Use-Case 기반 객체지향 프로젝트 스케줄링 기법

(A Use-Case Based Object-Oriented Project Scheduling Technique)

허 진 선 † 최 시 원 † 김 수 동 ††
(Jin Sun Her) (Si Won choi) (Soo Dong Kim)

요 약 객체지향 개발이 보편화 되었지만, 객체지향 프로젝트를 지원하는 소프트웨어 관리 기법에 대한 연구가 미흡하다. 또한, 기존의 소프트웨어 관리 기법을 객체지향 프로젝트에 적용하는 것 역시 어려움이 있다. 특히, 늘어가는 객체지향 기반의 대형 프로젝트에 적합한 프로젝트 계획 기법에 대한 연구가 미흡하다. 본 논문에서는 객체지향 프로젝트를 위한 관리 요소 중 스케줄링에 대한 체계화된 기법을 제안한다. 시스템의 기능적인 요구사항이 기술된 Use Case 다이어그램을 이용하여 객체지향 프로젝트 스케줄을 도출해가는 과정을 Use Case 식별, 상호의존성 분석을 통한 초기 PERT 차트 작성, 각 Use Case의 특성 규명, Iteration 개수 결정, Iteration에 Use Case 할당, 유용한 자원과 제약 사항 고려, Revised PERT 차트 작성의 7 단계로 나누어 제안한다. 각 단계에 대한 입력물과 중간 산출물, 그리고 수행 지침을 제시한다. 본 논문의 프로젝트 스케줄링 기법은 짧은 개발 기간 동안 고품질의 소프트웨어 생산에 목적을 둔 프로젝트 계획 단계에서 효율적인 기법으로 사용된다.

키워드 : OOP, 프로젝트 관리, 프로젝트 계획, 프로젝트 스케줄링, Use Case, PERT 차트

Abstract Object-oriented development has been generalized, but object-oriented project planning and scheduling techniques have not been studied enough. Furthermore, it is difficult to apply the conventional software management techniques to object-oriented projects. Especially, the large scaled projects are increasing, but the project planning techniques for these large scaled projects have not been proposed enough. In this paper, we propose systematic techniques for OO based project scheduling. We suggest a 7 step-process for deriving the OO project schedule from the use-case diagram which is describing the functional requirements of the system. The proposed process includes identifying use-cases, drawing preliminary PERT chart through interdependency analysis, identifying characteristics of each use-case, determining the number of iteration, assigning use-cases to iteration, considering available resource and constraints, drawing revised PERT chart. Each step has the explanation of the input, output, and the guidelines needed to perform the step. The project scheduling technique proposed in this paper can be used effectively in the planning phase which the purpose is to plan a development schedule to yield the high quality software in minimum time.

Key words : OOP, Project Management, Project Planning, Project Scheduling, Use-Case, PERT Chart

1. 서 론

1.1 동기

현재 UML을 이용한 OOA/D, JAVA, Enterprise Java Beans(EJB)를 이용한 소프트웨어 개발, 즉 OO기반의 프로젝트가 확대되고 있는 추세이다. 그러나 OO기반 프로젝트 관리에 대한 체계화된 기법이 부족하여 효과적인 프로젝트 진행에 어려움이 있다. 구조적(재래식)방법의 프로젝트 관리 기법에 대한 연구는 많이 이루어진 반면, OO기반의 프로젝트 관리 기법에 대한 연구는 미흡하다. 예를 들어, Function Point(FP)와 같은 문제 기반 생산성 추정 메

† 비 회 원 : 숭실대학교 컴퓨터학과
jsher@otlab.ssu.ac.kr
swchoi@otlab.ssu.ac.kr
†† 종신회원 : 숭실대학교 컴퓨터학과 교수
sdkim@comp.ssu.ac.kr
논문접수 : 2002년 4월 22일
심사완료 : 2002년 12월 17일

트릭이 OOSE(Object Oriented Software Engineering)에는 명확히 제시되어 있지 않다. 프로젝트 관리 기법 중 프로젝트 계획 기법은 프로젝트의 성패를 좌우할 만큼 중요하다. 따라서, 객체지향 프로젝트가 많아 짐에 따라 객체지향 기반 프로젝트 계획의 실용적인 기법과 지침에 대한 명확한 제시가 요구된다. 또한, 기본적으로 OO프로젝트 관리에 있어 기본적인 관리 기술뿐 아니라 OO기술에 대한 이해도 필요하다.

