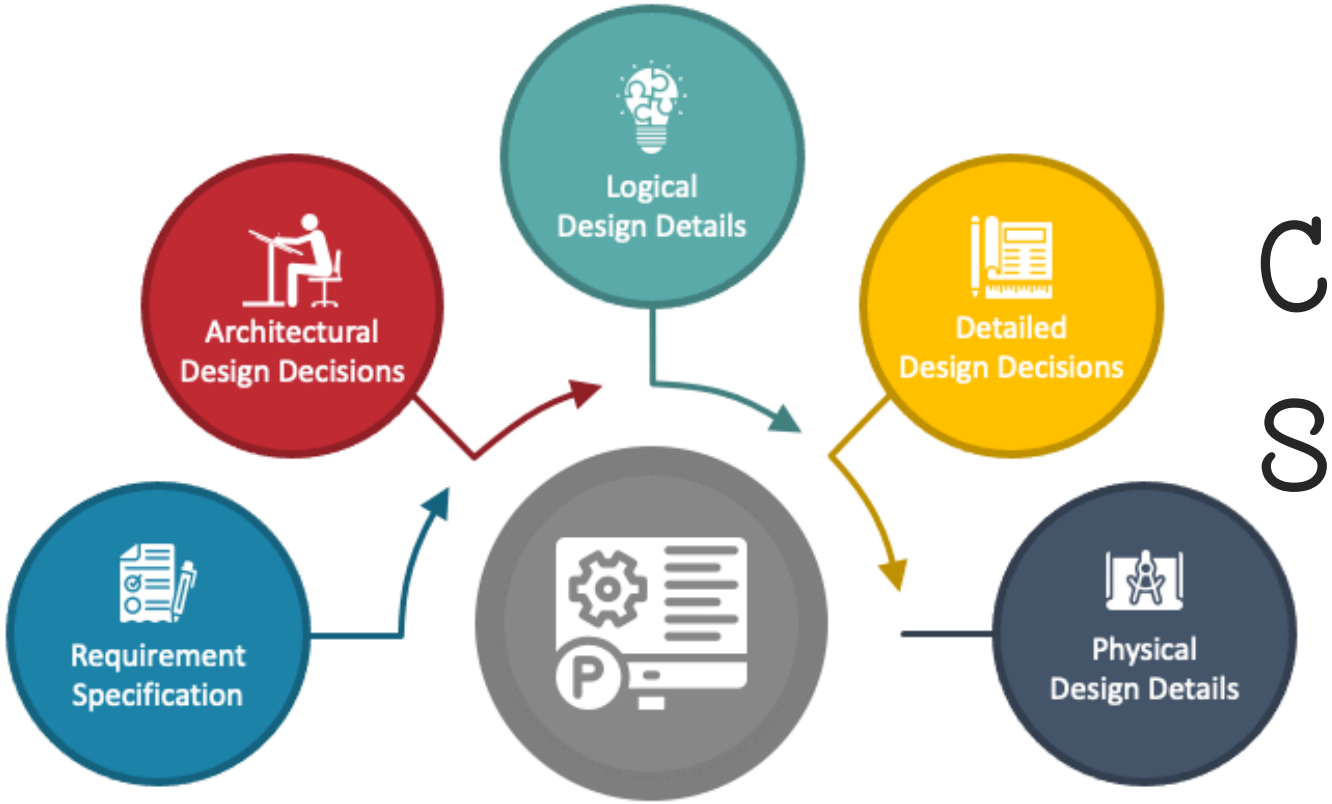


Chapter 10 : Software Design

ASSIT.PROF. JUTHAWUT CHANTHARAMALEE
CURRICULUM OF COMPUTER SCIENCE
FACULTY OF SCIENCE AND TECHNOLOGY,
SUAN DUSIT UNIVERSITY

Outline of this presentation

1. ความหมายของการออกแบบซอฟต์แวร์
2. กระบวนการออกแบบซอฟต์แวร์
3. สถาปัตยกรรมและโครงสร้างสถาปัตยกรรม
4. คุณภาพและการประเมินคุณภาพงานออกแบบซอฟต์แวร์
5. แนวคิดในการออกแบบซอฟต์แวร์
6. กลยุทธ์และระเบียบวิธีของการออกแบบซอฟต์แวร์
7. แบบจำลองที่ใช้ในการออกแบบ

