การปรับตัว (Adaptability):** ความยืดหยุ่นและการเรียนรู้สิ่งใหม่เมื่อสถานการณ์เปลี่ยนไป
- **ความฉลาดทางอารมณ์ (EQ):** การเข้าใจอารมณ์ตนเองและผู้อื่น รวมถึงการรับมือกับแรงกดดัน
- **การแก้ไขปัญหา (Problem Solving):** การมองปัญหาและหาทางออกอย่างสร้างสรรค์

ความแตกต่างระหว่าง Hard Skills และ Soft Skills

- **Hard Skills:** เป็นทักษะทางวิชาชีพที่เน้นความรู้เทคนิค วัดผลได้ชัดเจน ตรงไปตรงมา
- **Soft Skills:** เป็นทักษะด้านพฤติกรรม อารมณ์ และมนุษยสัมพันธ์ ซึ่งวัดผลเป็นคะแนนได้ยาก เช่น การทำงานเป็นทีม หรือการสื่อสาร

ทักษะเฉพาะทาง วิศวกรรมซอฟต์แวร์

1. **การเขียนโค้ด (Coding) :** ความรู้ความสามารถในการเขียนโค้ดอย่าง Java, Python, C++, PHP หรือ JavaScript เพื่อสั่งให้คอมพิวเตอร์ทำงานตามที่ต้องการ
2. **กระบวนทัศน์การเขียนโปรแกรม (Programming Paradigms) :** แนวคิดกับแนวทางการเขียนโค้ดที่แตกต่างกันหลายแบบ เช่นกระบวนทัศน์เชิงวัตถุ (Object-Oriented Programming) หรือกระบวนทัศน์เชิงฟังก์ชัน (Functional Programming)
3. **สถาปัตยกรรมซอฟต์แวร์ (Software Architecture) :** การออกแบบโครงสร้างและส่วนประกอบของซอฟต์แวร์ เพื่อให้ทำงานได้อย่างมีประสิทธิภาพ
4. **โครงสร้างข้อมูล และอัลกอริทึม (Data Structures/Algorithms) :** รูปแบบการจัดเก็บข้อมูลเพื่อให้เข้าถึงได้ง่าย และขั้นตอนการแก้ปัญหาที่กำหนดไว้อย่างเป็นระบบ
5. **ฐานข้อมูล (Database) :** การจัดการกับฐานข้อมูลขนาดใหญ่ เพื่อให้เข้าถึงข้อมูลได้สะดวกและรวดเร็ว

6. การทดสอบ (Testing) : กระบวนการตรวจสอบซอฟต์แวร์ เพื่อหาข้อผิดพลาดหรือความผิดปกติ
7. ความปลอดภัยของเว็บ (Web Security) : การป้องกันเว็บแอปพลิเคชัน จากภัยคุกคามด้านความปลอดภัยต่าง ๆ

เนื้อหาสำคัญจะถูกแบ่งออกเป็น 2 ส่วนหลัก ๆ ดังนี้

1. โครงสร้างข้อมูลพื้นฐาน (Data Structures)

เน้นเรียนรู้วิธีการจัดเก็บข้อมูลในหน่วยความจำของคอมพิวเตอร์อย่างเป็นระบบ เพื่อให้สามารถเข้าถึง ค้นหา หรือแก้ไขข้อมูลได้ในเวลาที่สั้นที่สุด:

- Linear Data Structures (โครงสร้างแบบเส้นตรง):
 - Array / Dynamic Array: การเก็บข้อมูลเรียงกันในหน่วยความจำ เข้าถึงด้วย Index ได้เร็วที่สุด
 - Linked List: การเก็บข้อมูลแบบกระจายตัวแต่เชื่อมโยงกันด้วย Pointer/Reference (เช่น Singly, Doubly, Circular Linked List)
 - Stack (LIFO): ข้อมูลที่เข้าทีหลังสุดจะถูกนำออกก่อน เช่น ระบบ Undo/Redo
 - Queue (FIFO): ข้อมูลที่เข้าก่อนจะออกก่อน เช่น ระบบจัดการคิวงาน (Job Scheduling)
- Non-Linear Data Structures (โครงสร้างแบบไม่เป็นเส้นตรง):
 - Tree & Binary Search Tree (BST): โครงสร้างแบบกิ่งก้าน ลำดับชั้น เหมาะกับการค้นหาข้อมูลปริมาณมาก
 - Graph: โครงสร้างเครือข่ายจุดเชื่อมโยง (Vertices & Edges) นิยมใช้กับระบบแผนที่หรือ Social Network
 - Hash Table / Hash Map: การเก็บข้อมูลรูปแบบ Key-Value ที่ให้ความเร็วในการค้นหาเกือบเป็นศูนย์ ($O(1)$)

