



Tamkang University Software Engineering Group
淡江軟體工程實驗室
<http://www.tkse.tku.edu.tw/>

物件導向系統分析與設計

教材：Introduction to Object-Oriented Analysis and Design

Stephen R. Schach 附ArgoUML光碟 滄海書局代理

Reporter : Ying-Hong Wang
E-mail : inhon@mail.tku.edu.tw



Tamkang University Software Engineering Group 淡江軟體工程實驗室 <http://www.tkse.tku.edu.tw/>

上課用書、參考用書暨相關規定

• 成績評定

- 出席: 15% 作業: 30% 期中報告: 15% 期末: 40%
- 點名成績係用以區隔同學每週出席與否的成績，故不接受各種形式未出席的原因，但每人可以有三次的緩衝機會，以因應不可抗拒的情況。缺席超過三次者，才開始扣分，反之，缺席少於三次者，可以獲得加分。
- 每週點名一次，點名超過一次者，即為加分點名，不在前述100%內累計。正規點名可以接受補點，加分點名則不接受補點
- 除期中與期末報告外，其餘作業可接受補交，自繳交截止時間點起，每24小時內補交者扣10分，扣至0分為止
- 請助教作業每次發還後隨即公佈登錄資料以供核對，有疑義者須於一週內完成補正，逾期不再受理

• 上課方式

- 投影片 + 問題案例研討、部分教材內容選自「軟體工程聯盟」開發之教材
- 校外專家演講

• 上課規定

- 手機請改設震動或關機、不要私下講話
- 鼓勵提問

2007年Cheers 雜誌就業能力指標

- 2007年Cheers雜誌對於台灣前1000大企業的畢業生調查就業能力評估指標
 - 專業知識與技術
 - 具有創新能力
 - 具有國際觀與外語能力
 - 具有解決問題能力
 - 具有融會貫通能力
 - 團隊合作
 - 穩定度與抗壓性
 - 學習意願與可塑性

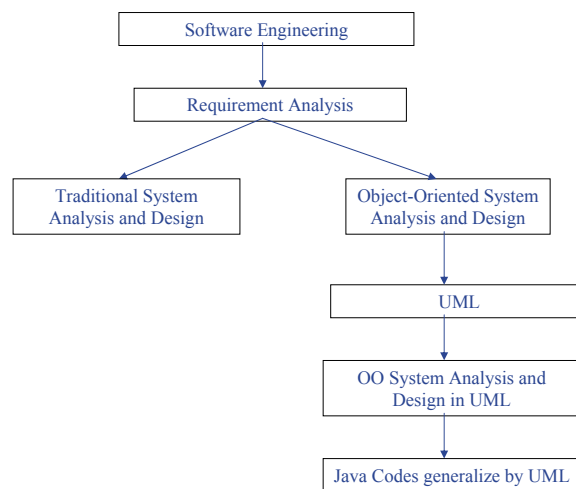
調整上課心態與學習態度

- 以前修課的想法可能是求過關就好
- 以前的學習態度可能是可以順利畢業就好
- 現在妳(你)應該要為即將就業作準備
- 現在的妳(你)應該要知道為什麼要學、要學什麼
- 現在的妳(你)應該要開始思考如何面對競爭
- 現在的妳(你)應該要準備如何打贏一場又一場的競賽
- 現在的妳(你)應該思考如何成為企業需要的人才

課程綱要

1. Introduction
2. Recall Software Engineering
3. System Analysis and Design
4. Object-Oriented Concepts
5. UML
6. Workflow of Requirement Analysis
7. Workflow of Object-Oriented System Analysis and Design

Learning Objects Architecture



Agenda

1. Introduction

2. Recall Software Engineering
3. System Analysis and Design
4. Object-Oriented Concepts
5. UML
6. Workflow of Requirement Analysis
7. Workflow of Object-Oriented System Analysis and Design

Introduction

- Definition of System:

- A **system** is a set of *artifacts* (components) that together achieve some outcomes

- Definition of Information System:

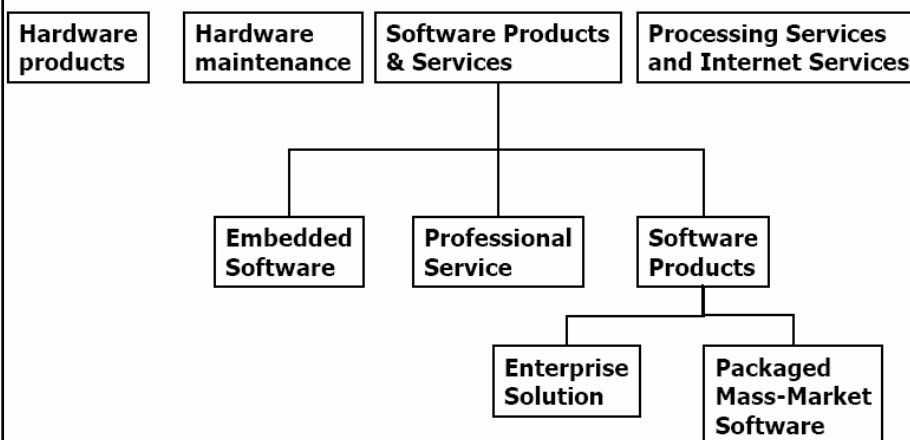
- An **information system** is a system that achieves a business outcome
- An **information system** collects, manipulates, stores, and reports information regarding the business activities of an organization, in order to assist the management of that organization in managing the operations of the business

Introduction

- Categories of Information Systems:
 - Custom Information Systems
 - Commercial off-the-shelf (COTS) packages
- Three main Stakeholders of a custom information system
 - The **Client**, who is paying for the information system to be developed
 - The **Future Users**, who are the users of the developing information system
 - The **Developers**, who are the team workers to develop the specific information system

Introduction

• IT Market



Introduction

- **Software Products and Services**

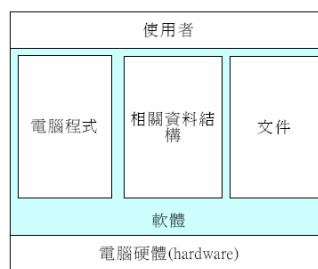
Professional Software Services	Enterprise Solutions	Packaged Mars-Market Software
Anderson Consulting	IBM	Microsoft
IBM	Oracle	IBM
EDS	Computer Associates	Computer Associates
CSC	SAP	Adobe
Science Applications	HP	Novell
Cap Gemini	Fujitsu	Symantec
Hp	Hitachi	Intuit
DEC	Parametric Technology	Autodesk
Fujitsu	People Soft	Apple
BSO Origin	Siemens	The Learning Company

Agenda

1. Introduction
- 2. Recall Software Engineering**
3. System Analysis and Design
4. Object-Oriented Concepts
5. UML
6. Workflow of Requirement Analysis
7. Workflow of Object-Oriented System Analysis and Design

Recall Software Engineering

- 何謂軟體？軟體的定義是什麼？
 - 控制電腦系統作業的詳細指令集
 - 在執行時可以提供所需功能及效能的指令(電腦程式)
 - 使程式適當地處理資訊的資料結構
 - 描述程式操作及使用的文件

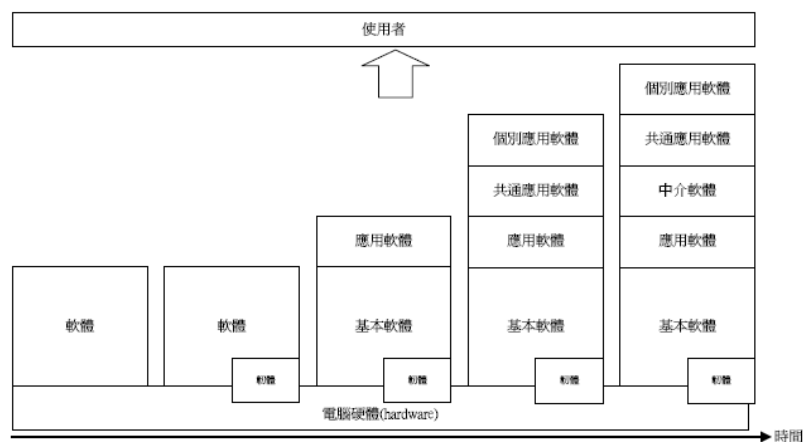


<inhon@mail.tku.edu.tw>

May 25, 2010 P 13

Recall Software Engineering

- 軟體的架構



<inhon@mail.tku.edu.tw>

May 25, 2010 P 14

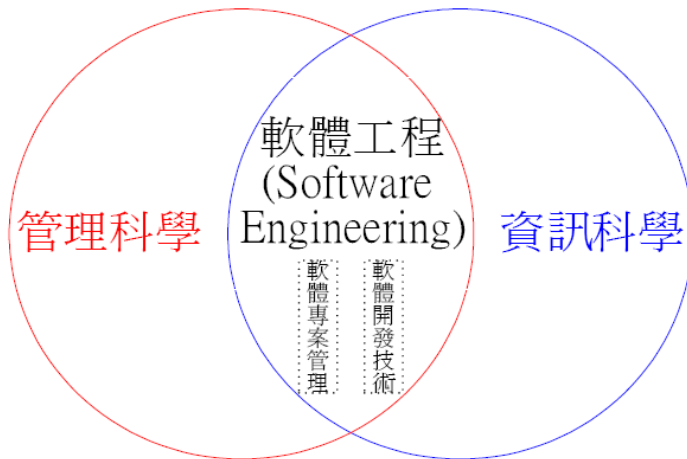
Recall Software Engineering

- 軟體工程(Software Engineering)
 - 第一位提出軟體工程(Software Engineering)專有名詞之學者為 Fritz Bauer
 - 軟體工程包含一個過程、管理、技術方法、和工具
 - 工程是經由運用科學知識並透過系統化的應用，來創造和建立具有經濟效益的方法，以便來解決人類的問題。而
 - **軟體工程**是工程的一部份，軟體工程運用電腦科學和數學的原理來針對軟體問題並提出低成本高效益的解決方法
 - 一門專業學門，針對在預算內生產並準時交付無錯誤且滿足客戶需求之軟體

Recall Software Engineering

- 軟體工程與其他種類之工程有何不同之處
 - 其最大不同在軟體工程經常處於不斷變動，即使軟體已開發至中途，也依然會有可能180度大改變。也因處於不斷變動，造成軟體開發時常變更設計進而造成開發成本提高、時間延誤、或(及)品質低落
- 軟體工程的定義
 - 軟體工程是一門以研究如何以系統化、規範化、和量化的工程原則和方法來進行俱經濟效益之軟體開發和維護的學科
- 軟體工程的內容
 - 軟體開發技術
 - 軟體專案管理

Recall Software Engineering



Recall Software Engineering

• What is the Problem?

- 84 % of all software projects do not finish on time and within budget (Survey conducted by Standish Group)
 - 8000 projects in US in 1995
 - More than 30 % of all projects were cancelled
 - 189 % over budget
- Key issues
 - Software firms are always pressured to perform under unrealistic deadlines.
 - The clients ask for new features, just before the end of the project, and unclear requirements.
 - Software itself is awfully complex.
 - Uncertainties throughout the development project.

Recall Software Engineering

- **Real Case**

- Trust business. 1982.
 - Spend 18 months in deep research & analysis of the target system.
 - Original budget: 20 million.
 - Original Schedule: 9 months, due on 1984/12/31.
 - Not until March-1987, and spent 60 million.
 - Lost 600 millions business
- Eventually, gave up the software system and 34 billion trust accounts transferred.

Recall Software Engineering

- **Software Myths**

- Management Myths
- Customer Myths
- Practitioner's Myths

Recall Software Engineering

• Management Myths

- We already have a book that's full of standards and procedures for building software. Won't that provide my people with everything they need to know?
- My people do have state-of-the-art software development tools; after all, we buy them the newest computers
- If we get behind schedule, we can add more programmers and catch up

Recall Software Engineering

• Customer Myths

- A general statement of objectives is sufficient to begin writing programs –we can fill in the details later
- Project requirements continually change, but change can be easily accommodated because software is flexible

Recall Software Engineering

• Practitioner's Myths

- Once we write the program and get it to work, our job is done
- Until I get the program “running” I really have no way of assessing its quality
- The only deliverable for a successful project is the working program

Recall Software Engineering

• Traditional Software Engineering Paradigm (傳統軟體工程典範)

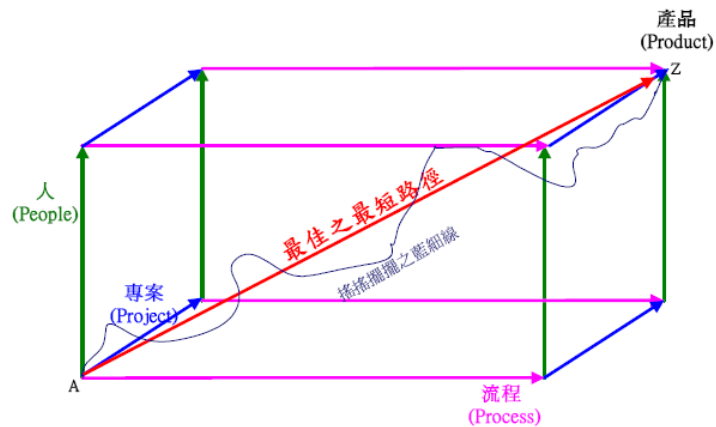
- 軟體工程在傳統上都是以電腦科學中的程式撰寫、編譯及連結的角度來切入，也因此傳統軟體工程亦被稱為結構化軟體發展 (*Structured Software Development*)

• 軟體工程的4 個構面

- 人(People)
- 流程 (Process)
- 專案(Project)
- 產品(Product)
- 這就是軟體工程中俗稱的4P

Recall Software Engineering

- 軟體工程的4個構面



<inhon@mail.tku.edu.tw>

May 25, 2010

P 25

Recall Software Engineering

- 軟體發展技術包括
 - 軟體發展方法學
 - 軟體工具
 - 軟體工程環境
- 軟體專案管理包括
 - 軟體度量
 - 專案估算
 - 時間控制管理
 - 人員組織管理
 - 配置管理
 - 專案計劃

<inhon@mail.tku.edu.tw>

May 25, 2010

P 26

Recall Software Engineering

- 美國專案管理協會(Project Management Institution, PMI)
對其專案管理劃分九大領域



Recall Software Engineering

- SEI (軟體工程協會, Software Engineer Institute) 對於
軟體開發流程的劃分
 - 客戶需求
 - 軟體規格
 - 計畫
 - 設計
 - 實施
 - 整合
 - 維護
 - 淘汰

Recall Software Engineering

- 軟體開發之生命週期 (Software Development Life Cycle, SDLC)
 - 也稱為系統開發之生命週期(System Development Life Cycle) , 或
 - 系統生命週期(System Life Cycle, SLC)

Recall Software Engineering

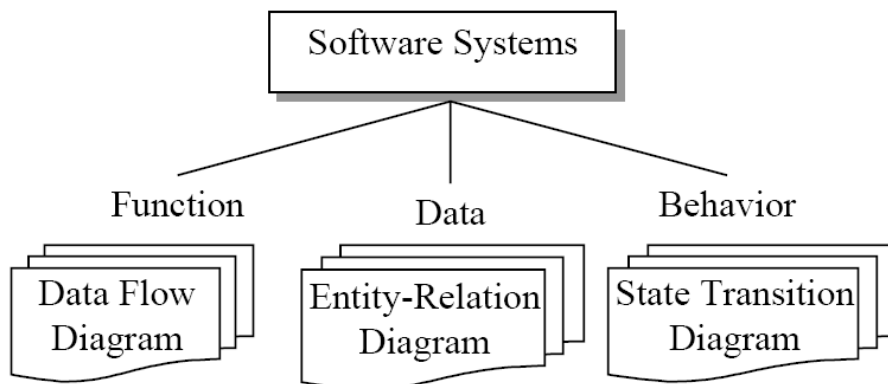
- 常見之軟體開發生命週期
 - 邊建邊修模式 (Build-and-Fix 或 Code-and-Fix)
 - 瀑布式模式 (Waterfall) (又稱傳統軟體生命週期)
 - 快速雛型式模式 (Rapid Prototyping)
 - 整合瀑布式與快速雛形 (Integrated Waterfall and Rapid Prototyping)
 - 漸增式模式 (Incremental)
 - 極致編程(Extreme programming)、敏捷方法論(Agile Method)
 - Synchronize-and-Stabilize
 - 螺旋式模式 (Spiral)

Recall Software Engineering

- 軟體開發生命週期在軟體開發流程上提供一個整體的架構 (Framework)
- 軟體開發人員雖然對於軟體開發流程有所瞭解，但是依然在觀念及作法上須其他的協助
- 此一協助稱為方法論(Methodology)，是一套特定規則、步驟、及流程來促進軟體開發團隊所有成員依循其規則、步驟、及流程，以便完成高效率之軟體開發活動及工作
- 每一種方法論包括
 - 模型(Model)
 - 工具(Tool)
 - 技巧(Technique)

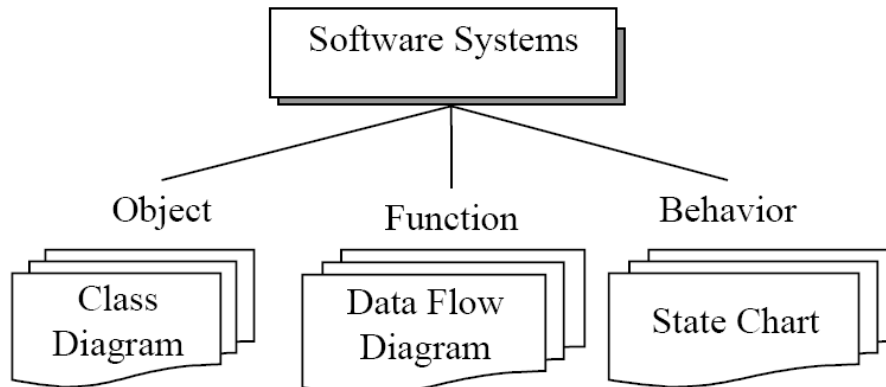
Recall Software Engineering

- **Traditional Software Engineering**



Recall Software Engineering

- **Object-Oriented Software Engineering**



<inhon@mail.tku.edu.tw>

May 25, 2010 P 33

Recall Software Engineering

- **Generic View**

- Definition: What
- Development: How
- Maintenance: Change

<inhon@mail.tku.edu.tw>

May 25, 2010 P 34

Recall Software Engineering

• Traditional Software Development Life Cycle

- Requirement acquisition (problem statements)
 - To describe the problem to be solved and providing a conceptual overview of the proposed system
- Requirement analysis
- Requirement specification
- System analysis
- System design, Detail design
- Coding
- Testing
- Maintenance

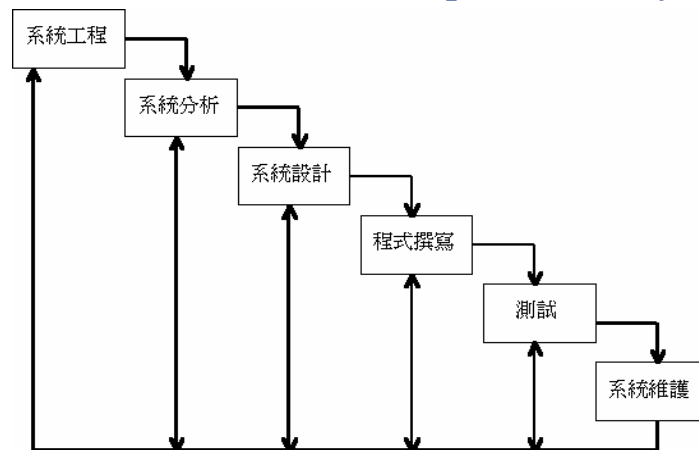
Recall Software Engineering

• Traditional Software Development Life Cycle

- Requirement Analysis Phase
 - Requirement acquisition (problem statements)
 - Requirement analysis
 - Requirement specification
 - System analysis
- Design Phase
 - System Design
 - Detail design
- Implement Phase
 - Coding
 - Testing
- Maintenance Phase

Recall Software Engineering

• Traditional Software Development Life Cycle



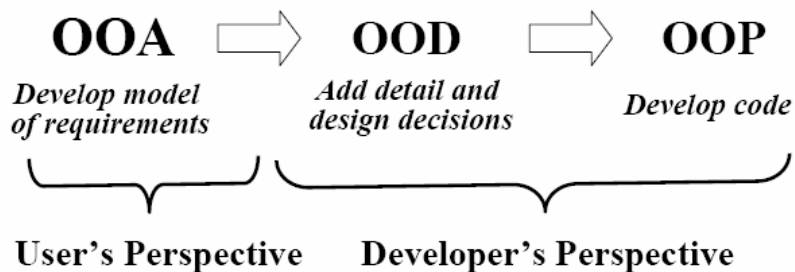
<inhon@mail.tku.edu.tw>

May 25, 2010

P 37

Recall Software Engineering

• Object-Oriented Software Development Life Cycle



<inhon@mail.tku.edu.tw>

May 25, 2010

P 38

Recall Software Engineering

• CMMI: Capability Maturity Model Integration

– Fundamental Concepts

- Software process *capability* describes the range of expected results that can be achieved by following a software process.
 - Predicting outcomes for the next project.
- Software process *performance* represents the actual results achieved by following a software process.
 - Focusing on the result achieved.
- Software process *maturity* is the degree to which a specific process is explicitly defined, managed, controlled and effective.

