Static Modeling(Class단위로 분석)

* Class들 사이의 구조적인 관계를 정의
* Class들과 그들의 관계를 Class Diagram으로 묘사
* Class들 사이의 관계
* Associations
* Composition/Aggregation
* Generalization/Specialization
* System의 정적인 구조를 표현
* ER(Information) Modeling에 기초
* Static Modeling
* In OO Analysis Modeling(분석에서의 Static Modeling)
* System의 Classes 정의
* Class의 attributes 정의
* Class사이의 relationships 정의
* In OO Design Modeling(설계에서의 Static Modeling)
* 각 Class의 Operations(method) 정의
* System Context Class Diagram
* 정의
* System과 External Environment 사이의 boundary 정의
* System
* 하나의 집계 <<system>> class로 묘사됨
* External Environment
* System이 interface하는 외부 Class
* Category of External Classes
* <<external user>>
* <<external I/O device>>
* <<external system>>
* <<external timer>>
* Association
* System Context Class Diagram은 System과 External Class사이의 Association 보여줌
* System Context Class Diagram은 Multiplicity of Association(1 to 1, 1..\* to 1, etc) 보여줌
* Association name
* <<external input device>> inputs to <<system>>
* <<system>> outputs to <<external output device>>
* <<external user>> interacts with <<system>>
* <<external system>> interfaces to <<system>>
* <<external timer>> awakens <<system>>
* Entity Classes
* 특징
* Data 집약적인 class
* persistent(long living) data 저장
* Information System을 위해서 중요(database intensive)
* real-time and distributed application을 위해서 중요
* Analysis Modeling중 하는 작업
* Problem domain 안의 Entity Class 모델링
* Attributes, Relationship

Object Structuring(개발할 Software를 Object 단위로 쪼갬)

* 특징
* Problem을 objects로 분해
* 문제의 특성과 판단에 기초함
* 올바른 단일 표현 없음
* System 안의 모든 Software Object 결정
* Object Structuring Criteria(Stereotype을 사용하여 묘사됨) 이용
* Object를 식별하기 위한 Guidelines
* Object들은 Categorized 됨
* 종류
* Interface Objects
* External Environment(Object)와 Interface
* 종류
* User Interface Object
  + standard I/O device를 통해 human user와 interface
  + 간단하거나 복잡한 User Interface 제공(CLI, GUI)
* Device Interface Object: I/O Device와 interface

1. Input Device Interface Object
2. Output Device Interface Object
3. I/O Device Interface Object

* System Interface Object
  + External System과 Interface
  + External System과 커뮤니케이션하는 Detail한 방법은 숨김
* External Class와 Interface Class의 묘사
* System Context Class Diagram로부터 시작
* 각 External Class는 반드시 Interface Class와 Interface 해야 함
* UML Package Notation을 이용하여 External Class는 Package 외부에, Interface Class는 Package 내부에 표현
* Entity Objects
* 정보를 저장하는 Long lasting Objects
* 같은 Object는 일반적으로 많은 Use Case에 의해 접근됨
* 정보는 여러 Use Case에 의한 접근을 통해 유지됨
* Static Model에서 보여지는 Entity Classes와 Relationships
* Entity Classes는 Design(설계)시에 RDB와 map됨
* Control Objects
* 특징
* Use case 실행을 위한 Overall Coordination 제공
* Use Case에 참여하는 Object들을 통합(연결)
* 언제, 어떤 순서로, 어떤 object들이 Use Case에 참여할지 결정
* 간단한 Use Case는 Control Object 필요 없음
* 복잡한 Use Case는 최소한 하나의 Control Object 가지고 있음
* 종류
* Coordinator Object
  + 일의 흐름을 연결
  + Use Case에 대한 Sequencing 제공
  + 상태 의존적이지 않음
* State Dependent Control Object
  + 상태와 Event에 따른 연결
  + Finite State Machine에 의해 정의됨(statechart, state transition table)
* Timer Object
  + 주기적으로 활성화 됨
* Application Logic Objects
* 종류
* Business Logic Object
  + Client Requesting을 처리하기 위한 business specific application logic 정의
  + 주로 하나 이상의 entity object 접근
* Algorithm Object
  + Problem Domain에서 이용되는 Algorithm 캡슐화
  + 과학적, 공학적, 실시간 Domain에서 일반적
* Subsystem
* Composite or Aggregate Object
* UML에서 Package로 묘사됨
* Subsystem Dependencies가 묘사될 수 있음
* advanced relationship을 위해 Subsystem을 Composite or Aggregate Class로 묘사
* Object Structuring에서는 쉽게 식별되는 Subsystem만 결정됨

