Software Engineering Knowledge

- *You often hear people say that software development knowledge has a 3-year half-life: half of what you need to know today will be obsolete within 3 years. In the domain of technology-related knowledge, that's probably about right. But there is another kind of software development knowledge—a kind that I think of as "software engineering principles"—that does not have a three-year half-life. These software engineering principles are likely to serve a professional programmer throughout his or her career.*

Steve McConnell

Principles that Guide Process - I

- **Principle #1. *Be agile.*** Whether the process model you choose is prescriptive or agile, the basic tenets of agile development should govern your approach.
- **Principle #2. *Focus on quality at every step.*** The exit condition for every process activity, action, and task should focus on the quality of the work product that has been produced.
- **Principle #3. *Be ready to adapt.*** Process is not a religious experience and dogma has no place in it. When necessary, adapt your approach to constraints imposed by the problem, the people, and the project itself.
- **Principle #4. *Build an effective team.*** Software engineering process and practice are important, but the bottom line is people. Build a self-organizing team that has mutual trust and respect.

Principles that Guide Process - II

- **Principle #5. *Establish mechanisms for communication and coordination.*** Projects fail because important information falls into the cracks and/or stakeholders fail to coordinate their efforts to create a successful end product.
- **Principle #6. *Manage change.*** The approach may be either formal or informal, but mechanisms must be established to manage the way changes are requested, assessed, approved and implemented.
- **Principle #7. *Assess risk.*** Lots of things can go wrong as software is being developed. It's essential that you establish contingency plans.
- **Principle #8. *Create work products that provide value for others.*** Create only those work products that provide value for other process activities, actions or tasks.

Principles that Action Practice

- **Principle #1. *Divide and conquer.*** Stated in a more technical manner, analysis and design should always emphasize *separation of concerns* (SoC).
- **Principle #2. *Understand the use of abstraction.*** At it core, an abstraction is a simplification of some complex element of a system used to communication meaning in a single phrase.
- **Principle #3. Strive for consistency.** A familiar context makes software easier to use.
- **Principle #4. *Focus on the transfer of information.*** Pay special attention to the analysis, design, construction, and testing of interfaces.

Principles that Action Practice

- **Principle #5. *Build software that exhibits effective modularity.*** Separation of concerns (Principle #1) establishes a philosophy for software. *Modularity* provides a mechanism for realizing the philosophy.
- **Principle #6. *Look for patterns.*** Brad Appleton [App00] suggests that: “The goal of patterns within the software community is to create a body of literature to help software developers resolve recurring problems encountered throughout all of software development.
- **Principle #7. *When possible, represent the problem and its solution from a number of different perspectives.***
- **Principle #8. *Remember that someone will maintain the software.***

Communication Principles

- **Principle #1.** ***Listen.*** Try to focus on the speaker's words, rather than formulating your response to those words.
- **Principle # 2.** ***Prepare before you communicate.*** Spend the time to understand the problem before you meet with others.
- **Principle # 3.** ***Someone should facilitate the activity.*** Every communication meeting should have a leader (a facilitator) to keep the conversation moving in a productive direction; (2) to mediate any conflict that does occur, and (3) to ensure than other principles are followed.
- **Principle #4.** ***Face-to-face communication is best.*** But it usually works better when some other representation of the relevant information is present.

Communication Principles

- **Principle # 5. *Take notes and document decisions.*** Someone participating in the communication should serve as a "recorder" and write down all important points and decisions.
- **Principle # 6. *Strive for collaboration.*** Collaboration and consensus occur when the collective knowledge of members of the team is combined …
- **Principle # 7. *Stay focused, modularize your discussion.*** The more people involved in any communication, the more likely that discussion will bounce from one topic to the next.
- **Principle # 8. *If something is unclear, draw a picture.***
- **Principle # 9. *(a) Once you agree to something, move on; (b) If you can't agree to something, move on; (c) If a feature or function is unclear and cannot be clarified at the moment, move on.***
- **Principle # 10. *Negotiation is not a contest or a game. It works best when both parties win.***

Planning Principles

- **Principle #1. *Understand the scope of the project.*** It's impossible to use a roadmap if you don't know where you're going. Scope provides the software team with a destination.
- **Principle #2. *Involve the customer in the planning activity.*** The customer defines priorities and establishes project constraints.
- **Principle #3. *Recognize that planning is iterative.*** A project plan is never engraved in stone. As work begins, it very likely that things will change.
- **Principle #4. *Estimate based on what you know.*** The intent of estimation is to provide an indication of effort, cost, and task duration, based on the team's current understanding of the work to be done.

Planning Principles

- **Principle #5.** ***Consider risk as you define the plan.*** If you have identified risks that have high impact and high probability, contingency planning is necessary.
- **Principle #6.** ***Be realistic.*** People don't work 100 percent of every day.
- **Principle #7.** ***Adjust granularity as you define the plan.*** *Granularity* refers to the level of detail that is introduced as a project plan is developed.
- **Principle #8.** ***Define how you intend to ensure quality.*** The plan should identify how the software team intends to ensure quality.
- **Principle #9.** ***Describe how you intend to accommodate change.*** Even the best planning can be obviated by uncontrolled change.
- **Principle #10.** ***Track the plan frequently and make adjustments as required.*** Software projects fall behind schedule one day at a time.

Requirements Modeling Principles

- **Principle #1.** ***The information domain of a problem must be represented and understood.***
- **Principle #2.** ***The functions that the software performs must be defined.***
- **Principle #3.** ***The behavior of the software (as a consequence of external events) must be represented.***
- **Principle #4.** ***The models that depict information, function, and behavior must be partitioned in a manner that uncovers detail in a layered (or hierarchical) fashion.***
- **Principle #5.** ***The analysis task should move from essential information toward implementation detail.***

Requirements Engineering-I

- Inception (建立) —ask a set of questions that establish …
  - basic understanding of the problem
  - the people who want a solution
  - the nature of the solution that is desired, and
  - the effectiveness of preliminary communication and collaboration between the customer and the developer
- Elicitation (啟發)—elicit requirements from all stakeholders
- Elaboration (制定)—create an analysis model that identifies data, function and behavioral requirements
- Negotiation (談判) —agree on a deliverable system that is realistic for developers and customers

Requirements Engineering-II

- Specification (規格)—can be any one (or more) of the following:
  - A written document
  - A set of models
  - A formal mathematical
  - A collection of user scenarios (use-cases)
  - A prototype
- Validation (驗證)—a review mechanism that looks for
  - errors in content or interpretation
  - areas where clarification may be required
  - missing information
  - inconsistencies (a major problem when large products or systems are engineered)
  - conflicting or unrealistic (unachievable) requirements.
- Requirements management