Recall Software Engineering

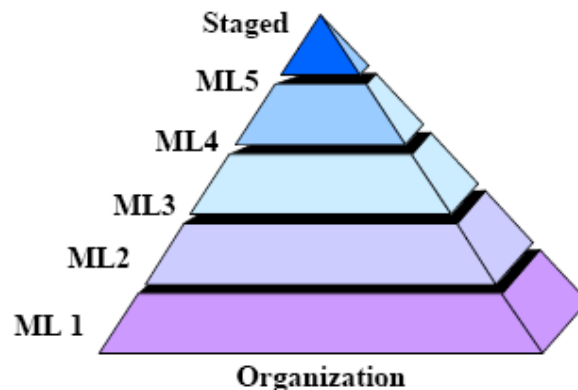
• CMMI: Capability Maturity Model Integration

– Fundamental Concepts

- *Infrastructure* is the underlying framework of an organization.
- *Culture* is the way doing thing.
- *Institutionalization* is the building of infrastructure and culture to support the methods, practices and procedures.
- Therefore, a mature software organization needs an infrastructure and culture to support its methods, practices and procedures so that they endure forever.

Recall Software Engineering

• CMMI: Capability Maturity Model Integration



<inhon@mail.tku.edu.tw>

May 25, 2010 P 41

Recall Software Engineering

• CMMI: Capability Maturity Model Integration

Level	Focus	Staged Organization of PAs
5 Optimizing	<i>Continuous Process Improvement</i>	Organizational Innovation and Deployment (OID) Causal Analysis and Resolution (CAR)
4 Quantitatively Managed	<i>Quantitative Management</i>	Organizational Process Performance (OPP) Quantitative Project Management (QPM)
3 Defined	<i>Process Standardization</i>	Requirements Development (RD) Technical Solution (TS) Product Integration (PI) Verification (VER) Validation (VAL) Organizational Process Focus (OPF) Organizational Process Definition (OPD) Organizational Training (OT) Integrated Project Management (IPM) Risk Management (RSKM) Integrated Teaming (IT) Integrated Supplier Management (ISM) Decision Analysis and Resolution (DAR) Organizational Environment for Integration (OEI)
2 Managed	<i>Basic Project Management</i>	Requirements Management (REQM) Project Planning (PP) Project Monitoring and Control (PMC) Supplier Agreement Management (SAM) Measurement and Analysis (MA) Process and Product Quality Assurance (PPQA) Configuration Management (CM)
1 Initial		

PA: Process Area

Agenda

1. Introduction
2. Recall Software Engineering
- 3. System Analysis and Design**
4. Object-Oriented Concepts
5. UML
6. Workflow of Requirement Analysis
7. Workflow of Object-Oriented System Analysis and Design

System Analysis and Design

● 需求擷取與分析

- 所謂使用者需求係指使用者期待系統解決的問題與希望從系統獲得之資訊。
- 使用者需求是資訊系統開發最關鍵、最重要且最容易發生錯誤的部分，亦是資訊系統失敗的主因之一。
- 需求分析是資訊系統發展過程中之一項非常重要的活動，亦是一個重要之階段。理論上，該階段之主要工作是應用已通過驗證之原理(Principles)、技術(Techniques)、語言(Languages)與工具(Tools)，以幫助分析師瞭解問題或描述新系統之外部行為。

System Analysis and Design

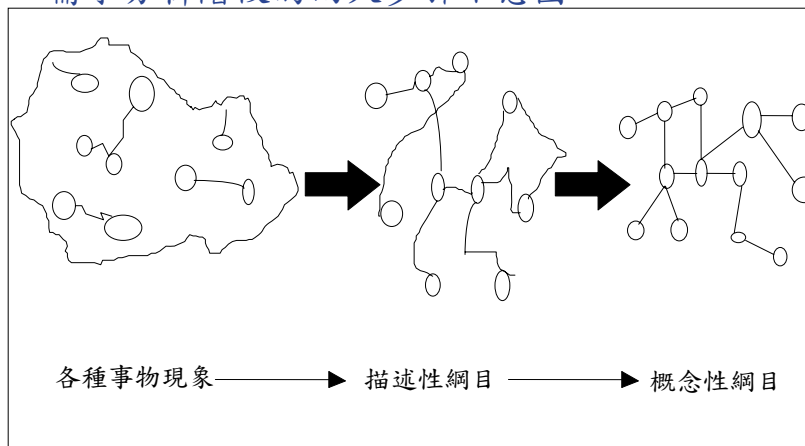
- 需求分析階段主要包括三個活動：
 - 需求判斷：強調如何判斷真正的需求及需求的正確性。
 - 需求分析：則重視在分析已有的需求下所產生的不一致、不完整、矛盾等問題。
 - 需求溝通：重視以最佳的方式組織及描述需求，以促使需求容易令人瞭解並經由相互溝通，達到需求確認的目的。

System Analysis and Design

- 需求分析階段可分為兩大步驟：
 - 需求擷取：主要是對系統範圍內之各種事物及相關現象加以瞭解、判斷和選擇，並設計成描述性綱目。
 - 需求轉換：主要將描述性綱目以系統模式語法轉換成概念性綱目。

System Analysis and Design

- 需求分析階段的兩大步驟示意圖：



<inhon@mail.tku.edu.tw>

May 25, 2010 P 47

System Analysis and Design

- 使用者需求可分為：

- **巨觀需求**：包括欲電腦化之環境、作業程序與範圍、輸出與輸入所需之資訊或表單及系統目標、限制、主要功能等，這些需求應盡可能的在需求分析階段中釐清與確定。
- **細部需求**：指的是電腦化之微觀範圍，可包括使用者介面之要求、例外狀況之處理與錯誤及輔助訊息之顯示等要求，這些需求通常需到設計階段才較容易處理，在此之前這些細部需求不易被掌握。

<inhon@mail.tku.edu.tw>

May 25, 2010 P 48

System Analysis and Design

- 需求分析階段之重要工作包括：
 - 瞭解現有問題；
 - 瞭解新系統目標；
 - 瞭解新系統之限制；
 - 瞭解使用者巨觀之需求等。

System Analysis and Design

- 需求擷取方式
 - 在擷取使用者需求之前，必須瞭解系統之潛在使用者及可能之人機互動。
 - 常用的需求擷取方式有查閱文件、觀察、問卷、訪談、開會討論與聯合開發等六種，這些方式可單獨應用亦可混合使用。

System Analysis and Design

- 需求擷取方式

- 查閱文件：

- 研究企業的內部工作說明書、企業表單(Business Forms)與手冊是瞭解企業運作邏輯之初步工作。
 - 一般來說，組織中很少有完整的文件詳細地描述出系統之全貌，加上系統可能已經過多次的修改，文件往往未能配合更新，因此以該方式蒐集之資訊常有過時之慮。

System Analysis and Design

- 需求擷取方式

- 觀察：

- 一般來說，**實地觀察**所獲得的資料正確性會比查閱文件為高，亦能驗證所蒐集資料之正確性及補充不完整的資料，透過觀察可以獲得第一手的資料。
 - 僅用觀察仍無法完整的反映出組織的真實情況與需求，例如被觀察者行為可能改變。
 - 選擇正常與例外情況之時機或對象來作觀察，可獲得更多的資料。

System Analysis and Design

- 需求擷取方式

- 問卷：

- 當潛在使用者太多或分布太廣時，可考慮以問卷之方式擷取需求。
 - 一般來說，問卷調查適合於大型企業或公眾資訊系統的設計，因為它所涉及的作業範圍或對象太廣，系統分析師無法逐一親自調查，故利用問卷方式來蒐集使用者需求較為可行。
 - 設計一份好的問卷需有相當的練習與經驗，因為問卷上的問題是以文字靜態的表達，故問題之語意與邏輯必須很清楚且有條理。

System Analysis and Design

- 需求擷取方式

- 問卷：

- 問卷設計時也可以用各種不同的方法來問同一個問題，以觀察各種可能的答案。
 - 正式問卷調查前需有先導測試。
 - 經由先導測試之檢討與回饋可進一步修飾問卷，及早發現問卷可能之問題，對提升問卷之效能有很大之幫助。

System Analysis and Design

- 需求擷取方式

- 問卷：

- 正式問卷調查時，必須決定那些族群是調查的對象。對象選取之抽樣方法主要分成：

- 機率抽樣(Probability Sampling)。
 - 非機率抽樣(Nonprobability Sampling)。

- 。

System Analysis and Design

- 需求擷取方式

- 訪談：

- 訪談是系統分析最有效且最普遍的資料蒐集方法，訪談時分析師親自與使用部門的主管或相關作業人員面對面討論實際作業的情況、報表和資訊需求等。
 - 在訪談期間，系統分析師蒐集到的可能是事實、選擇或推測，並且可觀察到人們的肢體語言、情緒和他們對於現行系統之觀感等。

System Analysis and Design

- 需求擷取方式

- 訪談：

- 訪談可分成兩種方式：

- 開放式訪談(Open Interview)

- » 分析師事先不預定表格、問卷或固定的標準程序，訪談過程全由使用者自由談論其工作。
 - » 適用於分析師對問題領域不熟悉或無法預期之情況。

- 結構化訪談(Structured Interview)

- » 結構化訪談又稱為標準化訪談或導向式訪談，其訪談過程近似於詢問(Interrogation)而非交談(Conversation)，所要求資訊的深度、專門程度亦較深。

System Analysis and Design

- 需求擷取方式

- 訪談：

- 結構化訪談(Structured Interview)

- 這種方式的特點是把問題標準化，然後由受訪者回答或選擇。所有的受訪者都回答同一形式的問題，其作法如下：
 - 每討論一個主題時，先由分析師依其現有瞭解的知識，提出簡短敘述。
 - 使用者針對主題作深度分析，但分析師應在適當知識深度時加以中止。
 - 對某個主題有初步瞭解，就可以進行另一主題的訪談。

System Analysis and Design

- 需求擷取方式

- 開會討論：

- 開會討論是一種很有效率的資料蒐集方式。使用者代表與系統開發人員聚集一堂，將所知道的事實、觀念說出，讓所有與會人員一起相互溝通意見。
 - 此方法的優點是較易獲得正確的資料，即使有不同的意見與觀念，經由眾人研究亦能加以修正。此外，亦可發揮腦力的效果。
 - 缺點是安排溝通與協調較費時。

System Analysis and Design

- 需求擷取方式

- 聯合開發：

- 聯合開發(Joint Application Development, JAD)主要之精神，是透過一個二至五天的集會，讓開發者與顧客能夠快速有效，而且深入的檢討需求並取得共識。
 - 聯合開發的具體結果是產生完整的需求文件。

System Analysis and Design

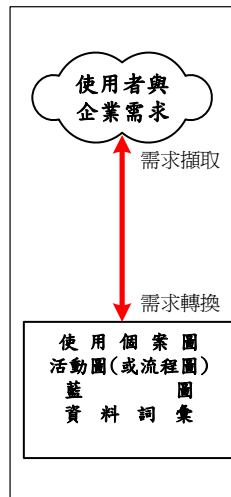
- 需求擷取方式
 - 聯合開發：
 - JAD依下列五個步驟來進行：
 - 範圍界定。
 - 關鍵人員的熟悉。
 - 會議（事前）準備。
 - 會議進行。
 - 文件產生。

System Analysis and Design

- 需求表達工具
 - 企業程序是企業處理有組織的結合，其結果需能完成某企業功能，例如行銷或生產管理等。
 - 結構化系統分析與設計主要是考量企業流程塑模與資料塑模。

System Analysis and Design

- 需求表達工具
 - 需求塑模



System Analysis and Design

- 需求表達工具
 - 流程塑模與資料塑模的工具包括：
 - 流程圖(Flow Chart)
 - 流程圖主要表達實體(Entity)與作業程序及所需資訊間之關係。
 - 處理描述(Process Description)
 - 處理描述則表示流程圖中之作業處理、程序與規則及其相關之資訊輸入與輸出等。

System Analysis and Design

• 需求表達工具

– 流程塑模與資料塑模的工具包括：

• 藍圖(Drawing)

– 藍圖主要表示資訊之展示格式與內容等，例如表單之格線與縱橫項目。


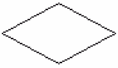


• 資料詞彙(Data Glossary)

– 資料詞彙主要描述藍圖內資訊之詳細內容與規則等。

System Analysis and Design

• 需求表達工具

– 流程圖之主要符號：

符號	意義
	表示作業處理
	表示流程控制
	表示資訊之展示與儲存
	表示作業處理或資訊進行方向

System Analysis and Design

• 需求表達工具

- 處理描述：

- 處理描述進一步的表達每個處理之執行步驟、法則、控制等，並說明其資料之輸入與輸出內容與所涉及之實體等。
- 每個處理描述之名稱應與流程圖中之處理同名，處理描述之表達形式可如下頁表

System Analysis and Design

• 需求表達工具

- 處理描述：

處理名稱	
執行程序與規則	1. 2. ...
資料輸入／來源	
資料輸出／目的地	
限制與備註	

System Analysis and Design

- 需求表達工具

- 藍圖：

- 藍圖(Drawing)主要用予表達流程圖中有關之表單、介面等各項資訊需求之名稱、展示位置、格線、圖表與說明等，這些資訊常無法在流程圖上具體的表達，因此須另以藍圖進一步的表示。

System Analysis and Design

- 需求表達工具

- 資料詞彙：

- 資料詞彙(Data Glossary)進一步說明藍圖所無法表達之內容如資訊之長度、型態、格式、公式、規則、範圍與限制等，並分別舉例說明之。
 - 資料詞彙之內容項目除了藍圖中之欄位編號與名稱，欄位資料之長度與形態，資料是否唯一，資料產生之規則／格式／範圍／公式等資訊均有助於系統分析與設計，因此均可列入資料詞彙中，其形式可表達如下頁表

System Analysis and Design

• 需求表達工具

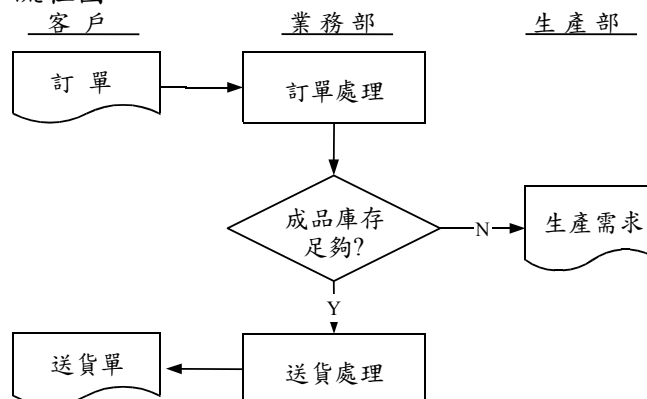
- 資料詞彙：

編號	欄位名稱	長度／形態	鍵	規則／格式／範圍／公式	範例
...

System Analysis and Design

• 範例

- 流程圖：



System Analysis and Design

• 範例

- 訂單處理描述：

處理名稱	訂單處理
執行程序與規則	1.業務部收到客戶訂單之後，需作客戶資料登錄與檢查。 2.業務部檢查訂貨之成品庫存，若成品庫存充足，則進行送貨處理；若成品庫存不足，則通知生產部進行生產計畫之規劃。
資料輸入／來源	訂單／客戶
資料輸出／目的地	送貨訊息／業務部或生產需求／生產部
限制與備註	

System Analysis and Design

• 範例

- 訂單藍圖：

夢幻企業股份有限公司

訂 單

客戶：	A	編號：	D
地址：	B	日期：	E
電話：	C		

貨品編號	品名	顏色	規格	尺寸	數量	單位	單價	金額
F	G	H	I	J	K	L	M	N
10000006	太空梭模型	綠	25kg	S	3	個	417.60	1,252.80
10000005	鐵釘	紅	25kg	L	1	支	200.00	200.00
10000006	太空梭模型	綠	25kg	S	1	個	200.00	200.00
10000002	坐墊	綠	50kg	S	1	粒	6,000.00	6,000.00
10000003	方向盤	黑	50kg	M	2	個	600.00	1,200.00
10000004	鐵蛋	紅	100kg	M	1	粒	200.00	200.00
客戶簽章：	總金額：						9,052.00	

註：套色區域表示須套印表單之部分。

System Analysis and Design

• 範例

- 訂單資料詞彙：

編號	欄位名稱	長度／ 形態	鍵	規則／格式／ 範圍／公式	範 例
A	客戶名稱	20C			王大明
B	地址	40C			高雄市鼓山區蓮海路70號
C	電話	10C			07-5252000
D	編號	8N	V	年+月+日+流水號 YYMMDD99	98090101
E	送貨日期	6D		日期	87年9月1日
F	貨品編號	8C		99999999	10000003
G	品名	10C			方向盤
H	顏色	5C			黑
I	規格	14C			50Kg
J	尺寸	4C			M
K	數量	10N			2
L	單位	4C			個
M	單價	10N		99,999,999.99	600.00
N	金額	10N		數量 × 單價： 9,999,999,999	1,200
O	總計	10N		999,999,999	9,052

註：C表字串；N表數字；D表日期。

<inhon@mail.tku.edu.tw>

May 25, 2010 P 75

System Analysis and Design

• 範例

- 需求分析文件：

需求分析文件樣板	
一、問題描述	
二、新系統目標	
三、新系統限制	
四、使用者需求	
(一) 流程圖 1	
處理描述 1-1	
.....	
處理描述 1-n	
.....	
藍圖 1-1	
資料詞彙 1-1	
.....	
藍圖 1-m	
資料詞彙 1-m	
(二) 流程圖 2	
.....	
.....	

<inhon@mail.tku.edu.tw>

May 25, 2010 P 76

System Analysis and Design

- 結構化技術

- 結構化技術起源於1960年代末期，主要強調如何應用一些概念、策略與工具，以提升系統分析與設計、程式設計與測試之效率與效能等。
- 結構化技術一般包括：
 - 結構化分析(Structured Analysis)。
 - 結構化設計(Structured Design)。
 - 結構化程式設計(Structured Programming)。
 - 由上而下發展(Top-down Development)。

System Analysis and Design

- 結構化分析與設計

- 結構化設計起源於1960年代末期(Yourdon, 1988; Lewis and Oman, 1990)，其主要目的是將資訊系統依由上而下發展，並將程式設計模組化與結構化。

- •

System Analysis and Design

• 結構化分析與設計

- 程式的模組(Module)是一連串指令的集合，一般來說，模組包括五個部分：
 - 模組名稱，名稱唯一且應有意義，應能表達模組的功能。
 - 輸入，是指模組被呼叫時，呼叫模組所需輸入的資料
 - 輸出，是指模組執行後所產生的輸出結果，它與輸入合稱為模組的介面
 - 處理邏輯，為達成模組功能，模組內必須具備的執行程序與邏輯。
 - 內部資料，該模組獨自擁有而不與其他模組共用的資料。

System Analysis and Design

• 結構化分析與設計

- 結構化設計包括文件工具、設計評估準則與設計經驗法則等。其中，文件包括：
 - 結構圖(Structure Chart)
 - HIPO圖 (Hierarchical Input Process Output)
 - 模組規格描述
 - 資料字典

System Analysis and Design

- 結構化分析與設計

- 結構化設計之實施有一些簡單的經驗法則(Rules of Thumb)可供參考，這些法則很有用，但不保證在所有的個案均可行。Yourdon(1988)認為最普遍的設計經驗法則常關係到：

- 模組大小(Module Size)。
- 控制間距(Span of Control)。
- 影響範圍(Scope of Effect)。
- 控制範圍(Scope of Control)。

System Analysis and Design

- 結構化分析與設計

- 結構化分析之圖形化文件工具包括：

- 事件列(Event List)。
- 環境圖(Context Diagram)。
- 資料流程圖(Data Flow Diagram, DFD)。
- 資料字典(Data Dictionary, DD)。
- 處理規格描述(Process Specification, PS)。
- 實體關係圖(Entity Relationship Diagram, ERD)。

System Analysis and Design

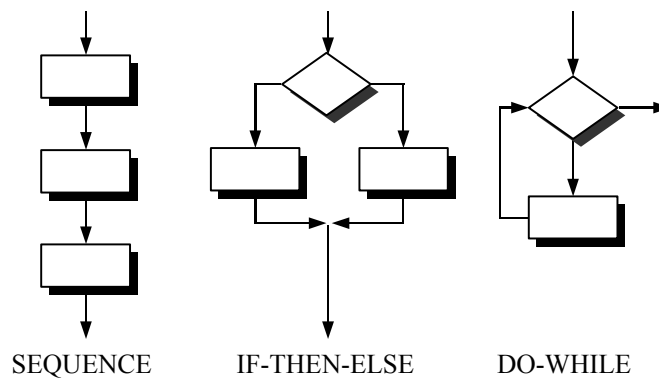
- 結構化分析與設計

- Dijkstra(1969)提出結構化程式設計的概念，以消除以往程式因GOTO而造成如麵條式的混亂狀態。
- 結構化程式的背後理論是任何程式邏輯僅有一輸入與一輸出，且均可由三種「結構」構成：循序、選擇與重複

System Analysis and Design

- 結構化分析與設計

- 結構化程式設計的邏輯概念



System Analysis and Design

- 結構化分析與設計

- 由上而下發展(Top-Down Development Model)

- 由上而下發展之技術起源於1960年代末期，此技術包含有三個相關但不同方面的「由上而下」：

- 由上而下設計(Top-Down Design)。

- 由上而下編碼(Top-Down Coding)。

- 由上而下實施(Top-Down Implementation)。

System Analysis and Design

- 結構化分析與設計

- 由上而下發展(Top-Down Development Model)