1. ความหมายของการออกแบบซอฟต์แวร์

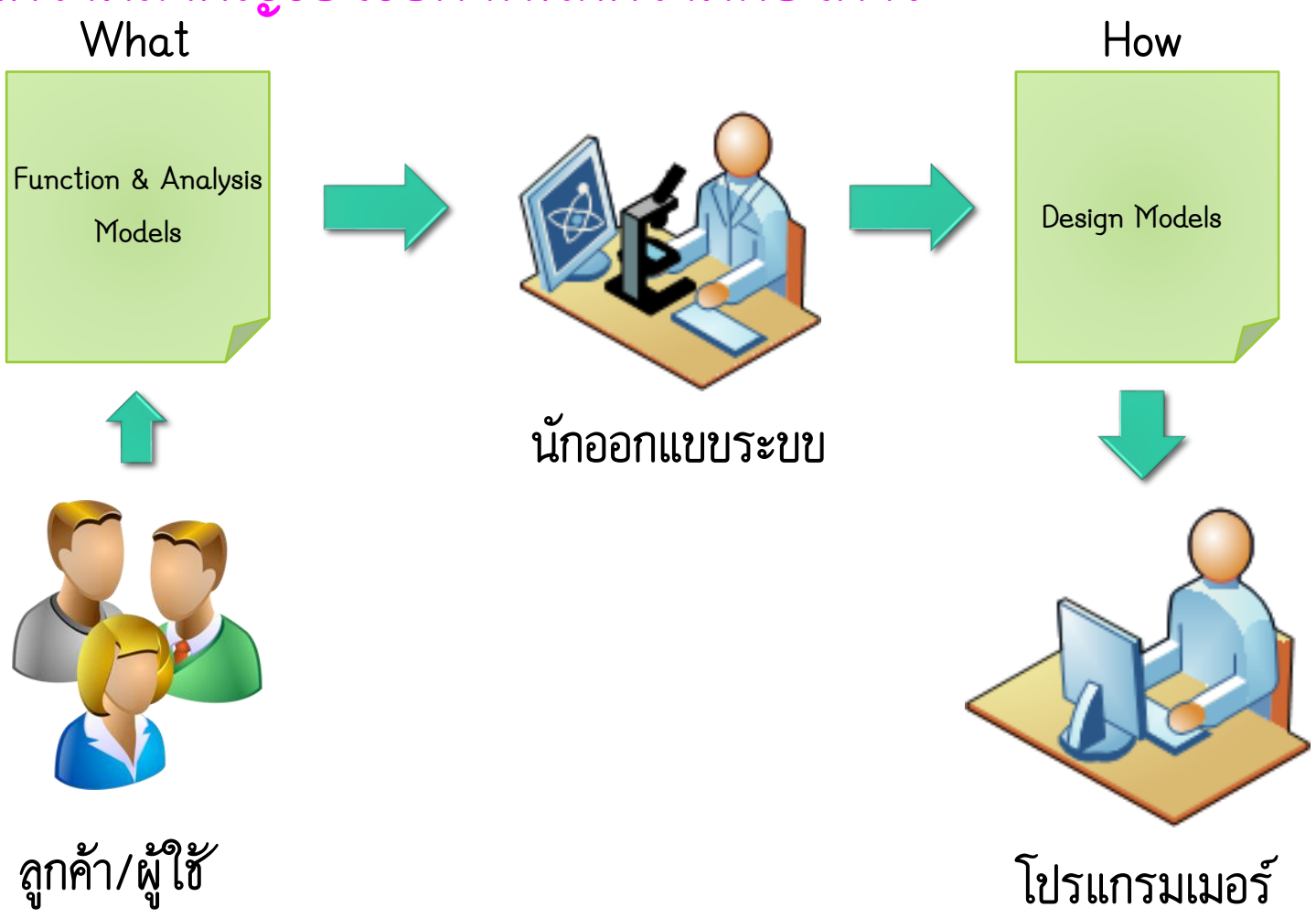
การออกแบบระบบ (System Design)

- เป็นการนำความต้องการของผู้ใช้มาแปลงให้อยู่ในรูปของแบบ (เปรียบได้กับพิมพ์เขียว) ก่อนนำไปสร้างเป็นผลิตภัณฑ์

- สิ่งที่ได้จากการออกแบบ คือ ข้อกำหนดเฉพาะของการออกแบบ (Specification Document)

- สิ่งจำเป็นที่สุดที่จะนำมาใช้ในการออกแบบ คือ ข้อกำหนดความต้องการของผู้ใช้และแบบจำลองที่ได้จากการวิเคราะห์ เพื่อนำมาสร้างเป็นแบบจำลองของการออกแบบที่มีรายละเอียดทางเทคนิคมากพอที่จะเป็นประโยชน์ในการเขียนโปรแกรม

ภาพจำลองความสำคัญของข้อกำหนดความต้องการ



แสดงการนำ Analysis Model มาใช้ในการออกแบบ

ด้านวิศวกรรมซอฟต์แวร์

1. มุ่งเน้นการผลิตซอฟต์แวร์เป็นหลัก
2. งานออกแบบซอฟต์แวร์เป็นหัวใจของกระบวนการผลิต

การออกแบบซอฟต์แวร์ (Software Design) คือกระบวนการกำหนดสถาปัตยกรรม ส่วนประกอบ ส่วนประสาน และลักษณะด้านอื่นๆ ของระบบหรือส่วนประกอบของระบบ โดยการออกแบบซอฟต์แวร์ยังมีความหมายรวมถึงสิ่งที่ได้จากการออกแบบ ซึ่งก็คือ แบบจำลองของการออกแบบ (Design Model) (IEEE610-12, 1990)

ในทางวิศวกรรมซอฟต์แวร์แล้วการนำความรู้ด้านวิศวกรรมซอฟต์แวร์มาประยุกต์ใช้กับการออกแบบ ก็คือ วิศวกรรมการออกแบบ ซึ่งมีเป้าหมายคือ การสร้างแบบร่างของระบบ หรือการนำเสนอระบบในแต่ละด้าน ให้มีคุณสมบัติที่ดี ได้แก่ Firmness (โปรแกรมที่ได้รับการออกแบบจะต้องไม่มีข้อผิดพลาด) Commodity (จะต้องตรงกับวัตถุประสงค์การใช้งาน) และ Delight (ต้องทำให้ผู้ใช้รู้สึกพอใจ) ทั้งหมดคือคุณภาพ

ปัจจัยที่ส่งผลให้การออกแบบมีคุณภาพ

1. ประสบการณ์ของบุคลากร
2. หลักการ
3. แนวทาง
4. เครื่องมือ
5. ระเบียบวิธีที่จะนำมาใช้
6. การกำหนดเงื่อนไขเพื่อวัดคุณภาพของงาน

