**แบบฝึกหัดเตรียมสอบปลายภาค วิชาวิศวกรรมซอฟต์แวร์ ภาคเรียน 2/2567**

**ส่วนที่ 1 Requirement Engineering วิศวกรรมความต้องการ**

1. อธิบายความหมายของ **วิศวกรรมความต้องการ (Requirements Engineering)** และเหตุผลที่ต้องมีวิศวกรรมความต้องการในการพัฒนาซอฟต์แวร์.

ตอบ วิศวกรรมความต้องการ (Requirements Engineering) คือ กระบวนการในการกำหนด วิเคราะห์ จัดทำเอกสาร ตรวจสอบ และจัดการความต้องการของระบบซอฟต์แวร์ เพื่อให้แน่ใจว่าระบบที่พัฒนาจะตรงกับความต้องการของผู้ใช้และผู้มีส่วนได้ส่วนเสีย

เหตุผลที่ต้องมี: เพื่อป้องกันความเข้าใจผิด ลดข้อผิดพลาด และช่วยให้การพัฒนาซอฟต์แวร์มีประสิทธิ

1. กระบวนการวิศวกรรมความต้องการประกอบด้วย 4 ขั้นตอนหลัก อธิบายแต่ละขั้นตอนโดยสังเขป

ตอบ

- **Elicitation (การรวบรวมความต้องการ)**: การสัมภาษณ์ สังเกต หรือใช้แบบสอบถามเพื่อรวบรวมข้อมูลจากผู้ใช้

**- Analysis (การวิเคราะห์ความต้องการ)**: การตรวจสอบความเป็นไปได้ของความต้องการ รวมถึงข้อขัดแย้งที่อาจเกิดขึ้น

* **Specification (การจัดทำเอกสารความต้องการ)**: การเขียนเอกสารข้อกำหนดความต้องการ (SRS)
* **Validation (การตรวจสอบความต้องการ)**: ตรวจสอบว่าความต้องการที่รวบรวมมาไม่มีข้อผิดพลาด

1. จงอธิบายความแตกต่างระหว่าง **ความต้องการเชิงหน้าที่ (Functional Requirements)** และ **ความต้องการที่ไม่ใช่เชิงหน้าที่ (Non-Functional Requirements)** พร้อมยกตัวอย่างของแต่ละประเภท

**-Functional Requirements (ความต้องการเชิงหน้าที่)**

กำหนดพฤติกรรมของระบบว่าต้องทำอะไร

ตัวอย่าง: "ผู้ใช้ต้องสามารถลงทะเบียนเข้าสู่ระบบได้"

**-Non-Functional Requirements (ความต้องการที่ไม่ใช่เชิงหน้าที่)**

\*กำหนดคุณลักษณะด้านคุณภาพของระบบ เช่น ความปลอดภัย, ประสิทธิภาพ, การรองรับโหลด

\*ตัวอย่าง: "ระบบต้องสามารถรองรับผู้ใช้พร้อมกันได้ 1,000 คน"

1. จงยกตัวอย่าง 3 วิธีที่ใช้ในการรวบรวมความต้องการ (Requirement Elicitation) และอธิบายข้อดีและข้อเสียของแต่ละวิธี

* **การสัมภาษณ์ (Interview)**
* **ข้อดี:** ได้ข้อมูลเชิงลึกจากผู้ใช้
* **ข้อเสีย:** ใช้เวลามากและอาจมีอคติจากผู้ตอบ
* **แบบสอบถาม (Questionnaire)**
* **ข้อดี:** รวบรวมข้อมูลจากกลุ่มเป้าหมายได้เป็นจำนวนมาก
* **ข้อเสีย:** อาจได้รับข้อมูลที่ไม่ละเอียดและไม่สามารถซักถามเพิ่มเติมได้
* **การสร้างต้นแบบ (Prototyping)**
* **ข้อดี:** ผู้ใช้สามารถเห็นภาพระบบและให้ข้อเสนอแนะได้ง่าย
* **ข้อเสีย:** ใช้เวลาและทรัพยากรมากในการสร้างต้นแบบ

1. ทำไมความต้องการของระบบจึงเปลี่ยนแปลงได้เสมอ? จงอธิบายปัจจัยที่ทำให้ความต้องการของระบบเปลี่ยนแปลง

 เทคโนโลยีเปลี่ยนแปลงตลอดเวลา

 นโยบายขององค์กรเปลี่ยนไป

 ความต้องการของลูกค้าเปลี่ยนแปลงตามตลาด

 ระบบที่พัฒนาอาจพบข้อผิดพลาดหรือข้อจำกัดใหม่ๆ

1. เอกสารข้อกำหนดความต้องการด้านซอฟต์แวร์ (SRS) มีองค์ประกอบหลักอะไรบ้าง? อธิบายโดยสังเขป

- บทนำ

- คำอธิบายระบบโดยรวม

- ข้อกำหนดเชิงหน้าที่ (Functional Requirements)

- ข้อกำหนดที่ไม่ใช่เชิงหน้าที่ (Non-Functional Requirements)

- ข้อจำกัดของระบบ

1. จงอธิบายแนวทางในการตรวจสอบความต้องการ (Requirement Validation) พร้อมยกตัวอย่างวิธีการตรวจสอบ
   1. **การตรวจสอบเอกสาร (Requirements Reviews)**

