บทที่ 3 การออกแบบซอฟต์แวร์ (เพิ่มเติม)

- การออกแบบซอฟต์แวร์
- วิศวกรรมการออกแบบ (Design Engineering)
- การออกแบบสถาปัตยกรรม(Architectural Design)

การออกแบบซอฟต์แวร์

- การออกแบบซอฟต์แวร์ หมายถึง กระบวนการกำหนดสถาปัตยกรรม ส่วนประกอบ ส่วนประสาน และลักษณะด้านอื่นๆ ของระบบ สิ่งที่ได้จากการออกแบบ ก็คือ แบบจำลองการออกแบบ นั่นเอง
- การออกแบบซอฟต์แวร์ เป็นการนำข้อกำหนดความต้องการของผู้ใช้มากำหนดรายละเอียดโครงสร้างภายในของซอฟต์แวร์เพื่อ นำไปใช้ในการเขียนและทดสอบโปรแกรมในระยะการสร้างซอฟต์แวร์

วิศวกรรมการออกแบบ

- วิศวกรรมการออกแบบรวบรวมหลักการ แนวความคิด และวิธีปฏิบัติที่นำไปสู่การพัฒนาระบบคอมพิวเตอร์ที่มีคุณภาพสูง การออกแบบเป็นกิจกรรมหลักอย่างหนึ่งของวิศวกรรมคอมพิวเตอร์

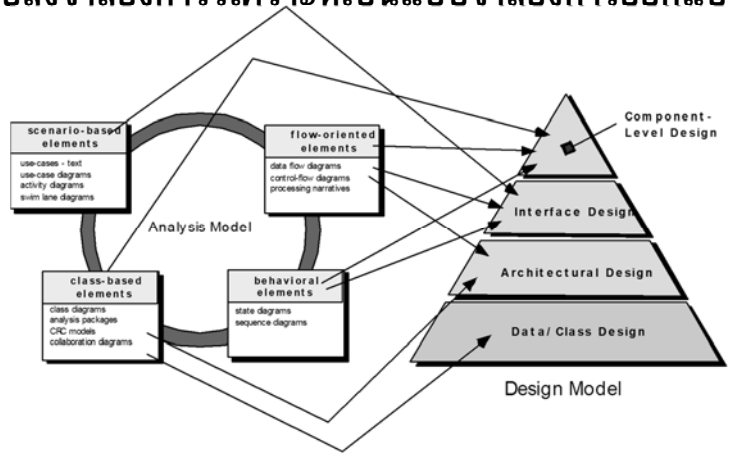
  โดยมีเป้าหมาย คือ การสร้างแบบร่างของระบบ หรือมีการนำเสนอระบบในแต่ละด้าน ให้มีคุณสมบัติ

  1. **firmness** การออกแบบไม่มีข้อผิดพลาด
  2. **commodity** ตรงกับวัตถุประสงค์การใช้งาน
  3. **delight** ทำให้ผู้ใช้รู้สึกพอใจ

การออกแบบ

- จากขั้นตอนการวิเคราะห์จะทำให้ได้ข้อมูล เพื่อจะนำไปสร้างแบบจำลองทั้ง 4 ประเภท ซึ่งจะนำไปใช้ต่อในขั้นตอนการออกแบบ
  - Scenerio-based elements องค์ประกอบเชิงฉากบรรยาย: use-case diagram
  - Class-based elememts องค์ประกอบเชิงคลาส : class diagram
  - Flow-oriented elements องค์ประกอบเชิงกระแส : Data flow diagram
  - Behavioral elements องค์ประกอบเชิงพฤติกรรม : State diagram, Sequence diagram

การแปลงจำลองการวิเคราะห์เป็นแบบจำลองการออกแบบ



แบบจำลองการออกแบบ (Design Model)

- Data/Class Design
- Architecture Design
- Interface Design
- Component-level Design

แบบจำลองการออกแบบ (Design Model)