2. การเขียนโปรแกรมและขั้นตอนวิธี (Coding & Algorithms)

เน้นการเขียนโค้ดเพื่อควบคุมโครงสร้างข้อมูลเหล่านั้น และการคิดแก้ปัญหาอย่างเป็นระบบ (Problem Solving)

- **Pointer & Memory Allocation:** สำหรับวิศวกรรมคอมพิวเตอร์ การทำความเข้าใจเรื่อง Pointer และการจองพื้นที่ใน Memory (Dynamic Memory) ถือเป็นหัวใจสำคัญในการสร้างโครงสร้างข้อมูลขึ้นมาเอง
- **Recursion:** การเขียนฟังก์ชันเรียกตัวเองเพื่อแก้ปัญหาซับซ้อน เช่น การเดินทางใน Tree
- **Sorting & Searching Algorithms:** เรียนรู้วิธีเรียงลำดับข้อมูล (Bubble, Insertion, Merge, Quick Sort) และการค้นหา (Linear, Binary Search)
- **Time & Space Complexity (Big-O Notation):** การวิเคราะห์ว่าโค้ดที่เราเขียนขึ้นมา ใช้เวลาประมวลผล (Time) และกินพื้นที่แรม (Space) มากน้อยแค่ไหนเมื่อข้อมูลมีขนาดใหญ่ขึ้น

กระบวนการพัฒนาซอฟต์แวร์" (Software Development Process) สำหรับสายวิศวกรรมคอมพิวเตอร์ จะเป็นการเรียนรู้วิธีการทำงานอย่างเป็นระบบเพื่อเปลี่ยนจาก "ไอเดีย" ให้กลายเป็น "ซอฟต์แวร์ที่ใช้งานได้จริง" มีคุณภาพ และส่งมอบได้ทันเวลา โดยเนื้อหาหลักจะแบ่งออกเป็นเฟสสำคัญตามวงจรชีวิตการพัฒนาซอฟต์แวร์ (SDLC - Software Development Life Cycle) และแนวคิดในการบริหารจัดการทีมดังนี้ครับ



1. วงจรชีวิตการพัฒนาซอฟต์แวร์ (SDLC Phases)

คุณจะได้เรียนรู้บทบาทและขั้นตอนในแต่ละเฟสของการทำระบบ ได้แก่:

1.1 วางแผน (Requirement Analysis)

ขั้นตอนแรกของ SDLC ที่เป็นการรวบรวม และวิเคราะห์ความต้องการจากผู้ที่มีส่วนได้ส่วนเสีย เพื่อระบุ requirement ของลูกค้าและกำหนดขอบเขตของโปรเจกต์ ขั้นตอนนี้คือจุดที่โปรเจกต์ส่วนใหญ่พลาดบ่อยที่สุด จึงมี บริการวางกระบวนการพัฒนา และเก็บ requirement ให้ครบก่อนเริ่มเขียนโค้ด

ทีมนักพัฒนาจะต้องสื่อสารกับลูกค้าและเก็บข้อมูลที่เกี่ยวข้องทั้งหมด เพื่อทำความเข้าใจวัตถุประสงค์ทางธุรกิจ ฟังก์ชันใช้งานที่ต้องการ และข้อจำกัดของระบบ รวมถึงรายละเอียดงบประมาณกับระยะเวลาของโปรเจกต์

เพื่อจัดทำเอกสารข้อกำหนดความต้องการ หรือ Software Requirements Specification (SRS) สำหรับใช้เป็นข้อตกลงร่วมกันระหว่างผู้พัฒนาซอฟต์แวร์กับผู้ว่าจ้าง