프로젝트의 대형화, 복잡화에 따른 더 효율적이고 실용적인 스케줄이 필요한 실정이다. OO기반이라 할지라도 규모가 작은 프로젝트일 경우 관리자가 가지고 있는 고유한 경험에 의해서 프로젝트의 계획이 어느 정도 가능하다. 그러나, 프로젝트 규모가 대형화 될수록 관리자의 경험만을 바탕으로 계획을 세우기에 한계점이 있다. 따라서, 본 논문에서는 프로젝트 계획 시의 어려운 점을 해결하기 위한 방안을 제시한다.

1.2. 논문의 범위 및 구성

기존의 연구에서는 OO 기반 프로젝트 계획 시 Use Case 다이어그램을 이용하여 프로젝트 계획을 도출하는 연구가 시도되었다. 그러나 기존의 연구는 관리자의 경험에 의존함으로 체계화된 유도 방법을 제시하지 못했다. 본 논문에서는 OO 기반 프로젝트 계발 계획을 위하여 요구사항 분석 모델을 통하여 보다 체계적이고 구체적인 프로젝트 스케줄을 도출하는 기법을 제시한다.

본 논문의 구성은 다음과 같다. 먼저, 2장에서는 관련 연구로서 기존 몇몇 학자들의 연구 진행 상황과 한계점을 살펴본다. 3장에서는 본 논문에서 사용되는 기본적인 개념들인 Use Case 모델과 스케줄링 차트에 대해 간략히 기술한다. 4장에서는 7가지 단계를 통해 Use Case 모델로부터 OO 스케줄이 도출되는 과정을 설명한다. 5장에서는 병원 업무를 도메인으로 4장에서 기술된 각 단계의 지침을 적용한 사례 연구를 진행한 뒤, 마지막으로 6장에서는 본 논문에서 제시한 기법에 대한 평가를 한다.

2. 관련 연구

2.1 Bruegge와 Dutoit의 연구[1]

Bruegge와 Dutoit의 연구에서는 객체지향 프로젝트 계획에 Gantt 차트와 PERT 차트를 사용하고 있으며 프로젝트 시작 시, 대략 다음의 세 단계를 거쳐 프로젝트 계획을 진행한다.

첫 번째 단계로 프로젝트 개발을 위한 태스크를 식별한다. 태스크 식별은 해당 문제에 대한 진술서, 라이프 사이클의 활동들, 과거의 경험들을 기반으로 시작한다. 사용자와 프로젝트 매니저는 프로젝트 진행사항 검토에 대한 동의 하에 각각의 산출물을 최소한 하나의 태스크에 할당한다. 작업들은 한 사람이 평가하고 수행하기 편리한 단위로 초기에 분해된다. 이렇게 초기에 분해되는 것을 Work Breakdown Structure(WBS)라고 하고 거대한 To Do 리스트와 비슷한 형태를 갖춘다. 보통의 경우, 요구사항과 산출물이 상세할수록, 더 정확한 태스크를 정의하기 쉬워진다. 잘 정의되지 않은 요구사항은 매니저가 태스크 계획을 재계획하고 정제하기 위한 추가적인 시간을 할당해야 하며 초기의 WBS는 지나치게 상세해서는 안 된다. 이것은 계획이 개정될 때, 추가적인 작업만을 만들 뿐이며 매니저는 짧은 주기만을 위한 상세한 WBS를 작성한다.

두 번째로 일단 태스크 식별을 마친 후에, 식별된 태스크들 사이의 상호의존도를 살펴보아야 한다. 상호의존도를 살펴봄으로써 더욱 효율적으로 자원을 태스크에 할당할 수 있게 된다. 각 태스크가 요구하는 작업 생성물을 살펴봄으로써 태스크들 사이의 상호의존도를 발견할 수 있다. 발견된 결과물을 이용하여 태스크의 집합과 상호관계로 구성된 태스크 모델을 만든다. 그 다음 스케줄을 만들어 이 태스크 모델을 자원과 시간에 매핑하면 된다.