ให้ทีมงานตรวจสอบเอกสารความต้องการ (SRS) เพื่อหาข้อผิดพลาด

**ตัวอย่าง:** การประชุมเพื่อตรวจสอบว่าคำอธิบายใน SRS ชัดเจนและตรงกับความต้องการของผู้ใช้

* 1. **การสร้างต้นแบบ (Prototyping)**

ใช้ต้นแบบเพื่อให้ผู้ใช้สามารถดูและให้ข้อเสนอแนะก่อนการพัฒนาเต็มรูปแบบ

**ตัวอย่าง:** สร้าง UI mockup ของระบบเพื่อให้ผู้ใช้ทดสอบและปรับปรุง

* 1. **การใช้ Checklists**

ใช้รายการตรวจสอบเพื่อให้แน่ใจว่าความต้องการที่ระบุครบถ้วนและไม่มีข้อผิดพลาด

**ตัวอย่าง:** ใช้ Checklist เพื่อตรวจสอบว่าเอกสาร SRS มีการระบุ Functional และ

Non-Functional Requirement ครบถ้วน

* 1. **การทดสอบต้นแบบหรือ Simulation**

จำลองพฤติกรรมของระบบเพื่อดูว่าสามารถทำงานได้ตรงตามที่ผู้ใช้ต้องการ

**ตัวอย่าง:** ใช้เครื่องมือจำลองการใช้งานระบบเพื่อตรวจสอบความเป็นไปได้ของความต้องการ

1. A screenshot of a computer

   AI-generated content may be incorrect.ปัญหาที่พบบ่อยในการกำหนดความต้องการของซอฟต์แวร์มีอะไรบ้าง? อธิบายและเสนอแนวทางแก้ไข
2. อธิบายแนวคิดของ **การจัดการความต้องการ (Requirement Management)** และเหตุใดจึงมีความสำคัญต่อโครงการซอฟต์แวร์

**Requirement Management คืออะไร?**

* กระบวนการบริหารจัดการความต้องการของระบบซอฟต์แวร์ตลอดวงจรชีวิตของโครงการ
* รวมถึงการจัดทำเอกสาร ติดตาม ตรวจสอบ และปรับปรุงความต้องการ

**ความสำคัญของ Requirement Management**

1. **ช่วยจัดการการเปลี่ยนแปลงของความต้องการ** – ระบบซอฟต์แวร์มีโอกาสที่ความต้องการจะเปลี่ยนแปลงอยู่เสมอ
2. **ช่วยลดข้อผิดพลาดในเอกสารความต้องการ** – ตรวจสอบให้แน่ใจว่าความต้องการถูกต้องและครบถ้วน
3. **ช่วยเพิ่มประสิทธิภาพในการพัฒนา** – ลดความเสี่ยงที่นักพัฒนาจะพัฒนาออกมาไม่ตรงกับความต้องการ
4. **ช่วยในการวิเคราะห์ผลกระทบของการเปลี่ยนแปลง** – ใช้ Traceability Matrix เพื่อตรวจสอบว่าการเปลี่ยนแปลงกระทบกับส่วนไหนของระบบ
5. พิจารณากรณีศึกษาของ ระบบจองตั๋วการแสดง
   1. ระบุ Functional Requirement 2 ข้อ
   2. ระบุ Non-Functional Requirement 2 ข้อ

**a. Functional Requirements**

1. ผู้ใช้สามารถเลือกที่นั่งและทำการจองตั๋วผ่านระบบออนไลน์ได้
2. ระบบต้องสามารถสร้าง QR Code สำหรับตั๋วที่จองสำเร็จ

**b. Non-Functional Requirements**

1. ระบบต้องรองรับผู้ใช้พร้อมกันได้อย่างน้อย 10,000 คน
2. ระบบต้องมีเวลาในการตอบสนอง (Response Time) ไม่เกิน 2 วินาทีต่อคำสั่ง

**ส่วนที่ 2 System Model แบบจำลองระบบ**

1. อธิบายวัตถุประสงค์ของ **แบบจำลองระบบ (System Model)** และเหตุผลที่ต้องใช้ในการพัฒนาซอฟต์แวร์

**วัตถุประสงค์:**

* ช่วยให้เข้าใจโครงสร้างและพฤติกรรมของระบบ
* เป็นแนวทางในการพัฒนาและออกแบบระบบ
* ลดความซับซ้อนของระบบโดยการแบ่งเป็นส่วนย่อย

**เหตุผลที่ต้องใช้:**

1. ช่วยให้ทีมพัฒนาและผู้ใช้เข้าใจระบบตรงกัน
2. ใช้เป็นเครื่องมือสำหรับการวิเคราะห์และออกแบบ
3. ช่วยลดข้อผิดพลาดในกระบวนการพัฒนา

2.แบบจำลองระบบสามารถแบ่งออกเป็นกี่ประเภท? อธิบายแต่ละประเภทโดยสังเขป

1. **แบบจำลองข้อมูล (Data Model)** – แสดงโครงสร้างข้อมูล เช่น ER Diagram
2. **แบบจำลองกระบวนการ (Process Model)** – แสดงการไหลของข้อมูล เช่น DFD
3. **แบบจำลองเชิงพฤติกรรม (Behavior Model)** – แสดงลำดับเหตุการณ์ เช่น State Diagram
4. แผนภาพ **Data Flow Diagram (DFD)** คืออะไร และมีหน้าที่อย่างไรในกระบวนการพัฒนาซอฟต์แวร์?