- 由上而下設計是一種設計策略，它把大而複雜的問題分解成較小且較不複雜的問題，直到原來的問題已可用一些容易且可理解的小問題組合來代表。例如：

- 先將主程式分為A、B、C三個子程式。

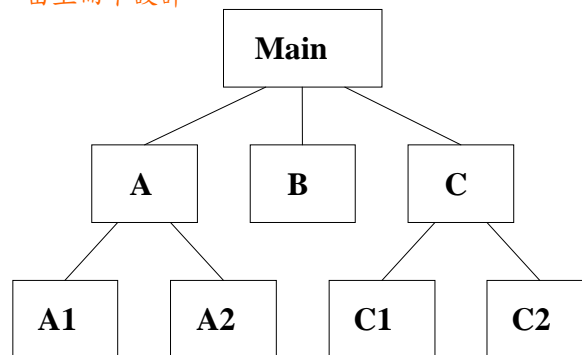
- 再劃分子程式A為A1與A2，子程式C為C1與C2，如圖

System Analysis and Design

- 結構化分析與設計

- 由上而下發展(Top-Down Development Model)

- 由上而下設計



<inhon@mail.tku.edu.tw>

May 25, 2010

P 87

System Analysis and Design

- 結構化分析與設計

- 由上而下發展(Top-Down Development Model)

- 由上而下設計的特色有：

- 一個層級化的設計

- » 先考慮程式的主要功能，再依次考慮其下的各個次要功能。

- 延緩考慮細節問題

- » 先考慮高層次之功能，再考慮其下之次功能。

- 與程式語言無關

- » 使用由上而下設計方式，可選用任何一種語言來撰寫，而不需更改設計內容。

- 便於驗證

- » 可根據需求分析的結果來檢驗程式的主要功能是否劃分完備，也可以根據各主要功能來檢查各次功能。

<inhon@mail.tku.edu.tw>

May 25, 2010

P 88

System Analysis and Design

- 結構化分析與設計

- 由上而下發展(Top-Down Development Model)

- 由上而下編碼：

- 程式編輯的方式很多，可採用由上而下編碼(Top-down Coding)，亦可採由下而上編碼(Button-up Coding)方式，或是採用兩者混合的方式。
 - 由上而下編碼是一種程式編輯策略，當高階模組被設計完成後，就先對高階模組編碼。
 - 一般的程式設計方式多採由上而下設計與由上而下程式編輯，當系統很大時，此種作法有助於縮短規劃時間。

System Analysis and Design

- 結構化分析與設計

- 由上而下發展(Top-Down Development Model)

- 由上而下實施：

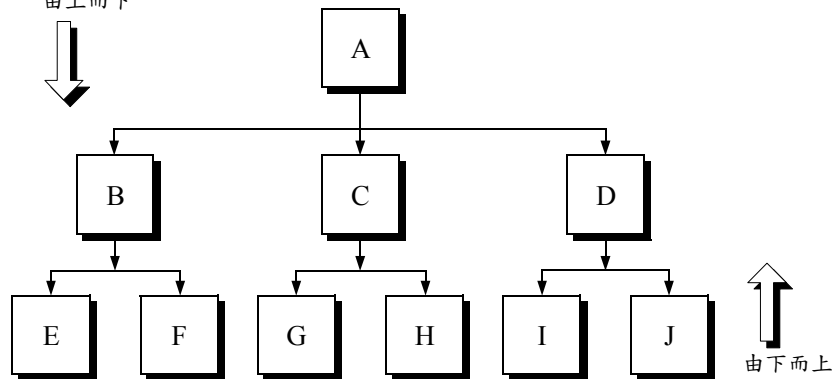
- 由上而下實施又稱由上而下整合測試(Top-Down Integration Test)。由上而下測試是在低階模組尚未完成程式撰寫前，亦可能尚未被設計前，有時甚至在其功能需求尚未被說明之前先測試系統的高階模組；而由下而上測試是由程式結構圖的最底端模組開始，逐次往上測試
 - 如圖示

System Analysis and Design

• 結構化分析與設計

• 由上而下實施：

由上而下



<inhon@mail.tku.edu.tw>

May 25, 2010

P 91

System Analysis and Design

• 結構化分析與設計

- 由上而下發展(Top-Down Development Model)

• 由上而下實施：

- 應用由上而下的整合測試，首先由結構圖上最頂端的模組開始進行測試，而以**殘根模組**(Stub Module)或**虛擬模組**(Dummy Module)暫時代替其下一層未完成的模組。以下圖為例，測試模組 A 時，C與 D是虛擬模組，同樣的，測試B時，E與F是虛擬模組。

<inhon@mail.tku.edu.tw>

May 25, 2010

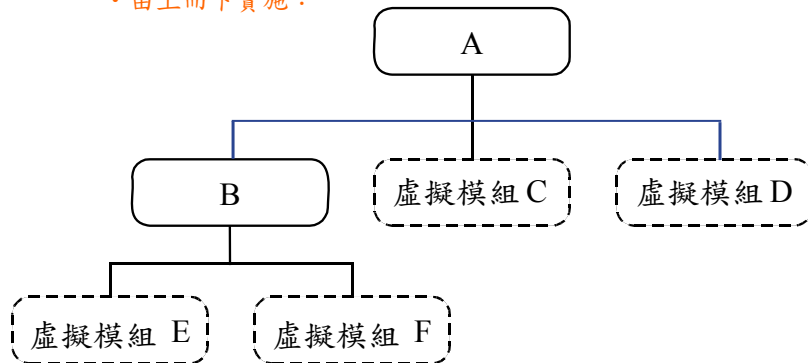
P 92

System Analysis and Design

- 結構化分析與設計

- 由上而下發展(Top-Down Development Model)

- 由上而下實施：



<inhon@mail.tku.edu.tw>

May 25, 2010

P 93

System Analysis and Design

- 結構化分析與設計

- 由上而下發展(Top-Down Development Model)

- 由上而下策略之優點

- 系統的整合測試可以減至最少。
 - 最高階層的介面最先被測試，且被測試的機會最多。
 - 高階層的模組是低階層模組最佳的測試**啟動(Driver)模組**。
 - 系統的錯誤若在上階層，則可及早發現。

<inhon@mail.tku.edu.tw>

May 25, 2010

P 94

System Analysis and Design

• 結構化分析與設計

- 由上而下發展(Top-Down Development Model)

• 由上而下策略之缺點

- 需要製造殘根或虛擬模組。
- 殘根或虛擬模組的設計通常比較複雜。
- 以殘根或虛擬模組執行輸入、輸出功能較困難。
- 測試個案的產生可能會很困難。
- 測試結果較難觀察。
- 易使人產生誤解，認為設計及測試可重疊進行，或認為某些模組的測試可延期完成。
- 低階層次模組，若想做平行測試，將會受制於其上階層模組是否已完成。

System Analysis and Design

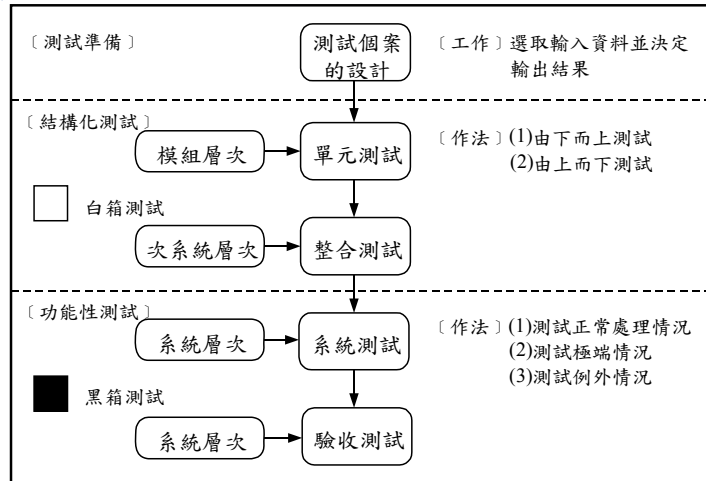
• 結構化分析與設計

- 由上而下發展(Top-Down Development Model)

- 系統測試與驗收測試傾向於使用**黑箱測試**，系統測試主要測試正常處理情況、極端與例外情況等，而驗收測試主要由使用者進行系統功能之測試（參圖4-6）。
- 測試的主要目的是從可執行的程式找出錯誤來。一般來說，測試的種類與進行順序分別是單元測試、整合測試、系統測試與驗收測試等。

System Analysis and Design

• 常用的測試種類與執行順序



<inhon@mail.tku.edu.tw>

May 25, 2010 P 97

System Analysis and Design

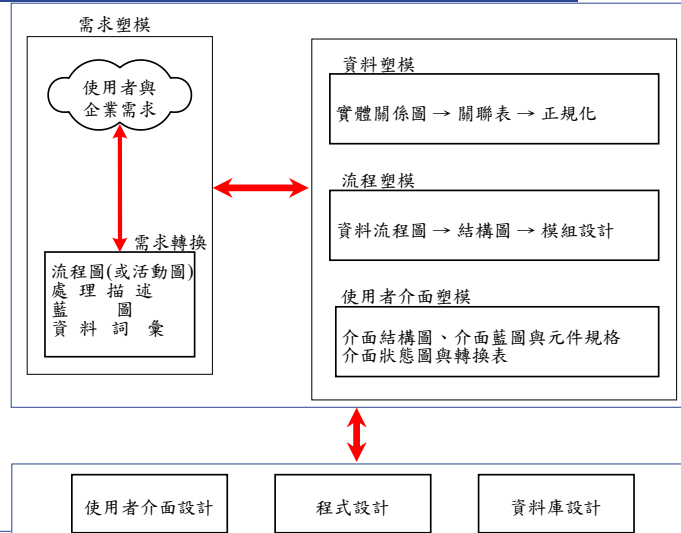
• 結構化分析與設計工具

- 基本上，結構化分析與設計是將資料與流程分開考慮，主要可分成三大部分：資料塑模、流程塑模與使用者介面塑模。其中，資料塑模主要是針對資訊系統的知識子系統，流程塑模主要是針對資訊系統的企業流程，而使用者介面塑模主要是針對使用者介面子系統。此三種塑模活動並沒有一定的進行順序，也就是說任何一種塑模活動均可視需要而先進行或以其他塑模活動交互進行。

<inhon@mail.tku.edu.tw>

May 25, 2010 P 98

結構化分析與設計及塑模工具



<inhon@mail.tku.edu.tw>

May 25, 2010

P 99

System Analysis and Design

• 結構化分析與設計工具

– 結構化分析與設計的文件工具包括：

- 事件。
- 環境圖。
- 資料流程圖。
- 資料字典。
- 結構圖與HIPO圖。
- 處理規格描述。
- 實體關係圖。

<inhon@mail.tku.edu.tw>

May 25, 2010

P 100









System Analysis and Design

• 結構化分析與設計工具

- 資料流程圖：

- 資料流程圖提供一種簡易的、圖形化的方式以表達系統之作業處理與資料流間之關係。
- 資料流程圖有四個基本元素：
 - 外部實體(Entity)。
 - 資料流(Data Flows)。
 - 處理(Process)。
 - 資料儲存(Data Stores)，其元素之表示主要有：
 - » DeMarco & Yourdon 及
 - » Gane & Sarson兩種方式

System Analysis and Design

表達方式 元 素	DeMarco & Your don	Gane & Sarson
外部實體		
資 料 流		
處 理		
資料儲存		

System Analysis and Design

• 結構化分析與設計工具

- 結構圖與HIPO圖：

- **結構圖**(Structure Charts) 與 **HIPO 圖**(Hierarchical Input Process Output)等文件工具的目的是用來表達系統的模組結構(Structure)及系統架構(Architecture)，而非針對程序邏輯(Procedural Logic)。
- 每個模組以矩形表示，矩形內有描述其功能之名稱，該名稱應能精確的反映該模組能做什麼。
- 模組之間以箭頭連結，而模組之呼叫順序是由左至右。及由上而下模組間靠參數傳遞做溝通，參數之型式有**資料耦合**(Data Couples)與**旗標**(Flags 或 Control Flags)。
- 資料耦合顯示資料在兩個模組間傳遞，其符號用空心之小圓圈連接向外之箭頭表示。

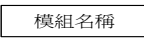
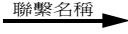
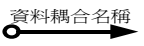
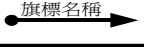
<inhon@mail.tku.edu.tw>

May 25, 2010 P 103

System Analysis and Design

• 結構化分析與設計工具

- 結構圖與HIPO圖：

符 號	意 義
	一個模組是執行某特定工作的一連串敘述，類似DFD的「處理」。例如有的模組包括COBOL敘述、PL/I程序、組合語言、外部副程式及以任任何語言完成的程式等。
	一個聯繫是連接兩個模組的符號，由監督模組（呼叫模組）到附屬模組（被呼叫模組）。因結構圖為階層式的，因此監督模組經常位於附屬模組之上，而聯繫的符號是將箭頭朝下。該符號同時應含當附屬模組停止時，則無條件的回到監督模組。
	一個耦合是一個由某一模組移動至另一模組的資料項。所有的或通過的資料必定以一個耦合出現。一個聯繫可能有數個耦合，每一個耦合必須要被填入資料字典(DD)中。耦合的名稱不需要是唯一，但應該定義清楚。
	一個旗標是一個測試的資料項，其目的是聯繫在某些條件下的訊息，以及其他資料項的聯繫。

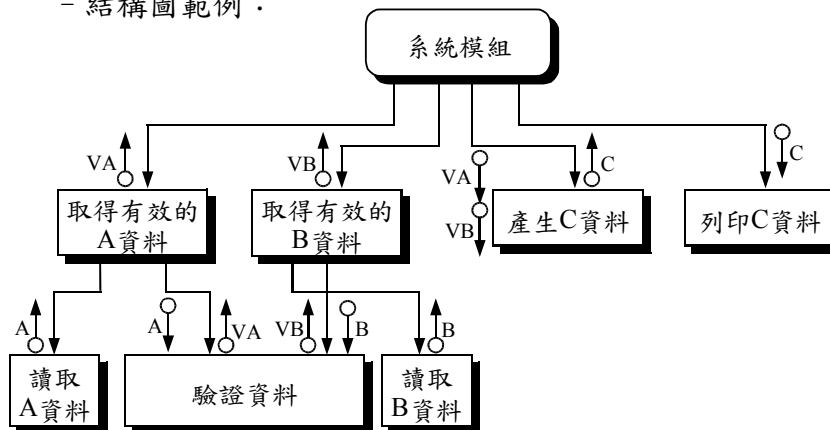
<inhon@mail.tku.edu.tw>

May 25, 2010 P 104

System Analysis and Design

• 結構化分析與設計工具

- 結構圖範例：



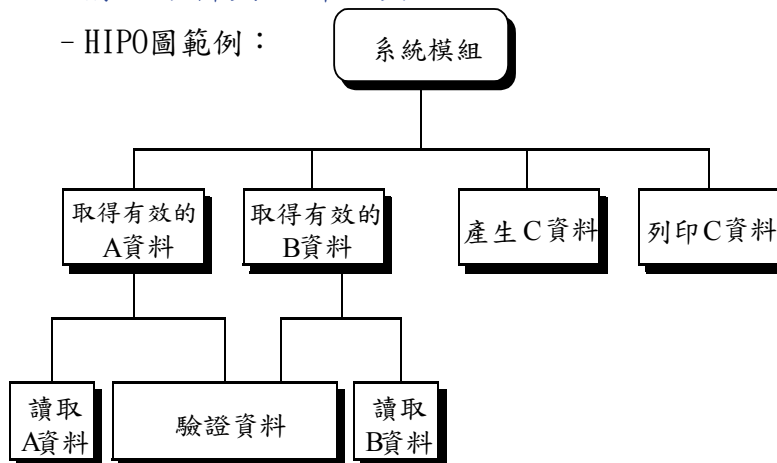
<inhon@mail.tku.edu.tw>

May 25, 2010 P 105

System Analysis and Design

• 結構化分析與設計工具

- HIPO圖範例：



<inhon@mail.tku.edu.tw>

May 25, 2010 P 106

System Analysis and Design

• 結構化分析與設計工具

- 處理規格描述：

- **處理規格描述**(Process Specification)允許分析師在資料流程圖最底層之每個基本處理單元精確的描述其商業規則（例如作業處理邏輯），許多不同的方法可被用於描述處理規格，例如：
 - 流程圖；
 - 法則；
 - 結構化英文(Structured English)；
 - 程式設計語言(Program Design Language, PDL)等。
- 目前結構化英文與程式設計語言較常被用於處理規格描述。

System Analysis and Design

• 塑模(Modeling)

- 結構化之分析與設計將所面對問題之流程與資料分開處理，並分別稱為流程塑模與資料塑模。
- 流程塑模主要是以資料流程圖(Data Flow Diagram, DFD)做為塑模之工具，將企業流程分解成具層級結構之模組。
- 資料塑模則是以實體關係圖(Entity-Relationship Diagram, E-R Diagram 或 ERD)做為塑模之工具，對組織或商業領域的**實體**(Entities)、**關聯**(Associations)及**資料元素**(Data Elements)提供概念性邏輯結構的表示。
- 以下僅介紹流程塑模。ERD各位在資料庫中學過。

System Analysis and Design

• 流程塑模

– 良好的結構化設計有三個特徵：

- 模組間有很好的分割。
- 階層式的系統架構。
- 獨立的模組功能。

– 要達到良好的系統設計與提升模組的品質，需考慮：

- 模組間的耦合力，是指一個系統內部各模組之間的相關程度。
- 模組的內聚力，是指一個模組內部所做事情之相關程度。
- 其他的考慮因素，如功能分割等。

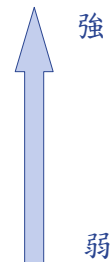
System Analysis and Design

• 流程塑模

– 內聚力(Cohesion)

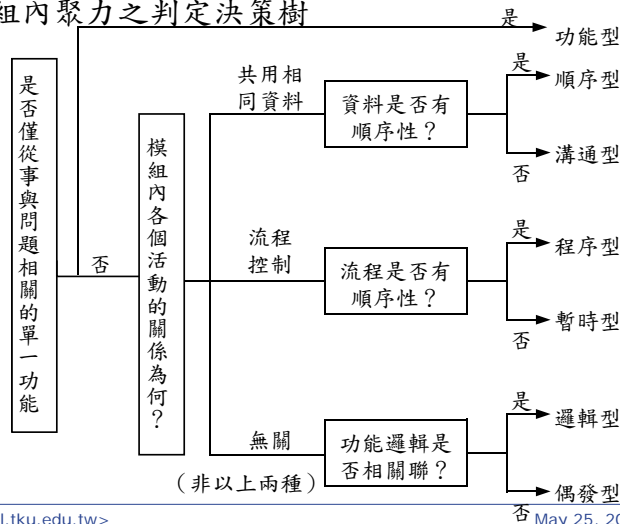
- **內聚力**是一種衡量模組內部之工作相關程度之方法。換句話說，模組的內聚力是衡量模組完成一件單一，且定義清楚之工作的程度。內聚力的種類大概可分為七種：

- 功能內聚力(Functional Cohesion)。
- 順序內聚力(Sequential Cohesion)。
- 溝通內聚力(Communication Cohesion)。
- 暫時內聚力(Temporal Cohesion)。
- 程序內聚力(Procedural Cohesion)。
- 邏輯內聚力(Logical Cohesion)。
- 偶發內聚力(Coincidental Cohesion)。



System Analysis and Design

• 模組內聚力之判定決策樹



<inhon@mail.tku.edu.tw>

May 25, 2010 P 111

System Analysis and Design

• 模組內聚力之評比因素與結果

內聚力種類	耦合力情形	模組撰寫難易	與其他程式之共用性	維護性	瞭解性
功能型	好	好	好	好	好
順序型	好	好	中等	好	好
溝通型	中等	好	差	中等	中等
程序型	變動	中等	差	變動	變動
暫時型	差	中等	很差	中等	中等
邏輯型	很差	很差	很差	很差	差
偶發型	很差	差	很差	很差	很差

<inhon@mail.tku.edu.tw>

May 25, 2010 P 112

System Analysis and Design

- 流程塑模

- 耦合力(Coupling)

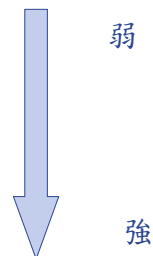
- 耦合力是一種衡量模組間相互關聯強度的方法。
 - 當解決了一模組內的錯誤狀況，而在其他的模組內引起了新的錯誤，這種現象稱為連鎖反應(Ripple Effect)。
 - 解決連鎖反應之可行方法是盡量使一個模組不與其它模組糾結在一起，即讓每個模組盡量的獨立。

System Analysis and Design

- 流程塑模

- 耦合力(Coupling)可分為五類：

- 資料耦合力(Data Coupling)。
 - 資料結構耦合力(Stamp Coupling)。
 - 控制耦合力(Control Coupling)。
 - 共同耦合力(Common Coupling)。
 - 內容耦合力(Content Coupling)。



System Analysis and Design

- 模組間耦合力之評比因素與結果

耦合力 種類	連鎖反應 狀況	修改難度	理解性	與其他系統 之共用性
資料型	變動	好	好	好
資料結構型	變動	中等	中等	中等
控制型	中等	差	差	差
共同型	差	中等	很差	很差
內容型	很差	很差	很差	很差