마지막으로 초기 스케줄을 작성한다. 초기 스케줄은 태스크 모델을 시간과 자원에 매핑하면서 생성된다. 태스크 상호의존도, 추정된 태스크 별 기간, 참가자의 수를 이용해서 초기 스케줄을 작성한다. 초기 스케줄은 사용자 인터뷰, 사용자 검토, 산출물 등을 포함한 클라이언트와 사용자 사이의 주요한 상호작용에 대한 일정을 계획하기 위해 사용된다. 이 일정은 할당 태스크의 변화가 있을 시에도, 전체 일정에 대한 변화가 있어서는 안 된다.

위의 Bruegge와 Dutoit의 연구에서는 WBS를 이용한 태스크 식별을 거쳐 태스크 사이의 의존도를 분석한 뒤, 태스크를 스케줄에 매핑 기법에 대해 제시하고 있다. 전체적인 프로젝트 계획의 3단계는 합리적이고 논리적으로 진행된다. 그러나, 객체지향 프로젝트의 특성을 고려한 계획 방법의 제시가 미흡했고 태스크간 의존도는 어떻게 식별하는 지에 대한 구체적인 기준과 상세 지침이 설명되어 있지 않다. 또한, 각 태스크 별 작업기간을 설정하기 위한 구체적인 기준이나 지침이 미약하며 무엇보다 중요한 태스크와 스케줄간의 매핑에 대한 적용 기준과 매핑방법에 대한 지침에 대한 한계점을 나타내고 있다.

2.2 Pooley와 Stevens의 연구[2]

Pooley와 Stevens의 연구에서는 객체지향 소프트웨어 개발 시, Use Case를 이용하여 프로젝트 계획을 한다. 먼저 추출된 Use Case에 대한 정보를 제시하면서 Use Case를 나열한다. 각 Use Case에 대한 정보로는 Use Case의 의미, Use Case의 요구 대상, Use Case의 요구량, Use Case의 위험 내포도, Use Case 구현 예상 소요 시간 등이 있다. 또한, 모든 Use Case 구현에 소요되는 시간이 전체 시스템에 보급되는 시간보다 짧아야 한다. 제공할 Use Case가 정해지면, 각 Use Case를 구현하는 순서와 Use Case가 어떠한 시스템 Iteration에 포함되는지를 결정한다.

Pooley와 Stevens의 연구에서는 비교적 객체지향 프로젝트에 적합한 기법을 제시하고 있다. 그러나, 앞에서 살펴본 바와 같이 아주 개략적인 지침만이 제시되어 있을 뿐, 더욱 상세한 지침에 대한 언급은 부족하다. 위에서 말한 바와 같이, 각 Use Case가 가져야 할 정보에 대한 나열만 있을 뿐, 구체적인 지침이 설명되어 있지 않다. 어떠한 방법으로 이러한 Use Case의 정보를 도출하는지, 그리고 각 Use Case를 나열한 뒤에 구현 순서를 어떤 방법으로 결정하는 지, Use Case가 어떤 Iteration에 어떤 기준을 가지고 배당이 되는 지에 대한 구체적인 설명이 부족하다는 한계점이 있다.

2.3 Cantor의 연구[3]

Cantor의 연구에서는 마스터 스케줄(Master Schedule), 마크로 스케줄(Macroschedule), 마이크로 스케줄(Microschedule)로 나누어 스케줄을 작성할 것을 제시하며 객체지향 프로젝트를 위해서는 여러 수준의 스케줄 작성을 제안한다. 이 세 가지 수준의 스케줄은 다음과 같다.