- Data/Class Design

  เป็นการออกแบบข้อมูล เนื้อหาที่จะใช้ในระบบ : แอตทริบิวส์ คลาส

- Architecture Design

  การออกแบบสถาปัตยกรรม -- ความสัมพันธ์ระหว่างส่วนประกอบเชิงโครงสร้างหลักๆ ของซอฟต์แวร์ สไตล์ และแบบรูปสถาปัตยกรรม เอามาจากข้อกำหนดระบบ แบบจำลองการวิเคราะห์

แบบจำลองการออกแบบ(Design Model)

- Interface Design

  การออกแบบส่วนติดต่อกับผู้ใช้ เพื่อการนำเสนอ และใช้งานซอฟต์แวร์ ต้องคำนึงถึงลำดับและขั้นตอนการทำงานของระบบ

- Component-level Design

  การออกแบบระดับรายละเอียด เป็นการออกแบบโปรแกรมย่อย หรือฟังก์ชั่นย่อยต่างๆ ของระบบ ที่จะประกอบเป็นระบบ

การออกแบบและคุณภาพ (Design and Quality)

- The design must implement all of the explicit requirements contained in the analysis model, and it must accommodate all of the implicit requirements desired by the customer.
- The design must be a readable, understandable guide for those who generate code and for those who test and subsequently support the software.
- The design should provide a complete picture of the software, addressing the data, functional, and behavioral domains from an implementation perspective

แนวทางด้านคุณภาพ (Quality Guidelines)

- A design should exhibit an architecture that
  - has been created using recognizable architectural styles or patterns,
  - is composed of components that exhibit good design characteristics and
  - can be implemented in an evolutionary fashion
  - For smaller systems, design can sometimes be developed linearly.
- A design should be modular; that is, the software should be logically partitioned into elements or subsystems
- A design should contain distinct representations of data, architecture, interfaces, and components.

แนวทางด้านคุณภาพ (Quality Guidelines)

- A design should lead to data structures that are appropriate for the classes to be implemented and are drawn from recognizable data patterns.
- A design should lead to components that exhibit independent functional characteristics.
- A design should lead to interfaces that reduce the complexity of connections between components and with the external environment.
- A design should be derived using a repeatable method that is driven by information obtained during software requirements analysis.
- A design should be represented using a notation that effectively communicates its meaning.

คุณลักษณะด้านคุณภาพ (Quality Attributes)

- Functionality - assessed by evaluating the feature set and capabilities of a program, generality of functions delivered, security of the overall system
- Usability - assessed by evaluating human factors (aesthetics, consistency, documentation)
- Reliability - evaluated by measuring MTBF(Mean Time between Failure) and severity of failures, MTTR (Mean Time to Repair), ability to recover from failure, predictability
- Performance - evaluated by measuring processing speed, response time, resource consumption, throughput, efficiency, etc.
- Supportability - extensibility, adaptability, serviceability, testability, compatibility, configurability

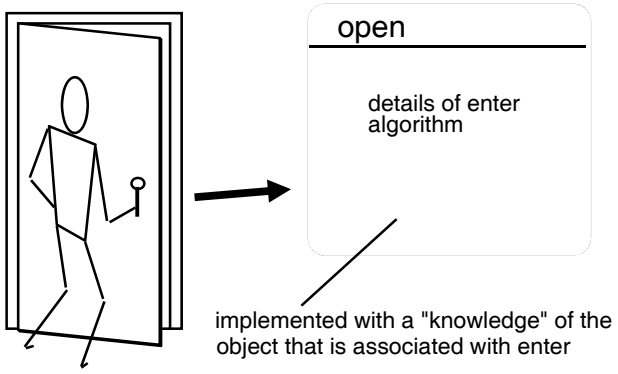
Fundamental Concepts

- **Abstraction** — data, procedure, control
- **Architecture** — the overall structure of the software
- **Patterns** — conveys the essence" of a proven design solution
- **Modularity** — compartmentalization of data and function
- **Hiding** — controlled interfaces
- **Functional independence** — single-minded function and low coupling
- **Refinement** — elaboration of detail for all abstractions
- **Refactoring** — a reorganization technique that simplifies the design