System Analysis and Design

- 流程塑模

- 耦合力愈弱愈好，內聚力愈強愈好。
- 模組間的耦合力有時可能不只是單純的一種情形，可能存在兩種以上的耦合力，此時這兩模組間的關係以較強的耦合力為準，例如兩個模組具有資料結構耦合力和共同耦合力的關係，則我們應以共同耦合力為準。
- 一般而言，可以接受的內聚力包含功能內聚力、順序內聚力與溝通內聚力，而在耦合力部分則是資料耦合力與資料結構耦合力。

System Analysis and Design

- 資料流程圖轉結構圖與模組設計

- 資料流程圖轉結構圖之步驟有四：

- 設立總裁(President)與副總裁(Vice Presidents)。
 - 設立較低層模組。
 - 模組設計與結構圖修改。
 - 進行評鑑。

System Analysis and Design

- 資料流程圖轉結構圖與模組設計

- 步驟一 設立總裁與副總裁

- 在結構圖中，設立一總裁，而在其下擺多位副總裁。環境圖上之系統可視為總裁，而第零階資料流程圖上之處理視為副總裁，資料流程圖上之資料流變成模組間必要的聯繫。處理聯繫時，暫時先忽略所有錯誤之發生情況、資料庫及其資料流等。

System Analysis and Design

- 資料流程圖轉結構圖與模組設計

- 步驟二 設立較低層模組

- 把第一階及其更低階資料流程圖上之處理，依序懸掛在結構圖上的副總裁底下，例如某第零階之資料流程圖下有更低階之資料流程圖，則須把第一階之處理掛在其第零階處理之下，同樣的，第二階之處理應掛在其所屬第一階處理之下。

System Analysis and Design

- 資料流程圖轉結構圖與模組設計

- 步驟三 模組設計與結構圖修改

- 完成第一版之結構圖後，應先對結構圖中之每一模組進行模組設計，再進一步修改結構圖使之更完美。這些工作包括：
 - 需加入資料流程圖中所沒有的例外狀況處理，出現錯誤時之錯誤訊息處理及操作時可能之輔助訊息處理等。
 - 將結構圖上較弱的地方再分解且加以重新組織。

System Analysis and Design

- 資料流程圖轉結構圖與模組設計

- 步驟四 進行評鑑

- 完成模組設計與結構圖修改後，接下來應確定結構圖的運作功能。也就是該結構圖應能正確的描述系統的行為，以完成流程圖上所描述之企業流程與規則。
 - 進行評鑑之目的是希望能儘早找出錯誤並及早修正，而不希望等到系統完成或在運作時發生錯誤再去修改它。

System Analysis and Design

- 案例

- 說明

- 以夢幻公司之管理資訊系統（以下簡稱夢幻系統）為例，首先應用『需求表達工具』描述夢幻系統之需求，再應用企業『流程塑模的概念與工具』，將需求分析之結果進行企業流程之系統分析與設計。
 - 夢幻公司是一家經營汽機車零件買賣之貿易公司，該公司亦擁有工廠，自行生產部分之零件。本系統之範圍包括銷售、生產管理與採購等三部分，其中銷售包括訂單、送貨、銷退、請款與登帳等作業；生產管理包括領料、退料、繳庫與盤點等作業，而採購包括訂貨、進貨與退貨等作業。

System Analysis and Design

● 需求分析

- 根據對使用者之需求訪談結果，得知夢幻公司之銷售管理作業如下：

- 業務部負責接訂貨單，接到客戶訂貨通知時須先進行訂貨資料登錄，並做成品庫存檢核，若成品庫存充足，則直接進行送貨處理；若成品庫存不足，則送生產需求通知給生產部，以便進行產品之生產計畫。
- 業務部亦負責送貨與進行送貨資料處理，如計算金額、送成品等，並產出送貨單給客戶確認。

System Analysis and Design

● 需求分析

- 根據對使用者之需求訪談結果，得知夢幻公司之銷售管理作業如下：

- 業務部收到客戶欲退回已銷售之成品通知（銷退單），需記錄客戶編號及銷退之成品數量、單價，並計算銷退單銷退總金額等。
- 業務部向客戶請款：
 - 每月請款一次，請款日期為每月25日。
 - 針對各客戶之本期送貨資料計算出本期應收帳款。
 - 合計上期未收款項及本期應收帳款列印請款單，請客戶付款。
- 業務部收到客戶之付款單，登錄客戶編號及付款資料。

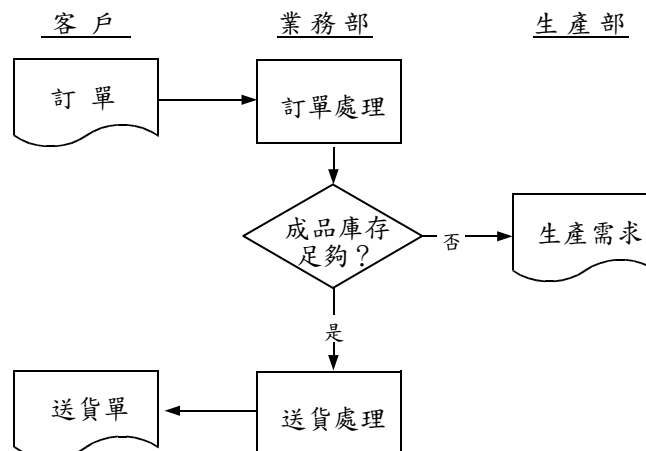
System Analysis and Design

• 流程圖範例

- 從上述之描述及訪談得知，前兩項作業可連續發生，也就是客戶訂貨，若有足夠庫存，則可馬上送貨，其餘三項作業均各自獨立。
- 在前兩項作業中，有三個外部實體參與：客戶、業務部與生產部。此外，前兩項作業中有訂貨與送貨兩個基本作業處理、一個庫存檢核控制及產出三張基本表單：訂單、送貨單與生產需求。
- 前兩項作業之流程圖可表示如下頁圖

System Analysis and Design

• 流程圖範例



System Analysis and Design

• 處理描述範例

- 以上述夢幻公司訂單送貨流程圖上之訂單處理為例（如上頁圖），其資料來源為客戶之訂單且產出為生產部之生產需求或通知出貨。
- 訂單處理之處理描述名稱可命名為訂單處理描述，該處理描述之執行程序與規則可從上述需求擷取之結果摘述如下頁表

System Analysis and Design

• 處理描述範例

處理名稱	訂單處理
執行程序與規則	<ol style="list-style-type: none"> 1. 業務部收到客戶訂單之後，需做客戶資料登錄與檢核。 2. 業務部檢查訂貨之成品庫存，若成品庫存充足，則進行送貨處理；若成品庫存不足，則通知生產部進行生產計畫。
資料輸入／來源	訂單／客戶
資料輸出／目的地	送貨訊息／業務部或生產需求／生產部
限制與備註	

System Analysis and Design

藍圖範例

- 以上述夢幻公司之訂單處理為例（參閱流程圖例），其訂單之藍圖可以該公司目前之訂單報表為基礎，再進一步對訂單上之每一欄位以由左至右與由上而下之原則編號，例如客戶編號為A、地址為B，依序至總金額為O等，詳如下頁表

System Analysis and Design

藍圖範例

夢幻企業股份有限公司
訂 單

客戶：	A	編號：	D
地址：	B	日期：	E
電話：	C		

成品編號 F	品名 G	顏色 H	規格 I	尺寸 J	數量 K	單位 L	單價 M	金額 N
10000006	太空梭模型	綠	25kg	S	3	個	417.60	1,252.80
10000005	鐵釘	紅	25kg	L	1	支	200.00	200.00
10000006	太空梭模型	綠	25kg	S	1	個	200.00	200.00
10000002	坐墊	綠	50kg	S	1	粒	6000.00	6,000.00
10000003	方向盤	黑	50kg	M	2	個	600.00	1,200.00
10000004	鐵蛋	紅	100kg	M	1	粒	200.00	200.00

客戶簽章： 總金額：9,052 O

註：灰色區域表示須套印表單之部份

System Analysis and Design

• 資料詞彙範例

- 如上所述，一張藍圖就應有一份資料詞彙，且藍圖中之每一欄位在資料詞彙中應有一記錄描述之，因此以夢幻公司之訂單藍圖為例（如前頁表單藍圖），且採用前述之資料詞彙樣板（參需求表達工具單元），再經由訪談整理，其訂單藍圖之資料詞彙可整理詳如下頁表

System Analysis and Design

• 資料詞彙範例

編號	欄位名稱	長度／形態	鍵	規則／格式／範圍／公式	範例
A	客戶名稱	20C			王大明
B	地址	40C			高雄市鼓山區蓮海路70號
C	電話	10C			07-5252000
D	編號	8N	✓	年+月+日+流水號 YYMMDD99	98090101
E	送貨日期	8D		YYYY年MM月DD日	1999年9月1日
F	成品編號	8C		99999999	10000003
G	品名	10C			方向盤
H	顏色	5C			黑
I	規格	14C			50Kg
J	尺寸	4C			M
K	數量	10N			2
L	單位	4C			個
M	單價	10N		99,999,999.99	600.00
N	金額	10N		數量 × 單價； 9,999,999,999	1,200
O	總計	10N		金額總和； 999,999,999	9,052

System Analysis and Design

- 企業流程塑模

- 完成了需求分析之工作，接下來是如何從分析結果進行企業流程塑模，以分析與設計該系統之各子系統。
- 本案例以資料流程圖為工具，並採用由中往外策略進行夢幻公司之企業流程塑模。進行步驟如下：

System Analysis and Design

- 企業流程塑模

- 步驟一 找出初步資料流程圖元素
 - 首先，從需求分析之結果
 - 找出外部實體
 - 找出處理
 - 找出資料儲存
 - 找出資料流

System Analysis and Design

• 企業流程塑模

- 步驟二 向上整合以建立資料流程圖

- 本個案之向上整合依管理功能之原則將步驟一之處理分成五群銷售管理(1.0)、生產管理(2.0)、採購管理(3.0)、基礎項目管理與(4.0)、綜合報表管理(5.0)等。
- 訂貨、送貨、銷退、請款與登帳等處理可整合成一企業程序，以完成該公司銷售管理之功能，其餘詳如下頁表

System Analysis and Design

• 企業流程塑模

- 處理分群範例

1.0	銷售管理
1.1	訂單處理
1.2	送貨處理
1.3	銷退處理
1.4	請款處理
1.5	登帳處理
2.0	生產管理
2.1	領料處理
2.2	退料處理
2.3	繳庫處理
2.4	盤點處理
3.0	採購管理
3.1	訂貨
3.2	進貨
3.3	退貨
4.0	基礎項目管理
4.1	基本資料處理
5.0	綜合報表管理
5.1	主管報表處理

System Analysis and Design

• 企業流程塑模

- 處理、資料檔與資料流向範例

處理	訂單 資料	送貨單 資料	銷退單 資料	請款單 資料	付款單 資料	客戶基 本資料	成品 資料	稅率 資料	客戶	業務部
訂單處理	↓↑					↓	↓		↓	↓↑
送貨處理	↓	↓↑				↓	↓↑	↓	↑	↓↑
銷退處理		↓	↓↑			↓	↓↑		↓	↓↑
請款處理		↓	↓	↓↑		↓			↑	↓↑
登帳處理				↓	↓↑	↓			↓	↓↑

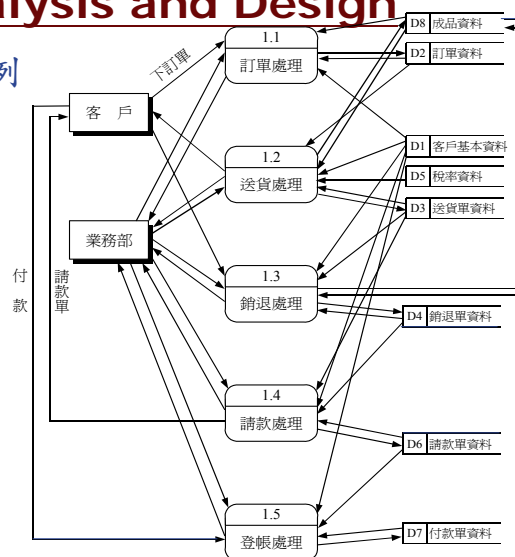
註：↓表由資料檔（實體）至系統；↑表由系統至資料檔（實體）；
↓↑表系統與資料檔（實體）間雙向交流。

<inhon@mail.tku.edu.tw>

May 25, 2010 P 137

System Analysis and Design

• 第一階DFD範例



<inhon@mail.tku.edu.tw>

May 25, 2010 P 138

System Analysis and Design

處理與資料流之整合範例

處理	訂單資料	送貨單資料	銷退單資料	請款單資料	付款單資料	客戶基本資料	成品資料	稅率資料	客戶	業務部
訂單處理	↓↑					↓	↓		↓	↓↑
送貨處理	↓	↓↑				↓	↓↑	↓	↑	↓↑
銷退處理			↓↑				↓↑		↓	↓↑
請款處理		↓	↓	↓↑		↓			↑	↓↑
登帳處理				↓	↓↑	↓			↓	↓↑

↓整合

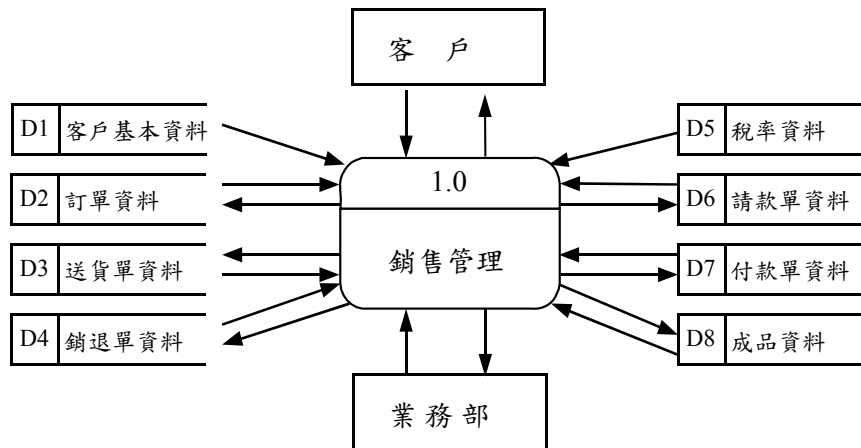
處理	訂單資料	送貨單資料	銷退單資料	請款單資料	付款單資料	客戶基本資料	成品資料	稅率資料	客戶	業務部
銷售管理	↓↑	↓↑	↓↑	↓↑	↓↑	↓	↓↑	↓	↓↑	↓↑

System Analysis and Design

第零階DFD範例

System Analysis and Design

• 第零階DFD範例

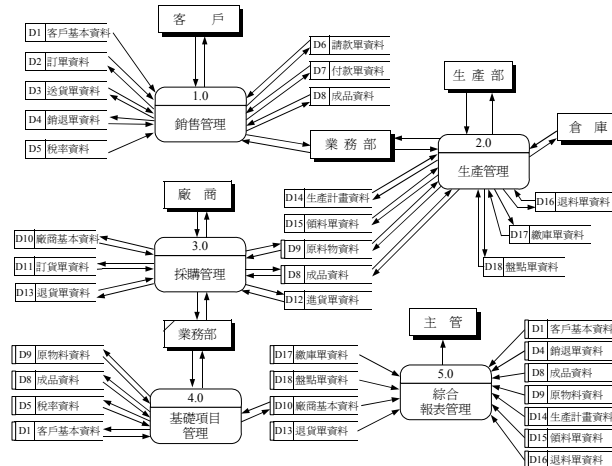


<inhon@mail.tku.edu.tw>

May 25, 2010 P 141

System Analysis and Design

• 第零階DFD範例全貌



<inhon@mail.tku.edu.tw>

May 25, 2010 P 142

System Analysis and Design

- 企業流程塑模

- 步驟三 向下分解以建立低層資料流程圖

- 向下分解之原則可依內聚力或程式碼之多寡（例如不要超過200行）來判定。
 - 以銷售管理子系統之送貨處理為例，送貨處理為步驟一產生之處理，從巨觀的角度來看，送貨處理僅做一件訂單相關之事情，已符合所謂的功能內聚力，可不必再分解，但若該處理中還包括新增、修改、刪除、查詢與列印等操作處理，則建議將送貨處理再依操作向下分解至第二階。

System Analysis and Design

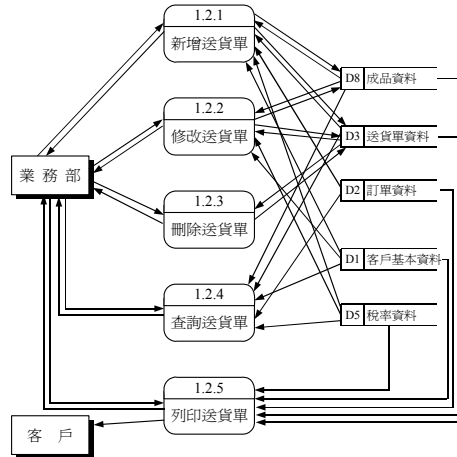
- 企業流程塑模

- 步驟三 向下分解以建立低層資料流程圖

- 以送貨單處理為例，其第二階 DFD 分析如下：
 - 處理
 - 資料儲存
 - 實體
 - 資料流

System Analysis and Design

• 第二階DFD範例(送貨處理)



<inhon@mail.tku.edu.tw>

May 25, 2010 P 145

System Analysis and Design

• 企業流程塑模

- 步驟三 向下分解以建立低層資料流程圖

- 某些較複雜之第二階處理而言，其程式碼數量可能過大，若再加入偵錯或例外狀況處理則將更大，因此可考慮將該之分解成更細之子處理。
- 以新增送貨單為例，可再分解成六個子處理，其第三階 DFD 分析如下：

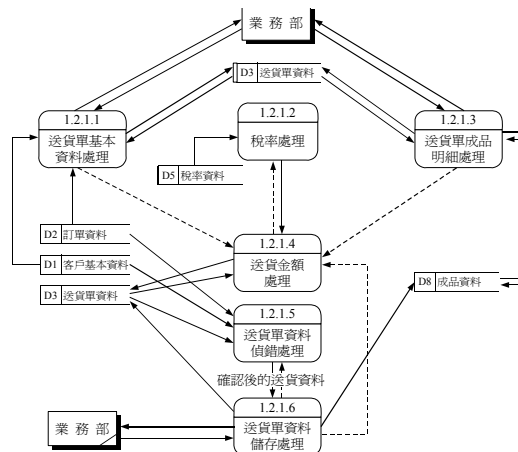
- 處理
- 資料儲存
- 實體
- 資料流

<inhon@mail.tku.edu.tw>

May 25, 2010 P 146

System Analysis and Design

• 第三階DFD範例(新增送貨單)



<inhon@mail.tku.edu.tw>

May 25, 2010 P 147

System Analysis and Design

• 處理規格描述範例

– 以送貨金額處理為例

Procedure 1.2.1.4 送貨金額處理

Begin

{11.計算送貨明細加總金額}

{7.計算稅前總金額}

{9.設定稅金額}

{8.計算稅後金額}

End;

{***11.計算送貨明細加總金額***}

Begin

移動送貨單的送貨明細資料到第一筆記錄

將送貨單的送貨明細加總金額初設為0

當送貨單的送貨明細資料還沒超過最後一筆時重複以下動作

Begin

送貨單的送貨明細加總金額=送貨明細(數量×售價)+原送貨單的送貨明細加總金額

移動送貨單的送貨明細資料到下一筆記錄

End;

End;

<inhon@mail.tku.edu.tw>

May 25, 2010 P 148

System Analysis and Design

- 處理規格描述與程式對應範例

- 以Delphi語言為例
- PDL轉為程式的註解

Procedure 1.2.1.4送貨金額處理

```
Begin
  {11.計算送貨明細加總金額}
  ComputeSummary;
  {7.計算稅前總金額}
  ComputeTotal;
  {9.設定稅金額}
  SetTaxMoney;
  {8.計算稅後金額}
  ComputeMoney;
End;
```

<inhon@mail.tku.edu.tw>

May 25, 2010 P 149

System Analysis and Design

- 處理規格描述與程式對應範例

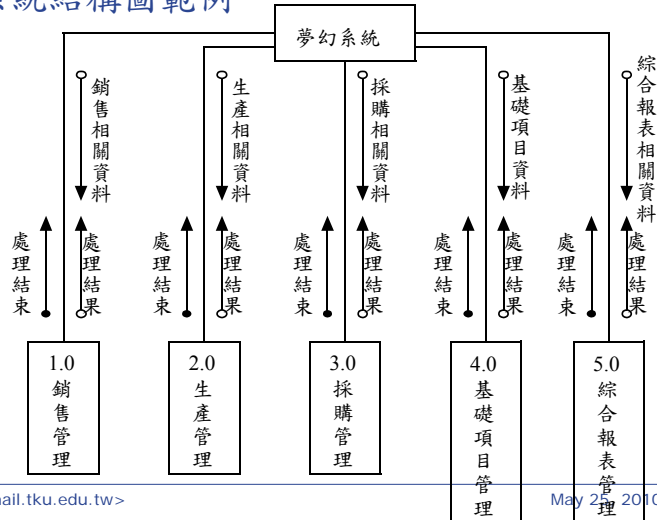
```
{***11.計算送貨明細加總金額*** PDL+程式碼 }
Procedure ComputeSummary; {計算送貨明細加總金額}
Begin
  移動送貨單的送貨明細資料到第一筆記錄
  DataMdlDeli.TblDeliProdSearch.First;
  將送貨單的送貨明細加總金額初設為0
  DataMdlDeli.TblDeliver.FieldByName('Summary').AsFloat := 0;
  當送貨單的送貨明細資料還沒超過最後一筆時重複以下動作
  While Not DataMdlDeli.TblDeliProdSearch.Eof Do
  Begin
    送貨單的送貨明細加總金額 = 送貨明細 (數量 × 售價) + 原送貨單的送貨明細加總金額
    DataMdlDeli.TblDeliver.FieldByName('Summary').AsFloat :=
      DataMdlDeli.TblDeliver.FieldByName('Summary').AsFloat +
      Round_(DataMdlDeli.TblDeliProdSearch.FieldByName('Price').AsFloat ×
        DataMdlDeli.TblDeliProdSearchNum.Value);
    移動送貨單的送貨明細資料到下一筆記錄
    DataMdlDeli.TblDeliProdSearch.Next;
  End;
End;
```