คือ DFD เป็นแผนภาพที่แสดงการไหลของข้อมูลภายในระบบ

มีหน้าที่ : ใช้ระบุว่าข้อมูลเข้าสู่ระบบจากแหล่งใด และไหลผ่านกระบวนการต่าง ๆ อย่างไร

1. แผนภาพ Context Diagram แตกต่างจาก **DFD Level 1** อย่างไร? และมีบทบาทอย่างไรในการออกแบบระบบ?

* **Context Diagram** แสดงเฉพาะระบบและการเชื่อมต่อกับภายนอก
* **DFD Level 1** แสดงรายละเอียดของกระบวนการที่เกิดขึ้นภายในระบ

1. Entity-Relationship Diagram (ERD) มีองค์ประกอบหลักอะไรบ้าง? อธิบายแต่ละองค์ประกอบ

* **Entity** – สิ่งที่ต้องเก็บข้อมูล
* **Attribute** – คุณสมบัติของ Entity
* **Relationship** – ความสัมพันธ์ระหว่าง Entity

1. จงอธิบายประเภทของความสัมพันธ์ใน **ER Diagram** และยกตัวอย่างกรณีใช้งานของแต่ละประเภท

* **One-to-One (1:1)** – เช่น ผู้ใช้หนึ่งคนมีบัตรสมาชิกหนึ่งใบ
* **One-to-Many (1:M)** – เช่น ลูกค้าหนึ่งคนสามารถสั่งซื้อสินค้าได้หลายรายการ
* **Many-to-Many (M:N)** – เช่น นักเรียนหลายคนเรียนในหลายวิชา

1. Use Case Diagram ใช้แสดงอะไร? อธิบายองค์ประกอบหลักของ Use Case Diagram

* แสดงการโต้ตอบระหว่างผู้ใช้กับระบบ
* องค์ประกอบหลัก: **Actor, Use Case, Relationship**

1. A screenshot of a computer

   AI-generated content may be incorrect.เปรียบเทียบ **Class Diagram กับ Entity-Relationship Diagram (ERD)** ว่าแตกต่างกันอย่างไร และมีการใช้งานอย่างไรบ้าง

1. Sequence Diagram คืออะไร? ใช้แสดงอะไร? และมีองค์ประกอบหลักอะไรบ้าง?

* **Sequence Diagram** เป็นแผนภาพใน **UML (Unified Modeling Language)** ที่ใช้แสดงลำดับของข้อความ (Messages) ที่ถูกส่งระหว่าง **วัตถุ (Objects) หรือ องค์ประกอบของระบบ** ในช่วงเวลาหนึ่ง โดยเน้นไปที่ลำดับการทำงานของแต่ละองค์ประกอบ และการโต้ตอบกันระหว่างผู้ใช้และระบบ

1. Activity Diagram ใช้ทำอะไร? และแตกต่างจาก **Flowchart** อย่างไร?

- **Activity Diagram** ใช้แสดงลำดับขั้นตอนการทำงานของระบบหรือกระบวนการ (Process)

โดยเน้นที่การไหลของกิจกรรม (Activity) และการตัดสินใจ (Decision).

- **แตกต่างจาก Flowchart**

Flowchart ใช้แสดงลำดับการทำงานทั่วไปของโปรแกรม ส่วน Activity Diagram

มุ่งเน้นที่การแสดงกระบวนการของ Use Case และมีความสามารถในการรองรับ Object Flow

1. State Chart Diagram ใช้แสดงข้อมูลอะไร? และมีความสำคัญอย่างไรในกระบวนการออกแบบซอฟต์แวร์?

- ใช้แสดงสถานะ (State) และการเปลี่ยนสถานะ (Transition) ของ Object ในระบบ

- สำคัญต่อการออกแบบระบบที่มี **Finite State Machine (FSM)**

เช่น เครื่องจำหน่ายสินค้าอัตโนมัติหรือ Workflow Management

1. Component Diagram และ Deployment Diagram แตกต่างกันอย่างไร? อธิบายการใช้งานของแต่ละแผนภาพ

* **Component Diagram** แสดงโครงสร้างของซอฟต์แวร์ เช่น โมดูล, ไลบรารี และอินเตอร์เฟซ
* **Deployment Diagram** แสดงการแจกจ่าย (Deployment) ของซอฟต์แวร์ในฮาร์ดแวร์จริง เช่น Server, Cloud, Database

1. แผนภาพ Collaboration Diagram มีหน้าที่อะไร? และแตกต่างจาก **Sequence Diagram** อย่างไร?

* Collaboration Diagram ใช้แสดงการสื่อสารระหว่าง Object ต่างๆ ในระบบ
* **แตกต่างจาก Sequence Diagram**
* Sequence Diagram เน้นลำดับเวลา (Time Order)
* Collaboration Diagram เน้นโครงสร้างของ Object และการส่งข้อความระหว่างกัน

1. Class Diagram ใช้แสดงอะไร? อธิบายองค์ประกอบหลักของ Class Diagram พร้อมยกตัวอย่าง

* ใช้แสดงโครงสร้างของคลาสและความสัมพันธ์ระหว่างคลาส
* **องค์ประกอบหลัก**
* **Class** (ชื่อคลาส, คุณสมบัติ, เมธอด)
* **Relationships** (Association, Generalization, Aggregation, Composition)