1.2 การออกแบบระบบ (System Design)

หลังจากที่กำหนด SRS ชัดเจนดีแล้ว ก็จะเริ่มกระบวนการออกแบบระบบซอฟต์แวร์สำหรับใช้พัฒนา โดยอิงตามเอกสารข้อกำหนดความต้องการที่ได้จากขั้นตอนก่อนหน้านี้

ขั้นตอนนี้เกี่ยวข้องกับการสร้างสถาปัตยกรรมของซอฟต์แวร์ การออกแบบโครงสร้างฐานข้อมูล อินเทอร์เฟซผู้ใช้ (UX/UI) ฯลฯ รวมถึงการกำหนดเทคโนโลยีและเครื่องมือที่ต้องใช้ในการพัฒนา โดยจะจัดทำเอกสารการออกแบบระบบ หรือ System Design Document ที่ระบุโครงสร้างกับการทำงานของซอฟต์แวร์อย่างละเอียด เพื่อใช้เป็นแนวทางสำหรับพัฒนาในขั้นตอนถัดไป

1.3 ดำเนินการพัฒนา (Implementation)

SDLC ขั้นตอนนี้ จะเป็นการเขียนโค้ดหรือการสร้างซอฟต์แวร์จริงตามเอกสารการออกแบบระบบ ที่ได้รับการอนุมัติแล้วจากขั้นตอนก่อนหน้านี้

นักพัฒนามักจะแบ่งงานออกเป็นหลายโมดูล และถูกพัฒนาโดยทีมหรือบุคคลที่แตกต่างกัน ขึ้นอยู่กับขนาดและความซับซ้อนของโปรเจกต์ หลังจากที่ได้แต่ละส่วนพัฒนาเสร็จสิ้นก็จะนำมารวมกันเป็นระบบใหญ่ วิธีการนี้อาจเกิดขึ้นหลายครั้งในระหว่างการพัฒนา เพื่อทดสอบแต่ละส่วนของโค้ดอย่างละเอียด (Unit Testing) และแก้ไขข้อผิดพลาดอย่างต่อเนื่อง ก่อนจะเข้าสู่ขั้นตอนการทดสอบระบบที่ครอบคลุมมากขึ้น

1.4 การทดสอบ (Testing)

หลังจากพัฒนาซอฟต์แวร์เสร็จ ขั้นตอนทดสอบก็จะเริ่มขึ้น เพื่อค้นหาข้อผิดพลาดและปัญหาภายในซอฟต์แวร์ ฝ่าย QA (Quality Assurance) จะทำการทดสอบหลายระดับ หลายประเภท และหลายเหตุการณ์ เพื่อให้มั่นใจว่าซอฟต์แวร์จะทำงานได้ตามที่คาดหวัง เสถียร และผ่านมาตรฐานความปลอดภัย เป็นขั้นตอนสำคัญในการประเมินคุณภาพก่อนที่จะถูกนำไปใช้งานจริง

1.5 ติดตั้งใช้จริง (Deployment)

ในกระบวนการ SDLC เมื่อซอฟต์แวร์ผ่านการทดสอบว่ามีคุณภาพ และตรงตาม requirement ของลูกค้าทุก
อย่าง ก็จะเข้าสู่ขั้นตอนการติดตั้งให้กับผู้ใช้

การติดตั้งที่ว่า หมายถึงการนำซอฟต์แวร์ออกจากสภาพแวดล้อมการพัฒนาทดสอบ และนำไปติดตั้งบน
เซิร์ฟเวอร์หรือระบบคอมพิวเตอร์ที่จะใช้งานจริง รวมถึงการตั้งค่าระบบให้ผู้ใช้ที่เกี่ยวข้องสามารถเข้าถึงได้

ระหว่างการติดตั้ง นักพัฒนาจำเป็นต้องดำเนินการอบรมผู้ใช้งาน เพื่อให้เข้าใจวิธีการใช้ซอฟต์แวร์ที่สร้าง
อย่างถูกต้อง อีกทั้งต้องจัดทำเอกสารคู่มืออธิบายรายละเอียดการใช้งานเบื้องต้น เพื่อให้ซอฟต์แวร์ถูกนำไปใช้ได้
ตามที่คาดหวัง

