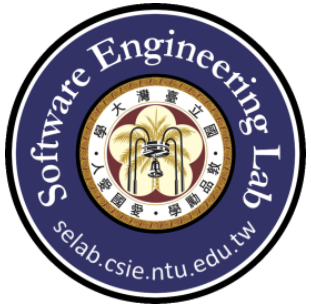
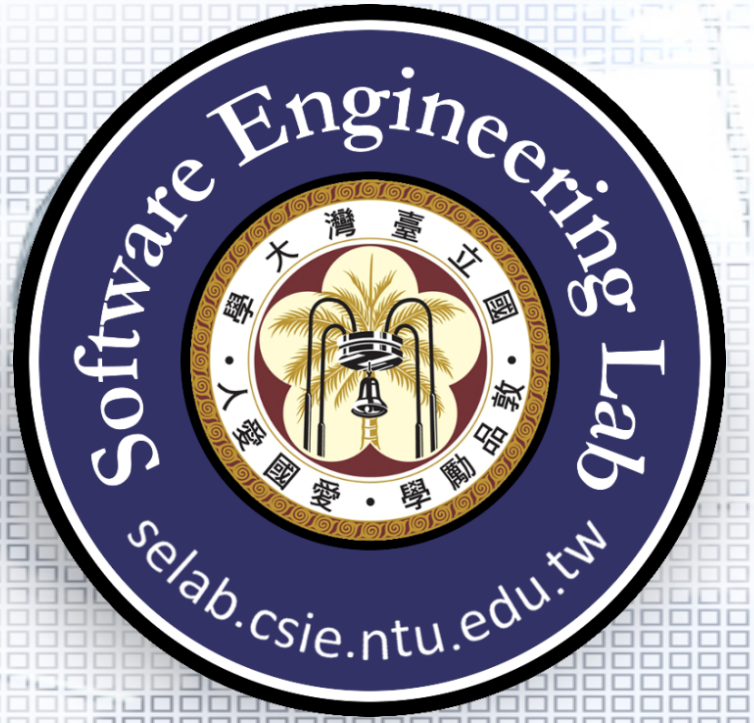


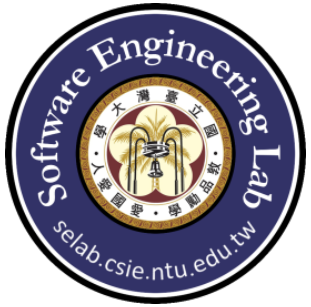
Homework

- Read Chapter 2 (due 10/17)
- Prepare a Software Requirements Specification (SRS) of your term project based on the SRS template, with the following features (due 10/24)
  - System architecture
  - Functional, nonfunctional, internal interface, and external interface requirements
  - Use case diagram, and use case specifications
  - Traceability matrix
- Prepare a presentation of Goal-Driven Use Case approach in Chapter 3.5 (due 10/24)



Requirements Engineering: Software Requirements Specification

Prof. Jonathan Lee
CSIE Department
National Taiwan University



What're the Problems?

- Stakeholders don't know what they really want.
- Stakeholders express requirements in their own terms.
- Stakeholders may have conflicting requirements, even for the same stakeholders.
- Organisational and political factors may influence the system requirements.
- The requirements change during the software development process. New stakeholders may emerge and the business environment changes.

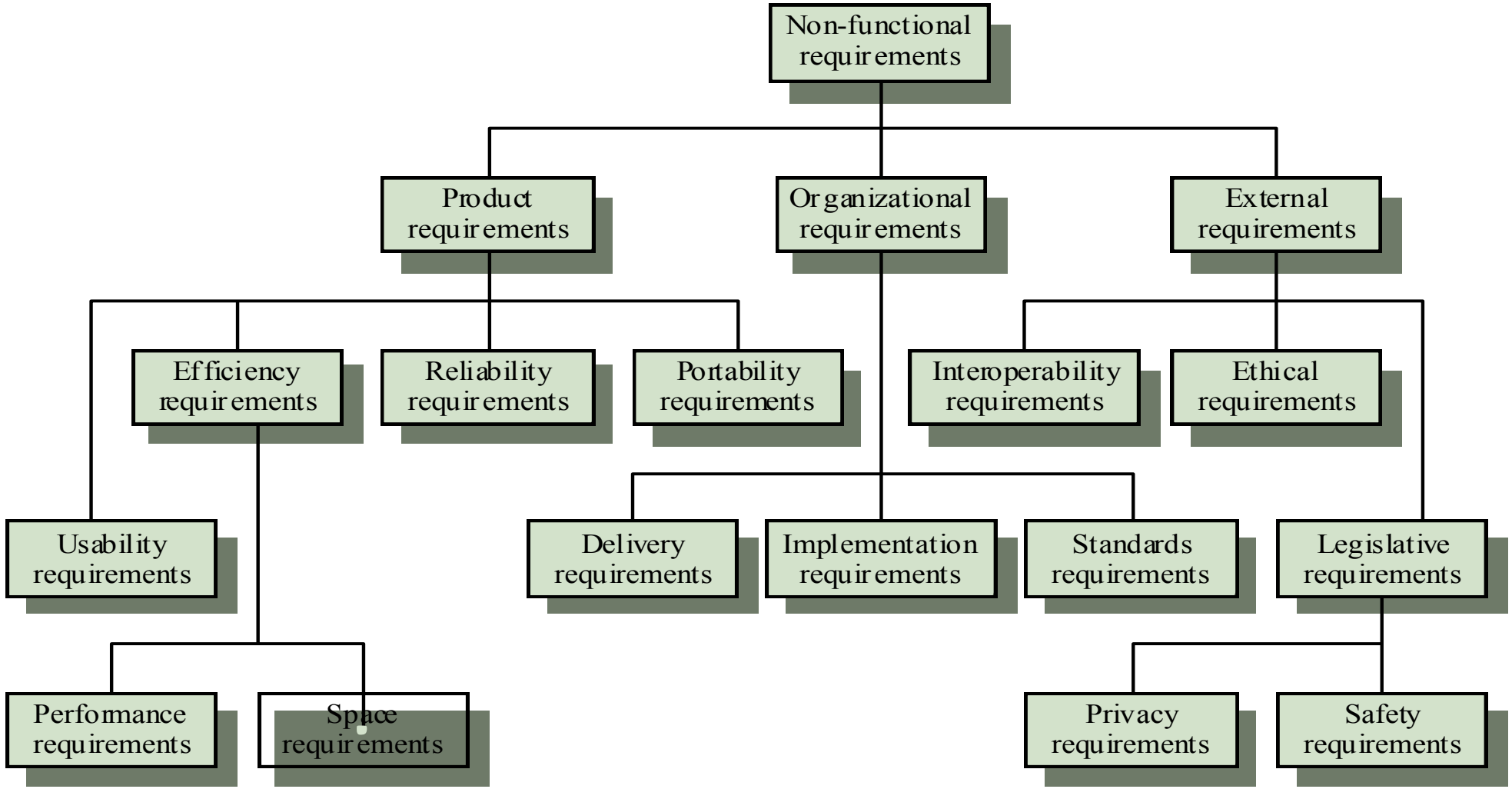
Why is Requirements Engineering Important?

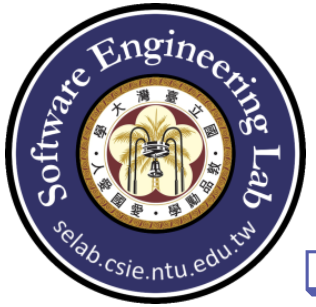
- Requirements address the **needs** of stakeholders
  - including those pertinent to various product lifecycle phases (e.g., acceptance testing criteria) and product attributes (e.g., safety, reliability, and maintainability).
- Requirements address **constraints** caused by the selection of design solutions
  - e.g., integration of commercial off-the-shelf products
  - Requirements are the basis for design.
- Requirements Management helps control the **evolution** of requirements

Software Requirements

- Requirements
  - Functional requirements describe system services or functions
  - Non-functional requirements are constraints or goals on the system or on the development process
  - Interface requirements define the message passing between subsystems and external environment
- User (Customer) requirements: A statement in natural language plus diagrams of the services the system provides and its operational constraints
- Requirements specification: A **structured document** for detail description of system services, system requirements specification.