กระบวนการออกแบบซอฟต์แวร์

กระบวนการออกแบบซอฟต์แวร์ (Software Design Process) จะมีลักษณะการทำงานแบบซ้ำ ๆ เนื่องจากต้องนำความต้องการของระบบที่ผ่านมาวิเคราะห์แล้วในแต่ละด้าน ทั้งด้านข้อมูล ฟังก์ชัน และส่วนประกอบ มาแปลงเป็นข้อกำหนดของการออกแบบ

ดังนั้น ข้อกำหนดการออกแบบจึงสอดคล้องกับข้อกำหนดความต้องการและสามารถใช้สื่อสารกับโปรแกรมเมอร์ได้ กระบวนการออกแบบนั้นจะประกอบไปด้วยการออกแบบใน 2 ระดับ ได้แก่ การออกแบบเชิงสถาปัตยกรรม และการออกแบบในรายละเอียด

1. การออกแบบเชิงสถาปัตยกรรม (Architectural Design) เรียกอีกอย่างหนึ่งว่า Top-Level Design เป็นการกำหนดลักษณะโครงสร้างของระบบหรือซอฟต์แวร์ในมุมมองระดับบน กล่าวคือ เป็นการแสดงให้เห็นส่วนประกอบต่าง ๆ ของซอฟต์แวร์ภายใต้โครงสร้างสถาปัตยกรรมรูปแบบใด ๆ

2. การออกแบบในรายละเอียด (Detail Design) เรียกอีกอย่างหนึ่งว่า Implementation Design เป็นการอธิบายรายละเอียดของแต่ละส่วนประกอบของซอฟต์แวร์ เพื่อเอื้ออำนวยต่อการเขียนโปรแกรมให้มากที่สุด

3. สถาปัตยกรรมและโครงสร้างสถาปัตยกรรมซอฟต์แวร์

สิ่งสำคัญที่ควรทราบสำหรับการออกแบบซอฟต์แวร์คือ สถาปัตยกรรม และโครงสร้างสถาปัตยกรรมซอฟต์แวร์ (Software Architecture and Architecture Structure)

สถาปัตยกรรมซอฟต์แวร์ (Software Architecture) หมายถึง การแสดงความสำพันธ์ระหว่างระบบย่อยและส่วนประกอบ (คอมโพเน้นท์) เพื่อกำหนดโครงสร้างหรือระบบภายในซอฟต์แวร์

โครงสร้างสถาปัตยกรรมและมุมมอง

โครงสร้างสถาปัตยกรรมนั้น เกิดขึ้นจากมุมมองและแนวคิดในการออกแบบที่มีความหลากหลายในปัจจุบัน เนื่องจากการออกแบบซอฟต์แวร์ ก็คือการกำหนดรายละเอียดในแต่ละมุมมองของซอฟต์แวร์ แล้วนำมาประกอบกันเป็นซอฟต์แวร์ ลักษณะของโครงสร้างเมื่อรวมส่วนประกอบย่อยต่างๆ ของซอฟต์แวร์เข้าด้วยกันแล้ว ก็คือ สถาปัตยกรรมซอฟต์แวร์ ชนิดต่างๆ ที่มีการทำงานแตกต่างกันออกไป แต่ละชนิดมีวัตถุประสงค์เพื่อให้ซอฟต์แวร์ที่งานในลักษณะเฉพาะที่แตกต่างกัน บางชนิดมีลักษณะโครงสร้างของการประกอบรวมซอฟต์แวร์เหมือนกันแต่ทำงานได้ต่างกัน โครงสร้างสถาปัตยกรรมซอฟต์แวร์ จึงถูกกำหนดขึ้น เพื่อควบคุมสถาปัตยกรรมซอฟต์แวร์ที่หลากหลายดังกล่าวนั้นเอง

รูปแบบสถาปัตยกรรม หมายถึง ข้อบังคับหรือกฎเกณฑ์ทางด้านสถาปัตยกรรม ที่จัดตั้งขึ้นมา เพื่อจำแนกกลุ่มหรือหมวดหมู่ของสถาปัตยกรรมซอฟต์แวร์ ปัจจุบันมีการกำหนดรูปแบบสถาปัตยกรรมขึ้นมาหลากหลาย สามารถแบ่งออกเป็น 5 กลุ่ม ดังนี้

1. กลุ่มสถาปัตยกรรมแบบโครงสร้างทั่วไป (General Structure) เช่น สถาปัตยกรรมแบบ Layer, Pipe and Filter และ Blackboard เป็นต้น

2. กลุ่มสถาปัตยกรรมแบบกระจาย (Distributed System) เช่น สถาปัตยกรรมแบบ Client/Server, Three Tiers และ Broker เป็นต้น

3. กลุ่มสถาปัตยกรรมระบบแบบโต้ตอบ (Interactive System) เช่น สถาปัตยกรรมแบบ Model-View-Controller และ Presentation-Abstraction-Control เป็นต้น

4. กลุ่มสถาปัตยกรรมระบบที่สามารถดัดแปลงได้ (Adaptable System) เช่น สถาปัตยกรรมแบบ Micro-Kernel และ Reflection เป็นต้น

5. กลุ่มสถาปัตยกรรมระบบแบบอื่น ๆ เช่น สถาปัตยกรรมแบบ Batch, Interpreter, Process Control และ Rule based เป็นต้น

แบบแผนการออกแบบ

แบบแผน (Patten) หมายถึง วิธีแก้ปัญหาที่มีรูปแบบเป็นกลาง เพื่อใช้กับปัญหาทั่วไปตามลักษณะของปัญหาที่ระบุในวิธีแก้ปัญหาแบบแผนการออกแบบในระดับจุลภาค ถูกกำหนดขึ้นเนื่องจากรูปแบบสถาปัตยกรรมแสดงถึงโครงสร้างในระดับบนของสถาปัตยกรรมเท่านั้น ไม่สามารถแสดงให้เห็นถึงระดับรายละเอียดปลีกย่อยอื่นๆ ที่ประกอบอยู่ภายในสถาปัตยกรรมรูปแบบนั้นได้