1.6 การบำรุงรักษา (Maintenance)

SDLC ขั้นตอนสุดท้ายหลังจากที่ซอฟต์แวร์ถูกติดตั้งใช้งานจริงเรียบร้อยแล้ว ก็ยังจำเป็นต้องได้รับการดูแลและ
ปรับปรุงอย่างต่อเนื่อง เพื่อให้แน่ใจว่าซอฟต์แวร์จะยังคงทำงานได้ดีไปตามการเปลี่ยนแปลงของผู้ใช้และธุรกิจ

การ Maintenance ไม่ใช่เพียงการแก้ไขข้อผิดพลาดที่อาจเกิดขึ้นเท่านั้น แต่รวมถึงการอัปเดต การเพิ่ม
ฟังก์ชันใหม่ และการพัฒนาระบบให้เข้ากับเทคโนโลยีล่าสุดอีกด้วย มักต้องเกิดขึ้นเรื่อย ๆ เพื่อคุณภาพกับความ
น่าเชื่อถือของซอฟต์แวร์ในระยะยาว

2. ประเภท SDLC Model ทั้ง 7 แบบ

SDLC Model คือแบบแผนการทำงานที่ใช้พัฒนาซอฟต์แวร์ ให้อยู่ในกรอบที่ควบคุมได้ แต่ละโมเดลของ
SDLC จะมีวิธีการกับขั้นตอนที่ต่างกัน แต่โมเดลทั้งหมดล้วนมีเป้าหมายร่วมกัน ในการทำให้กระบวนการพัฒนา
ซอฟต์แวร์เป็นไปได้อย่างราบรื่น และสามารถสร้างซอฟต์แวร์ที่ตอบสนองผู้ใช้ได้เป็นอย่างดี สำหรับ SDLC Model
ที่นิยมใช้และรู้จักกันดี จะมีอยู่ 7 โมเดล ตามต่อไปนี้

2.1 Big Bang Model

Big Bang model เป็นโมเดล SDLC ที่เน้นไปที่การเขียนโค้ดของซอฟต์แวร์โดยตรง ไม่เน้นการวางแผนหรือการทำงานทีละขั้นตอนอย่างเป็นระบบ การพัฒนาจะเริ่มต้นโดยไม่มีแผนที่ละเอียดและโครงสร้างไม่ชัดเจน เหมาะกับโปรเจกต์ขนาดเล็กที่มีนักพัฒนา 1-2 คนทำงานร่วมกันในระยะเวลาพัฒนาที่จำกัด มักจะทดสอบซอฟต์แวร์เมื่อเสร็จหรือใกล้เสร็จในที่เดียว

ข้อดีของ Big Bang Model

- ใช้งบประมาณไม่สูง เหมาะกับโปรเจกต์ขนาดเล็ก

- ความยืดหยุ่นในการทำงานสูง สามารถทำงานได้อย่างอิสระ

- ไม่ต้องเสียเวลาวางแผนนาน หรือจัดทำเอกสารให้วุ่นวาย

ข้อพิจารณาของ Big Bang Model

- มีความเสี่ยงสูงที่จะพบข้อผิดพลาด หรือบัคจำนวนมาก

- อาจไม่ได้ฟีเจอร์ตามที่ผู้จ้าง requirement มาทุกอย่าง

- ไม่เหมาะกับโปรเจกต์ขนาดใหญ่ หรือที่มีความซับซ้อนสูง

- หากต้องการเอกสาร SRS หรือคู่มือการใช้งานซอฟต์แวร์ โมเดลนี้จะไม่ให้

2.2 Waterfall Model

Waterfall model เป็นโมเดล SDLC เก่าแก่ แต่ยังคงได้รับความนิยมมาก เพราะมีลำดับขั้นตอนพัฒนาซอฟต์แวร์ที่ชัดเจน เรียงต่อกันเป็นลำดับ แต่ละขั้นต้องเสร็จเรียบร้อยก่อนถึงจะเริ่มขั้นถัดไปได้ โมเดลนี้มีโครงสร้างที่เข้าใจง่ายแต่ใช้เวลาทำนาน เหมาะกับโปรเจกต์ที่มีความต้องการกับข้อกำหนดชัดเจน แน่นนอน และไม่มีการเปลี่ยนแปลงบ่อย