마스터 스케줄은 전체적인 프로그램 관점을 보여주고 있으며 고객과 의사소통 시에 사용되는 수준으로 기술된다. 마스터 스케줄은 외부적으로 드러나는 계약사항과 주요한 상호의존도 정도만 나타낸다. Cantor의 연구에서는 Build라는 개념이 쓰이는데 빌드(Build)는 미리 결정된 기능성을 만족하고 한 개발 주기를 완전히 마친 시스템의 응집된 버전을 의미한다. 마스터 스케줄 작성 시, 각각의 점진적인 빌드에 포함될 내용을 Use Case로서 기술하는 것이 이상적이라 하고 있다.. 마스터 스케줄 작성시 중요시되는 4가지 원칙이 있다. 첫째, 시각적인 내용이다. 고객이 제일 필요로 하는 내용과 고객의 검증이 필요한 내용에 대한 우선순위를 부여하는 것이 한 원칙이 된다. 두 번째 원칙은 기술적인 불명확성으로써 개발 기간이 불명확한 컴포넌트는 일찍 개발을 시작하는 것이 좋음을 의미한다. 세 번째 원칙은 중대한 기능성으로써 완전하게 작동하는 시스템을 보급하는 데에 필요한 핵심적인 기능들은 첫 번째 빌드에 포함해야 함을 의미한다. 마지막으로 자원 균일화이다. 빌드 마다 인력변동을 최대한 줄이도록 기능성을 스케줄링 한다.

마크로 스케줄은 프로젝트 수준의 관점을 제시하고 있다. 현재 개발의 하루 별 관리를 위해 사용되며 최소한 한 달에 한번은 갱신되어야 한다. 마스터 스케줄의 내용을 모두 포함하고 있으며, 개발 단계, 내부적인 이정표, 내부적으로 사용되는 산출물, 시험 계획 활동, Infrastructure 관리에 대한 내용을 포함하고 있다

마이크로 스케줄은 서브시스템 팀들과 내부적인 상호의존도를 관리한다. 마이크로 스케줄은 계획 프로세스 중에 작성되지 않으며 개발 단계 수행 시에 작성된다. 마이크로 스케줄은 통합 시 필요한 Use Case에 대한 부분을 포함하고 있다. 이 스케줄은 유연성이 높고, 매우 상세하며, 프로젝트 외부에 절대 공개되지 않는다. Inception단계의 마이크로 스케줄에는 Use Case 완료, 시스템 시험 계획, 프로세스 태스크의 3가지 종류가 있으며, 특별히 Use Case 완료에서는 주마다 어떠한 Use Case를 완료해야 하는지, 어떠한 Use Case 다이어그램들을 완료해야 하는지, 사용자 인터페이스에서 어떠한 Use Case를 프로토타입해야 하는지, 누가 각 개발 완료에 대한 책임을 맡고 있는지를 계획해야 한다. Elaboration단계의 마이크로 스케줄은 패키지 레벨의 Use Case들과 클래스 다이어그램들, 시퀀스 다이어그램들, 서브 시스템 통합 시험 사례 등에 대한 것을 계획해야 한다.

Construction 단계의 마이크로 스케줄은 다음과 같이 작성된다. (1) Use Case 구현 순서를 계획한다. (2) Use Case구현 계획을 만족하기 위한 클래스 메소드 구현의 순서를 도출한다. (3) 클래스 메소드 구현의 상호의존도와 자원 레벨링을 정렬하고 노력과 상호의존도를 기반으로 클래스의 구현을 스케줄링 한다. (4) 개발 현실을 만족하기 위한 Use Case순서를 다시 짠다. (5) 새로운 Use Case 순서를 재검토한다. (6) 재검토 결과 만족스러우면, 마이크로 스케줄을 작성한다. 이렇게 6 단계를 거쳐 마이크로 스케줄을 작성하고, 또한 Svedlow 다이어그램을 작성한다. Svedlow 다이어그램은 Use Case와 클래스 메소드 간의 상호 도표로서 각 시스템 통합마다 구현되어야 할 클래스 메소드를 유도하는 데에 사용된다. Transition단계의 마이크로 스케줄은 스케줄의 초점이 Use Case 추적에서 결점(defect) 추적으로 전환된다.