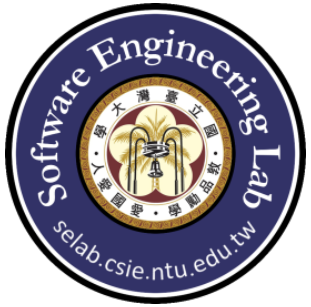
Non-Functional Requirements (Examples)





Examples of Requirements from Meeting Scheduler System

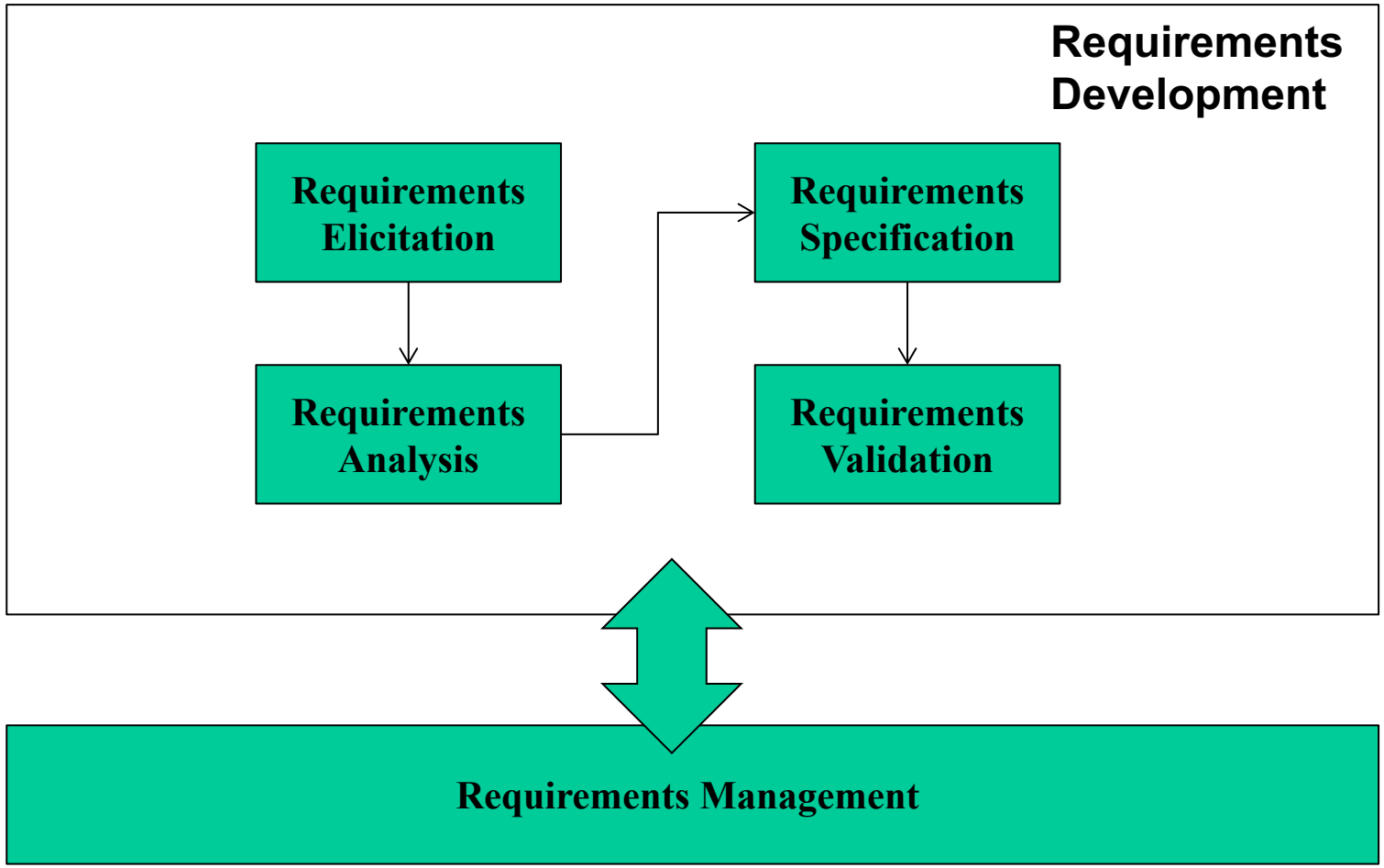
- Functional requirement: A Meeting initiator asks all potential meeting attendees for a set of dates on which they cannot attend the meeting, and a set of dates on which they would prefer the meeting to take place.
- Non-functional requirement: The system should provide an appropriate level of performance, for example: the elapsed time between the submission of a meeting request and the determination of the corresponding meeting date/location should be as small as possible.
- External interface requirement: There should be an access control mechanism to determine if a user is the administrator after ID and password are entered.
- Internal interface requirement: Meeting Scheduler Module retrieves the states of meetings and the states of participants from Database to perform Meeting Scheduler Algorithm and stores the result to Database.

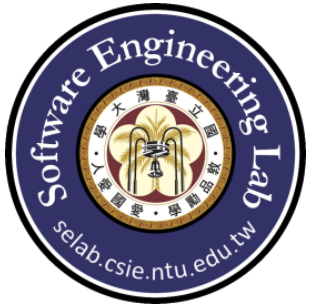


Characteristics of Requirements

- Incomplete Requirements
  - Most software systems are complex and change over time, developers can never fully capture all the requirements during the system development.
- Inconsistent Requirements
  - Different users have different requirements and priorities. There is a constantly shifting compromise in the requirements.
  - Prototyping is often required to clarify requirements.

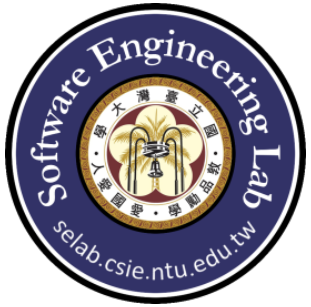
Requirements Engineering Process





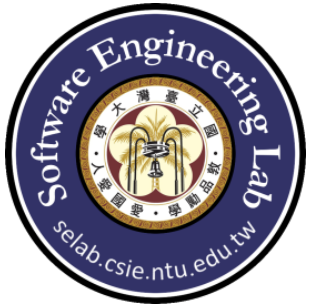
Requirements Engineering$_1$

- Requirements elicitation (acquisition): Determine what the customers require.
- Requirements analysis: Understand the relationships among various customers' requirements.
- Requirements negotiation:
  - Shape the relationships among various customer requirements to achieve a successful result
  - Research on requirements trade-off analysis (formulating as goals)



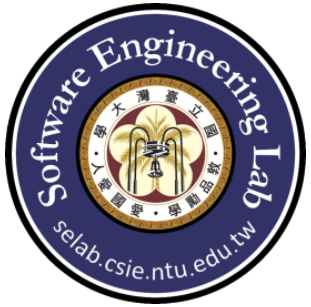
Requirements Engineering$_2$

- Requirements specification
  - Build a form of requirements.
  - Generate a representation of requirements that can be assessed for correctness, completeness, and consistency (Software Requirements Specifications, SRS).
- Requirements validation
  - Review the model against users' needs.
- Requirements management
  - Identify, control, and track requirements and their changes.