1. อธิบายแนวคิดของ **Generalization และ Aggregation** ใน Class Diagram และยกตัวอย่างการใช้งานจริง

- **Generalization**: ความสัมพันธ์แบบ Superclass-Subclass (สืบทอดคุณสมบัติ)

เช่น Vehicle -> Car, Bike

* **Aggregation**: ความสัมพันธ์แบบ Whole-Part (เป็นส่วนหนึ่ง แต่แยกกันได้)

เช่น Library - (has) -> Book

1. Activity Diagram มีการใช้สัญลักษณ์อะไรบ้าง? อธิบายความหมายของแต่ละสัญลักษณ์

* **Initial Node**: จุดเริ่มต้น (วงกลมดำ)
* **Action State**: กิจกรรมที่เกิดขึ้น (สี่เหลี่ยมมน)
* **Decision Node**: จุดตัดสินใจ (ข้าวหลามตัด)
* **Fork & Join**: แสดงการทำงานพร้อมกัน (เส้นหนา)

- **Final Node**: จุดสิ้นสุด (วงกลมดำล้อมด้วยวงกลมขาว)

1. การออกแบบ **Data Flow Diagram (DFD)** ควรคำนึงถึงปัจจัยใดบ้างเพื่อให้ได้แผนภาพที่มีประสิทธิภาพ?

* ความถูกต้องของกระแสข้อมูล
* การแบ่งระดับของ DFD (Context Diagram → Level 1 → Level 2)
* ความสัมพันธ์ของเอนทิตี ข้อมูล และกระบวนการ

1. แผนภาพ **System Flowchart** มีความสำคัญอย่างไรในการพัฒนาระบบสารสนเทศ

- แสดงการไหลของข้อมูลในระบบสารสนเทศ

- ใช้สื่อสารกับนักพัฒนาและลูกค้าเกี่ยวกับโครงสร้างระบบ

1. อธิบายความแตกต่างระหว่าง **Top-down Design** และ **Bottom-up Design** ในการออกแบบซอฟต์แวร์

 **Top-down**: ออกแบบจากภาพรวมลงรายละเอียด

 **Bottom-up**: เริ่มจากส่วนเล็กๆ แล้วประกอบเป็นระบบใหญ่

1. ในกระบวนการพัฒนาซอฟต์แวร์ ควรเลือกใช้แบบจำลองระบบ (System Model) แบบใดจึงจะเหมาะสมกับ **โครงการพัฒนาเว็บแอปพลิเคชันสำหรับการจองโรงแรมออนไลน์**? อธิบายเหตุผล

* **เลือกแบบจำลองระบบ (System Model) สำหรับเว็บแอปจองโรงแรมออนไลน์**

สำหรับ **โครงการพัฒนาเว็บแอปพลิเคชันสำหรับการจองโรงแรมออนไลน์** ควรใช้ **Agile Model ร่วมกับ DevOps** เพราะต้องรองรับความเปลี่ยนแปลงของระบบ และมีการพัฒนาอย่างต่อเนื่องเพื่อเพิ่มประสิทธิภาพและประสบการณ์ของผู้ใช้

**ส่วนที่ 3 Software Testing การทดสอบซอฟต์แวร์**

1. อธิบายความหมายของ **การทดสอบซอฟต์แวร์ (Software Testing)** และเหตุผลที่ต้องมีการทดสอบซอฟต์แวร์

* **Software Testing** คือ กระบวนการตรวจสอบและประเมินคุณภาพของซอฟต์แวร์เพื่อให้แน่ใจว่าซอฟต์แวร์ทำงานได้อย่างถูกต้อง ปลอดภัย และตรงตามข้อกำหนด
* **เหตุผลที่ต้องมีการทดสอบ**

1. **ป้องกันข้อผิดพลาด (Bug Prevention)** – ลดปัญหาที่อาจเกิดขึ้นเมื่อนำระบบไปใช้งานจริง
2. **ปรับปรุงคุณภาพซอฟต์แวร์ (Quality Assurance)** – ทำให้ซอฟต์แวร์มีเสถียรภาพมากขึ้น
3. **เพิ่มความพึงพอใจของผู้ใช้ (User Satisfaction)** – ลดปัญหาจากการใช้งานจริง
4. **ลดต้นทุนในการแก้ไขปัญหา (Cost Reduction)** – ตรวจพบปัญหาได้เร็ว ลดค่าใช้จ่ายในการแก้ไขภายหลัง
5. A screenshot of a computer

   AI-generated content may be incorrect.เปรียบเทียบ **Verification และ Validation** ในกระบวนการทดสอบซอฟต์แวร์
6. อธิบายประเภทของ **ข้อผิดพลาด (Errors) ในการพัฒนาโปรแกรม** พร้อมตัวอย่าง

**1. ข้อผิดพลาดทางไวยากรณ์ (Syntax Error)**

เป็นข้อผิดพลาดที่เกิดจากการใช้ไวยากรณ์ของภาษาโปรแกรมไม่ถูกต้อง เช่น ลืมวงเล็บปิด ลืมเครื่องหมายจุลภาค หรือใช้คำสั่งผิดรูปแบบ ทำให้คอมไพเลอร์หรืออินเตอร์พรีเตอร์ไม่สามารถแปลโค้ดได้