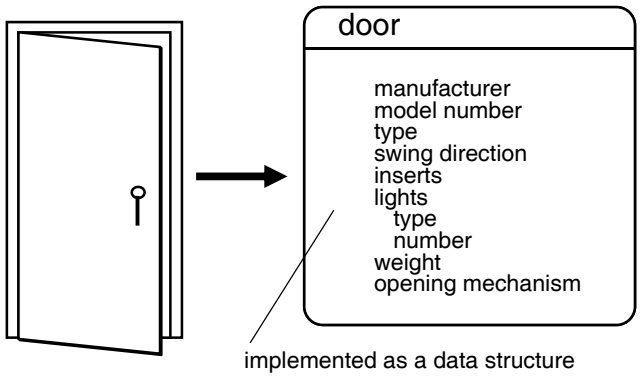
Abstraction

- การกำหนดสาระสำคัญ เป็นพื้นฐานทางความคิดในการออกแบบ เราสามารถกำหนดสาระสำคัญได้หลายระดับ ระดับบนสุดนั้นจะอธิบายในภาพรวมของปัญหาและสภาพแวดล้อมภายนอก ในระดับถัดมาจะอธิบายถึงวิธีการแก้ปัญหาค่อนข้างละเอียด
  - Procedural Abstraction -- เชิงกระบวนการทำงาน : ลำดับของคำสั่งที่ทำหน้าที่เฉพาะเจาะจงอย่างหนึ่ง
  - Data Abstraction -- เชิงข้อมูล : เป็นชื่อของข้อมูลที่อยู่ใน Procedural Abstraction

Procedural Abstraction



Data Abstraction



Architecture

- โครงสร้างทั้งหมดของซอฟต์แวร์ ที่แสดงให้เห็นถึงโครงสร้างของโปรแกรมย่อยหรือโมดูล และการทำงานร่วมกันของโปรแกรมย่อยเหล่านั้น นอกจากนี้ยังแสดงให้เห็นโครงสร้างของข้อมูลที่ถูกใช้ในแต่ละโปรแกรมย่อยด้วย
- การออกแบบสถาปัตยกรรม สามารถทำได้ด้วยแบบจำลอง
  - แบบจำลองโครงสร้าง (Structural Models)
  - แบบจำลองโครงแบบ (Framework Models)
  - แบบจำลองเชิงพลวัต (Dynamic Models)
  - แบบจำลองเชิงกระบวนการ (Process Models)

Patterns

- อธิบายโครงสร้างตัวแบบที่ช่วยแก้ปัญหาการออกแบบ หลักและวิธีการแก้ปัญหาชนิดหนึ่งชนิดใด ที่สามารถนำไปใช้กับปัญหาชนิดเดียวกันที่เกิดขึ้นซ้ำได้
- การใช้ pattern จะช่วยให้งานผลิตซอฟต์แวร์ดำเนินไปได้อย่างรวดเร็ว ประหยัดเวลาในการออกแบบ

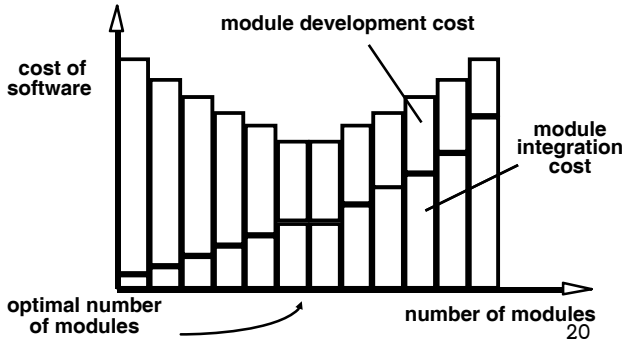
Modularity

- การแบ่งระบบหรือซอฟต์แวร์แยกออกเป็นส่วนๆ แต่ละส่วน เรียกว่า “โมดูล” (Module) ซึ่งจะประกอบกันได้เพื่อทำงานตามความต้องการ
- easier to build, easier to change, easier to fix
- การแบ่งระบบเป็นโมดูลจะช่วยให้การออกแบบงานในแต่ละส่วนง่ายขึ้น นอกจากนี้ยังช่วยให้การวางแผนพัฒนา การแก้ไขหรือเปลี่ยนแปลง ตลอดจนการทดสอบและซ่อมบำรุงเป็นเรื่องง่าย