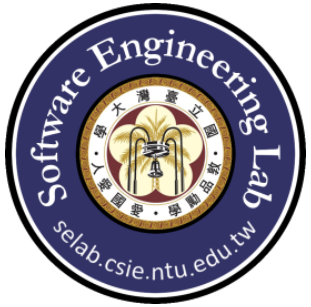
Requirements Elicitation$_1$

- Two sources of information for the requirements elicitation process
  - User (customer)
  - Application domain
- Asking
  - Ask users what they expect from the system
- Task analysis
  - High-level tasks can be decomposed into sub-tasks
- **Scenario-based analysis**
  - A scenario can be real or artificial and can be expressed in **use case** diagrams.



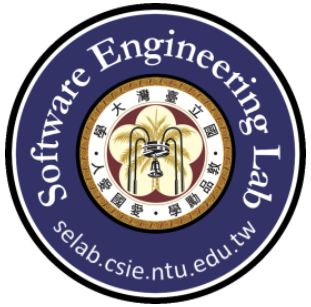
Requirements Elicitation$_2$

- **Form analysis**
  - A lot of information about the domain can be found in various forms
  - Forms provide us with information about the data objects of the domain, their properties, and their interrelations
- Natural language description
  - with background information to be used in conjunction with other elicitation techniques such as interviews
- Derivation from an existing system (**reverse engineering**)
- **Prototyping** to collect requirements.



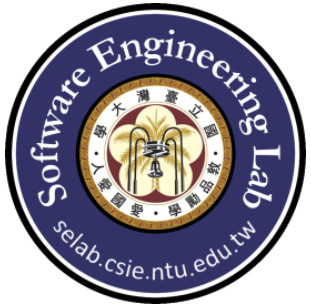
Requirements Analysis$_1$

- Structure, function and behavior representation
  - Structure: entity and relationship
  - Function: process transformation
  - Behavior: state transition diagram
- Interfaces definition
  - function/process interface
- Problem partition and abstraction
  - at different levels of abstraction
  - classification and assembly structure



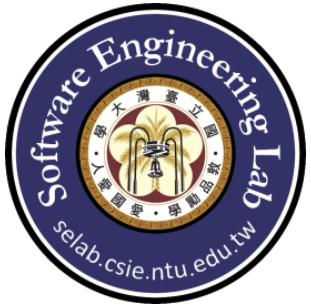
Requirements Analysis$_2$

- Application and Solution Domain
  - Application Domain (Requirements Analysis)
    - Involve **domain knowledge** in a specific application.
  - Solution Domain (System Design, Object Design)
    - The available technology to build the system, such as establishing a **system architecture** as a basis for defining the scope of the system, **an initial design** in class diagrams, and possibly a **redesign** of the class diagrams based on design principles and design patterns.



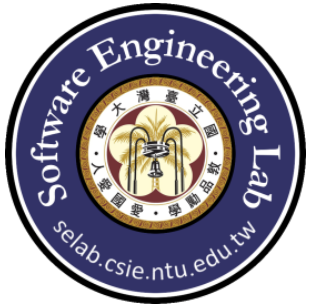
Requirements Specification

- Build a form of requirements
  - Informal specification: natural language narratives, such as user story in agile methods.
  - Semi-formal specification: graphical representation, such as UML
  - Formal specification: mathematical constructs or logical axioms, such as Z
- Generate software requirements specification (SRS)
  - Including system architecture, scenarios, requirements, acceptance criteria, traceability matrix, and etc.



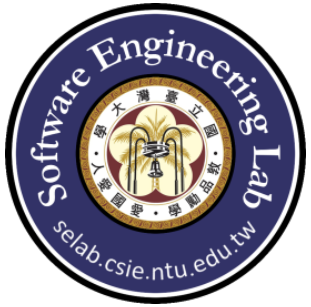
Build an SRS with Essential Elements

- Focus on semi-structured requirements specification approaches, such as object-oriented modeling UML.
- Use case diagrams, use case specification
- Sequence diagram



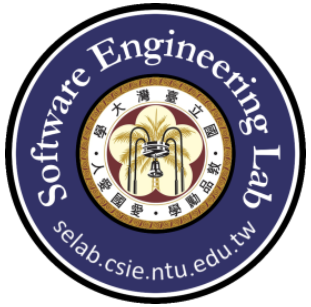
UML Diagrams

- A diagram is a view (perspective) of a model
  - Presented from the aspect of a particular stakeholder
  - Provides a partial representation of the system
  - Semantically consistent with other views
- There are 8 standard diagrams
  - **Use case diagram**
  - **Class (object) diagram**
  - Behavior diagram
    - activity, collaboration, **sequence**, state
  - Implementation diagrams
    - Component, Deployment



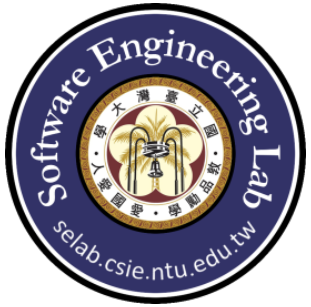
Use Case and Sequence Diagrams

- Use case diagrams
  - Describe the functional behavior of the system as seen by the user, that is, how to use the system
  - Illustrate actors, use cases, and their relationships
- Sequence diagrams
  - Describe the dynamic behavior between actors and the system and between objects of the system.
  - Illustrate objects and their relationships, including their chronologically structured messages exchange



Use Case Diagram

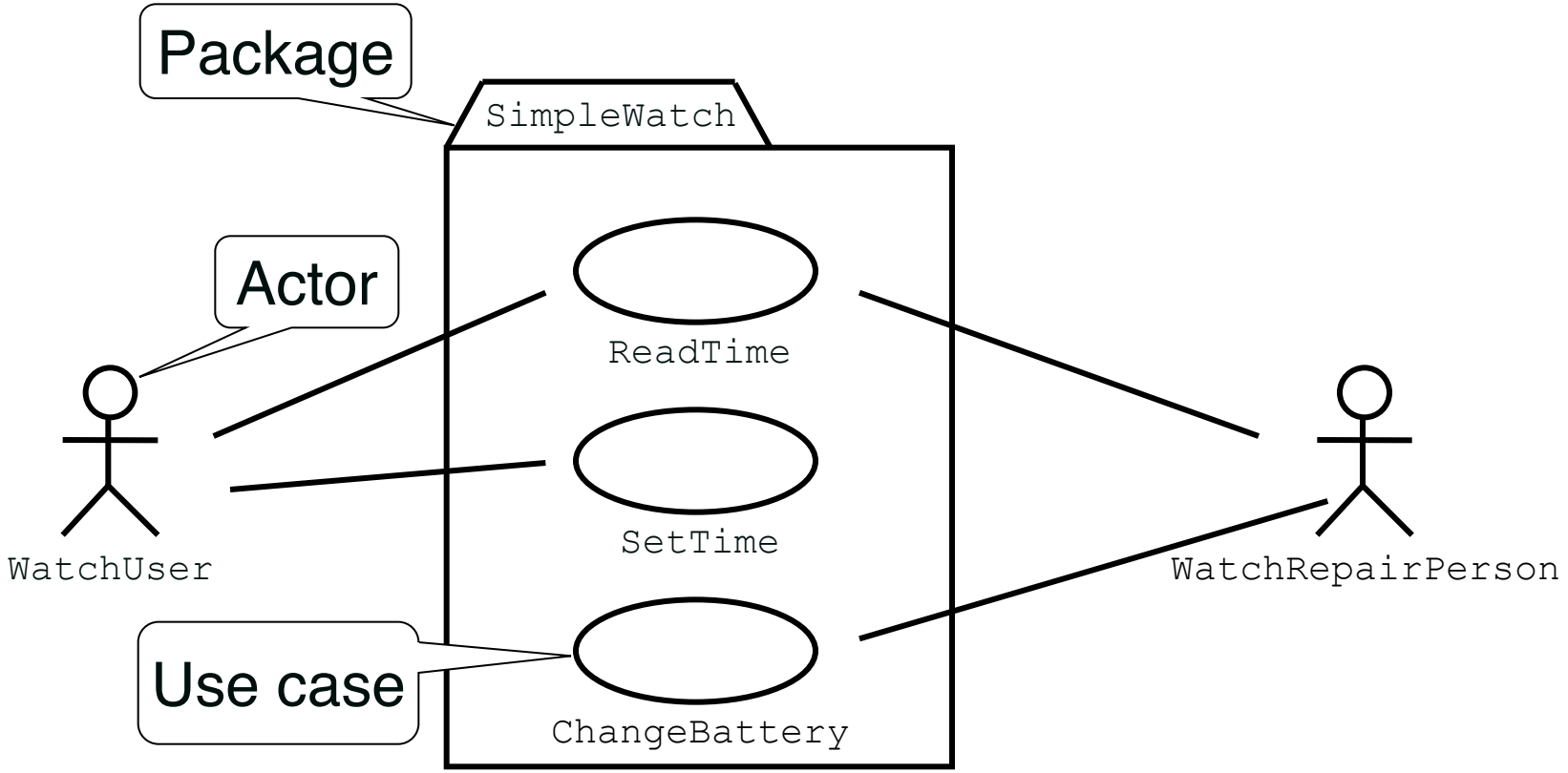
- A graph of **actors**, a set of **use cases**, and the **relationships** between these elements.
- The relationships are
  - associations between the actors and the use cases,
  - generalizations between the actors
  - generalizations, extends, and includes among the use cases.
- The use cases represent functionality of a system or a classifier, like a subsystem or a class.
- A Classifier represents a group of things with common properties.
- Developed by analysts and domain experts