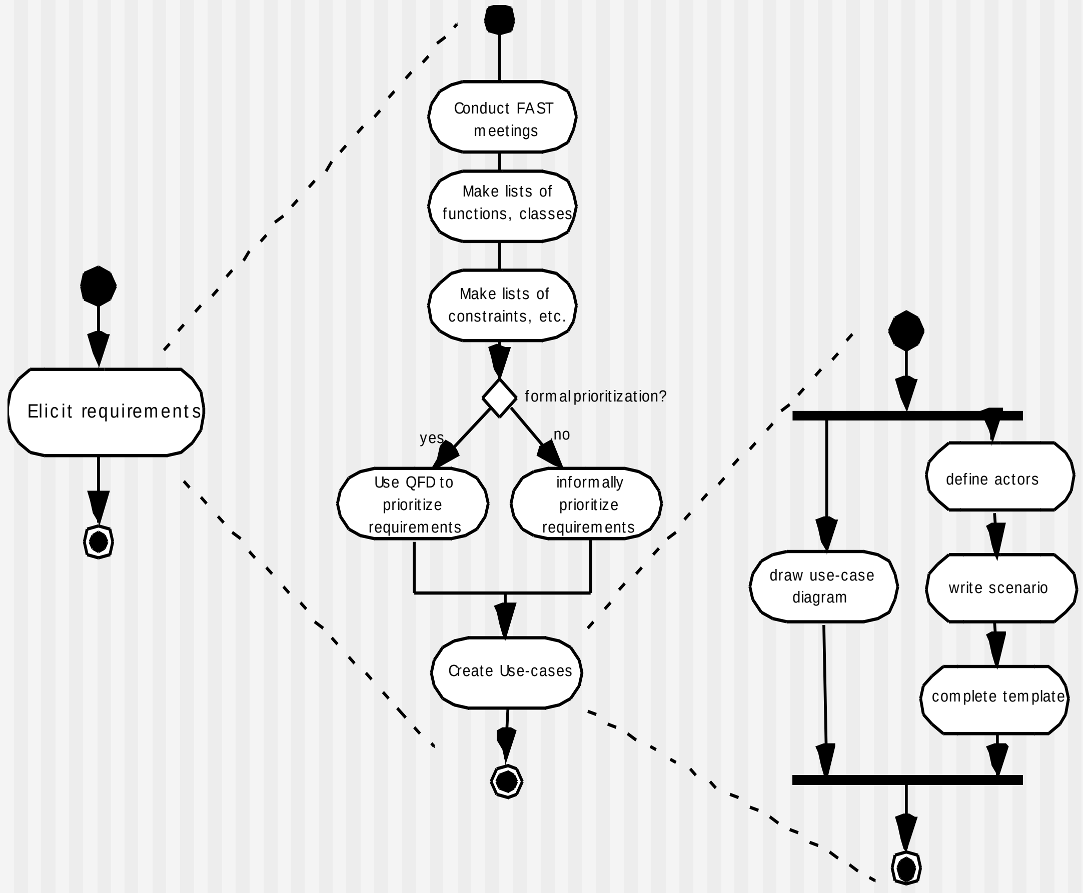
Inception

- Identify stakeholders
  - “who else do you think I should talk to?”
- Recognize multiple points of view
- Work toward collaboration
- The first questions
  - Who is behind the request for this work?
  - Who will use the solution?
  - What will be the economic benefit of a successful solution
  - Is there another source for the solution that you need?

Eliciting Requirements

- meetings are conducted and attended by both software engineers and customers
- rules for preparation and participation are established an agenda is suggested
- a "facilitator" (can be a customer, a developer, or an outsider) controls the meeting
- a "definition mechanism" (can be work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room or virtual forum) is used
- the goal is
  - to identify the problem
  - propose elements of the solution
  - negotiate different approaches, and
  - specify a preliminary set of solution requirements

Eliciting Requirements



Quality Function Deployment

- Function deployment determines the “value” (as perceived by the customer) of each function required of the system
- Information deployment identifies data objects and events
- Task deployment examines the behavior of the system
- Value analysis determines the relative priority of requirements

Elicitation Work Products

- a statement of need and feasibility.
- a bounded statement of scope for the system or product.
- a list of customers, users, and other stakeholders who participated in requirements elicitation
- a description of the system’s technical environment.
- a list of requirements (preferably organized by function) and the domain constraints that apply to each.
- a set of usage scenarios that provide insight into the use of the system or product under different operating conditions.
- any prototypes developed to better define requirements**.**

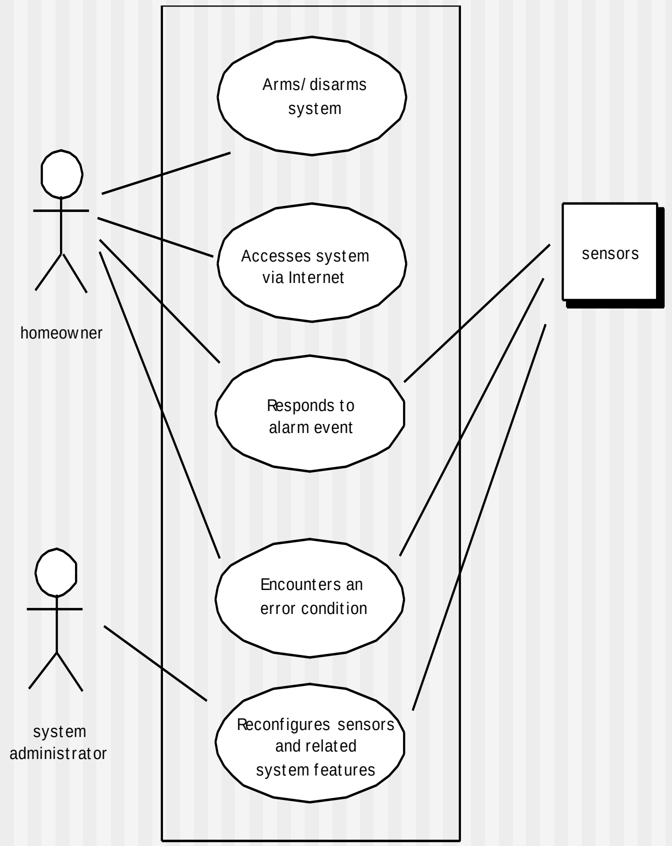
Building the Analysis Model

- Elements of the analysis model
  - Scenario-based elements
    - Functional—processing narratives for software functions
    - Use-case—descriptions of the interaction between an “actor” and the system
  - Class-based elements
    - Implied by scenarios
  - Behavioral elements
    - State diagram
  - Flow-oriented elements
    - Data flow diagram

Use-Cases

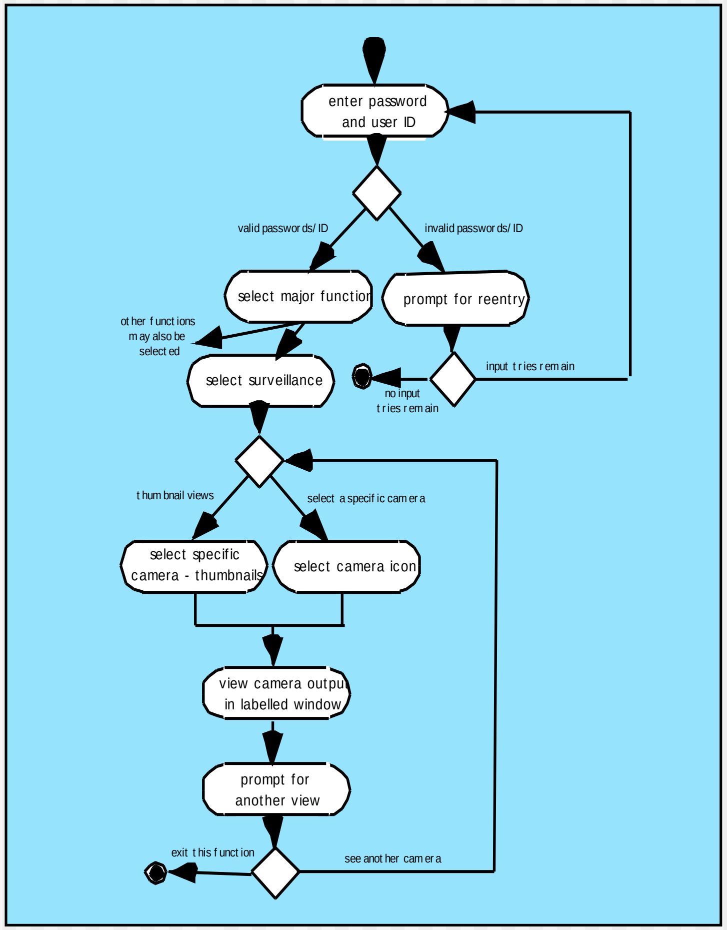
- A collection of user scenarios that describe the thread of usage of a system
- Each scenario is described from the point-of-view of an “actor”—a person or device that interacts with the software in some way
- Each scenario answers the following questions:
  - Who is the primary actor, the secondary actor (s)?
  - What are the actor’s goals?
  - What preconditions should exist before the story begins?
  - What main tasks or functions are performed by the actor?
  - What extensions might be considered as the story is described?
  - What variations in the actor’s interaction are possible?
  - What system information will the actor acquire, produce, or change?
  - Will the actor have to inform the system about changes in the external environment?
  - What information does the actor desire from the system?
  - Does the actor wish to be informed about unexpected changes?