Modularity

- จากกราฟ

  ค่าใช้จ่ายในการพัฒนาจะลงลดเมื่อจำนวนโมดูลเพิ่มขึ้น (ขนาดของโมดูลเล็กลง) แต่ค่าใช้จ่ายในการรวมโมดูลเข้าด้วยกันก็เพิ่มขึ้นตามจำนวนโมดูลที่เพิ่มขึ้นด้วย



Information Hiding

- โมดูลจะต้องซ่อนรายละเอียดการทำงานไว้ ไม่ว่าจะเป็นอัลกอริทึมหรือข้อมูลของโมดูล เพื่อป้องกันการเข้าถึงข้อมูลภายในโมดูลโดยไม่จำเป็น
- การใช้หลักการซ่อนข่าวสารในการออกแบบโมดูล ทำให้ง่ายต่อการปรับปรุง การทดสอบ และกิจกรรมภายหลัง เช่น การบำรุงรักษา

Functional independence

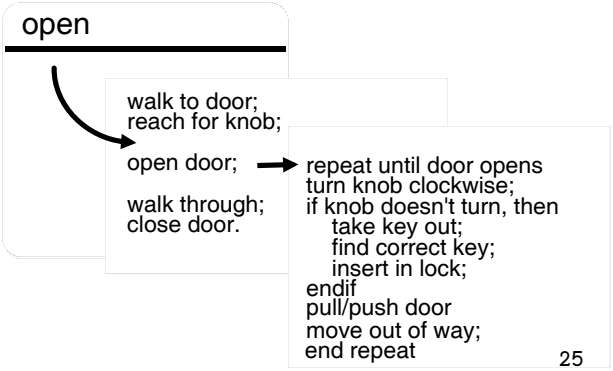
- การออกแบบให้โมดูลมีความเป็นอิสระต่อกัน โมดูลควรทำหน้าที่เดียว หลีกเลี่ยงการมีปฏิสัมพันธ์กับโมดูลอื่นๆ
- โมดูลที่เป็นอิสระต่อกันจะง่ายต่อการบำรุงรักษา เพราะผลกระทบจากการเปลี่ยนแปลงมีอยู่จำกัด ลดการแพร่กระจายความผิดพลาด และมีความเป็นไปได้ที่จะนำกลับมาใช้ใหม่
- การประเมินระดับของความเป็นอิสระของโมดูล ประเมินได้จาก
  - ความเชื่อมโยง (Coupling)
  - ความเชื่อมแน่น (Cohesion)

Functional independence

- Coupling เป็นการวัดความสัมพันธ์ระหว่างโมดูล 2 โมดูลว่ามีความซับซ้อนหรือมีระดับการขึ้นต่อกันของโมดูลมากน้อยเพียงใด การเชื่อมต่อของโมดูลจะผ่าน Interface โครงสร้างของโมดูลที่ดีจะต้องมีระดับการขึ้นต่อกันของโมดูลน้อย (Loosely Coupled)
- Cohesion เป็นการวัดระดับการยึดเกาะกันของหน้าที่หรือกิจกรรมในโมดูล เพื่อประมวลผลข้อมูลให้ได้เป็นผลลัพธ์ที่ต้องการ ลักษณะของโมดูลที่ดีจะต้องมีระดับการยึดเกาะกันของหน้าที่ในโมดูลสูง (High Cohesion) โดยที่มีการปฏิสัมพันธ์กับโมดูลอื่น กิจกรรมอื่นในโมดูล หรือระบบอื่นน้อยที่สุด