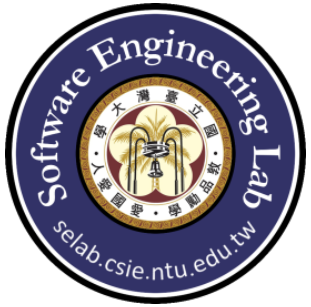


Purpose of Use Case Diagram

- Describe the interaction between a set of use cases and the actors
- Capture system functionality as seen by users
- Used during requirements elicitation
- Drive implementation and **generate test cases**

Use Case Diagram



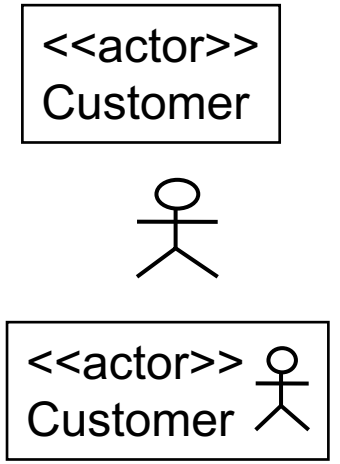


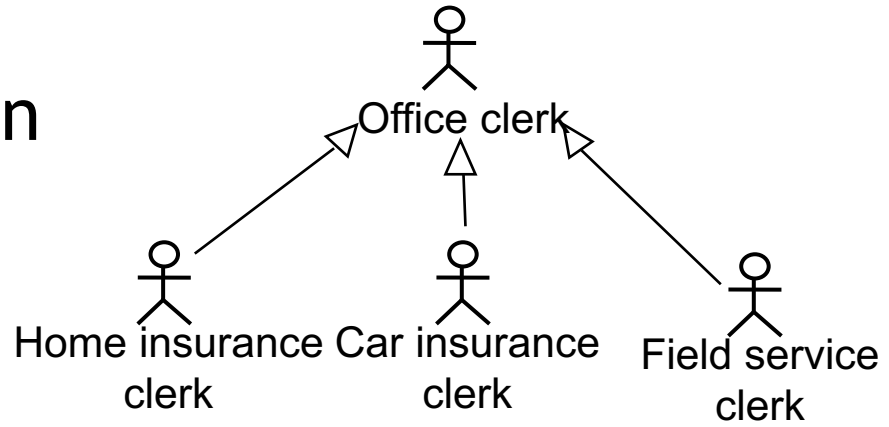
What is an Actor?

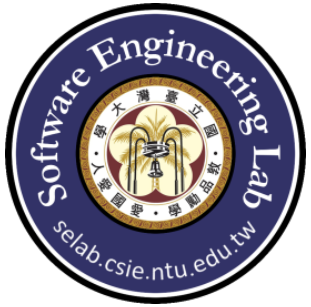
- An external entity communicates with the system by sending and receiving messages, and exchanging data and events.
  - user, external system, physical environment.
  - A description to represent roles that play the system functions
  - A person can play a number of roles (actors)
- Examples:
  - Passenger: A person in the train
  - GPS satellite: Provides the system with GPS coordinates
  - ATM: the client is an actor who can withdraw money from an account, transfer money to an account, or check balance of an account

Actor Notation

- Notation
  - Textual stereotyping
  - Visual stereotyping
  - Textual and visual stereotyping
- Generalization/specialization

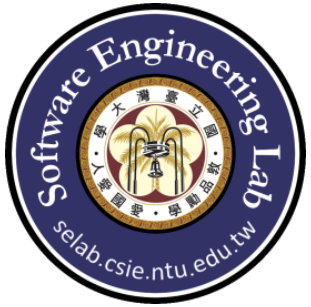






What is a Use Case?

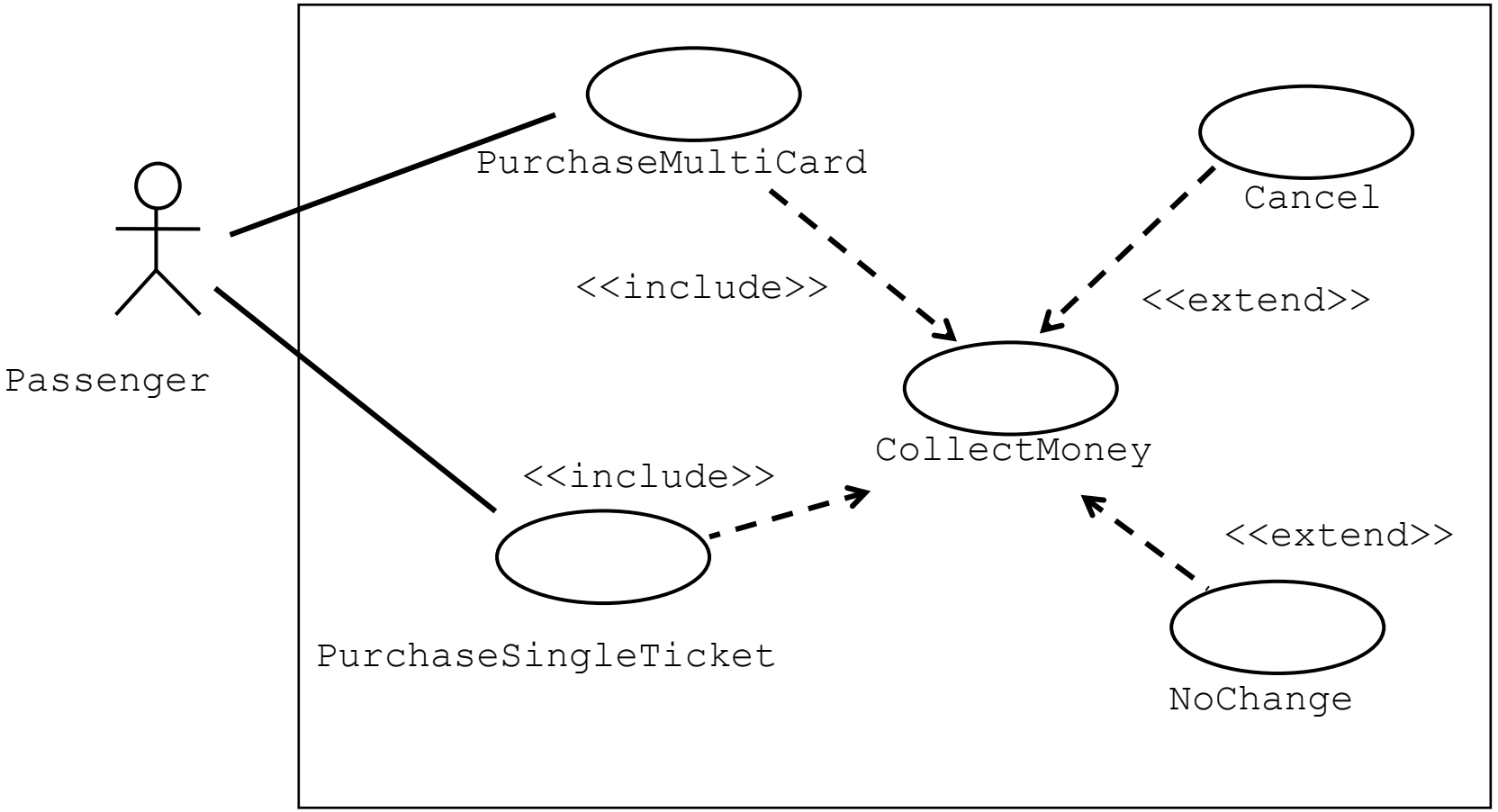
- A use case represents a coherent unit of functionality provided by a system, a subsystem, or a class.
- A sequence of messages exchanged among the system and the actors together with actions performed by the system.
- A scenario that describes a “thread of usage” for a system
- Describe a set of activities of a system from the point of view of its actors