Use-Case Diagram

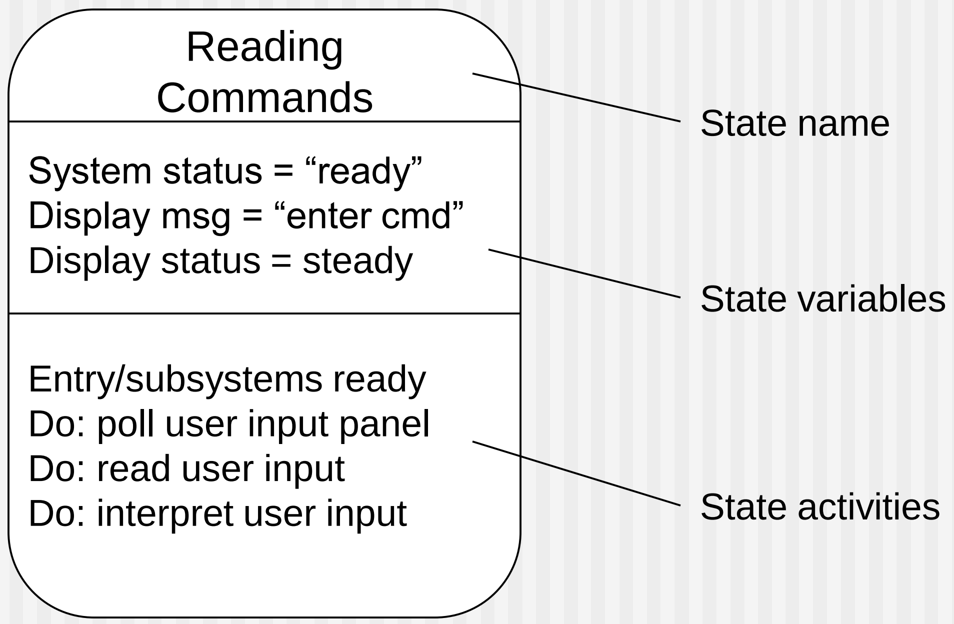

Arms/ disarms system
Accesses system via Internet
Responds to alarm event
Encounters an error condition
Reconfigures sensors and related system features
homeowner
system administrator
sensors

Activity Diagram

Supplements the use case by providing a graphical representation of the flow of interaction within a specific scenario

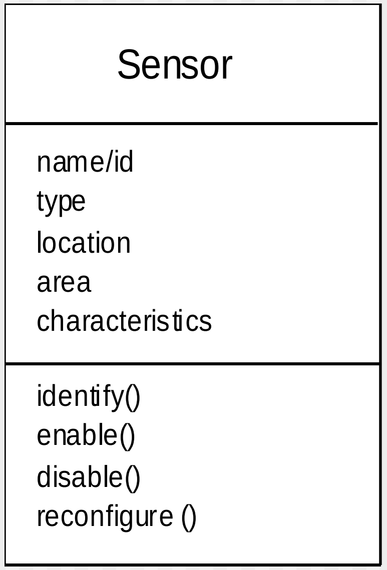


State Diagram

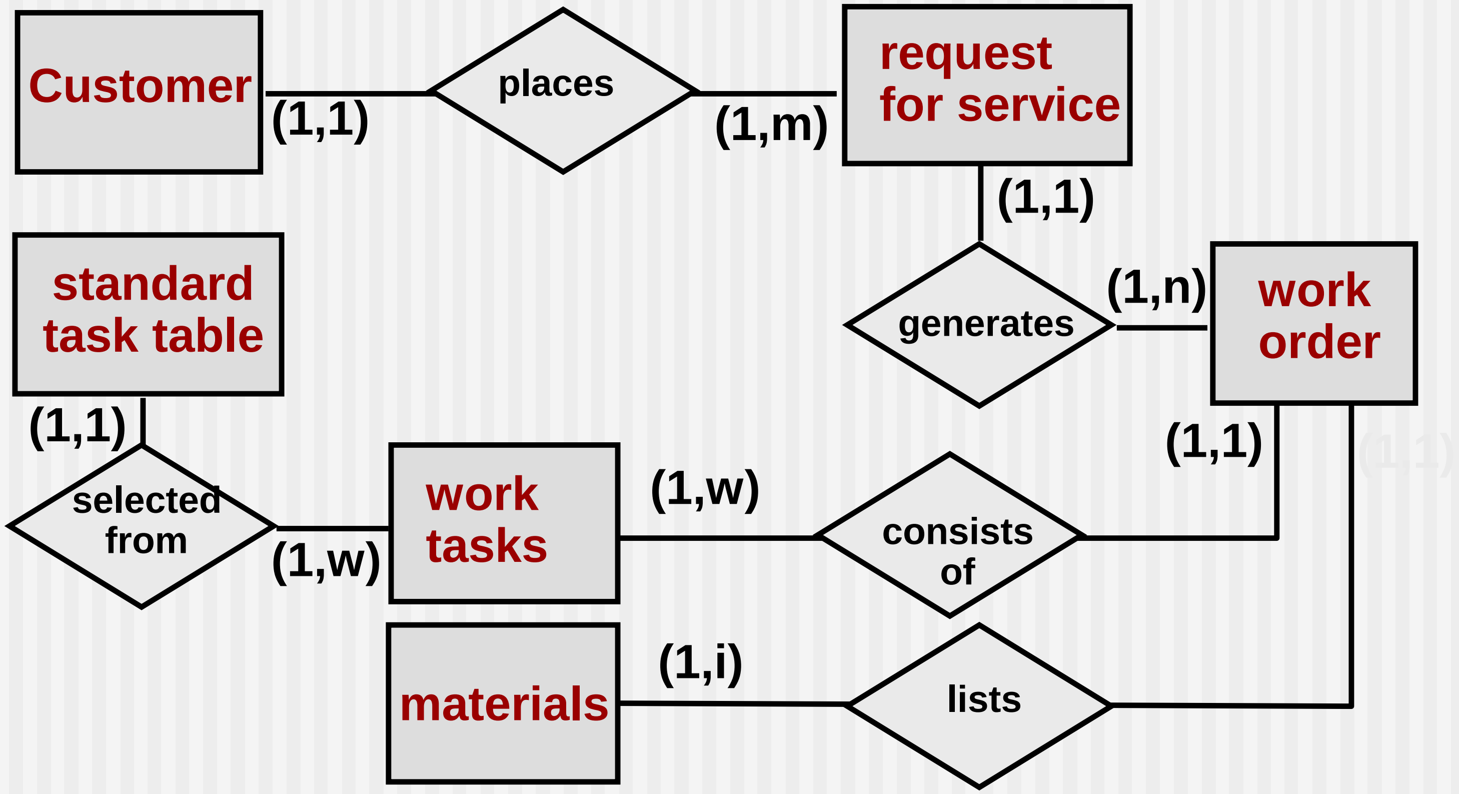


Class Diagram

From the *SafeHome* system …



The ERD: An Example



Negotiating Requirements

- Identify the key stakeholders
  - These are the people who will be involved in the negotiation
- Determine each of the stakeholders "win conditions"
  - Win conditions are not always obvious
- Negotiate
  - Work toward a set of requirements that lead to "win-win"

Validating Requirements - I

- Is each requirement consistent with the overall objective for the system/product?
- Have all requirements been specified at the proper level of abstraction? That is, do some requirements provide a level of technical detail that is inappropriate at this stage?
- Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?
- Is each requirement bounded and unambiguous?
- Does each requirement have attribution? That is, is a source (generally, a specific individual) noted for each requirement?
- Do any requirements conflict with other requirements?