Refinement

- การลงรายละเอียดเพิ่มเติมรายละเอียดกระบวนการทำงานจากประโยคที่ระบุหน้าที่ไปทีละขั้นตอนจนกว่าจะได้ประโยคภาษาโปรแกรม



Refactoring

- การแยกส่วนประกอบใหม่ เป็นเทคนิคในการปรับโครงสร้างการออกแบบ เป็นการจัดระเบียบใหม่ เพื่อให้งานออกแบบองค์ประกอบย่อย หรือตัวโค้ด ที่ลักษณะที่ง่ายขึ้น โดยไม่ไปเปลี่ยนแปลงพฤติกรรมของการทำงาน
- When software is refactored, the existing design is examined for:
  - redundancy
  - unused design elements
  - inefficient or unnecessary algorithms
  - poorly constructed or inappropriate data structures
  - or any other design failure that can be corrected to yield a better design.

Design Class

- คลาสของข้อมูล อธิบายส่วนประกอบของโดเมนปัญหาที่มองเห็นได้จากมุมมองของผู้ใช้งานหรือลูกค้า แสดงให้เห็นโครงสร้างภายใน คลาสออกแบบมี 5 ประเภท
  - User Interface Class
  - Business Domain Class
  - Process Class
  - Persistent Class
  - System Class

ลักษณะที่ดีของคลาสออกแบบ

- Complete and Sufficient
- Primitiveness
- High Cohesion
- Low Coupling

Design Model Elements

- Data elements
  - Data model --> data structures
  - Data model --> database architecture
- Architectural elements
  - Application domain
  - Analysis classes, their relationships, collaborations and behaviors are transformed into design realizations
  - Patterns and “styles” (Chapter 10)

Design Model Elements

- **Interface elements**
  - the user interface (UI)
  - external interfaces to other systems, devices, networks or other producers or consumers of information
  - internal interfaces between various design components.
- **Component elements**
- **Deployment elements**

สถาปัตยกรรมซอฟต์แวร์ (Software Architecture)

- The architecture is not the operational software. Rather, it is a representation that enables a software engineer to:

(1) analyze the effectiveness of the design in meeting its stated requirements,

(2) consider architectural alternatives at a stage when making design changes is still relatively easy, and

(3) reduce the risks associated with the construction of the software.

Why is Architecture Important?

- Representations of software architecture are an enabler for communication between all parties (stakeholders) interested in the development of a computer-based system.
- The architecture highlights early design decisions that will have a profound impact on all software engineering work that follows and, as important, on the ultimate success of the system as an operational entity.
- Architecture "constitutes a relatively small, intellectually graspable model of how the system is structured and how its components work together" [BAS03].

Data Design

- At the architectural level …
  - Design of one or more databases to support the application architecture
  - Design of methods for 'mining' the content of multiple databases
    - navigate through existing databases in an attempt to extract appropriate business-level information
    - Design of a data warehouse —a large, independent database that has access to the data that are stored in databases that serve the set of applications required by a business

Data Design

- At the component level …
  - refine data objects and develop a set of data abstractions
  - implement data object attributes as one or more data structures
  - review data structures to ensure that appropriate relationships have been established
  - simplify data structures as required

Data Design -- component level

1. The systematic analysis principles applied to function and behavior should also be applied to data.
2. All data structures and the operations to be performed on each should be identified.
3. A data dictionary should be established and used to define both data and program design.
4. Low level data design decisions should be deferred until late in the design process.
5. The representation of data structure should be known only to those modules that must make direct use of the data contained within the structure.
6. A library of useful data structures and the operations that may be applied to them should be developed.
7. A software design and programming language should support the specification and realization of abstract data types.