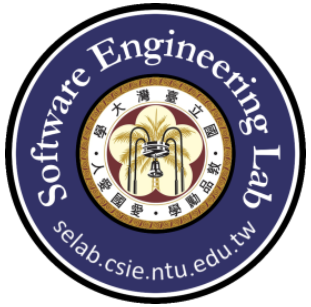


Three types of Use Cases

- Base use case: this is the case in which **all go smoothly**.
- Extension use case: this is the case to **extend** the behavior of a base case.
  - <<extend>> relation
- Inclusion use case: this is the case in which common functionality is factored out for **reuse**.
  - <<include>> relation

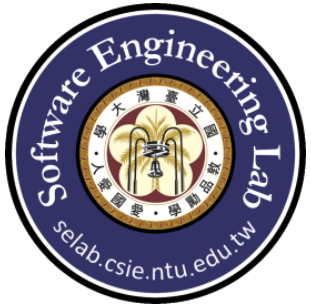
Use Case Example





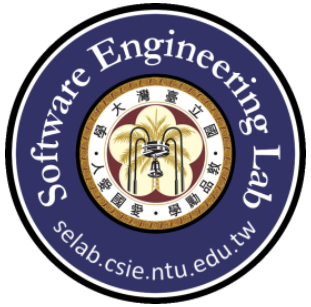
Abstract Actor and Use Case

- Actor generalization is typically used to pull out common requirements from several different actors to simplify modeling .
  - E.g. abstract Database Administrator, Backup Administrator, and Deployment Administrator into System Administrator
- Use case generalization is used to express some high-level functional needs of a system without going into specifics.
  - E.g. abstract Check-Passenger-Fingerprint, and Verify-Passenger-RFID-Tag into Verify-Passenger-Identity



Use Case Specification$_1$

- Unique name of use case
- **Goal (optional)**
- Requirements (corresponding)
- Use case description
- Participating actors
- Assumption
- Constraint
- Preconditions
- Post conditions
- **Basic Flow of events**
- Alternative Flow
- Exceptions, error situations

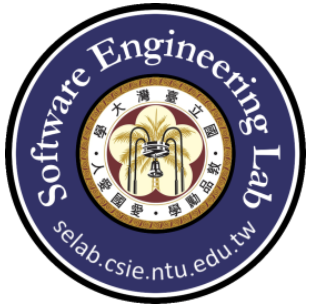


Use Case Specification$_2$

- Inclusion use case
- Extension use case
- Business rules, validation rules
- Artifact
- Use case glossary
- Notes / open questions: design decisions / design alternatives.

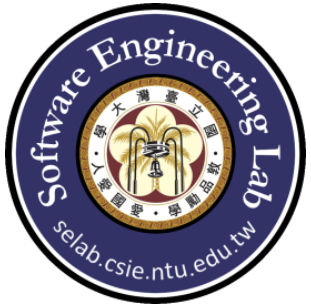
Use Case Specificaton$_3$

Use Case ID	**UC-IP-001**	
Use Case Name	規劃會議	
Goal	達成「滿足會議需求」	
Requirements	[FFR1: 發起會議] , [FFR3: 會議行事曆], [FFR4: 會議管理] , [BFR5: 訊息管理], [BFR6: 會議排程演算法]	
Description	使用者使用該功能，發起、維護一會議，並邀請人參與會議。	
Actor	召集人	
Assumptions		
Constraints		
Priority	High	
Pre-Conditions	1. 使用者必須為本系統認可使用者 2. 使用者必須先登入系統	
Post-Conditions	會議進入排程狀態，並寄送訊息給邀請之參與者。	
Basic Flow	**Actor**	**System**
	1. 使用者點選"發起會議" 2. 使用者點選發起會議 3. 編輯會議名稱、類型、角色 4. 使用者編輯會議項目、議程[UC-IP-003:建立討論議程] 5. 使用者新增邀請參與人[UC-IP-002:新增參與者] 6. 設定參與者重要性[UC-IP-007: 設定參與者重要性] 7. 使用者設定調查時間/地點範圍[UC-IP-006: 擴充時間/地點範圍] 8. 設定會議參與人數限制 9. 設定召集人[UC-IP-008: 設定召集人權限] 2. 使用者點選"會議管理" 2. 管理會議內容 3. 使用者點選送出	1.1 系統顯示會議發起單 2.1 系統顯示具有管理權限之欄位 3.1 系統將會議重新導入排程



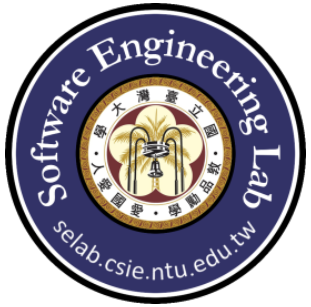
Use Case Specificaton$_4$

Alternative Flows	1.7.1 若人數限制不做調整，則預設為50% 1. 會議角色若為空，提醒使用者是否確認送出 1. 使用者點選「是」，則將召集人設定為主席，並送出會議發起單 2. 使用者點選「否」，則回原畫面，讓使用者繼續填寫
Exceptional Flows	2.1 若使用者完全不具此會議管理權限，則不顯示在可管理會議上面 1. 調查日期/地點、會議名稱若為空，則顯示警告訊息，提醒使用者填寫，或者停止發起此會議。
Inclusion Use Case	
Extension Use Case	
Business Rules	
Artifacts	會議發起單
Use Case Glossary	會議發起單 = 會議編號 + 會議等級 + 召集人編號 + 會議目的 + 工作項目及負責人 + 參與者編號 + 時間選項 + 地點選項 + 備註
Notes	



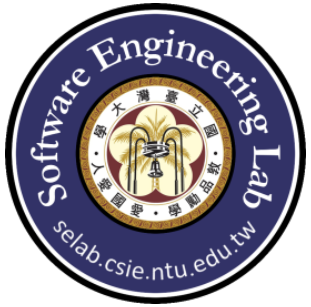
Sequence Diagram

- Capture dynamic behavior (time-oriented) in terms of interactions.
- Model flow of control
- Illustrate typical scenarios
- Complement the class diagrams
- **Extremely useful for debugging if the sequence diagram is in a fine-grain**