**2. ข้อผิดพลาดขณะรันไทม์ (Runtime Error)**เป็นข้อผิดพลาดที่เกิดขึ้นในขณะที่โปรแกรมกำลังทำงาน แม้ว่าโค้ดจะถูกต้องตามไวยากรณ์แล้วก็ตาม แต่มันอาจทำให้เกิดปัญหาขณะรัน เช่น การหารด้วยศูนย์ (Division by zero) หรือการเข้าถึงตัวแปรที่ไม่มีค่า

**3. ข้อผิดพลาดเชิงตรรกะ (Logical Error)**

เป็นข้อผิดพลาดที่เกิดจากตรรกะของโค้ดไม่ถูกต้อง ทำให้โปรแกรมทำงานผิดจากที่คาดไว้ ถึงแม้โค้ดจะไม่มีข้อผิดพลาดทางไวยากรณ์หรือรันไทม์ก็ตาม แต่ผลลัพธ์ที่ได้ไม่ถูกต้อง

**4. ข้อผิดพลาดทางตรรกะของการจัดการหน่วยความจำ (Memory Error)**

ข้อผิดพลาดที่เกิดจากการใช้หน่วยความจำมากเกินไป เช่น การสร้างลูปที่ไม่มีทางสิ้นสุด หรือการใช้ทรัพยากรมากเกินไปจนทำให้ระบบขาดหน่วยความจำ

1. อธิบาย **ประเภทของการทดสอบซอฟต์แวร์** แบ่งตามระดับการทดสอบ

* ตรวจสอบว่าสามารถใช้งานได้จริงตามความต้องการของผู้ใช้
* มี **Alpha Testing** และ **Beta Testing** (ดูข้อ 7)
* **ตัวอย่าง:** ให้ผู้ใช้ทดลองใช้แอปและให้ฟีดแบ็กก่อนเปิดตัวจริง

1. อธิบายแนวทางการทดสอบแบบ **White Box Testing** และ **Black Box Testing** พร้อมยกตัวอย่าง

**White Box Testing (การทดสอบกล่องขาว)**

- ทดสอบจากภายใน ดูโครงสร้างและโค้ดของระบบ

- ใช้โดยนักพัฒนาเพื่อตรวจสอบตรรกะการทำงาน

**- ตัวอย่าง:** ตรวจสอบว่าฟังก์ชัน if-else ทำงานครบทุกกรณี

**Black Box Testing (การทดสอบกล่องดำ)**

-ทดสอบจากมุมมองของผู้ใช้ โดยไม่สนใจโค้ดภายใน

- ใช้โดยทีม QA เพื่อดูผลลัพธ์ที่ตรงกับข้อกำหนดหรือไม่

**- ตัวอย่าง:** ทดสอบว่าปุ่ม "สมัครสมาชิก" ทำงานหรือไม่เมื่อกรอกข้อมูลครบถ้วน

1. อธิบายความหมายของ **Test Case และ Test Plan** และเหตุผลที่ต้องมี

* **Test Case** คือ ชุดของเงื่อนไขที่ใช้ตรวจสอบการทำงานของซอฟต์แวร์
  + ประกอบด้วย **Input, Expected Output และ Result**
  + **ตัวอย่าง:** ทดสอบว่าถ้ากรอกรหัสผ่านผิด 3 ครั้ง ระบบต้องล็อกบัญชี
* **Test Plan** คือ เอกสารที่กำหนดแนวทางการทดสอบทั้งหมด เช่น วิธีการ, เครื่องมือ, ตารางเวลา และทีมงานที่รับผิดชอบ
  + จำเป็นต้องมีเพื่อให้กระบวนการทดสอบเป็นระบบ
  + **ตัวอย่าง:** กำหนดว่าจะทดสอบฟีเจอร์ **Login** ก่อน **Payment** เพื่อความเป็นระเบียบ

1. อธิบายกลยุทธ์ **การทดสอบแบบ Alpha Testing และ Beta Testing**

**Alpha Testing**

* ทำในสภาพแวดล้อมที่ควบคุมได้ (โดยทีมพัฒนา)
* ทดสอบภายในก่อนส่งให้ผู้ใช้จริง
* **ตัวอย่าง:** นักพัฒนาและ QA ทดสอบแอปพลิเคชันในบริษัท
* **Beta Testing**
* ให้ผู้ใช้จริงทดสอบก่อนเปิดตัว
* ใช้เพื่อรับฟีดแบ็กและแก้ไขข้อผิดพลาดที่อาจมองข้าม
* **ตัวอย่าง:** ให้กลุ่มผู้ใช้ 1,000 คน ทดลองใช้แอปเวอร์ชันเบต้า

1. อธิบายแนวทาง **การดีบัก (Debugging) และวิธีที่ใช้ในการดีบัก**

**Debugging** คือ กระบวนการค้นหาและแก้ไขข้อผิดพลาดในโค้ด โดยมีแนวทางหลัก ๆ ดังนี้:

1. **ใช้ Debugger** (เช่น Visual Studio Debugger, Chrome DevTools)
2. **พิมพ์ค่าออกมาดู (print หรือ console.log)**
3. **ใช้เครื่องมือ Logging** เช่น Log4j หรือ Winston
4. **ใช้ Unit Test ตรวจสอบโค้ด**
5. **แบ่งโค้ดเป็นส่วนเล็ก ๆ เพื่อทดสอบทีละส่วน**
6. อธิบายความแตกต่างระหว่าง **Integration Testing แบบ Top-down และ Bottom-up**