โดย Design Pattern ในระดับจุลภาคถูกแบ่งออกเป็น 3 กลุ่ม ดังนี้

1. แบบแผนในกลุ่มการสร้าง (Creational Pattern) เช่น Builder, Factory, Prototype และ Singleton เป็นต้น

2. แบบแผนในกลุ่มโครงสร้าง (Structural Pattern) เช่น Adapter, Bridge, Composite, Decorator, Facade, Flyweight และ Proxy เป็นต้น

3. แบบแผนในกลุ่มพฤติกรรม (Behavioral Pattern) เช่น Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template, และ Visitor เป็นต้น

กลุ่มของซอฟต์แวร์และ Framework วิธีการอย่างหนึ่งที่ช่วยสนับสนุนการออกแบบซอฟต์แวร์และคอมโพเน้นของซอฟต์แวร์ ให้สามารถนำกลับมาใช้ใหม่ได้ก็คือ การกำหนดเป็นกลุ่มของซอฟต์แวร์ (Families of Software) ขึ้น ซึ่งรู้จักกันในชื่อ Software Product Line

4. คุณภาพและการประเมินคุณภาพงานออกแบบซอฟต์แวร์

เกณฑ์คุณภาพ (Quality Attribute)

1. การทำงานของโปรแกรม (Functionality) จะประเมินจากลักษณะ (Feature Set) และความสามารถ (Capability) ของโปรแกรม นอกจากนี้ ยังประเมินจากหน้าที่ทั่วไปของโปรแกรม และความปลอดภัยเมื่อต้องทำงานรวมเป็นระบบ

2. ความสามารถในการใช้งาน (Usability) พิจารณาจากผลตอบกลับจากการใช้งานของผู้ใช้ ไม่ว่าจะเป็นการใช้งานง่าย และเรียนรู้ง่าย

3. ความน่าเชื่อถือ (Reliability) วัดจากความถี่และความรุนแรงของความผิดพลาดที่เกิดขึ้น ความถูกต้องของผลลัพธ์ที่ได้ เวลาเฉลี่ยของความล้มเหลว (Mean-Time-To-Failure:MTTF) ความสามารถในการกู้คืนระบบ และความสามารถในการคาดการณ์ได้ของโปรแกรม

4. ประสิทธิภาพ (Performance) วัดจากความเร็วของการประมวลผล ระยะเวลาตอบสนอง ทรัพยากรที่ใช้ ปริมาณที่ทำได้ในช่วงเวลาหนึ่ง และประสิทธิผลในการทำงาน

5. ความสามารถในการสนับสนุนการใช้งาน (Supportability) และความสามารถในการบำรุงรักษา (Maintainability) พิจารณาจากความสามารถในการเพิ่มเติมส่วนกรทำงาน ความสามารถในการแปลงการทำงาน และการบริการ ยังพิจารณาจากความสามารถในการทดสอบ การทำงานข้ามระบบได้ และการจัดสภาพแวดล้อมของระบบด้วย

การวิเคราะห์และประเมินคุณภาพ (Quality Analysis and Evaluation)

การวิเคราะห์และการประเมินคุณภาพเป็นกิจกรรมที่ช่วยให้มั่นใจว่าซอฟต์แวร์ที่ถูกออกแบบไว้จะต้องมีคุณภาพโดยทีมงานสามารถใช้เครื่องมือและเทคนิคต่างๆ ในการวิเคราะห์และประเมิน ซึ่งแบ่งตามกิจกรรมได้ ดังนี้

1. ทบทวนงานออกแบบซอฟต์แวร์ (Software Design Review) เทคนิคที่ช่วยให้การทบทวนงานออกแบบมีประสิทธิภาพ ได้แก่ Group-Based Technique เป็นเทคนิคในการตรวจสอบและปรับปรุงคุณภาพของงานออกแบบ เช่น การทบทวนสถาปัตยกรรมซอฟต์แวร์ และการตรวจสอบอย่างละเอียด เป็นต้น

2. วิเคราะห์งานออกแบบ (Static Analysis) ผลที่ได้จากการวิเคราะห์จะนำไปประเมินคุณภาพของงานออกแบบ เทคนิคที่ช่วยใช้การวิเคราะห์มีประสิทธิภาพ เช่น Fault-tree Analysis, Auto-cross Checking เป็นต้น

3. การจำลองสถานการณ์และการสร้างต้นแบบ (Simulation and Prototyping) เช่น การจำลองเหตุการณ์ประสิทธิภาพของซอฟต์แวร์ และการสร้างต้นแบบซอฟต์แวร์เพื่อทดสอบความเป็นไปได้ เป็นต้น

การวัด (Measure)

การวัดสามารถใช้กับการประเมินหรือการประมาณการคุณลักษณะบางอย่าง เช่น ขนาด โครงสร้าง หรือคุณภาพ ของซอฟต์แวร์ได้ แต่การวัดคุณภาพของการออกแบบซอฟต์แวร์จะแบ่งออกเป็น 2 ประเภท ดังนี้

1. การวัดการออกแบบเชิงฟังก์ชัน (Function-Oriented Measure)