Architectural Styles

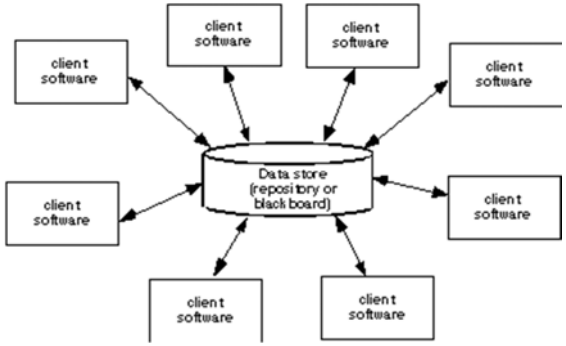
- Each style describes a system category that encompasses: (1) a set of components (e.g., a database, computational modules) that perform a function required by a system, (2) a set of connectors that enable "communication, coordination and cooperation" among components, (3) constraints that define how components can be integrated to form the system, and (4) semantic models that enable a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.

Architectural Styles

- Data-centered architectures
- Data flow architectures
- Call and return architectures
- Object-oriented architectures
- Layered architectures

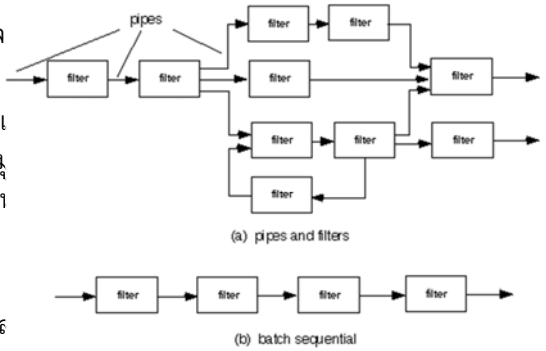
Data-Centered Architecture

- แหล่งเก็บข้อมูลเป็นศูนย์กลางของสถาปัตยกรรม
- จัดเก็บข้อมูลไว้ร่วมกัน ระบบย่อยสามารถเข้าถึงได้



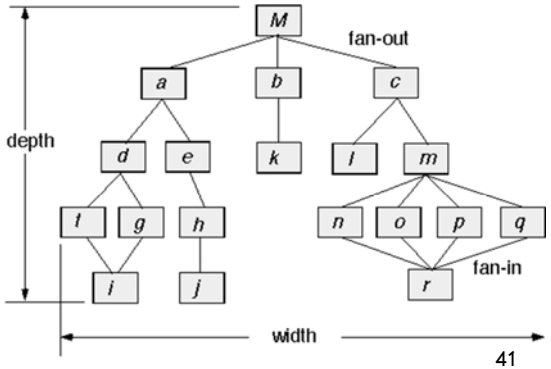
Data Flow Architecture

- แบ่งหน้าที่การประมวลผลข้อมูล ของกระบวนการ
- Pipe เป็นเสมือนท่อส่งข้อมูลนำเ ไปยังกระบวนการเปลี่ยนรูปข้อมู ซึ่งเรียกว่า Filter ที่เปรียบเสมือน การกรองข้อมูลจนได้ผลลัพธ์ที่ ต้องการ
- Batch sequential : การประมวล แบบเดียวกันเป็นกระแสเดียว



Call and Return Architecture

- มีลักษณะแบบ Top-down คือ จัดให้ระบบย่อยทำหน้าที่ควบคุมระดับย่อยอื่น
- ระดับย่อยที่อยู่ในระดับสูงสุดเรียกใช้ระบบย่อยอื่นที่อยู่ในระดับชั้นถัดลงมาด้านล่าง

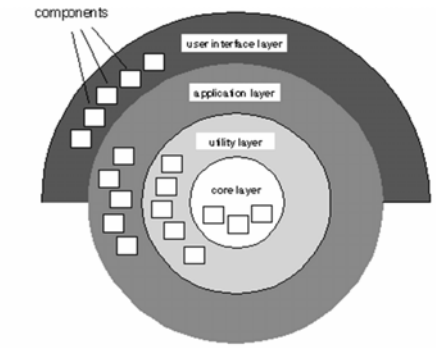


Object-oriented architectures

- คอมโพเน้นท์ของระบบ ห่อหุ้มเอาข้อมูลและตัวปฏิบัติการที่ทำงานกับข้อมูลเข้าไว้ด้วยกัน การสื่อสารและร่วมมือกันระหว่างคอมโพเน้นท์ ทำการส่งข้อความถึงกัน