Validating Requirements - II

- Is each requirement achievable in the technical environment that will house the system or product?
- Is each requirement testable, once implemented?
- Does the requirements model properly reflect the information, function and behavior of the system to be built.
- Has the requirements model been “partitioned” in a way that exposes progressively more detailed information about the system.
- Have requirements patterns been used to simplify the requirements model. Have all patterns been properly validated? Are all patterns consistent with customer requirements?

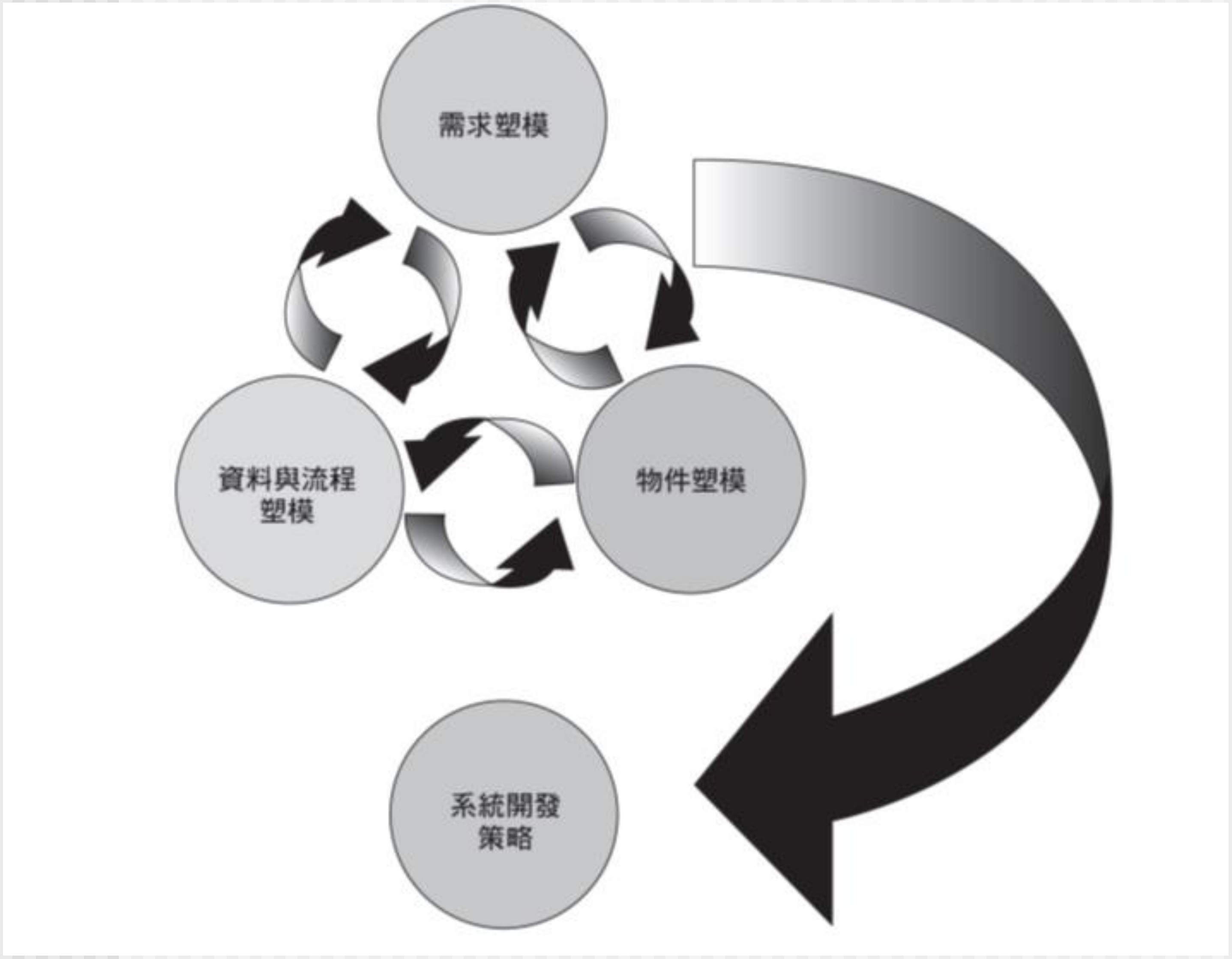
需求擷取與分析

系統分析階段綜述

- 一般而言，系統分析的目的是了解提議中的計畫，確定它滿足商業需求，並且為系統開發建構紮實的基礎。
- 我們將運用塑模以及文件化的工具，以視覺化或文字來描繪提案的系統。

系統分析階段綜述

- 系統分析的**活動**
  - 需求塑模
    - **輸入(inputs)**
    - **輸出(outputs)**
    - **流程(process)**
    - **性能(performance)**
    - **安全性(security)**
  - 資料與流程塑模
  - 物件塑模
  - 開發策略
  - 系統需求文件 **(system requirements document)**