**Integration Testing** มี 2 แนวทางหลัก:

1. **Top-down Testing**
   * เริ่มจากโมดูลระดับสูง แล้วค่อยเพิ่มโมดูลย่อย
   * ใช้ **Stubs** แทนโมดูลที่ยังไม่มี
   * **ตัวอย่าง:** ทดสอบเมนูหลักก่อน แล้วค่อยทดสอบปุ่มย่อยในเมนู
2. **Bottom-up Testing**
   * เริ่มจากโมดูลย่อยก่อน แล้วรวมเป็นระบบใหญ่
   * ใช้ **Drivers** เพื่อจำลองโมดูลระดับสูง
   * **ตัวอย่าง:** ทดสอบฟังก์ชันการคำนวณก่อน แล้วจึงรวมเข้ากับ UI
3. อธิบายเทคนิค **Stress Testing และ Performance Testing** พร้อมตัวอย่าง

* **Stress Testing**
* ทดสอบระบบเมื่อใช้งานหนักเกินปกติ เช่น รองรับโหลด 10 เท่าของปกติ
* **ตัวอย่าง:** ทดสอบเว็บขายตั๋วคอนเสิร์ตเมื่อมีผู้ใช้พร้อมกัน 100,000 คน
* **Performance Testing**
* ทดสอบความเร็วและประสิทธิภาพของระบบ
* **ตัวอย่าง:** ทดสอบเวลาโหลดหน้าเว็บควรต่ำกว่า 2 วินาที

**ส่วนที่ 4 ข้อสอบเชิงวิเคราะห์ (Analytical & Applied Questions) วิศวกรรมซอฟต์แวร์**

1. บริษัทซอฟต์แวร์แห่งหนึ่งต้องการพัฒนา ระบบจองตั๋วออนไลน์ ซึ่งมีข้อกำหนดด้านประสิทธิภาพสูง เช่น รองรับผู้ใช้จำนวนมากพร้อมกัน และต้องการให้สามารถขยายระบบได้ง่าย ในฐานะวิศวกรซอฟต์แวร์ คุณจะเลือก กระบวนการพัฒนาซอฟต์แวร์ (Software Process Model) แบบใดจึงจะเหมาะสมที่สุด? และเพราะเหตุใด?

**แนวทางการตอบ:**

a.วิเคราะห์ความต้องการของระบบ เช่น Scalability, Performance, Maintainability

1. **Scalability** – ระบบต้องสามารถรองรับผู้ใช้จำนวนมากพร้อมกัน เช่น ในช่วงเวลาที่มีการเปิดขายตั๋วคอนเสิร์ตหรืออีเวนท์สำคัญ
2. **Performance** – ต้องมี Latency ต่ำ และตอบสนองได้รวดเร็วแม้มีทราฟฟิกสูง
3. **Maintainability** – ต้องสามารถปรับปรุง เพิ่มฟีเจอร์ใหม่ และแก้ไขข้อผิดพลาดได้โดยไม่ส่งผลกระทบต่อระบบหลัก
4. **Reliability** – ระบบต้องมีความเสถียร ป้องกันการล่มเมื่อมีการใช้งานหนัก

b.เปรียบเทียบระหว่าง Waterfall, Agile, Incremental, Spiral, DevOps

A screenshot of a computer

AI-generated content may be incorrect.

* 1. แนะนำแนวทางที่เหมาะสม เช่น Agile + DevOps เพื่อรองรับความเปลี่ยนแปลงของระบบและให้สามารถปรับปรุงได้อย่างต่อเนื่อง

จากการเปรียบเทียบ **Agile + DevOps** เป็นแนวทางที่เหมาะสมที่สุดสำหรับระบบจองตั๋วออนไลน์ เนื่องจาก:

1. **Agile** ช่วยให้พัฒนาและปรับปรุงฟีเจอร์ได้อย่างรวดเร็ว
2. **DevOps** ช่วยให้การพัฒนา การทดสอบ และการนำระบบขึ้น Production เป็นไปโดยอัตโนมัติ ผ่าน **CI/CD (Continuous Integration/Continuous Deployment)**
3. **Microservices Architecture** สามารถใช้ร่วมกับ DevOps เพื่อให้แยกส่วนต่าง ๆ ของระบบ เช่น **Authentication, Payment, Ticket Management** ออกจากกัน ทำให้รองรับผู้ใช้จำนวนมากได้ดีขึ้น
   1. ยกตัวอย่างแนวทางการใช้ Scrum หรือ CI/CD

**Scrum ใน Agile Development**

**-Sprint Planning**: วางแผนพัฒนาเป็นรอบ ๆ (เช่น ทุก 2 สัปดาห์)

**- Daily Standup**: ประชุมสั้น ๆ เพื่อติดตามความคืบหน้า

**-Sprint Review & Retrospective**: ประเมินผลและปรับปรุงกระบวนการในแต่ละ Sprint

1. **CI/CD Pipeline ใน DevOps**

**-Continuous Integration**: ใช้ GitHub Actions, Jenkins, หรือ GitLab CI เพื่อตรวจสอบโค้ดทุกครั้งที่มีการ Merge

**-Automated Testing**: ใช้ Unit Test, Integration Test และ Load Testing

**- Continuous Deployment**: ใช้ Kubernetes หรือ Docker เพื่อ Deploy ระบบอัตโนมัติไปยัง Cloud

1. คุณได้รับมอบหมายให้เป็นผู้จัดการโครงการพัฒนา **ระบบจัดการคลังสินค้า** โดยมีระยะเวลาจำกัดและทรัพยากรจำกัด คุณจะใช้เทคนิค **Software Project Management และ Cost Estimation** อย่างไรเพื่อให้โครงการสำเร็จภายในขอบเขตที่กำหนด?