Cantor의 연구는 다른 연구에 비해 어느 정도 상세한 지침을 제시하고 있다. 3레벨로 나누어 스케줄을 작성하는 것은 유용하게 사용될 수 있는 제안이다. 그러나, 이

연구 또한, Use Case간의 상호의존성을 고려해야 함을 언급할 뿐, 상호의존성이 될 수 있는 요인, 상호의존성 도출 방법, 검증 방법에 대한 상세 지침이 미약하다. 그리고, 각 Use Case의 특성은 스케줄링 시에 중요한 반영 요소가 되지만, 이러한 특성에 대해 고려한 부분이 부족하다. 또한, 작업 사이의 상호관련성과 흐름을 한눈에 알아보기 쉬운 PERT 차트 이용을 배제함으로써 PERT 차트의 역할을 담당하는 스케줄이 요구되어 한계점을 드러내고 있다.

2.4 Fayad, Tsai, Fulghum의 연구[4]

Fayad, Tsai, Fulghum의 연구에서 OO로 변화된 프로세스에 대해 3단계로 나누어 기술한다. 즉, 계획 및 선 프로젝트 단계, 기술 삽입 단계, 프로젝트 관리 단계로 나눈다. 본 논문과 관련된 계획 및 선 프로젝트 단계는 효과적인 개발 계획과 기존 문화 변경이라는 크게 2개의 활동으로 나누어진다. 효과적인 개발 계획에서는 전이 계획과 소프트웨어 개발 계획이 필요하다. 기존 문화 변경 활동은 크게 4 단계로 구성된다. 첫째, 문화 변화에 대한 이해를 한 다음 두 번째, 관리 공약에 대한 진술을 한다. 세 번째, 고객에게 객체지향 소프트웨어 공학을 판매한다. 네 번째, 소프트웨어 개발팀을 준비한다.

[4]의 연구에서는 전체적이고 개략적인 객체지향으로의 전이 프로세스 프레임워크를 제공한다. 따라서, 계획에 대한 언급은 개략적인 프레임워크에서 일부분을 차지해 좀 더 자세한 설명이 요구된다. 효율적인 개발 계획 단계 시, 소프트웨어 개발 계획(SDP)을 작성하라고 말한다. 프로젝트의 범위와 개발자가 이행해야 할 약속, 사용되는 기술과 툴, 내포된 위험 요소 등을 기술한 가장 핵심적인 계획서이다. 그러나, 효과적인 계획 활동을 위해서 이보다 더욱 상세한 지침과 고려사항의 기술이 필요하다.

3. 배 경

3.1 Use-Case 모델

Use Case 모델은 '공통 요구사항 명세서'와 도메인 지식 등을 기반으로 기능적 요구사항을 Use-Case 단위로 표현한다. 즉, 시스템이 무엇을 수행하며, 시스템 주위에는 어떠한 것이 있는지를 나타내는 모델이다. Use Case 모델은 최종 사용자, 도메인 전문가와의 의사소통 시에 사용되며 이를 통해 반영되지 않은 요구사항이 있는지 점검할 수 있다.

Use Case 모델은 Use Case 다이어그램과 Use Case 명세의 2가지 구성요소로 이루어진다. Use Case 다이어그램은 시스템의 기능적인 요구사항을 설명하기 위한 다이어그램이다. 시스템의 기능적인 요구사항이란 개발자와 사용자 사이에 개발하려고 하는 시스템의 환경이나 제공되는 기능들의 계약이라고 할 수 있다. 즉 사용자의 요구에 따른 시스템의 행위 또는 반응을 말한다. 시스템의 기능은 Use Case로서, 시스템과 상호 작용하는 시스템 주변의 사용자 또는 시스템은 Actor로서 표현한다. 그리고 Use-Case 명세란, Use Case의 사건 흐름을 파악할 수 있게 하는 명세서이다. Actor와 Use Case가 요구사항에서 발견된 후, 시스템이 Actor와 어떻게 상호 작용하고 각각의 Use Case에서 무엇을 하는지를 설명한다[5].