ข้อดีของ Waterfall Model

- ไพลนซอฟต์แวร์มีคุณภาพสูง และได้รับตาม requirement ที่ต้องการทั้งหมด

ข้อผิดพลาด หรือบั๊กมีน้อยมากจนแทบไม่มี เพราะมีการทดสอบตลอดการพัฒนา และรับฟีดแบ็กมาปรับปรุงให้ดีขึ้นอยู่เสมอ

ข้อพิจารณาของ Waterfall Model

ไม่เหมาะกับโปรเจกต์ที่ requirement ไม่ชัดเจน เปลี่ยนแปลงบ่อย หรือมีเอกสาร SRS ไม่ครบถ้วน

ใช้ระยะเวลาพัฒนาซอฟต์แวร์นาน ทำให้งบประมาณต้องสูงขึ้นตามไปด้วย

แก้ไขได้ยุ่งยากหากพบปัญหา หรือข้อผิดพลาด เพราะไม่สามารถย้อนกลับไปทำขั้นตอนก่อนหน้านี้ได้

2.3 V-Shaped Model

V-Shaped model เป็นโมเดล SDLC ที่เน้นการทดสอบอย่างเข้มงวดในแต่ละขั้นตอน มีลำดับขั้นตอนที่ชัดเจน สามารถย้อนกลับไปทำขั้นก่อนหน้าได้ โมเดลนี้เป็นการขยายและปรับปรุงมาจาก Waterfall model ด้วยการเพิ่มส่วนการทดสอบเข้ามาในทุกขั้นตอน มีจุดเด่นเรื่องการรับประกัน ว่าในแต่ละขั้นตอนการพัฒนาจะได้รับการทดสอบอย่างถี่ถ้วน ทำให้ข้อผิดพลาดหรือบั๊กถูกตรวจพบ และแก้ไขได้ตั้งแต่ระยะเริ่มต้นพัฒนาซอฟต์แวร์

ข้อดีของ V-Shaped Model

การทดสอบอย่างเข้มงวดในแต่ละขั้นตอน ช่วยให้ซอฟต์แวร์มีคุณภาพสูง ข้อผิดพลาดมีน้อยมากหรือไม่มีเลย

ตรวจพบและแก้ไขปัญหาได้ตั้งแต่การพัฒนาระยะแรก ลดความซับซ้อนในการแก้ไขปัญหาที่อาจเกิดขึ้นในภายหลัง

มีขั้นตอนการทำงานที่เป็นระบบ สามารถติดตามความคืบหน้าของโปรเจกต์ได้ง่าย

ข้อพิจารณาของ V-Shaped Model

ไม่เหมาะกับโปรเจกต์ที่ requirement ไม่ชัดเจน หรือมีเอกสาร SRS ไม่ครบถ้วน

การเปลี่ยนแปลงความต้องการ หรือข้อกำหนดในภายหลังนั้นทำได้ยาก และต้องใช้งบประมาณสูง

2.4 Iterative Model

Iterative Model เป็นโมเดล SDLC ขนาดใหญ่ ที่แบ่งการทำงานออกเป็นส่วนย่อย ๆ เน้นการพัฒนาผ่านการทำวนซ้ำหลายครั้ง ในแต่ละรอบของการวนซ้ำจะมุ่งเน้นไปที่การปรับปรุงและเพิ่มเติมฟีเจอร์เฉพาะไปเรื่อย ๆ สิ่งนี้จะช่วยให้ทีมพัฒนาตอบสนองต่อ requirement ที่เปลี่ยนแปลงได้อย่างรวดเร็ว

SDLC Model นี้ยังช่วยให้ผู้ใช้กับผู้มีส่วนได้ส่วนเสียได้มีโอกาสให้ข้อเสนอแนะ และมีส่วนร่วมในกระบวนการพัฒนา ทำให้เฟออลซอฟต์แวร์ออกมาตอบสนองความต้องการของผู้ใช้ได้เป็นอย่างดี