**แนวทางการตอบ:**

1. อธิบายแนวทางการ **บริหารโครงการซอฟต์แวร์** เช่น Gantt Chart, PERT Chart

* **ใช้ Gantt Chart**
  + ใช้ **Gantt Chart** เพื่อแสดงตารางเวลาของแต่ละงาน ทำให้สามารถติดตามความคืบหน้าได้ง่าย
* **ใช้ PERT Chart (Program Evaluation Review Technique)**
  + กำหนดงานที่ต้องทำ **(Tasks)** พร้อมความสัมพันธ์ระหว่างงานเพื่อประเมินเวลาโครงการได้แม่นยำขึ้น

1. วิธีประเมินต้นทุนเช่น **COCOMO Model, Function Point Analysis**

- **COCOMO Model (Constructive Cost Model)**

* ใช้คำนวณต้นทุนจาก **ขนาดของโค้ด (Lines of Code - LOC)** และระดับความซับซ้อนของซอฟต์แวร์

- **Function Point Analysis (FPA)**

* ประเมินต้นทุนจากฟังก์ชันของระบบ เช่น จำนวนหน้าจอ, อินพุต, เอาต์พุต

1. เทคนิค **Critical Path Method (CPM) และ Resource Allocation**

**1 ใช้ CPM** เพื่อหางานที่สำคัญที่สุดของโครงการ

-วิเคราะห์งานที่ใช้เวลานานที่สุดและต้องทำให้เสร็จตรงเวลา

* 1. **Resource Allocation**

**-**กระจายทรัพยากรไปยังงานที่สำคัญโดยพิจารณาความสามารถของทีม

1. การใช้ Agile เพื่อช่วยในการจัดลำดับความสำคัญของงาน

- **ใช้ Scrum Framework**

* แบ่งโครงการเป็น **Sprint (รอบการพัฒนา 2-4 สัปดาห์)**
* มี **Daily Standup Meeting** เพื่อติดตามความคืบหน้า

-**ใช้ MVP (Minimum Viable Product) เพื่อส่งมอบฟังก์ชันหลักก่อน**

* เน้นพัฒนาส่วนที่จำเป็นที่สุดก่อน เช่น **Stock Management, Order Tracking**

1. ในโครงการพัฒนาระบบลงทะเบียนเรียนของมหาวิทยาลัย คุณพบว่าผู้ใช้ (เจ้าหน้าที่ทะเบียน) ไม่สามารถระบุความต้องการที่แน่นอนได้ตั้งแต่ต้น คุณจะใช้ เทคนิคการเก็บความต้องการ (Requirement Elicitation) แบบใดจึงจะเหมาะสม และทำไม?

**แนวทางการตอบ:**

* 1. วิเคราะห์ปัญหาว่าผู้ใช้ยังไม่สามารถให้ข้อมูลที่แน่ชัด

 ผู้ใช้ไม่มีความเชี่ยวชาญด้านเทคนิค อาจอธิบายความต้องการในรูปแบบทั่วไป เช่น "อยากให้ระบบใช้ง่าย"

 การใช้เอกสารหรือการสัมภาษณ์อย่างเดียวอาจไม่เพียงพอ

* 1. A screenshot of a computer

     AI-generated content may be incorrect.เปรียบเทียบระหว่าง Interview, Questionnaire, Prototyping, Observation
  2. อธิบายการใช้ Prototyping + Workshop เพื่อให้ผู้ใช้เห็นภาพจริงของระบบและปรับปรุงความต้องการ
* **สร้าง Prototyping ของระบบ** เพื่อให้ผู้ใช้เห็นตัวอย่างหน้าจอและฟังก์ชันการทำงาน
* **จัด Workshop** ร่วมกับเจ้าหน้าที่ทะเบียน ให้ทดลองใช้ต้นแบบและเสนอความคิดเห็น
* **ใช้ Iterative Development** ปรับปรุงตามฟีดแบ็กจากผู้ใช้

**ตัวอย่างการใช้ Prototyping:**

1. สร้าง UI สำหรับ **การค้นหารายวิชา** และให้ผู้ใช้ทดสอบ
2. ปรับปรุงระบบตามความคิดเห็น เช่น เพิ่มตัวกรองค้นหารายวิชาตามคณะ
3. ในโครงการพัฒนาระบบลงทะเบียนเรียนของมหาวิทยาลัย คุณพบว่าผู้ใช้ (เจ้าหน้าที่ทะเบียน) ไม่สามารถระบุความต้องการที่แน่นอนได้ตั้งแต่ต้น คุณจะใช้ เทคนิคการเก็บความต้องการ (Requirement Elicitation) แบบใดจึงจะเหมาะสม และทำไม?
   1. วิเคราะห์ปัญหาว่าผู้ใช้ยังไม่สามารถให้ข้อมูลที่แน่ชัด
4. **ผู้ใช้ไม่มีประสบการณ์ในการออกแบบระบบ**
   * เจ้าหน้าที่ทะเบียนอาจคุ้นเคยกับกระบวนการทำงานเดิม แต่ไม่สามารถระบุความต้องการของระบบใหม่ได้อย่างละเอียด
5. **ผู้ใช้ไม่แน่ใจว่าต้องการฟีเจอร์ใดบ้าง**
   * อาจรู้เพียงปัญหาที่พบในการทำงานปัจจุบัน แต่ไม่สามารถอธิบายการแก้ปัญหาผ่านระบบได้
6. **ความต้องการอาจเปลี่ยนแปลงได้ตลอดเวลา**
   * อาจมีข้อกำหนดใหม่จากฝ่ายบริหาร หรือกฎหมายที่เกี่ยวข้องกับระบบลงทะเบียน
   1. A screenshot of a computer