Dynamic

Dynamic Modeling

* 특징
* Object들이 각 Use Case에 어떻게 참여하는지 결정
* Use Case에서 Object interaction들의 순서를 표시
* Collaboration Diagram과 Sequence Diagram을 통해 묘사
* 종류
* Collaboration Diagram
* Use Case에 참여하는 Objects를 Graphically 묘사
* Object는 Box로 표시
* Message Interaction은 화살표로 표시
* Message의 순서를 Numbering
* Message= Event + Attributes
* ex) ATM card inserted (Card id, expiration date)
* 각 Use Case별로 하나씩 만들어짐
* Some Object는 하나의 Collaboration에만 나타남
* Some Object는 여러 Collaboration에서 나타남
* Sequence Diagram
* Use Case에서 Object interaction들의 순서를 표시
* object들 사이에서 전달되는 Message를 강조
* Objects는 Vertical Line으로 표시
  + Actor는 가장 왼쪽에 존재
* Messages는 Label이 있는 수평 화살표로 표시
* 페이지 상단에서 하단으로 갈수록 시간 경과
* Message Sequence Numbering은 optional
* Dynamic Analysis
* Use Case를 Support하기 위해 어떻게 Object들이 상호작용할지를 결정
* Actor로부터 온 External Event로 시작
* Use Case를 Support하기 위한 Objects 결정
* External Event 이후의 Internal Event의 순서 결정
* Collaboration Diagram으로 묘사
* Non-state dependent Dynamic Analysis
* State dependent Dynamic Analysis
* statechart에 의해 제어
* Control Object에 의해 실행
* Control Object가 다른 Objects를 활성화/비활성화

Software Architecture(Design)

* 설계는 두 가지로 나뉨
* Overall한 설계 : Architecture
* Detail한 설계(모듈 단위)
* Dimension of Complexity
* Technical Complexity
* Higher Technical Complexity: 기술적으로 어려움
* Lower Technical Complexity: 기술적으로 쉬움
* Management Complexity
* Higher Management Complexity: 코드가 길고 이해관계자가 많아 관리가 어려움
* Lower Management Complexity: 코드가 짧고 이해관계자가 적어 관리가 쉬움
* 이와 같은 복잡도들을 해결(분산)하기 위해 Architecture를 설계
* Software Architecture 정의
* 소프트웨어 시스템에 대한 상위 레벨의 구조적/행위적 모델
* 소프트웨어 솔루션 구조에 대한 커다란 그림을 보여줌
* 소프트웨어의 중요한 초기 설계 결정 사항들을 표현
* Common Architectural Style
* Style이란: Components + Connectors + Constraints
* Components: Computational Components
* Connectors: Components 사이의 상호작용들
* Constraints: 어떻게 결합 될지
* Pipe and Filter
* 어떤 것이 들어오면 Processing하고 그 다음으로 넘기는 스타일
* Components: Filters
* Connectors: Pipes
* Invariants
* Filter는 독립적인 Entity여야 하며, 다른 Filter와 상태를 공유해서는 안됨
* Filter는 upstream 및 downstream 필터의 identity를 모름
* Event Based, Implicit invocation(암시적 호출)
* Event에 의해서 Trigger 되기도 하고 Wait되기도 하며 일을 처리하는 스타일
* Components: Modules
* Module의 interface는 collections of procedures와, set of events 제공
* Connectors: Event registration and Procedure Invocation
* Invariants
* Event Announcer는 해당 이벤트의 영향을 받는 컴포넌트를 알지 못함
* Layered Systems
* Layer로 계층구조를 가지면서 각 계층 마다 일을 처리하는 스타일
* Components: Layers
* 각 Layer는 상위 계층에 Service를 제공하고, 하위 계층에 대한 Client 역할
* Connectors: Protocol
* Constraints
* Topological
* 인접한 Layer에 대한 interaction 제한
* Usage
* Communication Layer
* Database System and Operating System
* Repository
* 각각의 모듈들이 가운데 Repository를 가지고 일을 처리하는 스타일
* Repository는 Database가 될 수도 있고 Blackboard가 될 수도 있음
* Components
* Central Data Structure: 현재 상태를 나타냄
* A collection of independent components: 중앙 데이터 저장소에서 작동
* Traditional Database
* 입력 스트림의 transactions의 type은 실행할 프로세스를 선택하는 것을 트리거함
* Blackboard(하나의 파일)
* central data structure의 현재 상태는 실행할 프로세스를 선택하는 것의 주요 trigger
* Rule-based System
* Knowledge를 가져와서 Rule에 의해 Processing하는 스타일
* System Model: Virtual Machines
* Components: 추론 엔진 1개와 메모리 3개(지식기반, 작업 메모리, 추론 엔진 상태)
* Connectors: 데이터 접근 및 생산 규칙 적용
* Control
* 일반적으로 추론 엔진의 상태 전환
* 해석할 내용을 선택하기 위한 목표
* Software Architecture의 장점
* System 무결성
* 변경을 허용하면서 개념적 무결성을 보장
* System longevity(수명)을 지원
* 복잡도의 Control
* Component로 분해, 세부 구현을 숨김
* Component가 detail을 encapsulate, 서로 다른 Component는 서로 다른 전문지식을 가진 사람들에 의해 구현될 수 있음
* 예측가능성
* Architecture iteration은 주요 위험을 제거
* Architecture Prototype은 측정 지표의 모음(collection)을 허용
* 평가 가능성
* 잘 컴포넌트화 된 시스템은 진단, 추적, 에러 발견이 쉬움
* 재사용성
* Architecture는 대체 규칙을 정의
* Component Interface는 대체 경계를 정의
* Architecture는 다양하게 세분화된 재사용을 가능하게 함
* Component level에서 작은 규모의 재사용
* Subsystem, Product, Framework level에서 큰 규모의 재사용
* Communication
* Architecture는 이해당사자 사이의 Communication지원
* Architecture는 key Design Decision을 전달
* Architecture 설계의 근거는 trade-off를 전달
* 조직과 프로젝트 관리
* 조직 구조는 Architecture에 맞춰짐
* Component와 Subsystem은 개발, 테스트, delivery, upgrade 등의 단위로 이용됨
* Component/Subsystem은 동시에 개발됨