ใช้กับซอฟต์แวร์ที่ออกแบบด้วย แนวคิดเชิงโครงสร้าง ที่มีการแบ่งระบบใหญ่ออกเป็นระบบย่อยตามหน้าที่ เรียกว่า โมดูล และนำเสนอด้วยแผนภาพเชิงโครงสร้าง การวัดคุณภาพประเภทนี้จึงสามารถวัดได้จากคุณลักษณะของโมดูล เช่น Coupling Cohesion

2. การวัดการออกแบบเชิงวัตถุ (Object-Oriented Design Measure) ใช้กับซอฟต์แวร์ที่ถูกออกแบบด้วยแนวทางเชิงวัตถุ ที่มีการจัดให้ทุกสิ่งของระบบเป็นวัตถุ และนำเสนอโครงสร้างของซอฟต์แวร์ด้วย Class Diagram การวัดคุณภาพประเภทนี้ จึงสามารถวัดได้จากลักษณะภายในคลาส เช่น การวัดความสัมพันธ์ระหว่างคลาส การนับจำนวนการโต้ตอบกันระหว่างเมธอดของคลาส

หลักการออกแบบซอฟต์แวร์

เนื่องจากการออกแบบซอฟต์แวร์ต้องคำนึงถึงคุณภาพของซอฟต์แวร์ที่จะผลิตด้วย ดังนั้น ทีมงานจึงควรใช้แนวทางการออกแบบบางประการ เพื่อนำไปสู่การออกแบบที่ดี ดังนี้

1. การออกแบบควรแสดงให้เห็นถึงรูปแบบสถาปัตยกรรมที่เลือกใช้อย่างชัดเจนและมีแบบแผน
2. การออกแบบควรมีลักษณะเป็นโมดูล
3. การออกแบบควรนำเสนอด้านข้อมูล สถาปัตยกรรม ส่วนประสาน และคอมโพเน้นท์ที่ชัดเจน
4. ควรออกแบบคอมโพเน้นท์ให้มีอิสระต่อกัน
5. ควรออกแบบให้ส่วนประสานระหว่างคอมโพเน้นท์กับสภาพแวดล้อมภายนอกมีความซับซ้อนน้อยที่สุด

6. การออกแบบควรนำข้อมูลมาจากการวิเคราะห์ระบบ และใช้ระเบียบวิธีปฏิบัติเดียวกัน
7. สัญลักษณ์ที่ใช้ในการออกแบบควรสื่อความหมายได้ชัดเจน และเป็นมาตรฐาน
8. งานออกแบบควรมีโครงสร้างที่ดี เพื่อการแก้ไขที่ง่ายและใช้ต้นทุนน้อย
9. การออกแบบในระดับคอมโพเน้นท์มีลักษณะแบบ Functional Independence คือ ฟังก์ชันงานมีความเป็นอิสระต่อกัน ไม่ขึ้นต่อกัน
10 คอมโพเน้นท์ของซอฟต์แวร์จะต้องมีลักษณะการขึ้นต่อกันน้อยที่สุด (Loosely Coupled)

5. แนวคิดในการออกแบบซอฟต์แวร์

การคิดแบบนามธรรม (Abstraction)

เป็นพื้นฐานทางความคิดในการออกแบบอย่างหนึ่ง ที่ช่วยลดความซับซ้อนของระบบลงได้ เมื่อมีการพิจารณาถึงแนวทางแก้ไข ของแต่ละปัญหา จะเกิดการคิดแบบเป็นนามธรรมขึ้นเป็นระดับ ได้แก่ Procedural Abstraction เป็นการสร้างลำดับขั้นตอนของชุดคำสั่งของฟังก์ชันใดฟังก์ชันหนึ่งขึ้นมา โดยจะไม่ระบุถึงรายละเอียดภายในฟังก์ชั่น และ Data Abstraction คือ ชื่ออ็อบเจ็กต์ข้อมูลที่อยู่ใน Procedural Abstraction

สถาปัตยกรรม (Architecture)

เป้าหมายของการออกแบบสถาปัตยกรรม ก็เพื่อเป็นกรอบให้กับการออกแบบส่วนประกอบที่เหลือของระบบ ให้เป็นไปในทิศทางเดียวกัน และอยู่ขนสถาปัตยกรรมเดียวกันนั่นเอง การออกแบบโครงสร้างหรือสถาปัตยกรรมสามารถนำเสนอออกมาในรูปแบบจำลอง 4 ชนิด ได้แก่ Structural Model, Framework Model, Dynamic Model, Process Model, และ Functional Model

แบบแผน (Pattern)

คือหลักและวิธีแก้ไขปัญหาชนิดใดชนิดหนึ่งที่สามารถนำไปใช้กับปัญหาชนิดเดียวกันที่เกิดซ้ำได้ โดยจะต้องอธิบายโครงสร้างการออกแบบซอฟต์แวร์ไว้อย่างละเอียด ไม่ว่าจะเป็นชื่อแบบแผน วิธีแก้ปัญหา และผลที่ตามมา การใช้ Pattern จะช่วยให้งานผลิตซอฟต์แวร์จะดำเนินไปได้รวดเร็ว

การแบ่งระบบ (Modularity)

เป็นการแบ่งระบบหรือซอฟต์แวร์ออกเป็นส่วนย่อยๆ แต่ละส่วน ในระบบงานใดๆ ย่อมประกอบไปด้วยการทำงานหลายส่วนหากแบ่งออกแต่ละส่วนจะสามารถทำงานได้ง่ายขึ้น ลดความซักซ้อน

การซ่อนรายละเอียด (Information Hiding)