ข้อดีของ Iterative Model

ปรับเปลี่ยนและเพิ่มเติมฟีเจอร์ได้สะดวก เหมาะกับโปรเจกต์ซอฟต์แวร์ที่ requirement ไม่แน่นอน หรืออาจได้เปลี่ยนแปลงบ่อย

ปัญหาข้อผิดพลาดถูกตรวจเจอได้เร็ว จากการทำวนซ้ำขั้นตอนหลายรอบ

มีการรับฟีดแบ็คจากผู้ใช้งานระหว่างการพัฒนา และนำมาปรับปรุงซอฟต์แวร์ให้ตอบเจทย์ดีมากขึ้น

ข้อพิจารณาของ Iterative Model

ใช้เวลาและทรัพยากรสูงมาก เพื่อทำซ้ำขั้นตอนของ SDLC หลายครั้ง

งานเพิ่มจากการขยายขอบเขตไปเรื่อยแบบไม่มีสิ้นสุด (Scope Creep) การวนซ้ำแต่ละรอบจึงต้องอาศัยความระมัดระวังสูง

ต้นทุนโดยรวมของโปรเจกต์ที่ใช้อาจสูงมากขึ้น ตามการเปลี่ยนแปลงที่เกิดขึ้นต่อเนื่อง

2.5 RAD Model

RAD model หรือ Rapid Application Development โมเดล SDLC ที่เน้นขั้นตอนรวบรัด ด้วยการเลือกใช้เครื่องมือกับเทคนิคที่ช่วยให้การพัฒนาซอฟต์แวร์ดำเนินไปได้อย่างรวดเร็ว นักพัฒนาจะการสร้างต้นแบบ (Prototype) ของซอฟต์แวร์ขึ้นมา และนำไปทดสอบกับผู้ใช้ เพื่อรับฟีดแบ็คเป็นส่วนสำคัญมาพัฒนาต่อจนกว่าซอฟต์แวร์จะเสร็จพร้อมใช้

ข้อดีของ RAD Model

โมเดลนี้เน้นความเร็วการพัฒนาเป็นหลัก จึงส่งมอบซอฟต์แวร์ในเวลาอันสั้นได้

ผู้ใช้ได้เป็นส่วนสำคัญในระหว่างการพัฒนา

ข้อพิจารณาของ RAD Model

การลดขนาดระบบลง (Reduced Scalability) ทำให้มีความยืดหยุ่นน้อย เนื่องจากพัฒนามาจากระบบต้นแบบ

อาจไม่สมบูรณ์ได้ครบทุกฟีเจอร์ เพราะมีเวลาพัฒนาที่จำกัด

2.6 Spiral Model

Spiral model เป็นโมเดล SDLC ที่นำจุดแข็งของ Waterfall Model มาพัฒนาให้ดีขึ้น เพราะแต่เดิมหากขั้นตอนไหนผิดพลาดจะไม่สามารถแก้ไขได้ ต้องรอจบกระบวนการทั้งหมดก่อนถึงจะแก้ไขได้ตั้งแต่ขั้นแรกและเรียงลำดับขึ้นไปเรื่อย ๆ ทำให้มีความเสี่ยงที่การพัฒนาซอฟต์แวร์จะไม่สำเร็จ

จึงมีการปรับเปลี่ยนรูปแบบเป็น Spiral model หรือโมเดลรูปเกลียว ที่เน้นการประเมินความเสี่ยงของโปรเจกต์ด้วยการแบ่งโครงสร้างออกเป็นโปรเจกต์กลุ่มย่อย ช่วยให้การปรับเปลี่ยน requirement ในระหว่างพัฒนาทำได้ง่ายขึ้น ตลอดจนประเมินความเสี่ยงกับความต่อเนื่องของโปรเจกต์ในเรื่องค่าใช้จ่าย SDLC model นี้เหมาะกับโปรเจกต์ที่มีความซับซ้อนสูง และมีความเสี่ยงที่ต้องควบคุมอย่างเข้มงวด

ข้อดีของ Spiral Model

ลดความเสี่ยงได้ตั้งแต่ระยะเริ่มต้น จากการวิเคราะห์และประเมินอย่างละเอียด เพื่อแก้ไขได้ทันที