Task Structuring (Design)

* 특징
* System을 Task로 구조화
* Concurrent Task Structuring 기준
* Analysis Model을 Concurrent task로 구조화
* Task는 Active Object
* Task는 제어 thread를 가지고 있음
* System Activities의 Concurrent한 특성을 고려
* Concurrent task 결정
* Task Interface 정의
* Concurrent Collaboration Diagram, Task Behavior Specification 개발
* Active and Passive Object
* Object는 Active 또는 Passive임
* Active Object (Task)
* Concurrent Task : 제어 thread를 가짐
* Passive Object (Object)
* Information Hiding Object로 알려져 있음
* 제어 thread 없음
* Passive Object의 operation은 Task에 의해 실행되는데 Task의 제어 thread안에서 (Directly or Indirectly) 실행됨
* Concurrent Task를 위한 Run-time Support
* OS와 Language에서 Concurrent task 지원해야 함
* Operating System Services (Multitasking Kernel)
* Language Support for Concurrent Tasks
* Task Structuring Categories
* I/O task structuring criteria
* Internal task structuring criteria
* Task priority criteria
* Task clustering criteria
* I/O Task Structuring 기준
* Asynchronous I/O device interface task
* 각 Asynchronous I/O device에 대한 Task
* device의 I/O interrupt에 의해 활성화
* Asynchronous device는 Interrupt를 발생시킴
* Periodic I/O device interface task
* 각 폴링된 I/O device에 대한 Task
* 주기적으로 timer event에 의해 task 활성화
* 정기적으로 sampling되는 I/O device(일반적으로 Input)
* Passive I/O device interface task
* 각 Passive I/O device에 대한 Task(일반적으로 Output)
* 계산이 출력과 겹치는 경우
* Data를 생성하는 계산적인 task
* Resource Monitor task
* 여러 source들로부터 요청을 수신하는 각 I/O device에 대한 Task
* 이러한 I/O device는 요청을 조정하기 위한 Resource Monitor Task 필요
* I/O Device들의 특성
* Asynchronous I/O devices
* Input Device: 입력을 생성하면 interrupt
* Output Device: 출력을 완료하면 interrupt
* Passive(synchronous) I/O devices
* interrupt를 발생시키지 않음
* (정기적으로 polling, on demand하는)passive device로부터 input
* Passive Output Device는 주로 Passive Output Device Interface Task
* Passive Input Device는 polling될 가능성이 크므로 주로 Periodic input device interface Task
* Communications link
* Communication Protocol 사용
* Application task는 message를 사용하여 communication
* Internal Task Structuring Criteria
* Periodic Task
* 각 Periodic Activity에 대한 Task
* 주기적으로 timer event에 의해 task 활성화 됨
* Asynchronous Task
* 요청 시 실행되는 Activity에 대한 Task
* internal event나 다른 task에 의해 보내진 message의 요청에 의해 task 활성화됨
* Control Task
* statechart를 실행하는 task