3.2 스케줄링 차트

프로젝트를 계획할 때 관리자는 일련의 이정표(Milestone)를 설정해야 하고 이 이정표에 도달하면 현재 단계의 태스크를 완료했음을 의미한다. 이 이정표들이 각 단계를 구분하는 시점이며 각 단계의 이정표에서는 정기적인 진행보고서를 제출한다. 이정표를 설정하기 위해 진행 프로젝트를 위한 소프트웨어 프로세스를 여러 태스크로 구분한다. 프로젝트 일정계획은 프로젝트의 총 태스크를 각 태스크 별로 분리하여 완료되는데 소요되는 시간을 판단하는 것이다. 일반적으로 각 태스크 중에서 몇몇 태스크는 병행적으로 처리할 수 있으므로 관리자는 병행적인 태스크를 조정하여 계획한다. 어떤 태스크가 완료되지 않았기 때문에 전체 태스크가 지연되는 일이 발생하지 않도록 해야 한다.

PERT 차트는 프로젝트를 구성하는 여러 태스크들 사이의 관계 및 흐름을 그래픽으로 표현한 것으로 태스크 네트워크라고도 한다. PERT 차트에서는 태스크나 업무는 박스로 표현하며 화살표는 각 태스크간의 의존성을 보여주는데 사용된다. 즉, PERT 차트는 태스크간의 상호관련성, 결정경로, 경계시간을 보여준다. 또한 태스크 수행의 병렬성을 나타낼 수 있으므로 자원을 할당하는데 도움을 주며 관리자로 하여금 프로젝트를 감시하고 제어할 수 있도록 해 준다.

4. Use Case로부터의 OO 스케줄 도출 단계

프로젝트 관리의 핵심은 공통의 목적을 효과적으로 달성할 수 있도록 조력하는 것이다. 즉, 프로젝트 관리의 목적은 최소한의 자원으로 최소한의 시간에 최고의 품질을 가진 소프트웨어를 만들어내는 데에 있다. 또한, 프로젝트 수행 동안에 위험 요소(Risk)를 줄이는 데에 있다. 위험 요소를 줄이기 위하여 Iteration을 여러 번 두어 프로젝트 계획을 작성하는 형식으로 스케줄링을 진행시켜 나가는 것이 효과적이다[6][7]. 본 논문에서 제시하는 프로젝트

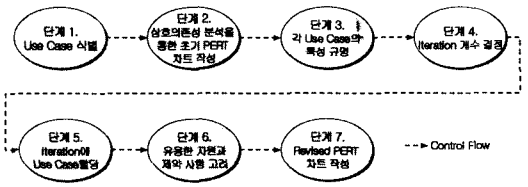

그림 1 객체지향 스케줄링 프로세스

스케줄링은 다음 그림 1의 7 단계를 통해 이루어진다. 일정계획은 태스크간의 관련성을 PERT 차트로 나타낸다.

그림 2는 객체지향 스케줄링 프로세스의 입력, 출력 흐름을 보여주는 DFD이다. 즉, 각 단계에서 나오는 산출물과 그 산출물들이 사용되는 단계를 나타낸다. 이 그림에서 제시된 흐름에 따라 각 단계에 대한 입력물, 중간 산출물, 그리고 지침에 대한 설명을 하고자 한다.