การพัฒนาหลายส่วนเป็นอิสระต่อกัน ทำให้แก้ไขหรือเพิ่มเติมตามความต้องการได้สะดวก

ได้รับฟีดแบ็กซอฟต์แวร์ที่ตรงกับความต้องการ และ requirement ทั้งหมด

ข้อพิจารณาของ Spiral Model

การนำ Spiral model มาใช้กับโปรเจกต์ อาจทำให้จบงานได้ล่าช้า เนื่องเพราะมีการแก้ไขตลอดเวลา จึงมักทำให้เกิดค่าใช้จ่ายที่มากขึ้นตามมาด้วย

เหมาะสำหรับโปรเจกต์ขนาดปานกลางถึงใหญ่ เพราะต้องใช้ทรัพยากรและเวลาค่อนข้างมาก หากเป็นโปรเจกต์ขนาดเล็ก การเลือกใช้โมเดลนี้จะทำให้สิ้นเปลืองโดยไม่จำเป็น

2.7 Agile Model

Agile model เป็น SDLC Model ที่เน้นความยืดหยุ่นและการทำงานเป็นทีม การสื่อสารที่คล่องตัวจะช่วยให้สามารถพัฒนาซอฟต์แวร์ที่ตอบสนองต่อ requirement ของลูกค้าได้เร็ว นักพัฒนาจะแบ่งโปรเจกต์เป็นส่วนย่อย ที่เรียกว่า Sprint ซึ่งแต่ละ Sprint จะใช้ระยะเวลาประมาณ 1-4 สัปดาห์ เพื่อให้ซอฟต์แวร์เสร็จภายในระยะสั้น ๆ แล้วนำไปทดสอบกับผู้ใช้ เพื่อรับฟีดแบคกลับมาปรับปรุงต่อ ใน Sprint ถัดไป

ข้อดีของ Agile Model

มีการสื่อสารกันภายในทีม ตลอดระยะการพัฒนา ช่วยให้ประสิทธิภาพการทำงานดีขึ้น

โมเดลนี้ช่วยให้นักพัฒนา (Dev) ได้คุยกับ PO หรือผู้ใช้โดยตรง แล้วนำมาอัปเดตให้คนในทีมทราบและเข้าใจตรงกัน

สามารถหยิบงานส่วนที่เคยมีคนทำไว้แล้วมาใช้ หรือพัฒนาต่อยอดได้ โดยที่ไม่ต้องเริ่มใหม่ทั้งหมด

ข้อพิจารณาของ Agile Model

ต้องอาศัยประสบการณ์ และทักษะของทีมนักพัฒนาสูง

โมเดลนี้จะเน้นการทำงานร่วมกันเป็นทีมจากทุกฝ่ายที่เกี่ยวข้องกัน หากมีปัญหาส่วนตัวกันภายในทีม หรือกับผู้ที่มีส่วนได้ส่วนเสีย เหล่านี้มักจะทำให้โปรเจกต์เกิดความล่าช้าได้

3. แนวปฏิบัติสมัยใหม่ (Modern Practices)

วิศวกรรมคอมพิวเตอร์ในปัจจุบันจะเน้นการนำเครื่องมือมาช่วยให้กระบวนการพัฒนาเป็นไปโดยอัตโนมัติ:

- **CI/CD (Continuous Integration / Continuous Deployment):** ระบบที่ช่วยทดสอบและนำโค้ดขึ้นเซิร์ฟเวอร์โดยอัตโนมัติทันทีที่มีการอัปเดต ช่วยลดความผิดพลาดจากมนุษย์ ข้อมูลเพิ่ม
- **DevOps Culture:** แนวคิดที่ผสานทีมพัฒนา (Development) และทีมดูแลระบบ (Operations) เข้าด้วยกัน เพื่อให้การปล่อยซอฟต์แวร์ทำได้รวดเร็วและเสถียรที่สุด

ถ้าคุณกำลังเตรียมตัวสำหรับโปรเจกต์หรือวิชานี้ สามารถบอกเพิ่มเติมได้นะคะว่า:

- อยากให้เจาะลึกที่ขั้นตอนไหนเป็นพิเศษไหม? (เช่น วิธีการทำ **Git Workflow** หรือ วิธีเขียน **UML Diagram**)