<inhon@mail.tku.edu.tw>

May 25, 2010 P 150

System Analysis and Design

系統結構圖範例

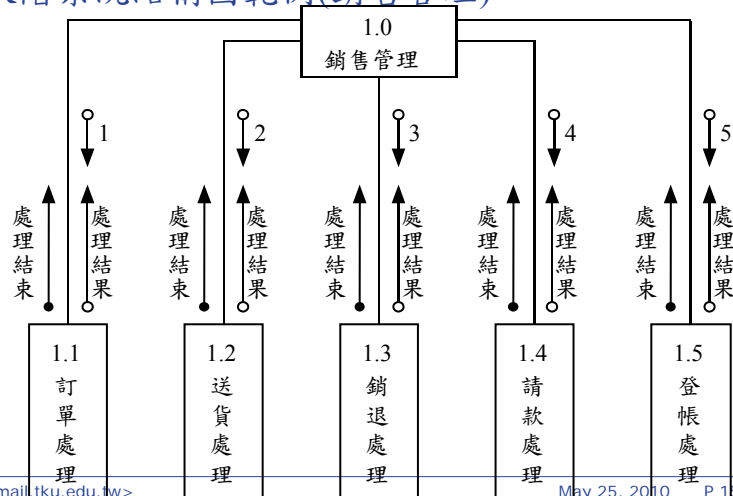


<inhon@mail.tku.edu.tw>

May 25, 2010 P 151

System Analysis and Design

次階系統結構圖範例(銷售管理)

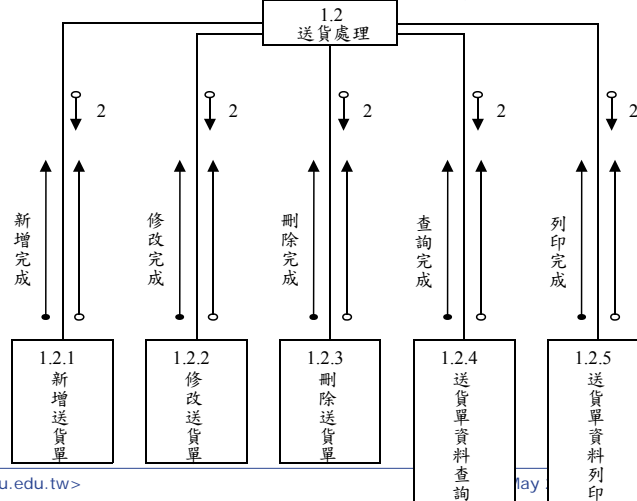


<inhon@mail.tku.edu.tw>

May 25, 2010 P 152

System Analysis and Design

• 再展開系統結構圖範例(送貨處理)

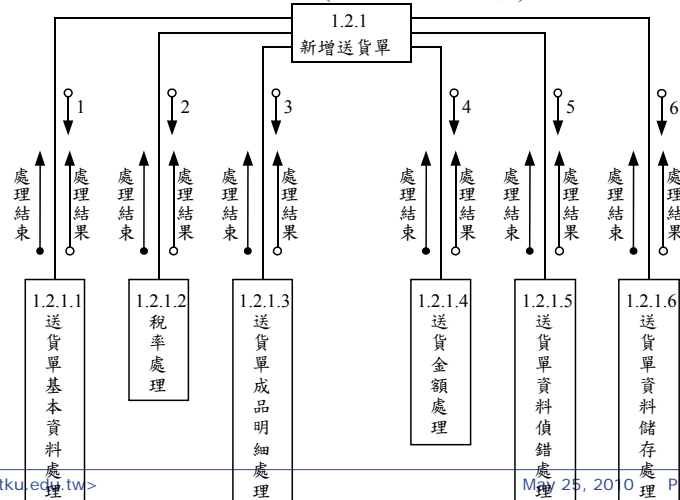


<inhon@mail.tku.edu.tw>

P 153

System Analysis and Design

• 再展開系統結構圖範例(新增送貨單)



<inhon@mail.tku.edu.tw>

M 25, 2010 P 154

System Analysis and Design

- Summary

- 需求分析之結果是SA&D的主要輸入，因此需求分析之表達是否完整，對SA&D之成敗有關鍵性影響。
- 對結構化之SA&D而言，流程塑模是程式設計之基礎，遵循科學化之方法論以進行流程塑模，可提升程式模組化與結構化程度，對程式之再用性與維護性有很大的幫助。

Agenda

1. Introduction
2. Recall Software Engineering
3. System Analysis and Design
- 4. Object-Oriented Concepts**
5. UML
6. Workflow of Requirement Analysis
7. Workflow of Object-Oriented System Analysis and Design

Object-Oriented Concepts

- The Vocabulary of Object Technology
 - Objects, Classes, and Instances
 - Message Passing and Associations
 - Generalization Relationship
 - Object Visibilities and Interfaces
 - Basic Principles of Object Orientation

Object-Oriented Concepts

- Objects, Classes, and Instances
 - An **Object** is an abstraction of a set of real world things [Shlaer et al. 1992].
 - “An object has state (attributes), exhibits some well-defined behavior (operations), and has a unique identity.” [Booch 1994], that is, an object can be characterized as: Object = Identity + State + Behavior.

Object-Oriented Concepts

- Objects, Classes, and Instances

- An **Instance** is an object created from a class. It has state, behavior, and identity like an object.
 - The terms instance and object are interchangeable; they are synonymous.
- **Instantiation** is the process whereby an object (instance) is created by a class.

Object-Oriented Concepts

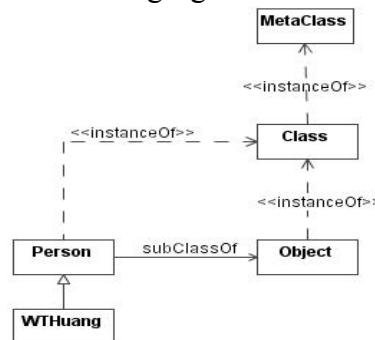
- Objects, Classes, and Instances

- **The Concept of Class.**
 - Abstract data type
 - Specifying the operational interface of its object instance
 - Object generator
 - Template for creating objects
 - Repository
 - For sharing resources, e.g., class-scope attribute and operation (underlined)
 - Object
 - Instance of a metaclass

Object-Oriented Concepts

• Objects, Classes, and Instances

- A **Metaclass** is a class whose instances are themselves classes, that is, a class that can be instantiated to other classes. The following figure shows a meta architecture.



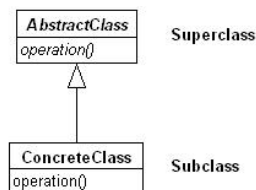
<inhon@mail.tku.edu.tw>

May 25, 2010 P 161

Object-Oriented Concepts

• Objects, Classes, and Instances

- A **Class** “is a set of objects that share a common structure and a common behavior.” [Booch 1994].
- **Abstract Classes** cannot be instantiated directly.
 - The main purpose of an abstract class is to define a common interface for its subclasses.
- **Concrete Classes** are not abstract and can have instances.



<inhon@mail.tku.edu.tw>

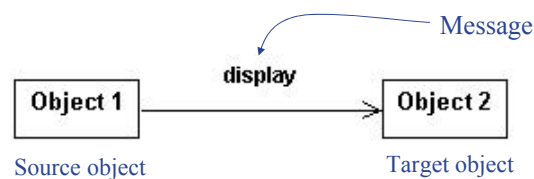
May 25, 2010 P 162

Object-Oriented Concepts

- Message Passing and Associations

- **Message Passing.**

- A **Message** is a communication element objects that triggers activity in the target objects.



Object-Oriented Concepts

- Message Passing and Associations

- An **Association** is a structural relationship that describes a set of links between classes (their objects).

- An association between classes usually means that the objects of two classes can send messages to another.
 - An association can have a name that describes the nature (semantic) of the relationship.

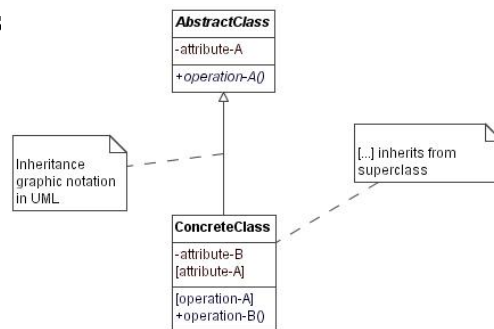
- A **Composite Object** is an object contain other objects.

- A message is sent to one object that acts the coordinator for several objects.
 - Aggregation: whole-part relationships

Object-Oriented Concepts

- Generalization Relationship

- **Class Hierarchies** are created so that more concrete classes inherit attributes and operations from more abstract class



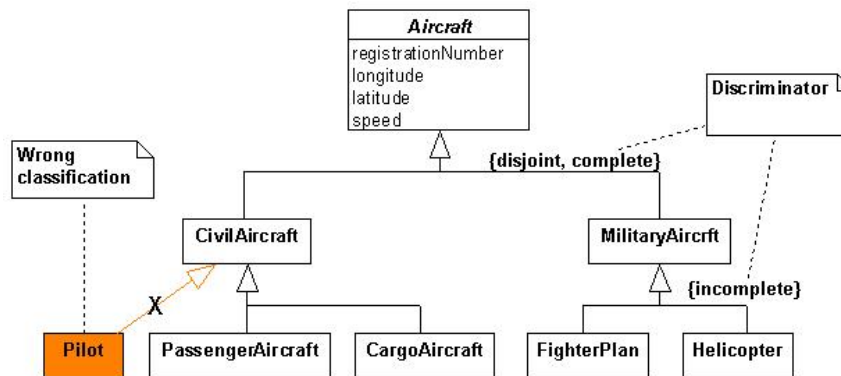
Object-Oriented Concepts

- Generalization-Specialization (Gen-Spec)

- **Generalization** is a relationship between a general thing, called superclass, and a more specific kind of that thing, called subclass.
 - Generalization is sometimes called “is-a” or “is-a-kind-of” relationship.

Object-Oriented Concepts

- Generalization-Specialization (Gen-Spec)



<inhon@mail.tku.edu.tw>

May 25, 2010 P 167

Object-Oriented Concepts

- Object Visibilities and Interfaces

- **Visibility** is how a name can be seen and used by others.
 - Visibility = Private + Protected + Public
 - Private: Only the class itself can use the features for the class.
 - Protected: Any descendant of the class can use the features.
 - Public: Any other class can use the features of the given class within the system.
 - An attribute of an operation can be specified their visibility as private, protected or public according to the developer's decision .

<inhon@mail.tku.edu.tw>

May 25, 2010 P 168

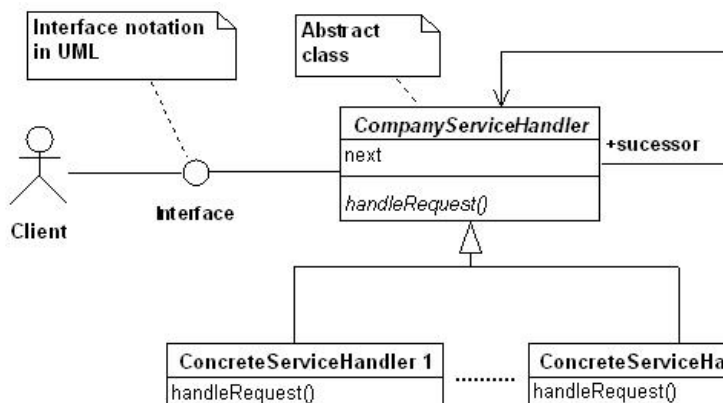
Object-Oriented Concepts

• Object Visibilities and Interfaces

- An **Interface** is a special type of class – an abstract class – that only provides a specification (not an implementation) for its members (abstract members).
 - A system can have many different interface classes
 - Example: Suppose a company offers some many services to its clients. How can the client get service he/she needs from the company?

Object-Oriented Concepts

• Object Visibilities and Interfaces



Object-Oriented Concepts

• Basic Principles of Object Orientation

– Object Orientation

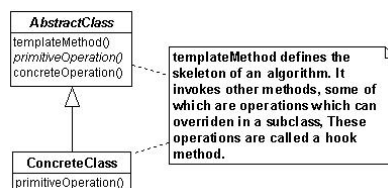
- Object orientation involves the elements of *abstraction*, *encapsulation*, *inheritance*, *polymorphism*, *classification*, and *identity*. If any of these elements is missing, we have something less than an object orientation. In this section, we briefly explain the concepts of these elements which will be explained more in detail in the chapters of “The Unified Modeling Language” and “Object-Oriented Development Process”.

Object-Oriented Concepts

• Basic Principles of Object Orientation

– Abstraction

- **Abstraction** is “the suppression of irrelevant detail”(*)
 - Modeling is the development of abstractions, thus, a model (say UML model) may describe a system that has not yet been implemented.
- Example: The Template Method pattern forms the basis of all patterns dealing with abstract classes.



Object-Oriented Concepts

• Basic Principles of Object Orientation

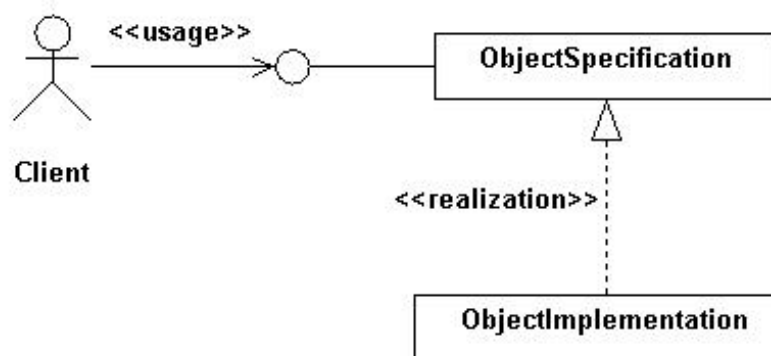
– Encapsulation

- **Encapsulation** is a mechanism that is used to hide the data, internal structure, and implementation details of an object. All interactions between objects is through public interface of operations.
 - In general, encapsulation means “any kind of hiding”.
- **The advantage**
 - Example: The separation of specification and implementation of an object. The implementation is hidden from client.

Object-Oriented Concepts

• Basic Principles of Object Orientation

– Encapsulation



Object-Oriented Concepts

• Basic Principles of Object Orientation

– Inheritance

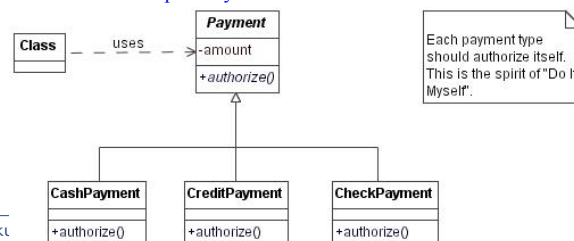
- **Inheritance** is a software mechanism to implement subtype conformance to supertype definitions.
- Gen-Spec vs. Inheritance.
 - Gen-Spec relationship
 - » is-a or is-a-kind-of relationship
 - Inheritance
 - » is a programming language concept for the implementation of a relation between a superclass and a subclass.
 - » usually implements gen-spec relationships.

Object-Oriented Concepts

• Basic Principles of Object Orientation

– Polymorphism

- The common ways to applying polymorphism.
 - “Do It Myself” [Coad 1997]
 - » “I (a software object) to those things that are normally done to the actual object that I’m an abstraction of.”
 - » Example: Payment

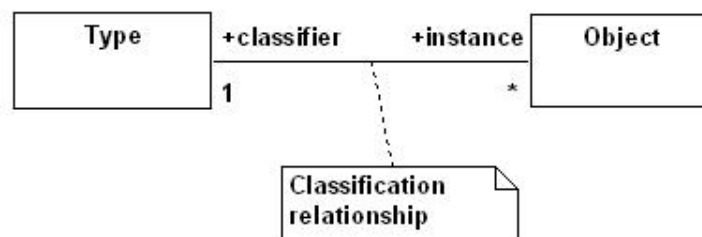


Object-Oriented Concepts

- Basic Principles of Object Orientation
 - Classification
 - “**Classification** is the act or result of applying a concept (i.e. type) to an object” [Martin et al. 1998].
 - When an object is classified, it becomes an instance of a specific type.
 - » For example, when the object Peter is employed, he is then classified as an Employee.
 - Classification is a relationship between types (or class) and objects.
 - Classification can occur in hierarchies, generalizations, aggregations.

Object-Oriented Concepts

- Basic Principles of Object Orientation
 - Classification



Object-Oriented Concepts

- Basic Principles of Object Orientation

- Identity

- “**Identity** is that property of an object which distinguishes it from all other objects.”
 - An object has a unique identity.

Object-Oriented Concepts

- Object-Oriented Lifecycle Model

- Problems arose when large information systems were developed using the traditional paradigm, especially when the resulting information systems were maintained.
 - By the mid-1980s, it had become clear that a better paradigm was needed.
 - The object-oriented paradigm proved to be the solution.
 - Over the next 10 years, more than 50 different object-oriented methodologies were published.

Object-Oriented Concepts

- Object-Oriented Lifecycle Model
 - Three of the most successful methodologies were
 - **Booch's Method**
 - **Jacobson's Objectory**
 - **Rumbaugh's OMT**

Object-Oriented Concepts

- Object-Oriented Lifecycle Model
 - The Software Development Process Framework – **Unified Process (UP)** [Jacobson et al. 1999].
 - The UP is designed to be a process framework from which customized processes could be derived.
 - The UP is an outgrowth of methodologies developed by Jacobson's "Objectory Methodology", "Booch Methodology", and Rumbaugh et al.'s "Object Modeling Technique" (OMT).

Object-Oriented Concepts

- Object-Oriented Lifecycle Model

- The Software Development Process Framework –
Unified Process (UP) [Jacobson et al. 1999].

- The UP is an *iterative* and *incremental* process with the strategy for developing a software product in small manageable steps:
 - Plan a little.
 - Specify, design, and implement a little.
 - Integrate, test, and run each iteration a little.

Object-Oriented Concepts

- Object-Oriented Lifecycle Model

- The UP is structured along two dimensions:
 - Time : divides the lifecycle into four phases and iterations
 - Disciplines (or process components): produce a specific set of artifacts with well-defined activities.

Object-Oriented Concepts

- Object-Oriented Lifecycle Model

- Time dimension:

- Inception Phase: specifies the project vision (the project scope);
 - Elaboration Phase: finalizes what's wanted and needed;
 - Construction Phase: builds the product;
 - Transition Phase: supplies the product to the user community.

Object-Oriented Concepts

- Object-Oriented Lifecycle Model

- Disciplines dimension:

- Business Modeling: reengineers the business process;
 - Requirements: captures a narration of what the system should do;
 - Analysis and Design: describe how the system will be realized in the implementation phase;
 - Implementation: produces code that will result in an executable system;

Object-Oriented Concepts

• Object-Oriented Lifecycle Model

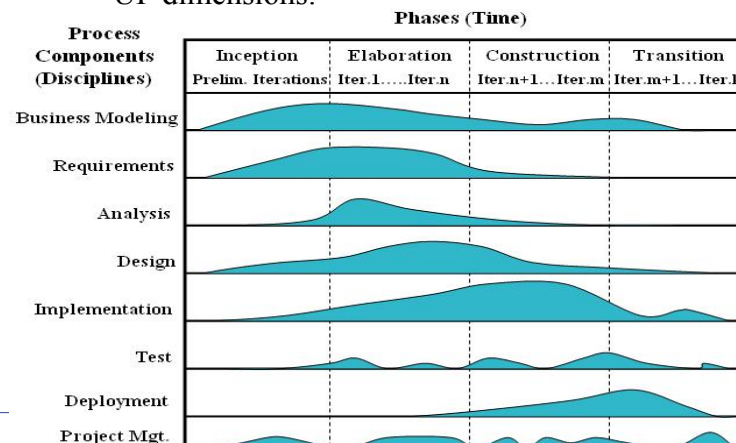
– Disciplines dimension:

- Test: verifies the entire system;
- Project management: provides
 - a framework for managing software-intensive projects;
 - practical guidelines for planning, staffing, executing, and monitoring project;
 - a framework for managing risk.