      AI-generated content may be incorrect.เปรียบเทียบระหว่าง Interview, Questionnaire, Prototyping, Observation
   2. อธิบายการใช้ Prototyping + Workshop เพื่อให้ผู้ใช้เห็นภาพจริงของระบบและปรับปรุงความต้องการ

**เหตุผลที่ควรใช้ Prototyping + Workshop**

* **ช่วยให้ผู้ใช้เข้าใจระบบใหม่ได้ดีขึ้น** โดยแสดงตัวอย่างหน้าจอและฟีเจอร์ของระบบ
* **ผู้ใช้สามารถทดลองใช้ต้นแบบและให้ฟีดแบ็ก** ช่วยให้สามารถแก้ไขข้อกำหนดที่ไม่ชัดเจนได้
* **ลดความเข้าใจผิดระหว่างทีมพัฒนาและผู้ใช้** ทำให้ได้ความต้องการที่แม่นยำมากขึ้น

**แนวทางการดำเนินการ:**

**พัฒนาต้นแบบ (Prototype)** ของระบบ เช่น หน้าจอการลงทะเบียนวิชา หน้าจออนุมัติจากเจ้าหน้าที่

1. **จัด Workshop** ให้เจ้าหน้าที่ทดลองใช้ต้นแบบและเสนอความคิดเห็น
2. **ปรับปรุงต้นแบบตามฟีดแบ็ก** และทำซ้ำจนกว่าผู้ใช้พอใจกับข้อกำหนดของระบบ
3. **ใช้ Agile Development** เพื่อนำข้อกำหนดใหม่ ๆ มาปรับปรุงระบบเป็นรอบๆ
4. ในโครงการพัฒนาระบบลงทะเบียนเรียนของมหาวิทยาลัย คุณพบว่าผู้ใช้ (เจ้าหน้าที่ทะเบียน) ไม่สามารถระบุความต้องการที่แน่นอนได้ตั้งแต่ต้น คุณจะใช้ เทคนิคการเก็บความต้องการ (Requirement Elicitation) แบบใดจึงจะเหมาะสม และทำไม?

**แนวทางการตอบ:**

* 1. วิเคราะห์ปัญหาว่าผู้ใช้ยังไม่สามารถให้ข้อมูลที่แน่ชัด
* **ผู้ใช้ไม่มีความเชี่ยวชาญในการออกแบบระบบ**
* เจ้าหน้าที่ทะเบียนคุ้นเคยกับกระบวนการทำงานปัจจุบัน แต่ไม่สามารถอธิบายความต้องการของระบบใหม่ได้ชัดเจน
* **ความต้องการอาจเปลี่ยนแปลงเมื่อเห็นตัวอย่างระบบ**
* บางครั้งผู้ใช้ไม่สามารถนึกถึงฟังก์ชันที่ต้องการได้ แต่เมื่อเห็นต้นแบบระบบอาจเสนอฟีเจอร์เพิ่มเติม
* **ข้อจำกัดของการสื่อสารผ่านคำพูดหรือเอกสาร**
* คำอธิบายอาจคลุมเครือ หรือมีความเข้าใจที่แตกต่างกันระหว่างทีมพัฒนาและผู้ใช้
  1. เปรียบเทียบระหว่าง Interview, Questionnaire, Prototyping, Observation

A screenshot of a computer

AI-generated content may be incorrect.

* 1. อธิบายการใช้ Prototyping + Workshop เพื่อให้ผู้ใช้เห็นภาพจริงของระบบและปรับปรุงความต้องการ

**เหตุผลที่ควรใช้ Prototyping + Workshop**

* **ช่วยให้ผู้ใช้เข้าใจระบบใหม่ได้ง่ายขึ้น** โดยการสร้างต้นแบบและให้ผู้ใช้ทดลอง
* **สามารถเก็บฟีดแบ็กและปรับปรุงได้ทันที** ซึ่งช่วยลดปัญหาความต้องการที่ไม่ชัดเจน
* **ลดความคลาดเคลื่อนระหว่างทีมพัฒนาและผู้ใช้** ทำให้การเก็บ Requirement มีประสิทธิภาพมากขึ้น

**แนวทางการดำเนินการ:**

1. **สร้าง Prototyping ของระบบ** เช่น หน้าจอลงทะเบียนวิชา หน้าจอสำหรับเจ้าหน้าที่
2. **จัด Workshop** เพื่อให้ผู้ใช้ทดลองใช้งานต้นแบบและเสนอความคิดเห็น
3. **เก็บฟีดแบ็กจากผู้ใช้** แล้วนำไปปรับปรุงต้นแบบ
4. **ทำซ้ำขั้นตอนนี้จนกว่าผู้ใช้พึงพอใจและระบุความต้องการได้ชัดเจน**