Layered Architecture

- แสดงให้เห็นการจัดโครงสร้างระบบย่อย ในมุมมองแบบระดับชั้น แต่ละระดับชั้นคือส่วนประกอบย่อยที่รับผิดชอบการทำงานในแต่ละด้านของซอฟต์แวร์



Architectural Patterns

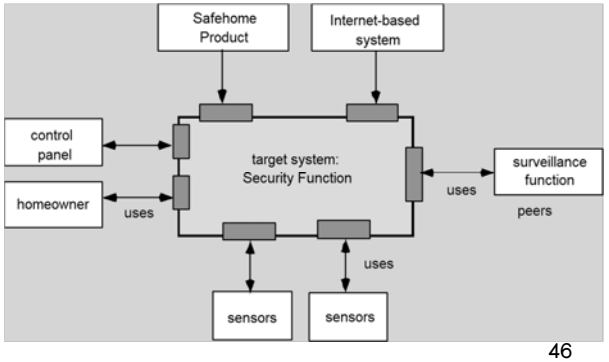
- Concurrency — applications must handle multiple tasks in a manner that enables parallelism
  - *operating system process management* pattern
  - *task scheduler* pattern
- Persistence — Data persists if it survives past the execution of the process that created it. Two patterns are common:
  - a *database management system* pattern that applies the storage and retrieval capability of a DBMS to the application architecture
  - an *application level persistence* pattern that builds persistence features into the application architecture
- Distribution — the manner in which systems or components within systems communicate with one another in a distributed environment
  - A *broker* acts as a 'middle-man' between the client component and a server component.

Architectural Design

- The software must be placed into context
  - the design should define the external entities (other systems, devices, people) that the software interacts with and the nature of the interaction
- A set of architectural archetypes should be identified
  - An ***archetype*** is an abstraction (similar to a class) that represents one element of system behavior
- The designer specifies the structure of the system by defining and refining software components that implement each archetype

Architectural Context

- ระบุเอนทิตี้ภายนอกที่ซอฟต์แวร์มีปฏิสัมพันธ์ด้วยและลักษณะของปฏิสัมพันธ์นั้น



Archetypes

- พื้นฐานของสถาปัตยกรรมที่เป็นนามธรรม คล้ายกับ Class แทนองค์ประกอบที่คงตัวของสถาปัตยกรรม

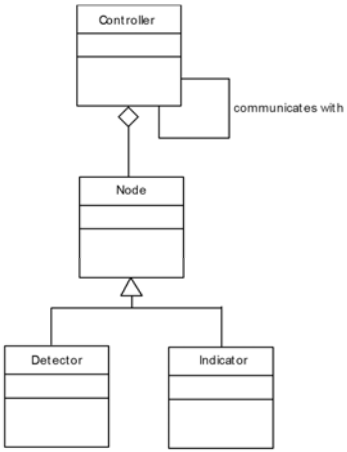
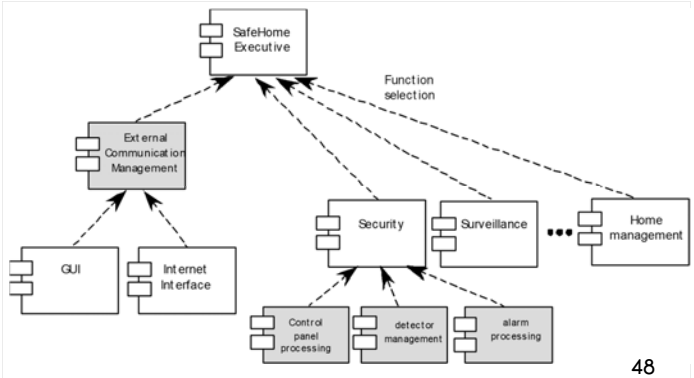


Figure 10.7 UML relationships for SafeHome security function archetypes (adapted from [BOS00])

Component Structure

- โครงสร้างสถาปัตยกรรมโดยรวม



Refined Component Structure

- ขยายความส่วนต่างๆ ของระบบ