Sequence Diagram Notation

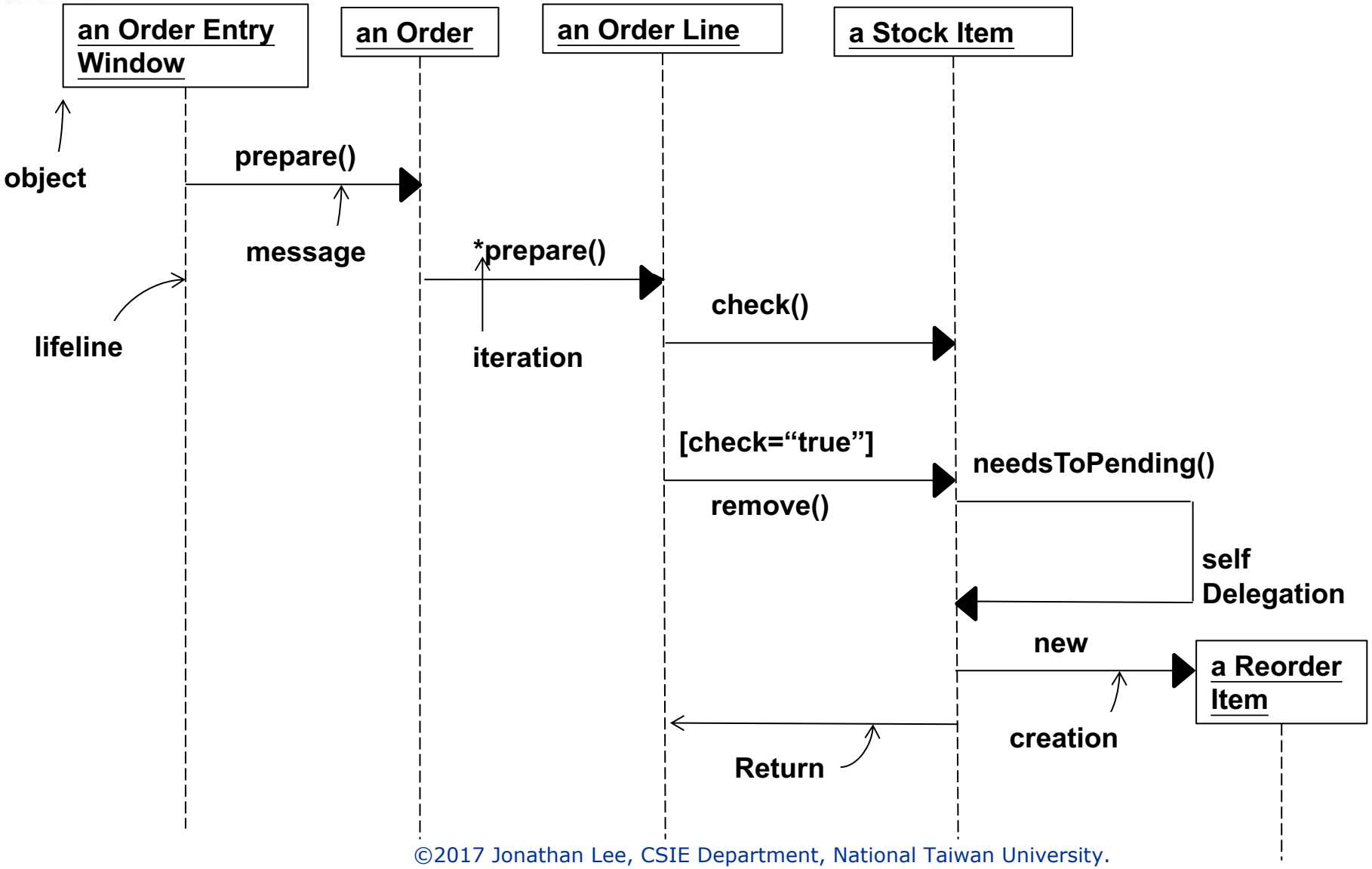
- Object instances are represented by columns
- Messages are represented by arrows
- Activations to indicate a method is active because it is either executing or waiting for a subroutine to return.
  - Best when used in concurrent situations
- Lifelines are represented by vertical dashed lines
- Self-delegation is a message that an object sends to itself.

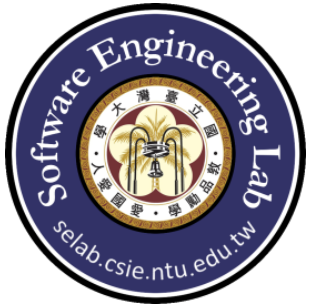


Notation

- Asynchronous message can be used to do one of the three things:
  - Create a new thread to link to the top of an activation
  - Create a new object
  - Communicate with a thread that is already running
- Object deletion for objects to self-destruct or to be destroyed by another message