需求塑模
資料與流程
塑模
物件塑模
系統開發
策略

系統分析階段綜述

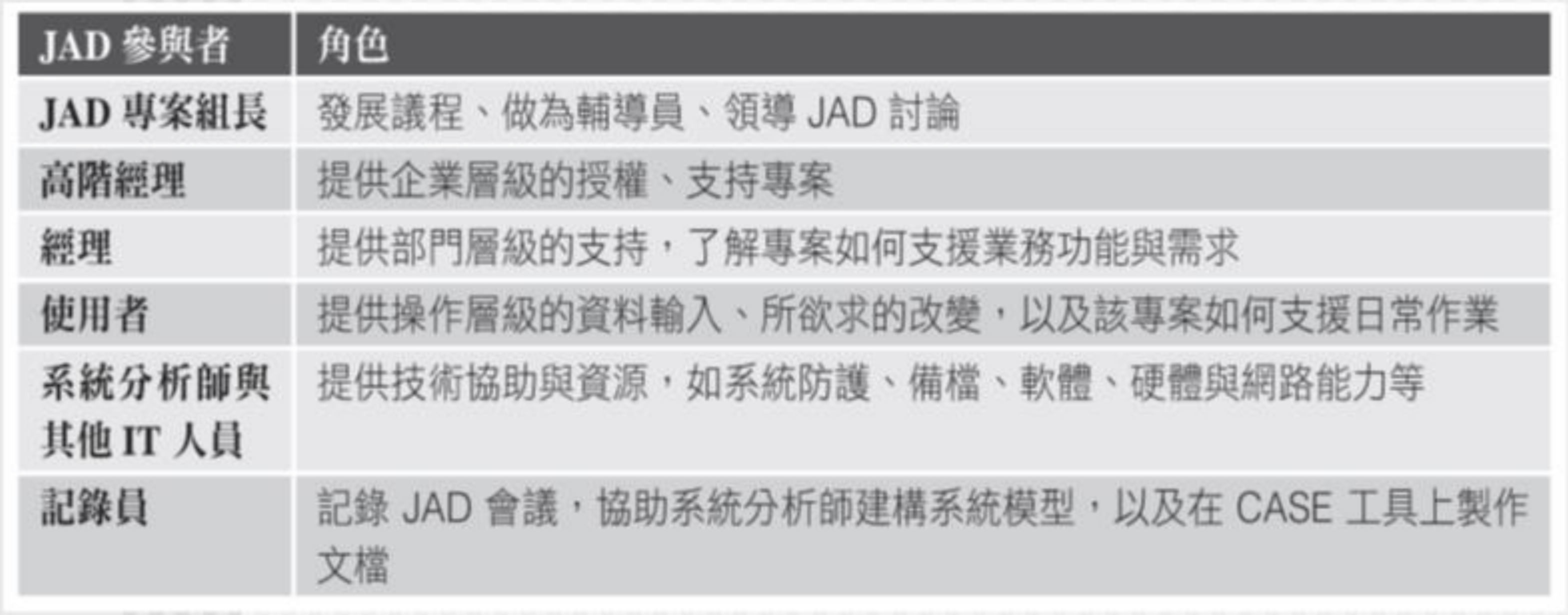
- 系統分析**技巧**
  - 分析能力 (analytical skills)
  - 人際能力 (interpersonal skills)
- **團隊技巧**：
  - 聯合應用開發 (joint application development, JAD)
  - 快速應用開發 (rapid application development, RAD)
  - 快捷法 (agile methods)

聯合應用開發(JAD)

- 是一種用來實情調查與建構需求模型的使用者導向技術。將使用者視為開發過程中的實際參與者。
- 使用者涉入
  - 使用者在資訊系統中扮演舉足輕重的角色，所以應該在系統開發過程中全程參與。
  - 成功的系統必須是使用者導向而且使用者需包含在內。

聯合應用開發(JAD)

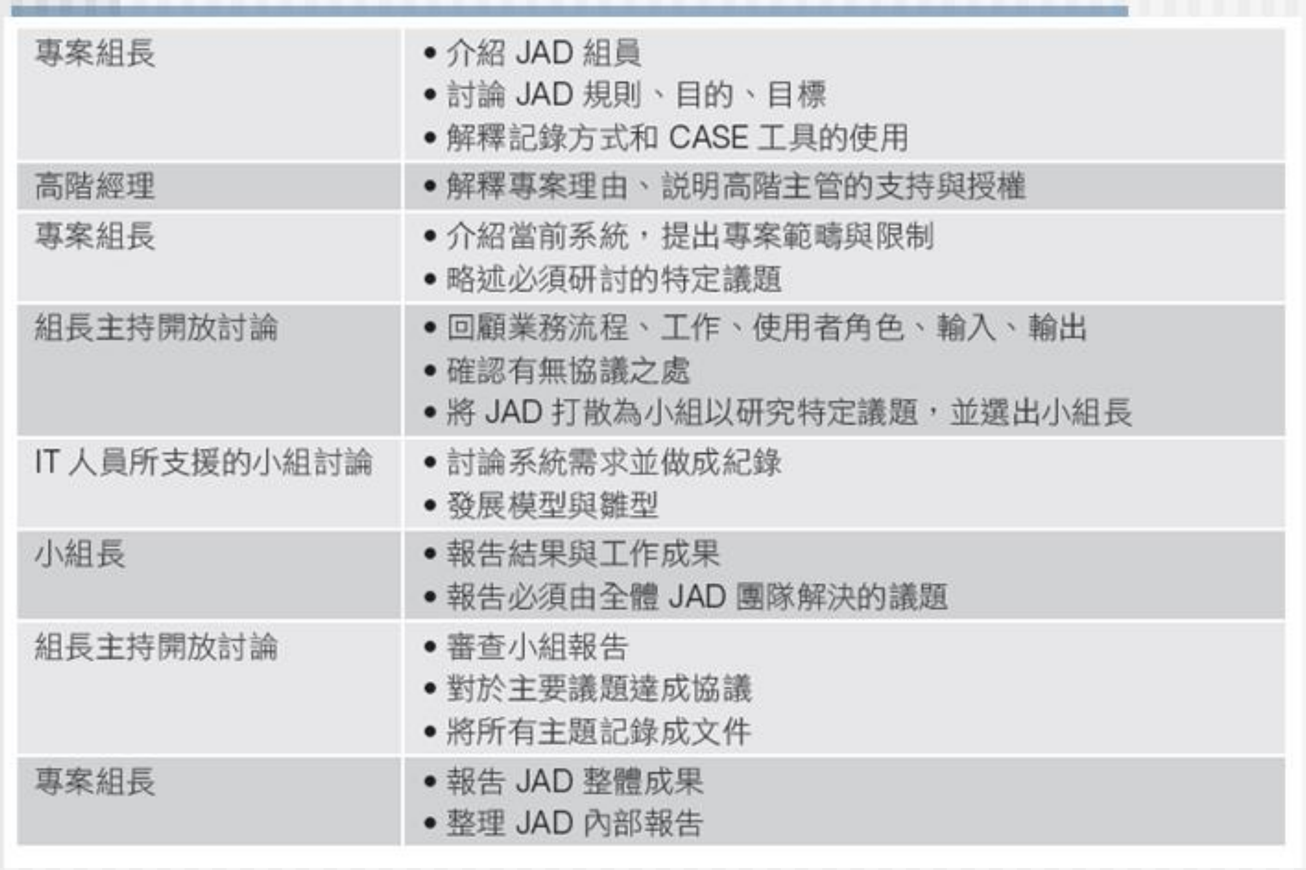
- JAD 參與者與角色



JAD 參與者	角色
JAD 專案組長	發展議程、做為輔導員、領導 JAD 討論
高階經理	提供企業層級的授權、支持專案
經理	提供部門層級的支持，了解專案如何支援業務功能與需求
使用者	提供操作層級的資料輸入、所欲求的改變，以及該專案如何支援日常作業
系統分析師與其他 IT 人員	提供技術協助與資源，如系統防護、備檔、軟體、硬體與網路能力等
記錄員	記錄 JAD 會議，協助系統分析師建構系統模型，以及在 CASE 工具上製作文檔

聯合應用開發(JAD)

- JAD 的開會議程



專案組長	• 介紹 JAD 組員 • 討論 JAD 規則、目的、目標 • 解釋記錄方式和 CASE 工具的使用
高階經理	• 解釋專案理由、説明高階主管的支持與授權
專案組長	• 介紹當前系統，提出專案範疇與限制 • 略述必須研討的特定議題
組長主持開放討論	• 回顧業務流程、工作、使用者角色、輸入、輸出 • 確認有無協議之處 • 將 JAD 打散為小組以研究特定議題，並選出小組長
IT 人員所支援的小組討論	• 討論系統需求並做成紀錄 • 發展模型與雛型
小組長	• 報告結果與工作成果 • 報告必須由全體 JAD 團隊解決的議題
組長主持開放討論	• 審查小組報告 • 對於主要議題達成協議 • 將所有主題記錄成文件
專案組長	• 報告 JAD 整體成果 • 整理 JAD 內部報告

聯合應用開發(JAD)