* statechart의 실행은 sequential
* 각 control object당 하나의 task
* 같은 type의 multiple task를 가질 수 있음
* User Interface Task
* 순차적 user activity당 하나의 task
* Task Priority Criteria
* 중요 고려사항
* Performance 분석
* Realtime Scheduling
* Time Critical
* hard deadline을 가지고 있는 Activity
* time critical task에 Mapping
* Non-time Critical computationally intensive
* Low priority activity
* Task Clustering Criteria: 도출된 task를 grouping
* Temporal Clustering
* 같은 이벤트에 의해 활성화되는 task들을 그룹화
* task들 사이에 sequential dependency는 없음
* Sequential Clustering
* 연속적으로 실행되는 task들을 그룹화
* sequence의 첫 task는 event에 의해 활성화되고, 그 이후 연속되는 task들은 delay없이 실행되어야 함
* Control Clustering
* Control Object에 의해 trigger되는 Action, Activity와 Control Object 그룹화
* Control Object에 의해 enable/disable 되는 Activity는 결합하지 않음
* Mutually Exclusive Clustering
* 동시에 일어날 수 없는 task들을 그룹화
* 한번에 하나의 task만 실행될 수 있음
* Task Inversion
* same type의 모든 object를 하나의 task로 mapping
* 각 Object에 대해 별도의 data record를 사용하여 state information 저장
* Task간의 Interface 정의
* Message: 교환을 하는데 데이터가 들어가 있는 내용들이 주고 받아짐
* Loosely coupled (asynchronous) message communication
* 메일, 메시지 개념(보냈다고 해서 상대방이 바로 확인할 필요 없음)
* Producer는 Message 보내고 나서 자기 자신의 일 진행
* Consumer는 메시지가 없는 경우 일시 중지되고 메시지가 도착하면 Activate됨
* Consumer쪽에 message queue가 생성될 수 있음
  + FIFO, Priority message queue
* 반 쪽짜리 화살표로 메시지 표현
* Tightly coupled (synchronous) message communication
* 전화와 비슷한 개념으로 상대방은 바로 확인해야함
* Producer task는 message를 보내고 reply를 기다림
* Consumer는 메시지가 없는 경우 일시 중지되고 메시지가 도착하면 Activate됨
* 완전한 화살표로 표시
  + with reply

Consumer는 reply를 생성하고 Producer에게 보냄

* + without reply

Consumer는 message를 accept하고, 일이 끝났을 때 producer를 release해줌

* Synchronous로 하게 되면 두 task는 서로 해당 message 작업에 묶여있게 됨
* data가 올 때마다 즉각즉각 display해야하는 상황에 synchronous 사용
* Event Synchronization: Data가 있는 것이 아닌 상대방을 깨우는 목적의 메시지
* Event 종류
* External Event(Interrupt)
* Timer Event
* Internal Event
* Two task는 그 들의 operation을 synchronize해야 될 수 있음
* data를 주고 받는 경우
* message가 data를 가지고 있지 않으면 internal event 사용가능
* Source Task가 Event Signal
* Destination Task는 Event가 Signal 될 때까지 기다림
* Passive object(Information Hiding Object)
* Data를 가지고 있음
* Data Structure의 내용 숨김
* 여러 task로부터 Operation을 통해 indirect하게 Data가 Access 됨
* Operation은 Data에 대한 access를 synchronize해야함