เนื่องจากการแบ่งระบบออกเป็นโมดูลย่อย นักออกแบบระบบได้เล็งเห็นปัญหาที่อาจเกิดขึ้นเมื่อมีการนำมาประสานเพื่อทำให้ทำงานร่วมกัน จึงเกิดความยุ่งยากในการใช้งานร่วมกัน จึงซ่อนรายละเอียดไว้เพื่อป้องกันการเข้าถึง ซึ่งอาจส่งผลให้ผิดพลาดได้

ความเป็นอิสระต่อกันในการทำงาน (Functional Independence)

Coupling เป็นการวัดความสัมพันธ์ระหว่างโมดูล 2 โมดูลว่ามีความซับซ้อนหรือมีระดับการขึ้นต่อกันของโมดูลมากน้อยเพียงใด

Cohesion เป็นการวัดระดับการยึดเกาะกันของหน้าที่หรือกิจกรรมในโมดูล เพื่อประมวลข้อมูลเป็นผลลัพธ์ที่ต้องการ ลักษณะโครงสร้างที่ดีจะต้องมีระดับการยึดเกาะกันของหน้าที่ในโมดูลสูง

การกลั่นกรอง (Refinement)

การกลั่นกรองเป็นการบรรยายรายละเอียดของแต่ละฟังก์ชันเป็นลำดับขั้น เริ่มต้นจากชื่อฟังก์ชันที่จะถูกกำหนดขึ้นในระดับบนสุดของการคิดแบบนามธรรม ซึ่งเป็นเพียงการกำหนดชื่อฟังก์ชันเท่านั้น ยังไม่มีรายละเอียดการทำงานภายในข้อมูล การกลั่นกรองเป็นการนำชื่อฟังก์ชันเหล่านั้น มาเพิ่มเติมรายละเอียดการทำงานภายในให้ชัดเจนยิ่งขึ้นในแต่ละระดับของการกลั่นกรอง

การปรับโครงสร้างการออกแบบ (Refactoring)

เป็นเทคนิคในการปรับโครงสร้างการออกแบบภายในของคอมโพเน้นท์ โดยไม่ต้องเปลี่ยนฟังก์ชันหรือพฤติกรรมของคอมโพเน้นท์ โดยเมื่อซอฟต์แวร์ถูกปรับโครงสร้าง จะเริ่มต้นจากการนำงานออกแบบเดิมมาพิจารณาถึงความซ้ำซ้อน ส่วนประกอบที่ไม่ได้ถูกใช้งาน อัลกอริธึมที่ไม่มีประสิทธิภาพหรือไม่จำเป็น ตลอดจนโครงสร้างข้อมูลที่ไม่เหมาะสม หรือข้อผิดพลาดจากการออกแบบอื่นๆ แล้วนำมาแก้ไขให้มีประสิทธิภาพและถูกต้องมากขึ้น

Design Class

สำหรับการวิเคราะห์ระบบด้วยแนวทางเชิงวัตถุ เมื่อจบขั้นตอนการวิเคราะห์ระบบแล้ว สิ่งที่ได้คือ แบบจำลองคลาส ที่แสดงเพียงมุมมองของระบบในระดับบนเท่านั้น เมื่อมาถึงขั้นตอนการออกแบบ วิศวกรซอฟต์แวร์จะต้องนำแบบจำลองคลาสเหล่านั้น มากลั่นกรองเพื่อกำหนดรายละเอียดเชิงลึกของแต่ละคลาสอีกครั้ง เพื่อให้เขียนโค้ดได้ง่ายขึ้น และต้องสร้างแบบจำลองคลาสที่แสดงให้เห็นถึงโครงสร้างภายในที่สนับสนุนกระบวนการทางธุรกิจด้วย

แนวคิดในการออกแบบซอฟต์แวร์

Design Class

สิ่งที่ได้คือ Design Class ซึ่งประกอบไปด้วย Design Class 5 ชนิด แบ่งตาม Layer ของสถาปัตยกรรมระบบ ได้แก่

1. User Interface Class
2. Business Domain Class
3. Process Class
4. Persistent Class
5. System Class

6. กลยุทธ์และระเบียบวิธีของการออกแบบซอฟต์แวร์

กลยุทธ์ในการอกแบบซอฟต์แวร์ เป็นเพียงหลักและแนวทางในการปฏิบัติงานแบบซอฟต์แวร์เท่านั้น ไม่ได้ระบุถึงวิธีทำงานอย่างชัดเจน แต่สำหรับระเบียบวิธี ในการออกแบบซอฟต์แวร์จะระบุถึงรายละเอียดของวิธีการทำงานอย่างชัดเจน พร้อมกับเตรียมสัญลักษณ์ต่างๆ ของแบบจำลองเฉพาะระเบียบวิธีนั้นไว้ให้ใช้งานด้วย ทำให้ทีมวิศวกรซอฟต์แวร์ทำงานได้ง่ายขึ้น ปัจจุบัน กลยุทธ์และระเบียบวิธีในการออกแบบซอฟต์แวร์มีหลายวิธี สรุปได้ดังนี้

กลยุทธ์ทั่วไปในการออกแบบซอฟต์แวร์ (General Strategy)

กลยุทธ์ทั่วไปในการออกแบบซอฟต์แวร์ ได้แก่ Divide-and Conquer, Stepwise Refinement, Top-down and Bottom-up Strategy, Data Abstraction and Information Hiding, Heuristic และ Design Pattern เป็นต้น

การออกแบบเชิงฟังก์ชัน (Function-Oriented Design)

การออกแบบเชิงโครงสร้าง ซึ่งเป็นระเบียบวิธีที่ได้รับความนิยมมาตั้งแต่อดีตจนถึงปัจจุบันเป็นวิธีในการพิจารณาถึงฟังก์ชันของซอฟต์แวร์เป็นเกณฑ์ในการแบ่งส่วนซอฟต์แวร์ออกเป็นส่วนย่อย จากนั้นจะกำหนดรายละเอียดในแต่ละส่วนย่อยของซอฟต์แวร์และปรับปรุงในลักษณะโครงสร้างลำดับขั้นจากบนลงล่าง