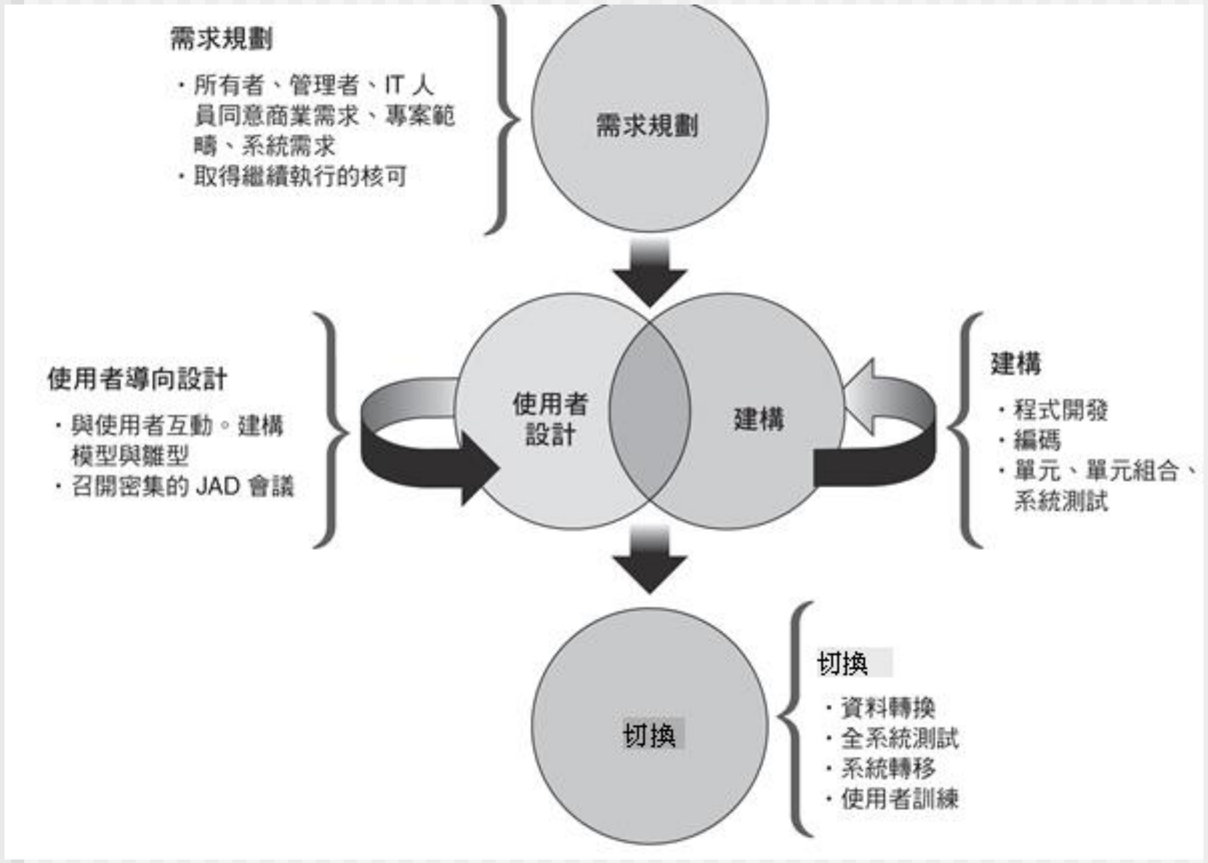
- JAD 的優劣
  - 和傳統方法相比，JAD 成本較高。
  - 可以讓相關人等在建構需求模型時能夠有效地參與。
  - JAD 若使用得當，將能夠產生比較正確的系統需求，也能讓大家了解系統的目的，提高對新系統的投入程度。

快速應用開發(RAD)

- 使用者會參與其中的各個環節
- JAD的最終產品是一個需求模型；RAD則是一套新的資訊系統。
- 和JAD不同的是，RAD提供完整系統開發的快速方法。

快速應用開發(RAD)

- RAD 階段與工作



快速應用開發(RAD)

- RAD 目標
  - RAD的主要目標是讓使用者參與系統開發的各個階段，以縮短開發時間與支出。
  - 成功的 RAD 團隊還必須擁有 IT 資源、技巧及管理層的支持。
  - 非常適合需要大量使用者互動的系統。

快速應用開發(RAD)

- RAD 的優劣
  - 優點是可以用更快且成本更低的方式開發系統。
  - 缺點是過於注重系統本身，而可能會忽略公司的策略需求。
  - 可能因為倉促行事而無法發展出品質、整體性、設計的標準。

快捷法 (Agile)

- 以累加的方法建構出一系列雛型。
- 包括了一系列的快捷法工具。
- 有些快捷法開發師 (agile developer) 完全不用 CASE 工具，他們認為這樣可以強化快捷法所訴求的目標：簡單、迅速、彈性及使用者取向。

快捷法

- 快捷法的優劣
  - 快捷法有彈性、有效率，適合處理變化。
  - 能夠不斷確認專業的正確性並且降低風險。
  - 團隊成員個個都必須具備極好的技術與人際能力。
  - 專案範疇也可能因為使用者需求的變化而變動。

需求擷取

- 分析階段的首要工作
  - 分析階段的首要工作在於需求的擷取與分析。
  - 根據許多的研究顯示，一個計畫之所以會失敗的原因之一常常是肇因於計畫初期，對於即將開發之系統所應提供的功能沒有切確的了解與掌握。基本上，一個無法滿足使用者需求的系統，不論所採用之資訊技術為何，其最終的命運均可想而知。
- 領域分析
  - 需求擷取可以從兩方面來著手。首先，你必須要從事領域分析(domain analysis)的工作。根據卡內基美隆(Carnegie Mellon)軟體工程學院對於領域分析所下的定義：
    - 根據既有系統及其開發的歷史、領域專家的知識，背後的理論，辨別，收集以及組織相關資訊的過程。
  - 簡單地說就是也就是從各方面對目前既有的系統做資料蒐集。

需求擷取

- 蒐集範圍與種類
  - 在資料搜集的過程中，不要限定範圍或是種類，舉凡是跟領域問題有關聯的都要蒐集起來。任何資料都能讓你對於計劃領域有更深一層的了解。
  - 針對特定的需求，決定你要蒐集的方向，也就是跟計畫密切相關的資料。
- 領域專家
  - 領域分析是系統開發前的預備知識。對於待開發產品之背景資料分析與蒐集、對於產品領域知識與操作越了解，越能在將來做出好的決策。
  - 領域分析過程中，你會接觸到許多的人。對領域有深度了解的人稱為領域專家(Domain expert)，這些人大都是計畫的客戶、既有系統的使用者等等。他們對企業的各項活動以及業務流程最熟悉。而這些人也將在系統開發的分析階段扮演著重要的腳色。