Sequence Diagram Example

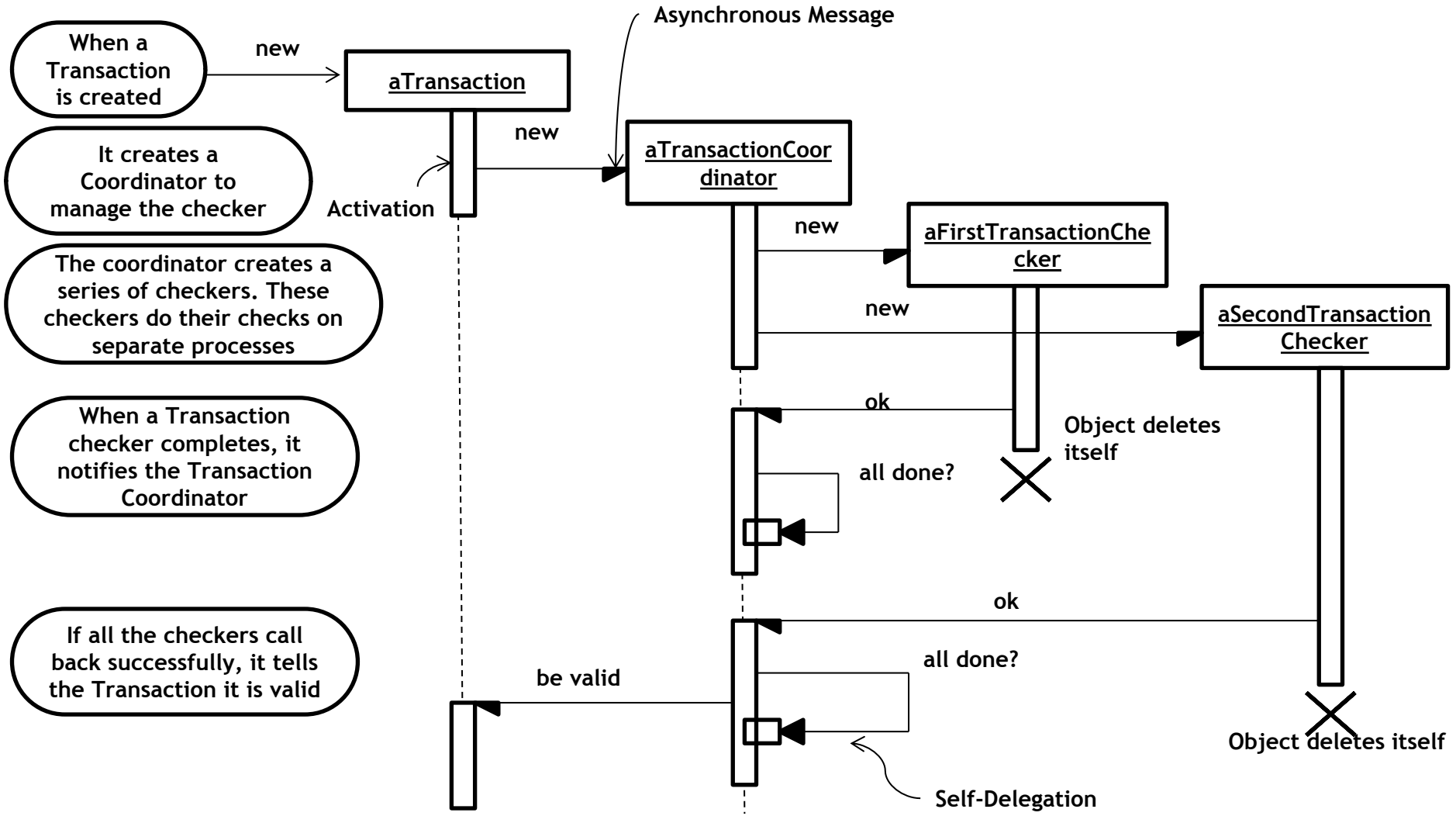


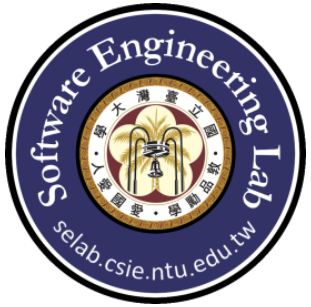


Concurrent Process and Activation: An Example

- When a Transaction is created, it creates a Transaction Coordinator to coordinate the checking of the Transaction.
- This coordinator creates a sequence of Transaction Checkers objects, each of which is responsible for a particular check. Each checker is called asynchronously and proceeds in parallel.
- When a Transaction Checked completes, it notifies the Transaction Coordinator. The coordinator looks to see if all the checkers called back. If not, the coordinator does nothing. If they have, and all of them are successful, then the coordinator notifies the Transaction that all are valid.

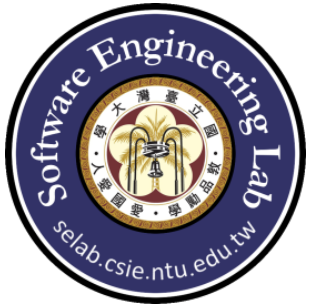
Concurrent Process Example





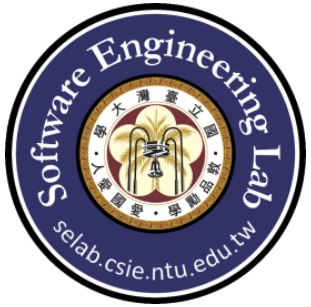
System Requirements Specification (SRS)

- System Architecture
- Requirements Description
  - Functional requirements
  - Non-functional requirements
  - Interface requirements: external and internal
- Scenarios and Operational concept
  - Use case diagram
  - Use case specifications
  - Sequence diagrams (optional but useful to visualize the interactions between object instances when programming)
- Traceability matrix



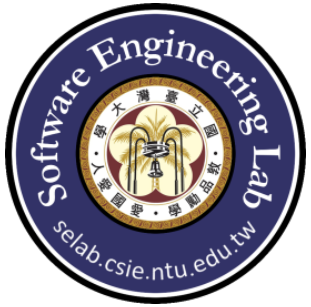
Requirements Validation$_1$

- Concerned with demonstrating that the requirements define the system that the customers really want.
- Fix a requirement error after delivery may cost up to 100 times the cost of fixing an implementation error.



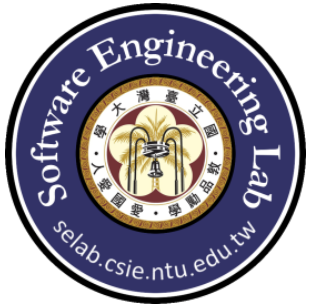
Requirements Validation$_2$

- Requirements checking
  - Validity: Does the system provide the functions which best support the customers' needs?
  - Consistency: Are there any conflicting requirements?
  - Completeness: Are all functions required by the customer included?
  - Realism: Can the requirements be implemented given available budget, time, and technology?
  - Verifiability: Can the requirements be checked?



Requirements Validation$_3$

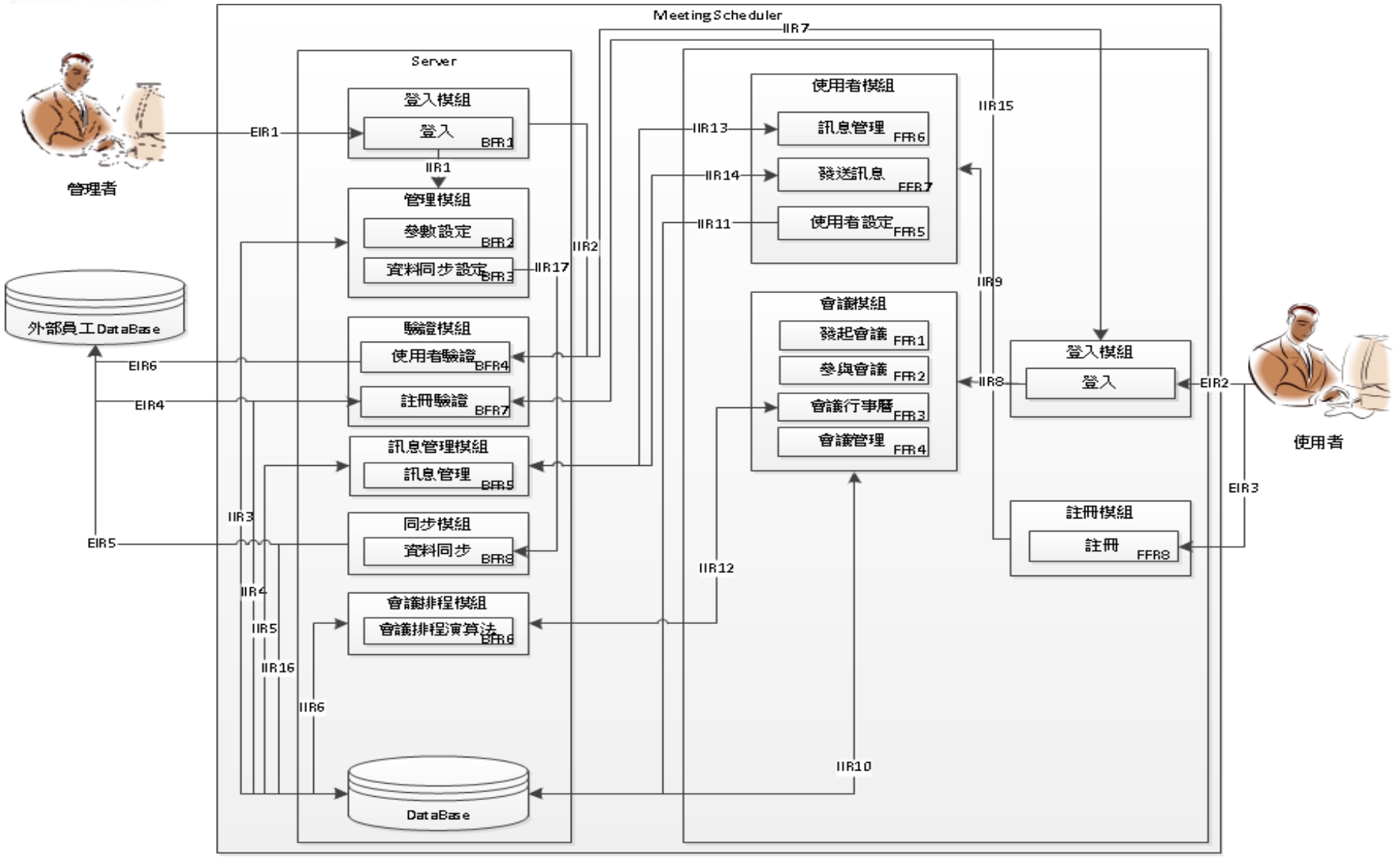
- Techniques
  - Requirements reviews
    - Systematic manual analysis of the requirements.
    - Good communications between developers, customers and users can resolve problems at an early stage.
  - Prototyping
    - Use an executable model of the system to check requirements.
  - Test-case generation
    - Develop test cases for requirements.