Software Testing

* 모든 개발활동마다, 그에 따른 test activity가 있음(V Model)
* 정의
* 에러를 발견하려는 의도를 가지고 Program을 실행하는 프로세스
* 요구 조건과, 실제와의 차이를 발견하고, software item의 feature를 평가하기 위해 software item을 분석하는 프로세스
* 시스템, 컴포넌트가 지정된 조건에서 실행되고, 결과가 관찰 및 기록되고, 일부 측면에 대한 평가가 이루어지는 활동
* 용어
* Testing: 알지 못하는 결함을 발견하는 활동
* Debugging: 발견된 결함의 위치를 파악하고 수정하는 일련의 활동
* Verification: 어떤 것을 기준에 맞게 제대로 하는 중인지 확인
* Validation: 실제로 제대로 된 것을 하고 있는지 확인
* Error: 개발자의 설계 오류
* Fault: 시스템의 Bug로 Debugging을 해서 고쳐야 하는 것
* Failure: 시스템의 이상 동작을 감지한 것
* Static Testing/ Dynamic Testing
* Static Testing
* Static Analysis: 구조, 내용, Documentation, code를 기초로 System을 평가하는 프로세스
* Review: 회의를 포함할 수 있는 프로세스, 의견 및 승인을 위해 작업 결과물이 일부 이해 관계자들에게 제공됨
* Dynamic Testing
* Dynamic Analysis: Program의 실행 중 동작을 기반으로 System을 평가하는 프로세스(resource 사용, memory 누수 등)
* Dynamic Testing: Test item의 실행이 필요한 Test
* Testing Principles
* 완전한 테스팅은 불가능
* Testing은 Bug가 존재하지 않음을 보여줄 수는 없음
* 결함 집중 현상(Bug를 찾은 곳에 Bug가 더 많이 있을 확률이 높음)
* 기존 테스트케이스에 대한 내성효과로 새로운 테스트케이스의 개발이 필요함
* 결함은 조기에 빨리 발견하는 것이 좋음
* 암묵적/명시적 Needs를 충족해야 하며 단순히 Error만 없다고 되는 것은 아님
* Testing은 Context Dependent(상황에 따라 서로 다른 Testing Needs를 가짐)
* Test Level, Test Phase
* Unit Testing
* 소프트웨어 각 단위(Component, module, function, class)가 개별적으로 테스트 됨
* 테스트 수행을 위해서는 Stub, Driver가 필요함
* 주로 개발 당사자에 의해 수행됨
* Integration Testing
* 각 Unit을 통합하여 테스트 수행
* 개발자 및 테스터에 의해 수행됨
* SW Architecture를 고려하여, 테스트 대상과, 테스트 방법, 통합 순서, 테스트 환경의 사전 계획이 필요함
* System Testing
* 전체 시스템을 대상으로 테스트 수행
* 기능 뿐만 아니라 Security, Performance, Reliablity, Usability Portability등 다양한 품질 요소를 테스트
* 실 환경과 동일하거나 유사한 환경(사용자 관점)에서 테스트
* 독립적인 테스터에 의해 수행
* Acceptance Testing
* 전체 시스템이, 실사용 환경 또는 그와 유사한 환경에서 테스트
* 주로 사용자, 고객에 의해 수행 되나, 독립적인 테스트 조직/회사에 의해 수행되기도 함
* Maintenance Testing
* 시스템이 Release된 이후, 시스템에 변경이 발생했을 때 수행하는 테스트
* 변경된 부분에 대한 테스트도 중요하지만, 전체 시스템에 대한 영향 평가 및 Regression Testing(변경 전에 잘되던 것이 되는지 테스트)이 중요
* Test Design Approach
* Specification-based testing(Black-box testing)
* 요구사항 및 명세서에 기반한 테스팅
* 소프트웨어 내부 구조에 대한 지식은 필요없음
* 소프트웨어 내부 구조를 Black Box로 간주
* 소프트웨어 개발 프로세스의 모든 단계에 적용가능
* 아무리 테스트를 해도 실제로 코드의 얼마가 테스트 되었는지는 모름
* 완전한 테스팅은 불가능(Input의 일부의 조합만 테스트 할 수 있음)
* 기법 적용을 통해 Random하게 생성된 것보다 효과적/효율적인 Test Case의 도출이 가능함
* Structure-based testing(White-box testing)
* 내부 흐름, 구조 등 SW 구현 코드에 기반한 테스트
* 프로그래밍 스킬이 요구됨
* 소프트웨어 개발의 모든 단계에 적용가능하지만 주로 개발자의 Unit Testing시 수행
* 요구사항에 대한 미구현 오류를 발견할 수 없음
* Experience-based testing
* Error Guessing, Checklist-based testing, Defect-based Test
* Exploratory Testing
* 정해진 테스트목적에 대해서 시스템에 대한 학습, 테스트 설계, 테스트 수행이 동시에 이루어지는 테스트 방법
* Risk-based Testing
* Risk를 평가하여 Risk에 따라 test 수준을 다르게 함
* Risk Exposure=Technical Risk(발생가능성) \* Business Risk(영향)
* Software Testing Process
* Planning – Monitoring and Control – Analysis – Design – Implementation – Execution – Completion