Object-Oriented Concepts

• Object-Oriented Lifecycle Model

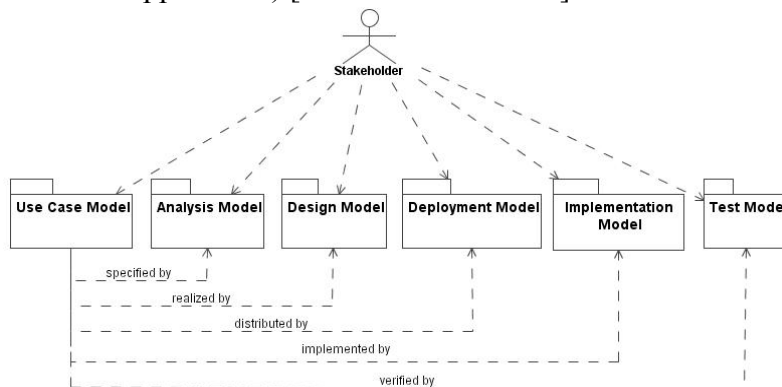
– UP dimensions:



Object-Oriented Concepts

- Object-Oriented Lifecycle Model

- The *primary model set* of the UP (views of the application) [Jacobson et al. 1999].



P 189

Object-Oriented Concepts

- Object-Oriented Development Process

- Analysis

- The purpose of analysis is to analyze the requirements in the form of an analysis model in order to
 - achieve a more precise understanding of the requirements,
 - achieve a description of the requirements that is easy to maintain.
- **Object-oriented analysis** is an activity that emphasizes finding and describing the objects, or concepts, in the problem domain.

Object-Oriented Concepts

- Object-Oriented Development Process

- **Commonality Analysis.**

- “Commonality analysis is the search for common elements that helps us understand how family members are the same”
 - ‘Family member’ means elements related to each other by the situation in which they appear or the function they perform.
 - Example: If people play music with a piano, an a violin, you might say they all have in common is these are “music instruments.” The process you perform to identify the piano and violin in a common manner is commonality analysis.
 - “Commonality analysis seeks structure that is unlikely to change over time.”

Object-Oriented Concepts

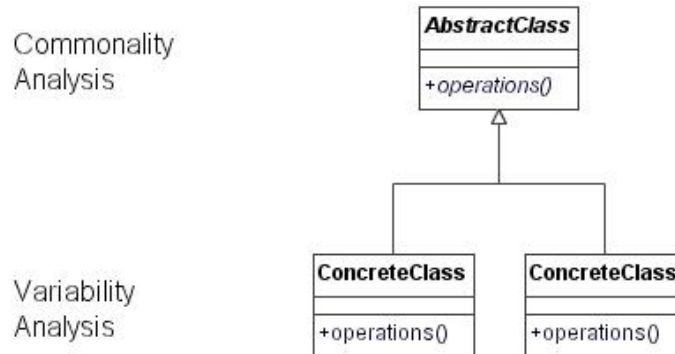
- Object-Oriented Development Process

- **Variability Analysis**

- **Variability Analysis** reveals how family members vary.
 - Example: piano and violin are the variation.
 - “Variability analysis captures structure that likely to change.”
 - The common concepts found by commonality analysis will be represented by abstract class. The variation found by variability analysis will be implemented by the concrete classes.

Object-Oriented Concepts

- Object-Oriented Development Process
 - Commonality Analysis and Variability Analysis



<inhon@mail.tku.edu.tw>

May 25, 2010 P 193

Object-Oriented Concepts

- Object-Oriented Development Process
 - We may use commonality and variability analysis as a tool in creating objects during finding objects in OO analysis and design process.
 - “Traditionally”, an OO designer begins by looking in the problem domain by identifying nouns and verbs relating to those nouns to be the candidates of objects. [Abbot 1983]

<inhon@mail.tku.edu.tw>

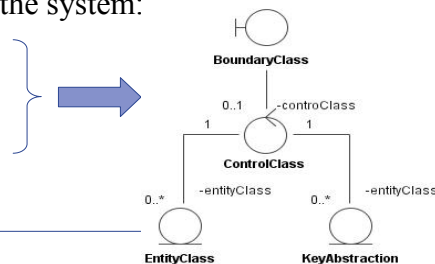
May 25, 2010 P 194

Object-Oriented Concepts

Object-Oriented Development Process

- An **Analysis Model** is a visual representation of conceptual classes or real-world objects in a domain of interest [Fowler 1997].
- In the analysis model, there are three different types of classes describing the system:

- Boundary classes
- Entity classes
- Control classes.



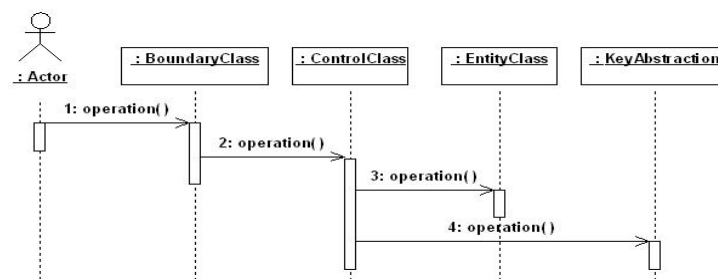
<inhon@mail.tku.edu.tw>

P 195

Object-Oriented Concepts

Object-Oriented Development Process

- The interaction among boundary, control, and entity classes.



where key abstraction is a class or object that forms part of the vocabulary of the problem domain.

<inhon@mail.tku.edu.tw>

May 25, 2010 P 196

Object-Oriented Concepts

- Object-Oriented Development Process

- A **Boundary Class** represents the system interface with the actor.
 - The boundary class collects the information from the actor and translates it into a form that can be used by the entity classes and control classes. It represents the boundary between actor and the system.
 - Boundary classes are often represent abstractions of screens (or windows) that are used to represent complete user interface instances with which the actors work.

Object-Oriented Concepts

- Object-Oriented Development Process

- An **Entity Class** is used to model the information that the system will handle over a longer time and often persistent.
 - The entity class is identified from the use cases (scenarios), that is, entity classes are the participants to realize the use cases.

Object-Oriented Concepts

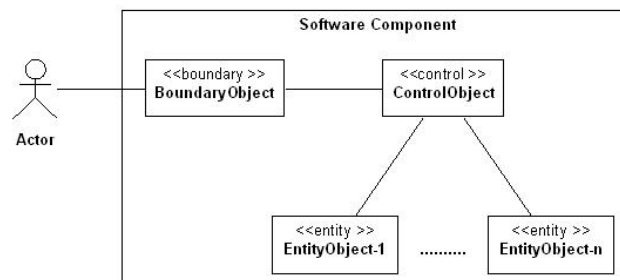
• Object-Oriented Development Process

- A **Control Class** is responsible for coordinating boundary and entity classes. It collects information from boundary classes and dispatches it to entity classes.
 - Types of functionality placed in the control classes: transaction related behavior, control sequences to use case(s), to separate entity class from the boundary classes.
- Functionality of a use case
 - = $\cup \{ \{ \text{Boundary Classes} \}, \{ \text{Entity Classes} \}, \{ \text{Control Classes} \} \}$

Object-Oriented Concepts

• Object-Oriented Development Process

- The decoupling of the boundary, control, and entity classes represents the separation of three aspects of the system (MVC pattern). This makes the system more tolerant to change and maintain.



Object-Oriented Concepts

• Object-Oriented Development Process

– Design

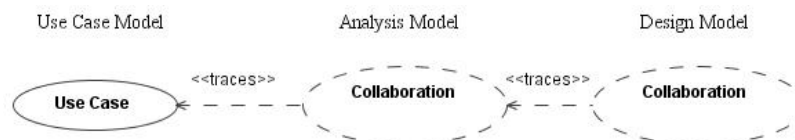
- **Design** is a process that uses the products of analysis to produce a blueprint (design model) for implementing a system. It is a logical description of how a system will work.
- **Object-oriented design** is an activity that emphasizes on defining software objects and how they collaborate to fulfill the requirements [Larman 2002].

Object-Oriented Concepts

• Object-Oriented Development Process

– Design

- Analysis and Design are summarized:
 - Analysis: do the right thing
 - Design: do the thing right



Object-Oriented Concepts

- Object-Oriented Development Process

- Design starts with the determination of the software architecture, which defines the general structure of the application.
- **Architecture** is a description of the organization, motivation, and structure of a system
 - Architectural patterns relate to the large-scale and coarse-grained design.

Object-Oriented Concepts

- Object-Oriented Development Process

- Design starts with the determination of the software architecture, which defines the general structure of the application.
- **Architecture** is a description of the organization, motivation, and structure of a system
 - Architectural patterns relate to the large-scale and coarse-grained design.

Object-Oriented Concepts

• Object-Oriented Development Process

- Architectural Pattern - Layers
- The **Layers patterns**, which structures a system into major layers. [Buschman et al. 1995]
 - Organize the large-scale logical structure of a system in to discrete layers; each layer is at a particular level of abstraction.
 - Collaboration and coupling is from higher to lower layers; lower-to-higher layer coupling is avoided. A CRC card shows this mechanism.

Class: Layer N	Collaborator
Responsibilities <ul style="list-style-type: none"> • Provides services for Layer N+1. • Delegates subtasks to Layer N-1. 	<ul style="list-style-type: none"> • Layer N-1

<inhon@mail.tku.edu.tw>

May 25, 2010 P 205

Object-Oriented Concepts

• Object-Oriented Development Process

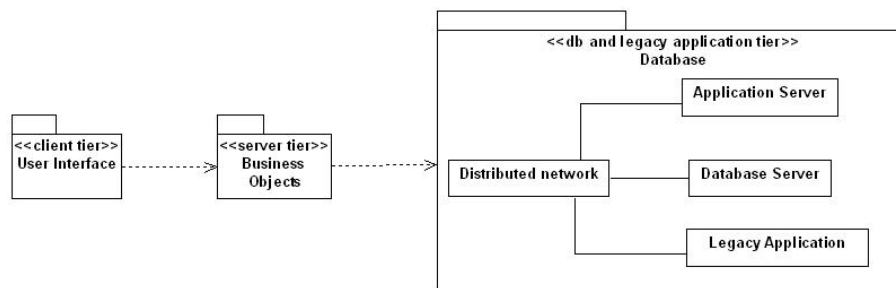
- Architectural Pattern - Layers
 - Three-Tier Architecture – a common layers (logical architecture) in information system
 - Presentation (client tier): windows, report, etc.
 - Application logic (Internet server tier): business objects and operations or business rules
 - Storage (database and legacy application tier)

<inhon@mail.tku.edu.tw>

May 25, 2010 P 206

Object-Oriented Concepts

- Object-Oriented Development Process
 - Architectural Pattern - Layers
 - Three-Tier Architecture

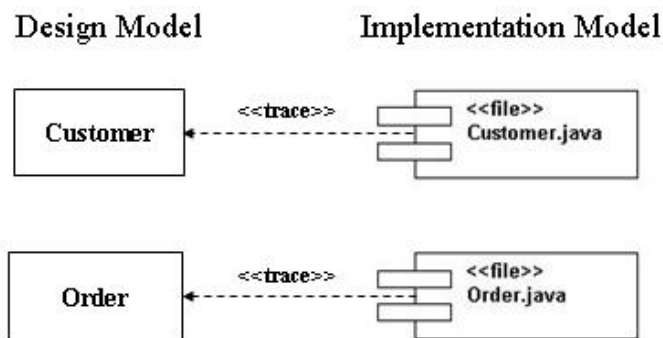


<inhon@mail.tku.edu.tw>

May 25, 2010 P 207

Object-Oriented Concepts

- Object-Oriented Development Process
 - Implementation
 - Components tracing to a design classes



<inhon@mail.tku.edu.tw>

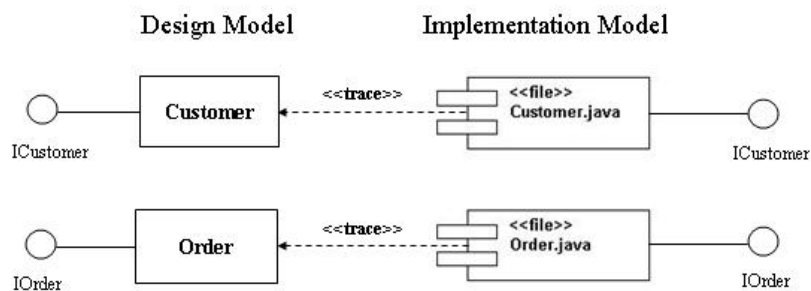
May 25, 2010 P 208

Object-Oriented Concepts

• Object-Oriented Development Process

– Implementation

- Components provide the same interfaces as the classes they implement.



<inhon@mail.tku.edu.tw>

May 25, 2010 P 209

Object-Oriented Concepts

• Object-Oriented Development Process

– Test

- Bottom-up testing.
 - Sending message to a class and asking it for what you expect it to do.
 - Example: You send message to the customer class and ask it for the credit limit.
 - If the class is a subclass, you are only testing a portion of the hierarchy.
- Top-down testing.
 - Top-down testing starts with use cases
 - » Incrementally test through all of the scenarios that are derived from the use case descriptions

<inhon@mail.tku.edu.tw>

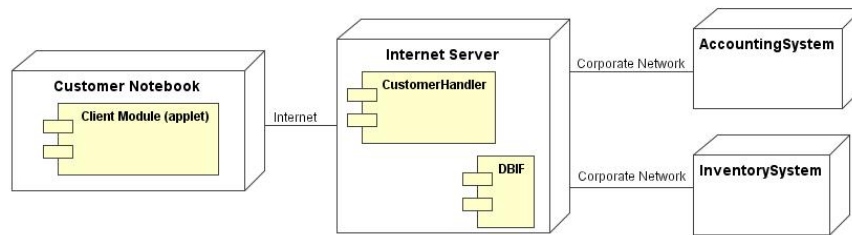
May 25, 2010 P 210

Object-Oriented Concepts

- Object-Oriented Development Process

- Deployment

- Deployment & Component Diagram for an Order Processing System

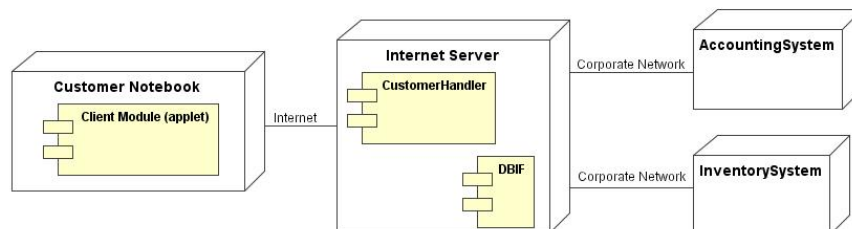


Object-Oriented Concepts

- Object-Oriented Development Process

- Deployment

- Deployment & Component Diagram for an Order Processing System



Object-Oriented Concepts

- Object-Oriented Development Process

- Agile Methods on Object-Oriented Software Development

- **Agile methods** aim at improving human communication and testing in software development.
 - By human communication, written documentation (a means of indirect communication) is minimized, software quality is thus improved and development time shortened. By testing, errors in human thinking are caught early. This assures software quality.

Object-Oriented Concepts

- Agile Methods on Object-Oriented Software Development

- User Story, Small Release, and Acceptance Test Cases

- A *user story* is a system function description consisting of Chinese sentences with English terms. (In this material, however, only English is used)
 - Be very concise - only half page per story.
 - This is just a reminder of requirement to help human communication between customer and developer. It is not a requirement document.

Object-Oriented Concepts

- Agile Methods on Object-Oriented Software Development
 - User Story, Small Release, and Acceptance Test Cases
 - For each user story (system function), write acceptance test cases (called system tests) for the function. For example:
 - Two test cases for the function of “borrow books”:
 - Test case 1:
 - » input: borrower: Jen-Yen CHEN , book name: ABC
 - » output: borrowed
 - Test case 2:
 - » input: borrower: Jen-Yen CHEN , book name: XYZ
 - » output: not borrowed

Object-Oriented Concepts

- Agile Methods on Object-Oriented Software Development
 - User Story, Small Release, and Acceptance Test Cases
 - In practice, we need customer physically be with development team, so that human communication can take place.
 - The team can communicate with customer at any time.
 - This eliminates requirement document.
 - The customer should do:
 - User story, acceptance test cases,
 - acceptance testing, etc.
 - Most importantly, he or she must decide, based on market considerations, which user stories are to be developed in this iteration – called a *small release*.

Object-Oriented Concepts

- Agile Methods on Object-Oriented Software Development

- OO Thinking with CRC Session

- After deciding on the small release, a CRC (class, responsibility, collaborator) session is conducted in a brainstorming way to determine classes of the system. That is, to design the system jointly and iteratively.
 - In this session, an object is played (simulated) by a developer. And all the developers together trace the acceptance test cases to identify all the objects, their responsibilities (public methods), and their collaborators (objects they called).

Object-Oriented Concepts

- Agile Methods on Object-Oriented Software Development

- OO Thinking with CRC Session

- Six people, at most, sit around table in a CRC session.
 - Allocate one object for someone to play (and use one card to take down the object).
 - Simulate interaction of the objects.
 - At any time take down:
 - (1) messages from objects and their parameters, and
 - (2) message receivers.

Object-Oriented Concepts

- Agile Methods on Object-Oriented Software Development
 - OO Thinking with CRC Session
 - Simulate all user stories and acceptance test cases, and adjust records of the objects all the times.
 - The final CRC cards represent architecture (design) of the system.

Object-Oriented Concepts

- Agile Methods on Object-Oriented Software Development
 - Pre-conditions and Post-conditions
 - When developing software, programming pairs (in this method, developers work in pairs for highest possible human communication) communicate with each other looking for preconditions, post-conditions, and invariants of the public method.
 - You can write pre- and post-conditions in natural language, and verify them by human.
 - You also can use formal method to write pre and post-conditions. Here, you can use tool to verify them.

Object-Oriented Concepts

- Agile Methods on Object-Oriented Software Development
 - Creating Unit Test Cases
 - Based on the preconditions and post-conditions, try to figure out unit test cases. (i.e. inputs and expected outputs for the unit)
 - It is recommended to write test cases for public method only, but not for private methods in order to simplify development process. This saves a lot of testing efforts.

Object-Oriented Concepts

- Agile Methods on Object-Oriented Software Development
 - Data Structuring
 - Note that cognitive theory says :
Everyone finish a fixed-number line of source code in a working day, regardless of the abstraction level of code.
 - Also note that, if abstract data structure is used (for example, Java reusable class “TreeSet”) then less code is needed to achieve the same function.

Object-Oriented Concepts

- Agile Methods on Object-Oriented Software Development
 - Data Structuring
 - Otherwise, if primitive data types (e.g., integer, Boolean) or simple data structure (e.g., array) are used, more code is needed.
 - By the theory above, that prolongs development time. Quality of code will thus be low, as developer gets tired in long working day.

Object-Oriented Concepts

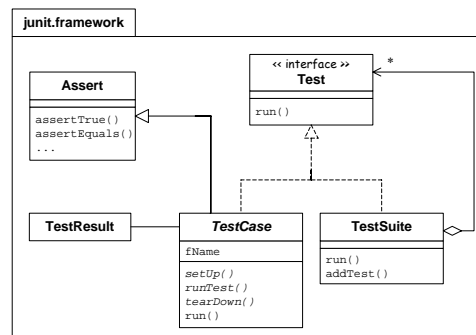
- Agile Methods on Object-Oriented Software Development
 - Pseudo Coding with Design Sketching
 - In the thinking process of software development, design sketching and pseudo coding are often proceeded in a interleaved way, this method thus combines the two into one step.
 - In physical arrangement of source code file, design sketch can be placed ahead of pseudo code to help understandability.

Object-Oriented Concepts

- Agile Methods on Object-Oriented Software Development
 - Coding
 - Java coding is used in this methodology.
 - No detailed discussion on Java will be given here, as Java textbooks are readily available everywhere.

Object-Oriented Concepts

- Agile Methods on Object-Oriented Software Development
 - Unit Testing (JUnit Framework)



Object-Oriented Concepts

- Agile Methods on Object-Oriented Software Development
 - Unit Testing (JUnit Framework)
 - As shown above, JUnit will be used to set up test suite (test cases) that will automatically tests source code.
 - JUnit will assert if each test case has passed or not, and will tabulate test results.

Object-Oriented Concepts

- Agile Methods on Object-Oriented Software Development
 - Continuous Integration
 - After unit testing, you have to integrate the unit into the system immediately, and make it pass acceptance tests.
 - “Hammer the iron (unit) while it is still hot” – meaning that while programming pair still remembers unit details clearly, it is rather easy to change code of the unit to fix integration problems.
 - Never postpone it till tomorrow! As people forget things after a night’s sleep.

Agenda

1. Introduction
2. Recall Software Engineering
3. System Analysis and Design
4. Object-Oriented Concepts
- 5. UML**
6. Workflow of Requirement Analysis
7. Workflow of Object-Oriented System Analysis and Design

UML

- The UML is an OMG's standard visual language (a drawing notation with semantics) used to create models of programs. It is for:
 - Visualizing
 - Specifying
 - Constructing
 - Documenting
- The artifacts of a software-intensive system [Booch et al. 1999]. But, it is not a method.

UML

- The Unified Modeling Language (UML) has become the de-facto standard for building Object-Oriented software.
- UML 2.1 builds on the already highly successful UML 2.0 standard, which has become an industry standard for modeling, design and construction of software systems as well as more generalized business and scientific processes.

UML

- UML 2.1 defines thirteen basic diagram types, divided into two general sets: structural modeling diagrams and behavioral modeling diagrams.
- The Object Management Group (OMG) specification states:
 - *“The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML offers a standard way to write a system’s blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components.”*

UML

- The important point to note here is UML is a “language” for specifying and not a method or procedure.
- However, UML can serve as a basis for different methods.
- The UML is used to define a software system – to detail the artifacts in the systems, to document and construct; it is the language the blueprint is written in.
- The UML may be used in a variety of ways to support a software development methodology (such as the Rational Unified Process), but in itself does not specify that methodology or process.