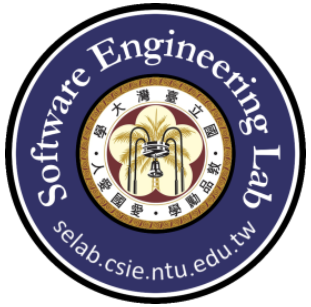


Requirements Management

- Large amounts of information need to be versioned and maintained in a consistent state.
- Changes to the requirements document must be propagated through other items that depend on the changed items in order to maintain their mutual consistency.
- Consistency maintenance requires the management of traceability links among items and propagating changes along such links.
- Build horizontal and vertical traceability matrix for requirements, designs, and codes.

MSS Example: System Architecture



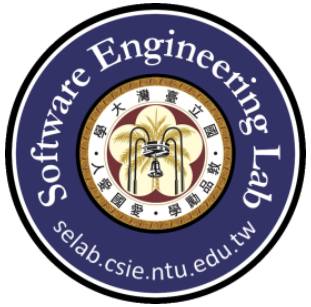


MSS: Functional Requirements

會議功能模組	
FFR1	發起會議
	使用者透過此功能，在系統上發起一項會議，發起會議中，可以選擇此會議之類型、以及選擇欲調查之時間/地點範圍，決定討論項目以及負責人，以及選擇所欲邀請之參與人。
FFR2	參與會議
	若使用者接收到其他使用者發起會議之參與，此功能會提醒使用者填寫參與意願，並且將調查結果送往後端進行會議排程。
FFR3	會議行事曆
	提供使用者一介面，呈現該使用者之參與會議，可能參與會議之排程狀態。
FFR4	會議管理
	使使用者管理具有管理權限之會議。
使用者功能模組	
FFR5	使用者設定
	此功能管理使用者個人基本資料，使用者可以透過此功能，來維護使用者本身資訊。
FFR6	訊息管理
	此功能提供使用者管理站內非同步訊息，接收來自各個使用者所傳遞之訊息，以及調閱歷史訊息記錄。
FFR7	發送訊息
	此功能提供使用者一介面，方便使用者與其他使用者溝通，傳遞訊息給其他使用者。
註冊模組	
FFR8	註冊
	提供新使用者一介面，註冊成為本系統使用者。

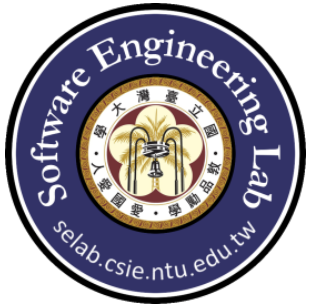
MSS: Functional Requirements

登入模組	
BFR1	登入功能
	此功能提供使用介面讓使用者輸入登入基本資訊，將資料送到後端進行使用者驗證。
管理模組	
BFR2	參數設定
	此功能提供介面讓系統管理者針對會議角色、會議等級、使用者等級這三項資訊新增、修改和刪除等管理選項。
BFR3	資料同步設定
	提供系統管理者一個介面，使系統管理者管理與外部員工資料庫的資料欄位設定，確保資料同步順利。
驗證模組	
BFR4	使用者驗證
	此功能接受登入之使用者，判別是否為本系統之使用者，並將驗證結果向上回傳，若為此系統之使用者，則向系統註冊登入，讓使用者進入系統進行功能操作。
BFR7	註冊驗證
	系統驗證新註冊使用者，透過外部員工資料庫進行身分比對，在發送訊息確認使用者註冊。
訊息管理模組	
BFR5	訊息管理
	此功能管理使用者藉由前端訊息功能所互相傳遞之訊息管理，使訊息能正確配送以及儲存至database，訊息傳遞採用非同步傳遞方式。
同步模組	
BFR8	資料同步
	固定時間與外部資料庫與系統資料庫進行員工資料同步
會議排程功能模組	
BFR6	會議排程演算法
	此功能提供一演算法，管理與排程使用者所發起之會議。 此會議排程，會針對參與者之參與的時間/地點意願來進行排程 會議排程給予會議一個會議排程等級，用於解決時間/地點衝突時的優先權。 當一會議參與者參與意願填寫至一定權重時，會啟動此會議的排程，將該會議導入排程。 當一會議到達到某一時限，會啟動此會議的排程，將該會議導入排程。 當一會議到達排程時限時，會將此會議鎖定排程，並將排程結果發送給會議之相關使用者。



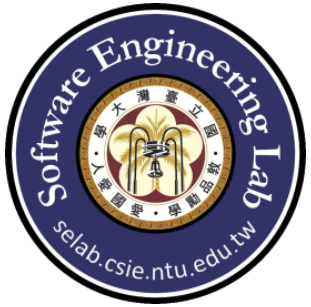
MSS: External Interface Requirements

EIR1	管理者身分驗證系統介面
	管理者輸入帳號與密碼，以判斷是否有資格進入後端管理其他子系統。
EIR2	使用者身分驗證系統介面
	使用者輸入帳號與密碼，已判斷是否有資格進入系統觀看或者使用其他子系統。
EIR3	使用者與註冊系統
	新使用者，透過此介面，輸入E-mail送至後端進行身分驗證及註冊。
EIR4	註冊驗證以及外部員工資料庫
	前端新使用者註冊後，透過註冊驗證，像外部員工資料庫查詢E-mail是否為有效mail。
EIR5	同步模組與外部員工資料庫
	同步時，系統讀取外部員工資料庫資訊。
EIR6	使用者驗證與外部員工資料庫
	當一使用者登入，向外部員工資料庫所以員工密碼特徵。



MSS: Internal Interface Requirements

IIR1	Admin 系統與其他管理系統介面
	網站管理員通過登入確認具有管理權限後，可以進入後端對各個子系統進行各種動作。
IIR2	管理者與驗證系統
	管理者透過系統提供之登入功能，將登入資料送往使用者驗證系統，由使用者驗證系統判斷是否准許進入後台管理系統。
IIR3	管理模組與Database連線
	管理者透過系統提供的後端管理功能，讓管理者管理會議等級，管理會議角色，管理使用者等級以及同步資料的參數設定。
IIR4	註冊驗證與Database
	經由外部database驗證後，確認註冊時向系統database註冊個人基本資料。
IIR5	訊息管理與Database
	將使用者傳送的訊息，經由訊息管理此子系統管理後，將訊息暫存至database。以及向database讀取暫存之訊息。
IIR6	會議排程與Database
	會議排程演算，向database中索取其他會議狀況、參與者狀態來進行會議排程管理，並將排程結果暫存至database。
IIR7	使用者與使用者驗證系統
	使用者透過系統提供之登入功能，將登入資料送往使用者驗證系統，由使用者驗證系統判斷是否准許使用前端子系統。



MSS: Non-Functional Requirements

NFR1	提供彈性
	提供使用者彈性參與會議，參與想要或者重要性大之會議
NFR 2	適當效能
	設定門檻，規定會議最晚該進入排程，以及預設參與人意願，使會議排程效能最大化。
NFR 3	最大方便度議程
	使討論項目參與者，皆能參與會議。
NFR 4	最大參與人數
	使所有系統內的會議，在排程中可以每個會議參與人數達到最高。