Test Design

* Test Case: Precondition, Input, Expected Result로 구성
* Specification Based Testing
* Equivalence Partitioning Testing(동등분할 테스팅)
* 명세서를 기준으로 여러 가능한 Input에 대해 동일한 결과를 값들을 하나의 Equivalence class로 구분하고, 각 Class 별로 Input을 선택하는 테스트
* Domain 분석을 통해 Valid, Invalid Class로 Decompose하는 것이 핵심
* 적절한 Coverage를 만족하면서 체계적으로 Test Case를 줄일 수 있음
* Coverage = (Number of Covered Partitions/Total Number of Partitions)\*100
* Boundary Value Analysis Testing(경계값 분석 테스팅)
* 결함은 경계 값에서 많이 나타난다는 것에 착안한 방법
* Min-1, Min, Min+1, Normal, Max-1, Max, Max+1
* Exception handling 확인에 좋음
* Combinatorial Testing(조합 테스팅)
* 하나의 결과에 영향을 미치는 여러 Parameter의 조합을 테스트
* Interdependent Parameter일 때 사용
* Full Combination은 현실적으로 불가능한 경우가 대부분
* 일반적으로 Pair-wise Testing 많이 사용
  + 2개 요소의 상호작용에 대한 테스트가 결함을 보다 많이 발견한다는 경험에 기반
  + 테스트 대상 아이템의 모든 Parameter 중에 두 개 Parameter의 모든 가능한 값을 조합
* Decision Table Testing(의사 결정 테이블 테스팅)
* 시스템이 구현해야하는 복잡한 비즈니스 규칙을 Condition에 따른 Action Table로 구성하여 테스트케이스 개발
* State-Transition Testing(상태 전이 테스팅)
* State Transition Diagram을 기반으로 각 state 또는 state를 발생시키는 event, state간의 transition의 수행을 만족하는 테스트를 수행함
* 각 state를 최소한 한번 이상 수행하는 테스팅
* 각 transition을 최소한 한번 이상 수행하는 테스팅
* transition의 path를 고려한 테스팅
* Tip
* 시스템의 상태에 따른 제약이 있는 Event 테스트에 용이
* Use Case 명세를 기반으로 State-transition diagram을 작성하여 테스트에 활용
* Information System에서는 시스템이 관리하는 주요 데이터의 상태에 대해 적용가능
* Use Case Testing(유스케이스 테스팅)
* Use Case의 흐름을 테스트 시나리오로 활용하여 테스트를 수행
* 기본 흐름 뿐만 아니라 대안 흐름, 예외 흐름에 대해서도 테스트가 수행되어야 함
* Tip
* Use Case별로 Activity Diagram을 작성하여 Test Case 추출
* 개별 Use Case의 Testing 분만 아니라, 관련 Use Case간의 상호작용에 대한 통합 시나리오 Testing을 수행할 수 있음

Project Management

* 정의
* 혼자서 달성 할 수 없는 목표를 달성할 수 있도록 하기 위하여 다른 사람의 활동을 계획하고 조정하는 것
* 관리 활동의 5가지 Type
* Plan: 할 일, 일정, 필요한 자원을 결정하는 것
* Organize: 필요한 역할과 관계를 결정하는 것
* Staff: 역할 수행(달성)에 필요한 인력을 확보하는 것
* Direct: 사람의 역할 수행(달성)을 위해 Guide
* Control: 작업활동을 모니터링하고 적절한(올바른) Action을 적용하는 것
* Project Manager의 역할
* Plan the work
* Estimate
* Prepare schedule and budget
* Identify risk factors
* Make commitments
* Acquire Resource
* Measure the following
* Control the work activities
* Lead and direct
* Represent the project
* Man Month: 한 사람이 한 달 동안에 할 수 있는 작업의 양, 시스템이나 프로그램을 개발할 때 기획, 입안 단계에서 지정해야하는 중요한 요소
* 요약
* Software Project를 관리하는 것은 어려운 작업
* 좋은 프로젝트 관리는 소프트웨어 프로젝트르 성공적으로 완료하고 성공적인 소프트웨어 집약 시스템을 Deliver할 수 있도록 함