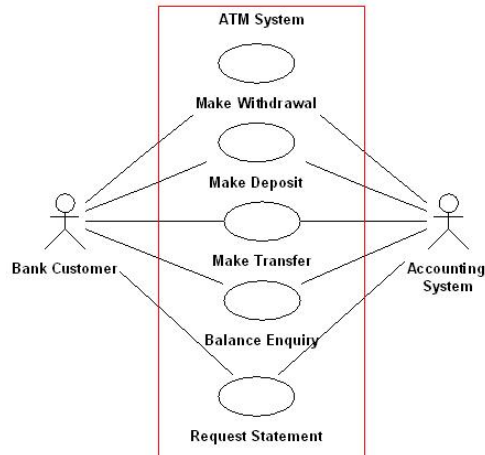
UML

- Domain using UML
 - Enterprise information system
 - Banking and financial services
 - Telecommunications
 - Defense/Aerospace
 - Medical electronics
 - Distributed web-based services
 - Hardware design
 - Non-software systems
 - etc.

UML

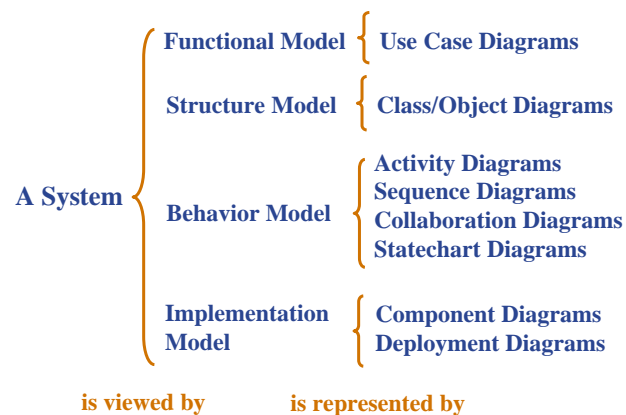
• Visual Modeling

- **Visual Modeling** is the process of thinking about the problems in the real-world using models and some kind of standard set of graphical notations, such as the UML.
- Example: Using use case diagram to visualize an ATM system.



UML

• Different Perspectives of a System



UML

- UML Notations
 - Notation for
 - **Functional Model**
 - **Structural Model**
 - **Behavior Model**
 - **Implementation Model**

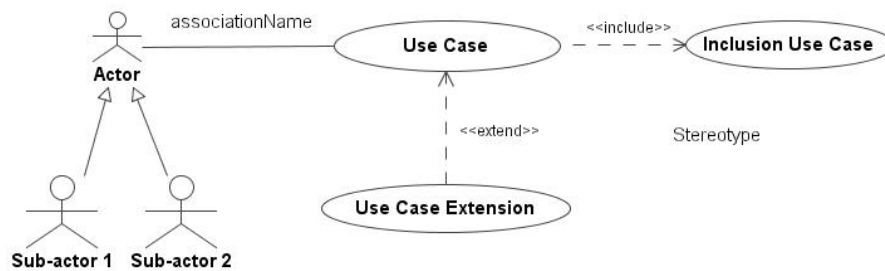
UML

- This notation summary provides a simple overview of what has been covered in this courseware.
- Readers may learn a more complete UML features we have omitted by referring to:
<http://www.omg.org/uml>, [Booch et al. 1999], [Rumbaugh et al. 1999] or a concise book [Fowler 2002].
- The semantics of the notations are based on [Rumbaugh et al. 1999] and [Fowler 2002].

UML

• Notations for Functional Model

- **Use Case Diagrams:** address the static functionality view of a system.



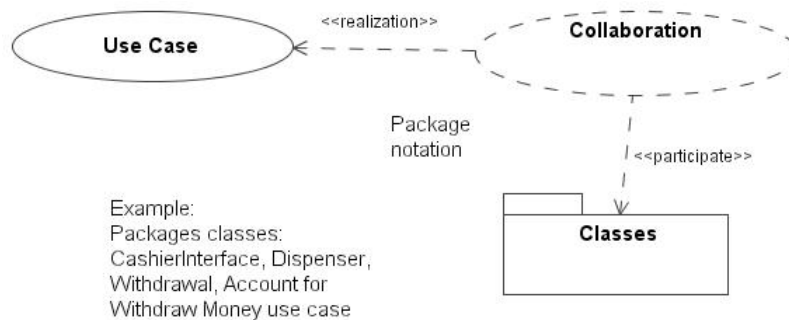
<inhon@mail.tku.edu.tw>

May 25, 2010 P 239

UML

• Notations for Functional Model

- **Use Case Realization:** shows the relationship between specification (use case) and its implementation (collaboration).



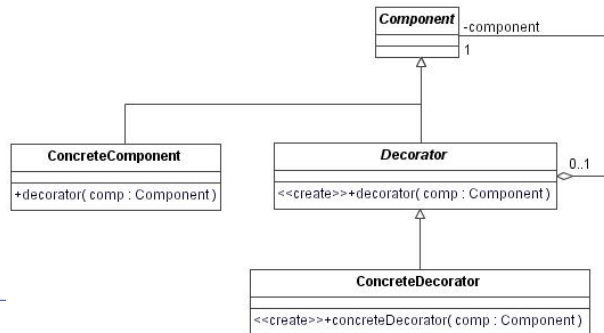
<inhon@mail.tku.edu.tw>

May 25, 2010 P 240

UML

• Notations for Structural Model

- **Class Diagram:** “shows a set of classes, interfaces, and collaborations and their relationships; it addresses static design view of a system.” For example, the Decorator pattern class diagram. [MagicDraw UML 9.0]

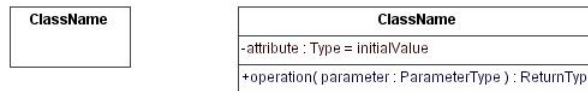


May 25, 2010 P 241

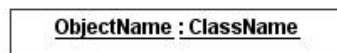
UML

• Notations for Structural Model

- **Class:** is the structure and the behavior of a set of objects.



- **Object:** An instance of a class, describing the set of possible objects that can exist.



UML

• Notations for Structural Model

- **Constraint:** A semantic condition or restriction expressed in some textual language.
{description of constraint in OCL or natural language}
- **Stereotype:** A variation of an existing model element with the same form.
<<stereotype name>>
- **Tagged Value:** A selector value pair that may be attached to any element.
{description of tagged value}

UML

• Notations for Structural Model

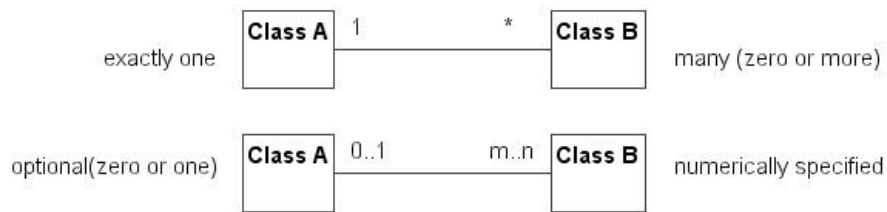
- **Associations:** represent relationships between instances of classes.
 - There are two roles of each association; each role is a direction on the association. For example, the role direction Class A to Class B is called role B; the role direction Class B to Class A is called role A.



UML

• Notations for Structural Model

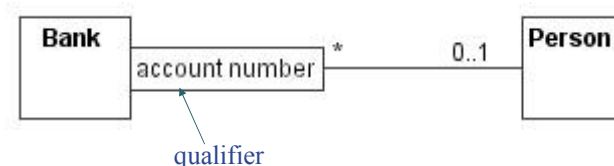
- **Multiplicities:** indicate how many objects may participate in the given relationships.



UML

• Notations for Structural Model

- **Qualified Association:** A qualifier used to select an object or objects from a set of objects related to an object by an association. For example, a Person can be associated with many Bank account number.

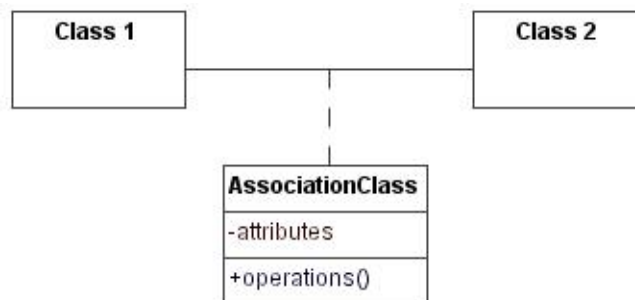


person *has* many (bank, account number)
 (bank, account number) *belongs* to 0 or 1 person

UML

• Notations for Structural Model

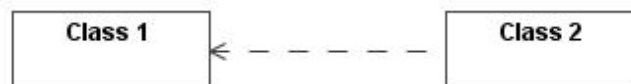
- **Association Class**: is a class attached to an association to provide extra information about the connection. It is a normal class.



UML

• Notations for Structural Model

- **Dependency**: is a relationship between two elements in which a change to independent element may affect information that needed by the dependent element.



UML

- Notations for Structural Model

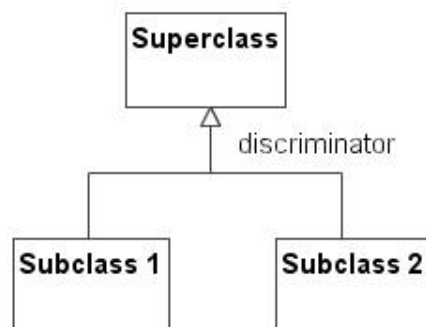
- **Navigability**: indicates that Source has a responsibility to tell which Target it is for, but a Target has no corresponding ability to tell which Source it has.



UML

- Notations for Structural Model

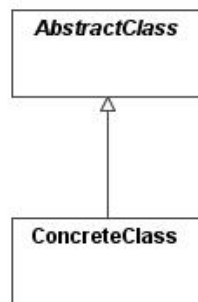
- **Generalization**: is a taxonomic relationship between a more general element and a more specific element.



UML

• Notations for Structural Model

- **Abstract Class**: is a class that may not be instantiated.
- **Concrete Class**: is a generalizable class that can be directly instantiated.



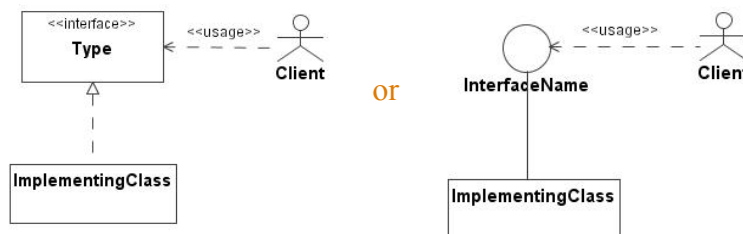
<inhon@mail.tku.edu.tw>

May 25, 2010 P 251

UML

• Notations for Structural Model

- **Interface**: is a named set of operations that characterize the behavior of a class.



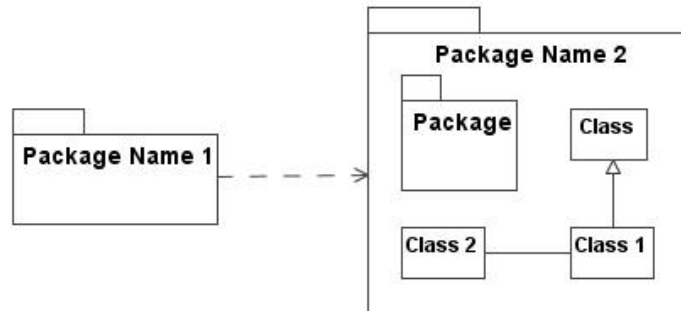
<inhon@mail.tku.edu.tw>

May 25, 2010 P 252

UML

• Notations for Structural Model

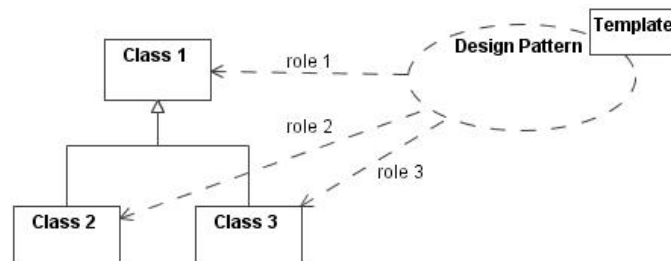
- Package: is a grouping of model elements and diagrams.



UML

• Notations for Structural Model

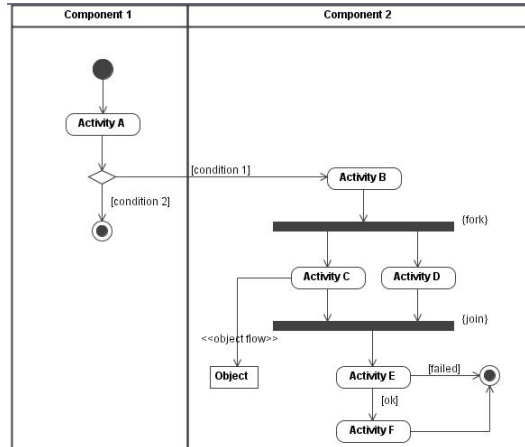
- **Design Pattern**: is “systematically names, motivations, and explains a general design that address a recurring design problem in object-oriented systems.” [GoF 1995]
- Modeling a Design Pattern:



UML

• Notations for Behavior Model

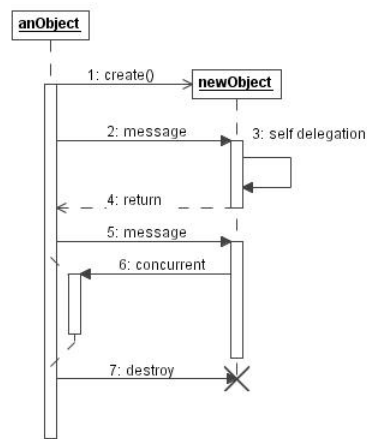
- **Activity Diagram:** is a diagram that shows the flow from activity to activity.



UML

• Notations for Behavior Model

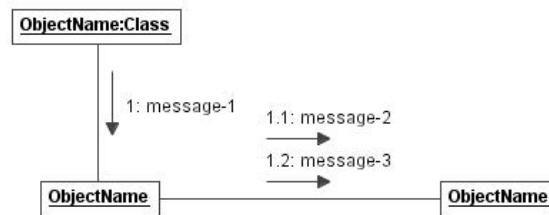
- **Sequence Diagram:** is an interaction diagram that emphasizes the time ordering of messages.



UML

• Notations for Behavior Model

- **Collaboration Diagram:** “is an interaction diagram that emphasizes the structural organization of the object that send and receive messages.” [Booch et al. 1999]



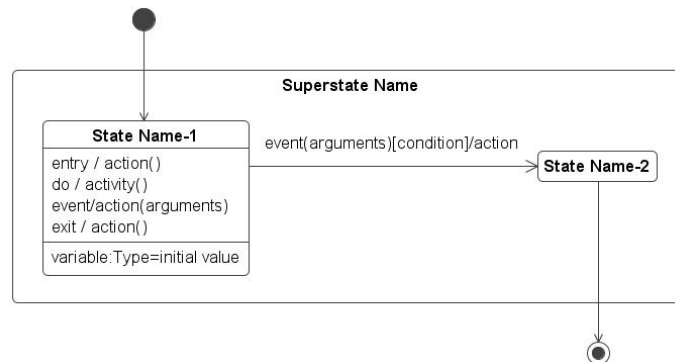
<inhon@mail.tku.edu.tw>

May 25, 2010 P 257

UML

• Notations for Behavior Model

- **Statechart Diagram:** addresses the dynamic view of system.



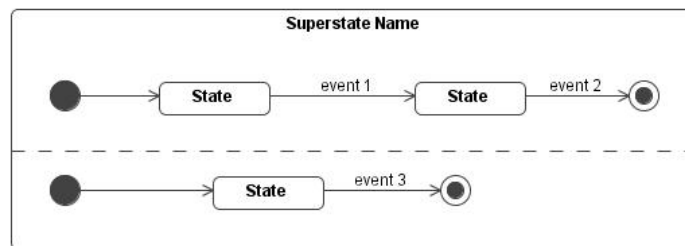
<inhon@mail.tku.edu.tw>

May 25, 2010 P 258

UML

• Notations for Behavior Model

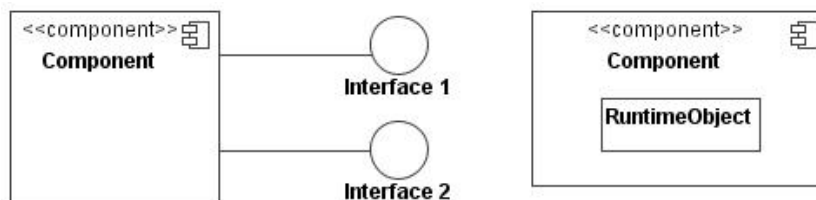
- **Concurrent States**: is “an orthogonal substate that can be held simultaneously with other substates contained in the same composite state.” [Booch et al. 1999]



UML

• Notations for Implementation Model

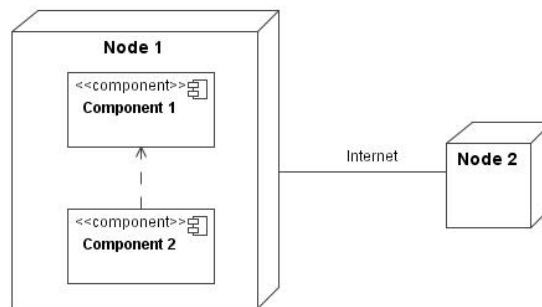
- **Component Diagram** (UML v2.0): “shows the organization of and dependencies among a set of components.” [Booch et al. 1999]



UML

- Notations for Implementation Model

- **Deployment Diagram:** “shows the configuration of run time processing nodes and the components that live on them.” [Booch et al. 1999]



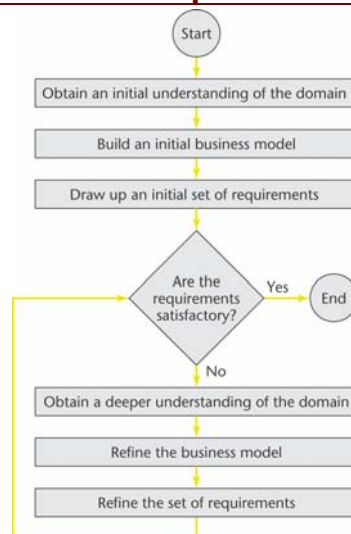
Agenda

1. Introduction
2. Recall Software Engineering
3. System Analysis and Design
4. Object-Oriented Concepts
5. UML
- 6. Workflow of Requirement Analysis**
7. Workflow of Object-Oriented System Analysis and Design

Workflow of Requirement Analysis

- Overview of the Requirement Analysis based on Unified Process (UP)
 - Understanding the *Application Domain*
 - Build a *Business Model*, using UML diagrams
 - Determine what the *Client's Requirements* are
 - *Iterate* the above steps

Flowchart of the Requirements Workflow



Workflow of Requirement Analysis

- Discovering the client's requirements
 - *Requirements elicitation* (or *requirements capture*)
 - Methods include interviews and surveys

- Refining and extending the initial requirements
 - *Requirements analysis*

Workflow of Requirement Analysis

- Understanding the Domain
 - Every member of the development team must become fully familiar application domain
 - Correct terminology is essential
 - We must build a glossary
 - That is, a list of technical words used in the domain, and their meaning

Workflow of Requirement Analysis

- Glossary, for example An information System for Art Dealer

Landscape	A painting of a scene in nature.
Masterpiece	A painting of undoubted excellence.
Masterwork	An inferior painting by an artist who previously or subsequently has painted a masterpiece.
Medium	A classification criterion. The material with which an artwork is painted. See also: oil, watercolor.
Oil	A medium. Abbreviation for "oil-based paint."
Other painting	A painting that is neither a masterpiece nor a masterwork.
Other subject	A subject that is not a landscape, a portrait, or a still life.
Portrait	A painting of one or more people.
Quality	A classification criterion. A painting is classified as a masterpiece, masterwork, or other painting, depending on its quality.
Still life	A painting of inanimate objects.
Subject	A classification criterion. Subjects include landscape, portrait, and still life.
Watercolor	A medium. Abbreviation for "water-based paint."

Workflow of Requirement Analysis

- Business Model
 - A *business model* is a description of the business processes of an organization
 - The business model gives an understanding of the client's business as a whole
 - This knowledge is essential for advising the client regarding computerization
 - The systems analyst needs to obtain a detailed understanding of the various business processes
 - Different techniques are used, primarily interviewing

Workflow of Requirement Analysis

- Interviewing

- The requirements team meet with the client and users to extract all relevant information
- There are two types of questions
 - *Close-ended* questions requires a specific answer
 - *Open-ended* questions are asked to encourage the person being interviewed to speak out
- There are two types of interviews
 - In a structured interview, specific preplanned questions are asked, frequently close-ended
 - In an unstructured interview, questions are posed in response to the answers received, frequently open-ended

Workflow of Requirement Analysis

- Interviewing

- Interviewing is not easy
 - An interview that is too unstructured will not yield much relevant information
 - The interviewer must be fully familiar with the application domain
 - The interviewer must remain open-minded at all times
- After the interview, the interviewer must prepare a written report
 - It is strongly advisable to give a copy of the report to the person who was interviewed

Workflow of Requirement Analysis

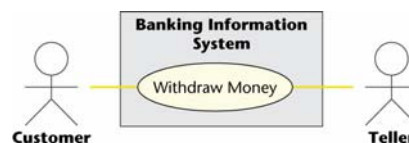
• Use Case Model

- An **Use Case Model** is a model that describes the functional requirements of a system to be developed, containing *actors*, *use cases*, and their *relationships*.
 - Requirements describe a system and its interaction with the surrounding environment, such as users and other systems.
 - A system is described by models from various kinds of view (see section 3.1.6), which are written by using well-defined language, such as the UML.
 - Use Case Diagrams describes Use Case Model.