需求擷取

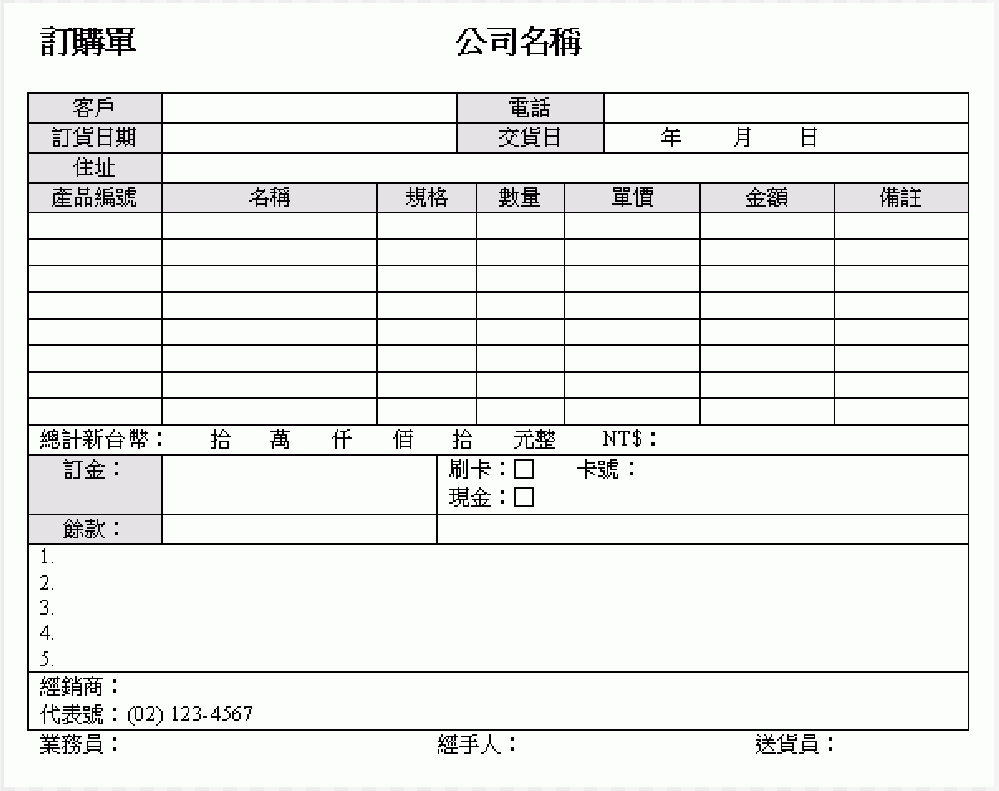
- 了解領域知識的好處
  - 可以有效率的和客戶溝通
    - 一般來說，從事領域分析的大多是系統分析師。參予計畫的每一個成員都必須對相關的領域有基本的認識與了解。因為這可以增進開發團隊與客戶之間的溝通。
  - 了解企業既有的(As-is)流程活動，對於接下來即將(To-be)開發的系統會有實質的幫助。
    - 這一點，尤其對經常變動的企業邏輯 (business logic)或是商業規則(business rule)這方面尤其重要。
  - 留未來新系統的開發空間
    - 從系統開發顧問公司的角度來看，在你為客戶開發系統的同時，你會對其業務有更深一層的了解。你甚至會了解到，客戶的許多既有的企業業務流程可以利用資訊系統的優點來改進，幫助企業改進執行的效率、工作生產力、降低成本開銷等等。對於這些觀察，可以經由與客戶高階執行階層討論與提出建議，以進而向客戶提出計畫書。為另一個新的計畫做準備。

擷取方式

- 了解了需求擷取的重要性，我們接著來看一下需求擷取有哪些方式。需求擷取的方式有很多，比較常用的方式如下
  - 企業既有的報表、表單、操作流程相關文件
  - 訪談
  - 小組討論
  - 腦力激盪

擷取方式

- 既有的報表、表單
  - 對於任何與開發中計畫相關之報表，表單之搜集。這些資料的主要來源為企業內既有的文件、業務流程說明、工作內容、作業描述等等。仔細閱讀這些資料以了解企業的核心流程，企業功能以及企業規則，幫助自己建立對該企業的一個普遍認識。對於人工作業時所採用之單據更需要仔細蒐集和保留。



訂購單 **公司名稱**

客戶			電話			
訂貨日期			交貨日	年 月 日		
住址						
產品編號	名稱	規格	數量	單價	金額	備註
總計新台幣： 拾 萬 仟 佰 拾 元整 NT$：						
訂金：			刷卡：☐ 卡號： 現金：☐			
餘款：						

1.
2.
3.
4.
5.

經銷商：
代表號：(02) 123-4567

業務員： 經手人： 送貨員：

擷取方式

- 訪談
  - 訪談是需求擷取方法中相當常用的一種方式。
  - 透過訪談你可以跟使用者有面對面討論與溝通機會，對於使用者的需求能夠有較真實的體驗。
  - 訪談之前必須要有詳細規劃，訪談之後必須將其結果詳實地紀錄下來。

擷取方式

- 訪談
  - 第一步：決定訪談對象
    - **非正式結構(informal structure)**
  - 第二步：建立訪談目的
    - 決定訪談的範圍
    - 確定需了解哪些事項
  - 第三步：設計訪談問題
    - 對訪談的問題進行標準化可以讓訪談有所依循，也避免節外生枝。
    - 須避免所謂的**導引式問題 (leading questions)**
    - **開放式問題(open-ended questions)**
    - **封閉式問題(closed-ended questions)**
    - **範圍式問題(range-of-response questions)**

- 訪談
  - 第四步：準備訪談
    - 訪談是很重要的會晤，不是閒聊，因此準備工作十分重要。
    - 最好限於一個小時之內完成。
    - 在訪談數天前必須將訪談主題寄給受訪者。
    - 如果想要查看資料，也應該告知受訪者事前備妥。
  - 第五步：執行訪談
    - 在訪談時先做自我介紹、描述專案內容，並解釋訪談目的。
    - **專注傾聽 (engaged listening)** 。
    - 給對方足夠時間思考。
    - 在所有問題問完之後，應該彙整訪談重點，並解釋接下來會做什麼。例如，寄送下次的訪談通知。
    - 訪談結束時，要感謝對方，並鼓勵對方如果有新的資訊或任何問題都與我們聯繫，也可以詢問對方是否有其他主題可以討論。

- 訪談
  - 第六步：記錄訪談
    - 最好盡量不要記筆記，筆記只做為簡單提示。
    - 在做完訪談之後，就必須馬上做記錄，並且評析所蒐集到的資訊。
    - 訪談之後可以寄感謝卡，載明訪談時間、地點、目的與主題，這樣受訪者就可以提供其意見。
  - 第七步：評量訪談
    - 除了獲取資訊之外，訪談時也要注意訪談是否有所失當。

- 失敗的訪談：讓訪談有個目標，這樣才不至於讓訪談變成聊天 。

擷取方式

- 小組定期開會討論
  - 定期開會討論與訪談的方式很類似。
  - 系統開發的過程中通常都會訂定有每週定期的開會時間表用以
    - 檢視計畫的進度
    - 確認工作細節
    - 確認需求的正確性
    - 各項相關任務的分派
    - 人力資源的調度等等事項。
  - 腦力激盪(brainstorming)
  - 針對特定問題、機會與議題的小組討論。
  - 此方法可以鼓勵新的構想、允許團隊參與、讓成員的想法與輸入彼此互為所用。
  - 結構化與非結構化腦力激盪。

擷取方式

- 其他需求擷取的方式還包括有：
  - 觀察
    - 指實地觀察系統運作。它可以讓我們對系統有更進一步的了解。
    - 先準備好觀察項目以及所欲詢問的問題清單。
  - 問卷調查
    - 設計問卷最重要的是確定我們得到的資訊可以用做未來實情調查之用。
  - 抽樣調查
    - **系統抽樣 (systematic sample)** 。
    - **階層抽樣 (stratified sample)**。
    - **隨機抽樣 (random sample)** 。
    - 抽樣的主要目的是確保樣本能代表整個母體。

需求分析

- 利用各種需求擷取的方式，你獲得了許多跟即將開發的系統相關的資料。這些資料可能很凌亂，雜亂無章，因此你必須將這些資料做整理與分析。
- 需求的種類
  - 對於所擷取之需求資料可以分為兩大類：
    - 功能的需求 (functional requirement)
    - 非功能的需求(non-functional requirement)