สิ่งที่ใช้ประกอบการออกแบบเชิงโครงสร้าง คือ แผนภาพกระแสข้อมูล (Data Flow Diagram :DFD) และรายละเอียดของกระบวนการ (Process Description)

การออกแบบเชิงวัตถุ (Object-oriented Design)

ระเบียบวิธีเชิงวัตถุในปัจจุบันมีหลากหลายวิธี ในยุคแรกของระเบียบวิธีการออกแบบเชิงวัตถุจะพิจารณาหาวัตถุในโดเมนที่สนใจจากคำอธิบายความต้องการของลูกค้า และจัดโครงสร้างของอ็อบเจ็กต์แบบ Inheritance และ Polymorphism ต่อมา ได้มีระเบียบวิธีการออกแบบซอฟต์แวร์แบบคอมโพเน้นท์ (Component-base Design) ขึ้นมา เพื่อช่วยให้การผลิตซอฟต์แวร์รวดเร็วขึ้น

การออกแบบโดยใช้ข้อมูลเป็นศูนย์กลาง (Data-structure Centered Design)

เป็นวิธีการออกแบบโดยใช้ข้อมูลที่ฟังก์ชันจะนำมาประมวลผลเป็นหลัก เริ่มต้นจากการแสดงโครงสร้างข้อมูล ทั้งที่เป็นข้อมูลนำเข้าและผลลัพธ์ (Input and Output Data) โดยสร้างเป็นแผนภาพเพื่อจำลองโครงสร้างของข้อมูลเหล่านั้น ยกตัวอย่างแผนภาพเช่น Jackson Diagram เป็นต้น จากนั้น ทีมงานจะนำแผนภาพดังกล่าวไปออกแบบโครงสร้างควบคุมการทำงานของโปรแกรมต่อไป

การออกแบบคอมโพเน้นท์ (Component-base Design: CBD)

เป็นวิธีการออกแบบซอฟต์แวร์ด้วยการแบ่งเป็นส่วนประกอบย่อยที่เรียกว่า คอมโพเน้นท์ จะทำงานเป็นอิสระต่อกัน ทำงานได้ด้วยตนเอง และสามารถประกอบกับคอมโพเน้นท์อื่นเพื่อเติมเต็มการทำงานให้กับซอฟต์แวร์ได้ ดังนั้น จึงต้องมีการสื่อสารระหว่างคอมโพเน้นท์ผ่านทาง Interface ได้ถูกพัฒนาขึ้นเพื่อตอบสนองความต้องการผลิตซอฟต์แวร์ที่สามารถนำกลับมาใช้ใหม่ได้

7. แบบจำลองที่ใช้ในการออกแบบ

จำแนกได้เป็น 2 กลุ่ม ดังนี้

1. กลุ่ม Structural Description (Static View)
2. กลุ่ม Behavioral Description (Dynamic View)

1. กลุ่ม Structural Description (Static View)

เป็นแบบจำลองที่ใช้อธิบายมุมมองด้านโครงสร้างของซอฟต์แวร์ โดยแสดงให้เห็นรายละเอียดของแต่ละคอมโพเน้นท์และความสัมพันธ์ระหว่างคอมโพเน้นท์ด้วย แบบจำลองในกลุ่มนี้ อาจอธิบายโครงสร้างด้วยแผนภาพหรือข้อความ ได้แก่

1.1 Architecture Description Language: ADL ใช้อธิบายสถาปัตยกรรมซอฟต์แวร์แบบคอมโพเน้นท์และการเชื่อมต่อคอมโพเน้นท์

- Class And Object Diagram แผนภาพแสดงโครงสร้างของคลาส (อ็อบเจ็กต์) และความสัมพันธ์ระหว่างคลาส

- Component Diagram แผนภาพแสดงคอมโพเน้นท์ที่เป็นส่วนประกอบของระบบ และแสดงความสัมพันธ์ระหว่างคอมโพเน้นท์ นอกจากนี้ยังแสดงให้เห็น Interface ของคอมโพเน้นท์ด้วย

1.2 Collaboration Responsibility Card: CRC ใช้บันทึกชื่อคอมโพเน้นท์ (คลาส) พร้อมกับคอมโพเน้นท์ที่มีความสัมพันธ์กันและหน้าที่ของคอมโพเน้นท์ด้วย

- Deployment Diagram แผนภาพแสดงโครงสร้างทางด้านฮาร์ดแวร์ (โหนด) ของระบบและความสัมพันธ์ระหว่างโหนดชนิดต่างๆ

- Entity Relationship Diagram ERD แผนภาพแสดงความสัมพันธ์ระหว่าง Entity ใช้แสดงโครงร่างของฐานข้อมูล

1.3 Interface Description Language: IDL มีลักษณะคล้ายกับการเขียนคำสั่งในโปรแกรม ใช้กำหนดรายละเอียดของ Interface

- Jackson Structure Diagram แผนภาพแสดงโครงสร้างควบคุมการประมวลผลข้อมูลแบบเรียงลำดับแบบเลือกทำ

- Structure Chart แผนภาพแสดงโครงสร้างของโปรแกรม แสดงให้เห็นการเรียกใช้โมดูล

2. กลุ่ม Behavioral Description (Dynamic View)