Workflow of Requirement Analysis

• Use Case Diagrams

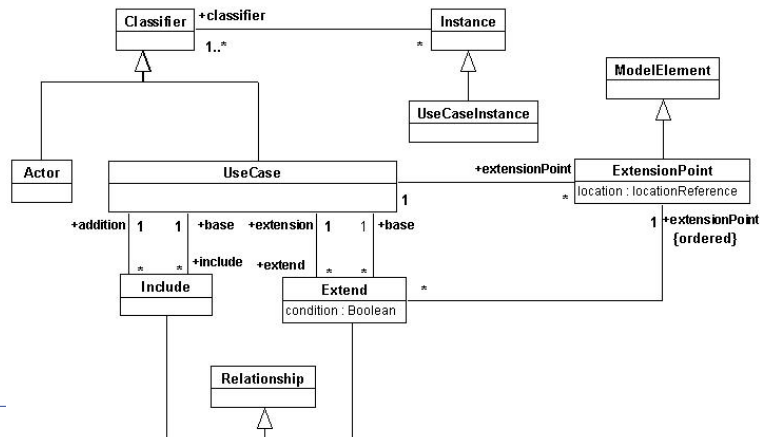
- An **Use Case Diagram** shows the relationships among actors and use cases within a system” [OMG-UML v1.4].
- Use case diagrams commonly contain:
 - Actors
 - Use Cases
 - Association, generalization, dependency, and realization relationships.
 - For example,



Workflow of Requirement Analysis

• Use Case Diagrams

- Simplified use case diagram metamodel [OMG 2001]



P 273

Workflow of Requirement Analysis

• Use Case Diagrams

- Use case diagrams are used
 - to model the context of a system;
 - to model the requirements of a system
 - to be used as a tool for communication between customers and developers.

Workflow of Requirement Analysis

- Use Case Diagrams

- A use case shows the interaction between
 - The information system and
 - The environment in which the information system operates
- Each use case models one type of interaction
 - There can be just a few use cases for an information system, or there can be hundreds
- The rectangle in the use case represents the information system itself

Workflow of Requirement Analysis

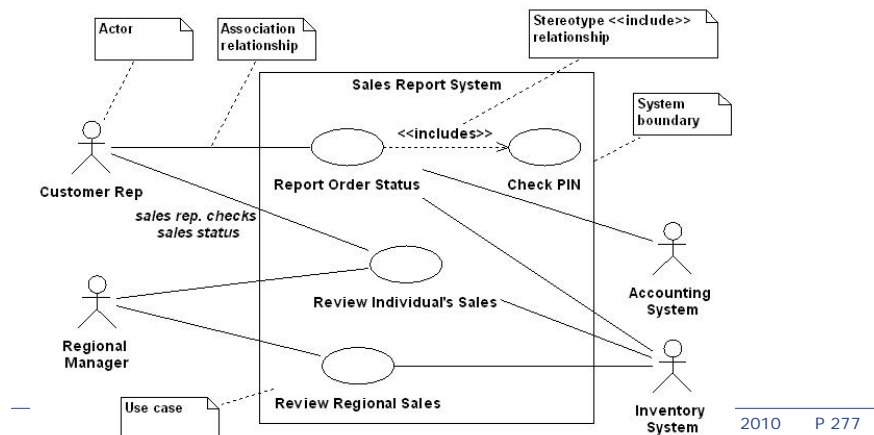
- Use Case Diagrams

- An actor is a member of the world outside the information system
- It is usually easy to identify an actor
 - An actor is frequently a user of the information system
- In general, an actor plays a role with regard to the information system. This role is
 - As a user; or
 - As an initiator; or
 - As someone who plays a critical part in the use case
- A user of the system can play more than one role

Workflow of Requirement Analysis

• Use Case Diagrams

- Example: A Sales Report System



Workflow of Requirement Analysis

• Initial Requirements

- The initial requirements are based on the initial business model
- Then they are refined
- The requirements are dynamic—there are frequent changes
 - Maintain a list of likely requirements, together with use cases of requirements approved by the client

Workflow of Requirement Analysis

- Initial Requirements

- There are two categories of requirements
- A *functional requirement* specifies an action that the information system must be able to perform
 - Often expressed in terms of inputs and outputs
- A *nonfunctional requirement* specifies properties of the information system itself, such as
 - Platform constraints
 - Response times
 - Reliability

Workflow of Requirement Analysis

- Initial Requirements

- Functional requirements are handled as part of the requirements and analysis workflows
- Some nonfunctional requirements have to wait until the design workflow
 - The detailed information for some nonfunctional requirements is not available until the requirements and analysis workflows have been completed

Workflow of Requirement Analysis

- Initial Requirements

- For example, Buy a painting use case

Brief Description

The Buy a Painting use case enables Osbert Oglesby to buy a painting.

Step-by-Step Description

1. Osbert inputs details of the painting he is considering buying.
2. The information system responds with the maximum purchase price he should offer.
3. If the seller accepts Osbert's offer to buy the painting, Osbert enters further details.

Note: Details of the algorithm for determining the maximum price will be obtained later.

Agenda

1. Introduction
2. Recall Software Engineering
3. System Analysis and Design
4. Object-Oriented Concepts
5. UML
6. Workflow of Requirement Analysis
- 7. Workflow of Object-Oriented System Analysis and Design**

Workflow of Object-Oriented System Analysis and Design

- Analysis Workflow
- Design Workflow

Workflow of Object-Oriented System Analysis and Design

- Analysis Workflow
 - Extracting Entity Classes
 - Initial Functional Model
 - Initial Class diagram
 - Initial Dynamic Model
 - Extracting Boundary Classes
 - Extracting Control Classes
 - Refining the Use Cases
 - Use-Case Realization
 - Incrementing the Class Diagram

Workflow of Object-Oriented System Analysis and Design

• Classes

- The three class types in the Unified Process are
 - Entity classes
 - Boundary classes
 - Control classes
- UML notation



Workflow of Object-Oriented System Analysis and Design

• Entity Classes

- An *entity class* models information that is long lived
- Examples:
 - **Account Class** in a banking information system
 - **Painting Class** in the Osbert Oglesby information system
 - **Mortgage Class** and **Investment Class** in the MSG Foundation information system
- Instances of all these classes remain in the information system for years

Workflow of Object-Oriented System Analysis and Design

- Boundary Classes

- A *boundary class* models the interaction between the information system and its actors
- Boundary classes are generally associated with input and output
- Examples:
 - **Purchases Report Class** and **Sales Report Class** in the Osbert Oglesby information system
 - **Mortgage Listing Class** and **Investment Listing Class** in the MSG Foundation information system

Workflow of Object-Oriented System Analysis and Design

- Control Classes

- A *control class* models complex computations and algorithms
- Examples:
 - **Compute Masterpiece Price Class,**
 - **Compute Masterwork Price Class, and**
 - **Compute Other Painting Price Class**

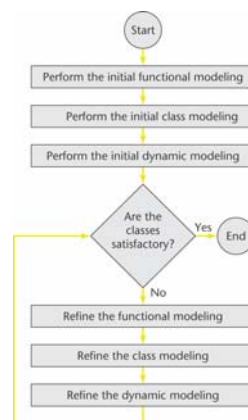
Workflow of Object-Oriented System Analysis and Design

• Extracting Entity Classes

- Entity class extraction consists of three steps that are carried out iteratively and incrementally:
 - Functional modeling
 - Present scenarios of all the use cases (a *scenario* is an instance of a use case)
 - Class modeling
 - Determine the entity classes and their attributes
 - Determine the interrelationships and interactions between the entity classes
 - Present this information in the form of a *class diagram*
 - Dynamic modeling
 - Determine the operations performed by or to each entity class
 - Present this information in the form of a *statechart*

Workflow of Object-Oriented System Analysis and Design

• Flowchart for Extracting Entity Classes



Workflow of Object-Oriented System Analysis and Design

- Extracting Entity Classes

- Functional modeling

- Present the scenarios

- For example,

Osbert Oglesby wishes to buy a masterpiece.

1. Osbert enters the description of the painting.
2. The information system scans the auction records to find the price and year of the sale of the most similar work by the same artist.
3. The information system computes the maximum purchase price by adding 8.5 percent, compounded annually, for each year since the auction of the most similar work.
Osbert makes an offer below the maximum purchase price—the offer is accepted by the seller.
4. Osbert enters sales information (name and address of seller, purchase price).

<inhon@mail.tku.edu.tw>

May 25, 2010 P 291

Workflow of Object-Oriented System Analysis and Design

- Extracting Entity Classes

- Class modeling

- Determine the entity classes and their attributes

- Determine the interrelationships and interactions between the entity classes

- Present this information in the form of a *class diagram*

<inhon@mail.tku.edu.tw>

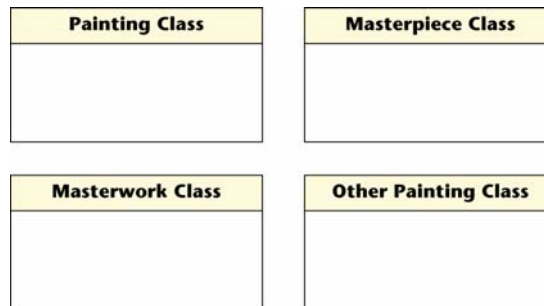
May 25, 2010 P 292

Workflow of Object-Oriented System Analysis and Design

- Extracting Entity Classes

- Class modeling

- For example,



<inhon@mail.tku.edu.tw>

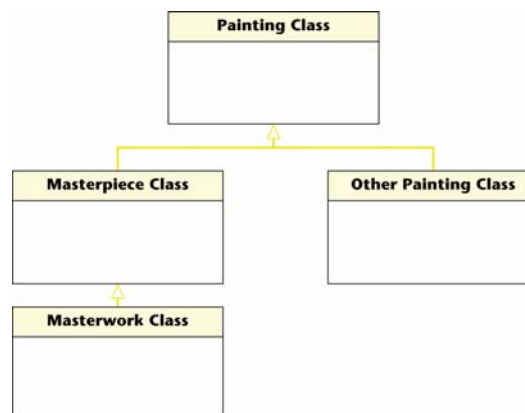
May 25, 2010 P 293

Workflow of Object-Oriented System Analysis and Design

- Extracting Entity Classes

- Class modeling

- For example
(Continuously),



<inhon@mail.tku.edu.tw>

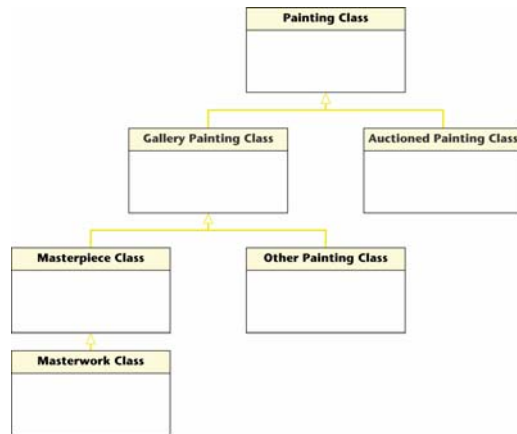
May 25, 2010 P 294

Workflow of Object-Oriented System Analysis and Design

- Extracting Entity Classes

- Class modeling

- For example (Continuously),



<inhon@mail.tku.edu.tw>

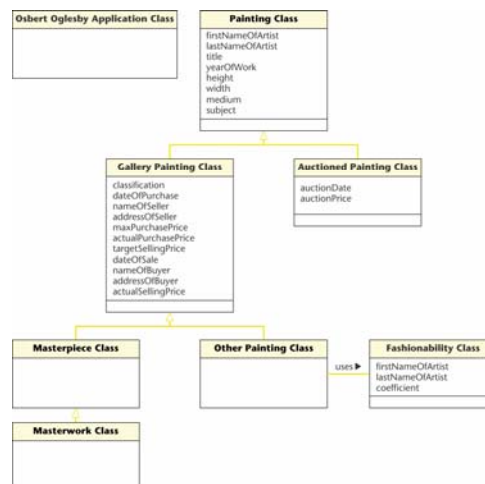
May 25, 2010 P 295

Workflow of Object-Oriented System Analysis and Design

- Extracting Entity Classes

- Class modeling

- For example (Continuously),



<inhon@mail.tku.edu.tw>

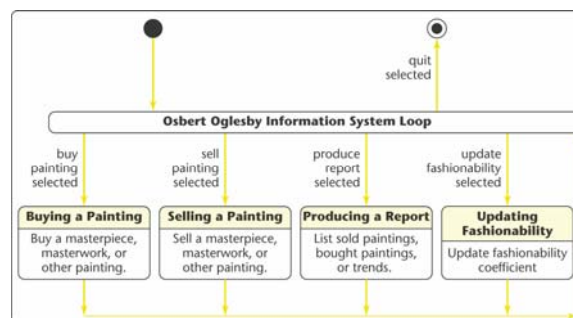
May 25, 2010 P 296

Workflow of Object-Oriented System Analysis and Design

- Extracting Entity Classes
 - Dynamic modeling
 - Determine the operations performed by or to each entity class
 - Present this information in the form of a *statechart*

Workflow of Object-Oriented System Analysis and Design

- Extracting Entity Classes
 - Dynamic modeling
 - For Example,



Workflow of Object-Oriented System Analysis and Design

- Extracting Boundary Classes

- Boundary Classes

- For Example,

**User Interface Class
Purchases Report Class
Sales Report Class
Future Trends Report Class**

Workflow of Object-Oriented System Analysis and Design

- Extracting Control Classes

- Control Classes

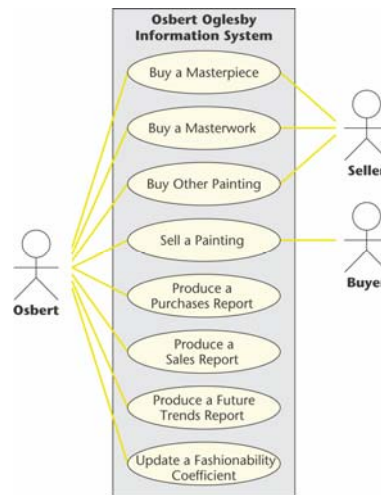
- For Example,

**Compute Masterpiece Price Class
Compute Masterwork Price Class
Compute Other Painting Price Class
Compute Future Trends Class**

Workflow of Object-Oriented System Analysis and Design

- Refining Use Cases

- For Example,



<inhon@mail.tku.edu.tw>

May 25, 2010 P 301

Workflow of Object-Oriented System Analysis and Design

- Use Case Realization

- The process of extending and refining use cases is called *use-case realization*
 - In the phrase “realize a use case,” the word “realize”,
 - It means to accomplish (or achieve) the use case
 - The realization of a specific scenario of a use case is depicted using an interaction diagram
 - Either a sequence diagram or collaboration diagram

<inhon@mail.tku.edu.tw>

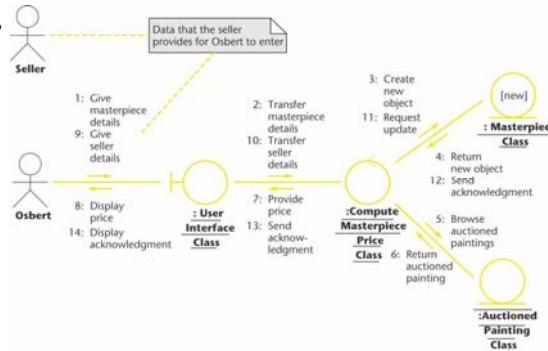
May 25, 2010 P 302

Workflow of Object-Oriented System Analysis and Design

- Use Case Realization

- A collaboration diagram

- For example,



<inhon@mail.tku.edu.tw>

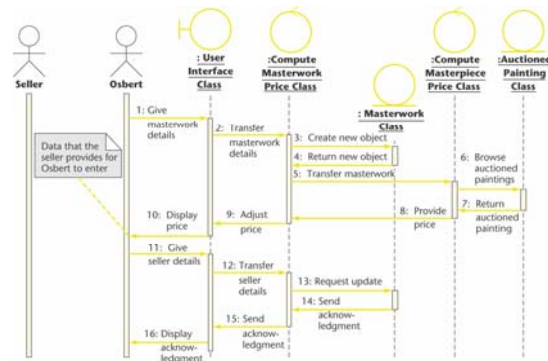
May 25, 2010 P 303

Workflow of Object-Oriented System Analysis and Design

- Use Case Realization

- A sequence diagram

- For example,



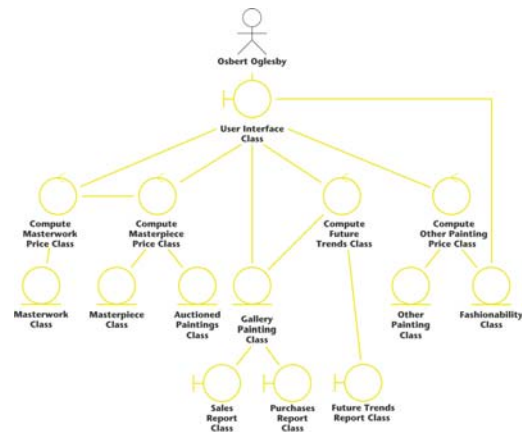
<inhon@mail.tku.edu.tw>

May 25, 2010 P 304

Workflow of Object-Oriented System Analysis and Design

• Final Class Diagram

– For example,



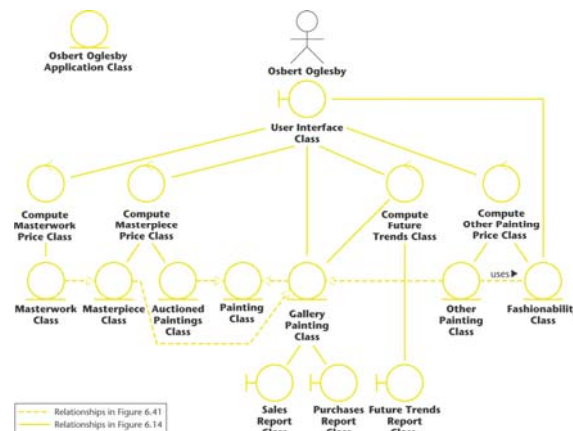
<inhon@mail.tku.edu.tw>

May 25, 2010 P 305

Workflow of Object-Oriented System Analysis and Design

• Final Class Diagram

– For example,



Relationships in Figure 6.14
Relationships in Figure 6.14

<inhon@mail.tku.edu.tw>

May 25, 2010 P 306

Workflow of Object-Oriented System Analysis and Design

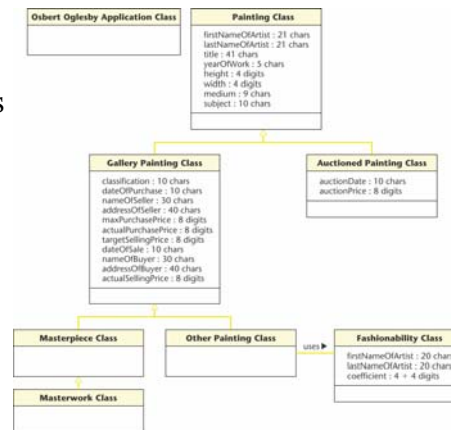
- Design Workflow
 - Formats of the Attributes
 - Allocation of Operations to Classes
 - CRC Cards
 - Class-Responsibility-Collaboration (CRC)

<inhon@mail.tku.edu.tw>

May 25, 2010 P 307

Workflow of Object-Oriented System Analysis and Design

- Design Workflow
 - Formats of the Attributes
 - For example,



<inhon@mail.tku.edu.tw>

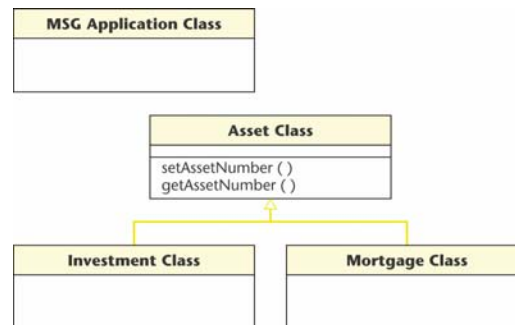
May 25, 2010 P 308

Workflow of Object-Oriented System Analysis and Design

- Design Workflow

- Allocation of Operations to Classes

- For example,



<inhon@mail.tku.edu.tw>

May 25, 2010 P 309

Workflow of Object-Oriented System Analysis and Design

- Design Workflow

- CRC cards

- For example,

CLASS	
Mortgage Class	
RESPONSIBILITY	COLLABORATION
Compute estimated grants and payments for week	Estimate Funds for Week Class
Initialize, update, and delete mortgages	Manage an Asset Class
Generate list of mortgages	User Interface Class
Print list of mortgages	Mortgages Report Class

<inhon@mail.tku.edu.tw>

May 25, 2010 P 310