需求分析

- 功能的需求
  - 功能的需求主要是在描述系統該做什麼。也就是系統要提供給使用者的服務項目。
  - 對於系統所提供之功能的描述可以包括什麼樣的輸入是這個功能所必須的、這個功能的處理流程與步驟、以及經過資料處理後，這個功能的輸出為何等等。
- 非功能的需求
  - 非功能的需求是指跟系統的執行效率，效能之需求，且可以量度的(measurable)的項目。下面列舉之非功能性需求參考項目：

- 參考項目
  - 反應時間(response time) ：對於使用者所觸發之事件的執行所將花費之時間。
  - 使用性(usability)：描述對於一個正常的使用者所需之訓練時間。
  - 可靠度(reliability)：可靠度可包含許多要項，最常見的是描述系統的失敗率。
  - 效能(performance)：表現度可包含許多要項，最常見的是描述系統在每秒鐘可以處理的交易量。
  - 維護性(maintainability)：描述任何可以增進系統維護之相關項目。比如說，編碼的準則，命名的標準等等。

需求分析

- 需求分析描述
  - 對於需求擷取所獲得的資料，你應該定義出所要解決的問題。定義問題的原則是：儘量使用淺顯易懂的句子，不要長篇大論，也不要太抽象，相關之流程敘述要記載明確。問題的敘述最好是從使用者的觀點出發。
- 例子
  - 對於一個電影院(火車、客運等等)訂票系統來說，我們可以將使用者的需求定義如下：
    - 使用者必須登入以使用此系統。
    - 使用者可以利用瀏覽器找尋相關電影並預訂電影票。
  - 如果你用“實現訂票流程自動化”來描述這個系統的功能，那就把問題的範圍定義變得太廣，很抽象。

- ATM例子
  - 大部分的人都使用過自動提款機(ATM)來從事提款，轉帳等事項。如果你仔細地觀察一下ATM的畫面，你會發現到一部ATM的主畫面以及其他可用的選項都會對應到一個事件上。比如說：提款，存款，餘額查詢，轉帳等等。這些不都是一部提款機所提供給使用者的功能嗎？你，一個使用者，必須製造某些事件來觸發或是告知系統你到底想要它做什麼。

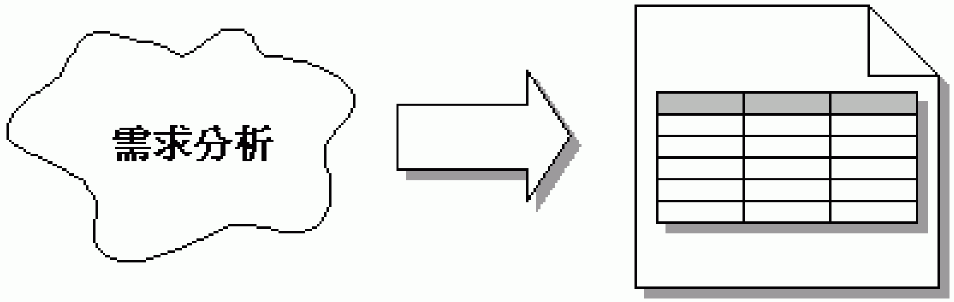
需求分析

- 需求描述
  - 對於功能性需求的描述，你應該可以將它們歸納成事件。
  - 事件的寫法儘量用：
    - 主詞+動詞+受詞的形式。
  - 你可以把事件當作是需求描述的精簡版或是摘要(summary)。

需求分析

- 事件表
  - 當你整理歸納出一大堆事件之後，可以將之紀錄於表格中，成為事件表。
  - 事件表是用來記錄系統功能很有用的工具。不要擔心功能到底需要怎麼被實現出來，先把系統當成一個黑盒子。這樣做的好處是讓參與計畫的人員能將焦點放在系統高層次的觀點，從外部來看系統，而不是系統內部的運作情況。在很多的經驗中我們發現人們一般都把焦點放在系統的How, 而不是系統的What。

需求分析



- 事件表欄位說明
  - 事件名稱：寫出造成系統去完成某事的事件。利用主詞+動詞+(受詞)的型式。
  - 觸發器(trigger)：引發事件的資料。
  - 來源：資料來源
  - 活動：事件發生時， 系統要執行的任務。活動要以動詞為開端。 例如：查詢 (Retrieve)…、更新(Update)…、產生(Create)…、取消(Delete)…。
  - 回應：什麼樣的輸出如果有的話？由哪裡產生？
  - 目的地：輸出到哪裡去。

事件名稱	觸發器	來源	活動	回應	目的地

需求分析

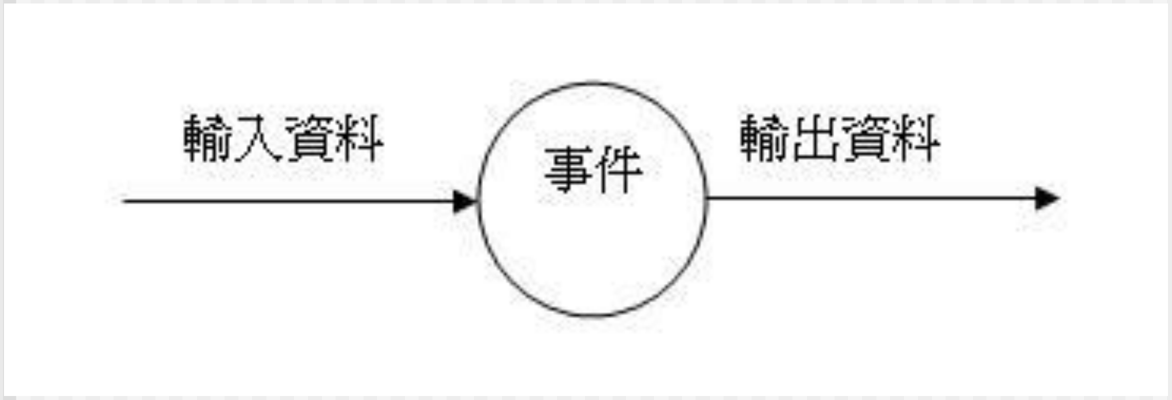
- 舉例說明
  - 一個線上購物系統，最明顯的事件就是”顧客下訂單”。我們可以分析出：事件名稱就是顧客下訂單。訂單是顧客下的，所以事件的來源來自於顧客。這個事件被觸發是因為訂單資料的到來，這是來自來源的一種要求。對於這個事件，系統應該要產生一筆新的訂單記錄，以記錄這個活動。所以，”產生一筆訂單”是活動的名稱。而”訂單編號”是這個活動的回應，此回應的目的地是使用者。

訂單產生之後要將訂單資料傳送到出貨部門

事件名稱	觸發器	來源	活動	回應	目的地
顧客下訂單	訂單	顧客	產生一筆新 訂單	訂單編號 訂單	使用者 出貨部門

需求分析

- 事件圖思考方向
  - 嘗試著鑑別所有需要被儲存的資料。
  - 由建立(Create)、存取(Retrieve)、更新(Update)以及刪除(Delete)這四個動作來檢視系統可能發生的事件行為。這四個動作統稱為CRUD 。
  - 想一想有沒有以資料為導向的事件。
  - 想一想有沒有以時間為導向的事件。
  - 利用主詞+動詞+(受詞)的型式來歸納出事件。
  - 嘗試著鑑別各事件稍後會使用到的任何資訊。



軟體需求規格文件

- 軟體需求規格文件(Software Requirement Specification)記載著下列的項目
  - 系統目標
  - 系統範圍
  - 系統整體描述
  - 功能需求
  - 非功能需求
  - 系統所需提供之使用者、軟體、硬體等切面之需求
  - 其他資料