เป็นแบบจำลองที่ใช้อธิบายมุมมองด้านพฤติกรรมการทำงานของซอฟต์แวร์และคอมโพเน้นท์ แสดงให้เห็นพฤติกรรม การทำงานที่เปลี่ยนแปลงไปเมื่อเกิดเหตุการณ์ใดเหตุการณ์หนึ่ง

Activity Diagram แผนภาพแสดงลำดับการดำเนินกิจกรรมของระบบที่เกิดจากการทำงานของอ็อบเจ็กต์

Collaborative Diagram แผนภาพแสดงให้เห็นถึงการปฏิสัมพันธ์ระหว่างอ็อบเจ็กต์ (เมื่อไม่เป็นไปตามลำดับเวลา)

Data Flow Diagram แผนภาพแสดงการไหลของข้อมูล จากกระบวนการหนึ่งไปอีกกระบวนการหนึ่ง

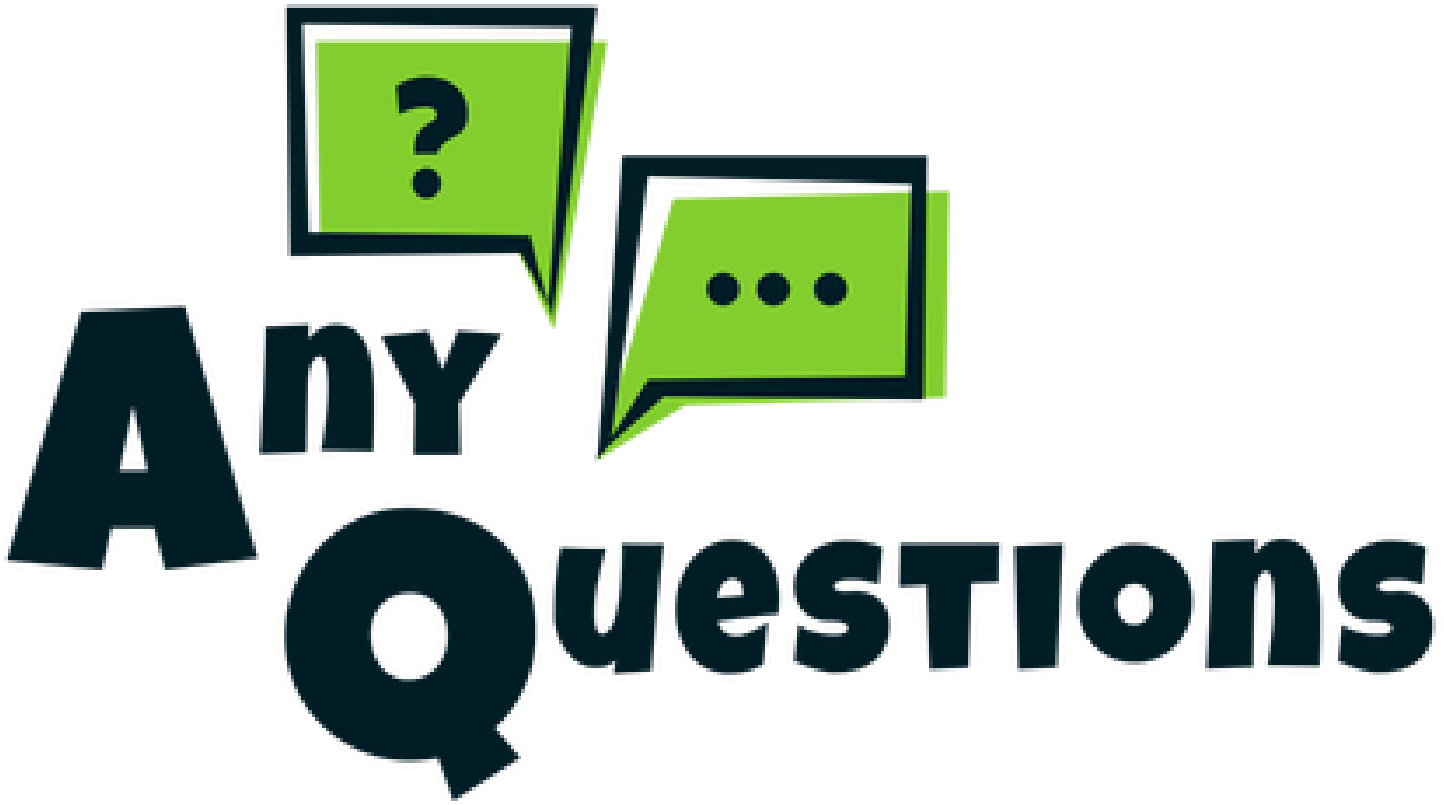
Decision Table and Diagram ตารางการตัดสินใจใช้แสดงให้เห็นการตัดสินใจดำเนินกิจกรรมอย่างใดอย่างหนึ่งของระบบภายใต้เงื่อนไขที่ซับซ้อน

Flowchart and Structure Flowchart แผนภาพแสดงลำดับการดำเนินกิจกรรม

Sequence Diagram แผนภาพแสดงปฏิสัมพันธ์ระหว่างอ็อบเจ็กต์โดยแสดงถึงสถานะและการเปลี่ยนแปลงสถานะของอ็อบเจ็กต์ที่มีต่อเหตุการณ์ใดเหตุการณ์หนึ่ง

Formal Specification Language ใช้กำหนดรายละเอียดของ Interface และพฤติกรรมของคอมโพเน้นท์

Pseudo-code and Program Design Language PDL มีลักษณะคล้ายกับการเขียนคำสั่งในโปรแกรม เรียกว่า รหัสเทียม จำลองการทำงานของฟังก์ชัน โพรซีเดอร์ หรือเมธอด


Any Questions