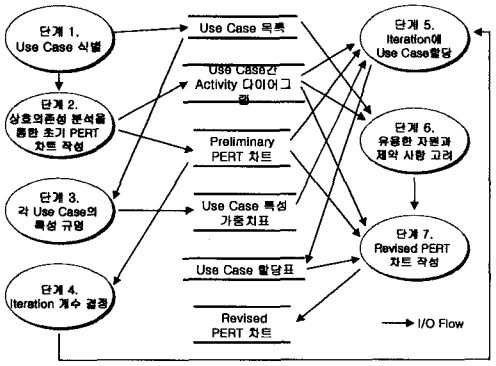

그림 2 단계간 산출물 흐름

4.1 단계 1:Use Case 식별

단계 1은 Use Case 식별 단계이다. 먼저 요구사항이 분석된 후, 개발해야 할 기능 모델인 Use Case 다이어그램을 입력물로 받아 Use Case 다이어그램에 기술된 Use Case들을 나열한 뒤 기능적 고찰을 수행한다. 각 Use Case의 기능에 대해 파악하는 과정에서 Use Case의 Granularity를 검사해야 한다. Use Case 중 지나치게 기능성이 크거나, 작거나, 추상화 수준에 차이가 있을 경우에 Granularity가 다르다는 것을 의미한다[8]. Granularity가 비슷하면 계획을 유도하기에 편리하지만 각 Use Case 마다 Granularity의 차이가 클 경우에는 Use Case 다이어그램을 정제하여 맞추어야 한다. 또한, 본 단계에서 개발하지 않을 Use Case를 식별한다. 즉, Use Case간 Generalization관계의 Parent Use Case는 개발 시에 배제할 것을 미리 고려한다. 단계 1의 결과물은 다음 표 1과 같은 Use Case 목록이 된다. Use Case는 개발 시스템의 모든 기능을 포함하고 있기 때문에 Use Case만을 통한 프로젝트 계획은 타당하다.

표 1 Use Case 목록

UC ID	Use Case	설 명
UC1	A	...
UC2	B	...

4.2 단계 2: 상호의존성 분석을 통한 초기 PERT 차트 작성

단계 2 에서는 Use Case들 간의 상호의존성을 분석한다. 분석된 상호의존성을 기반으로 전체적인 Use Case의 개발 순서를 파악하여 계획 시에 반영한다. 상호의존성이 의미하는 것과 작업 개발 순서가 의미하는 것은 다르다. 상호의존성이란 어떠한 Use Case를 개발하기 위해서 이 Use Case가 의존하고 있는 다른 Use Case를 반드시 그 이전에 개발해야 함을 의미한다. 어떠한 Use Case가 다른 Use Case보다 먼저 개발되었다고 하여 그 두 Use Case간에 상호의존성을 갖는 것은 아니다. 상호의존성을 파악하기 위한 몇 가지 지침이 있는데, 먼저 Use Case간의 전체적인 실행 흐름을 고려한다. 이는 우선적으로 매니저의 요구사항에 대한 전체적이고 포괄적인 이해가 요구된다. 전체 Use Case간의 실행순서에 대한 개요를 작성한 후, 병렬적으로 수행 가능한 Use Case 군으로 분류하여 분류된 Use Case 내에서 실행순서가 명확히 드러나는 Use Case를 표시한다.

Use Case 다이어그램을 입력물로 받아 Use Case간의 상호의존성을 고려할 때, 우선적으로 인접한 기간에 개발을 수행하는 것이 적절한 관계가 있다. 즉, Use Case 다이어그램에 명시적으로 드러나는 Use Case간의 실행 순서를 알 수 있는 의존성의 타입으로는 Include, Extend, Generalization 등이 있다[9]. 이들 Use Case간의 의존성을 아래의 표 2와 같이 표기하고 각각의 타입에 대해 아래에 설명된 기준을 적용한다.

표 2 Use Case간 의존성 추출표

의존성 타입	추출 사항
Include	UC01 ??→ UC02
Extend	UC01 ??→ UC02
Generalization	UC14, UC15 ??→ UC13

??→ : 의존성

위의 표 2에서 밑줄 그은 UC는 우선 개발이 기대되

는 Use Case이다. 의존성 타입 Include의 추출사항을 보면, UC01이 UC02를 Include하는 관계로서 UC01은 UC02에 의존적이다. 즉, UC02를 개발한 이후에 UC01을 개발하는 것이 좋으며 이들의 개발 시기를 인접해서 두는 것이 좋다. 의존성 타입 Extend의 추출사항은 UC01이 UC02를 Extend한 것으로 UC01은 UC02에 의존적임을 나타낸다. 즉, UC02를 개발한 이후에 UC01을 개발하는 것이 좋으며 이 두 Use Case 또한 비슷한 시기에 개발하는 것이 좋다. 의존성 타입 *Generalization*의 추출사항을 보면, UC14와 UC15를 일반화한 것이 UC13임을 나타내는 것으로 상위 Use Case인 UC13에 대한 실제적인 개발을 하지 않고 UC14, UC15를 개발한다. UC14와 UC15는 같은 기능을 다른 방식으로 수행하는 것이므로 로직이 비슷한 경우가 대부분이다. 따라서 가능한 같은 개발자가 비슷한 시기에 이 두 Use Case를 개발하는 것이 좋다.

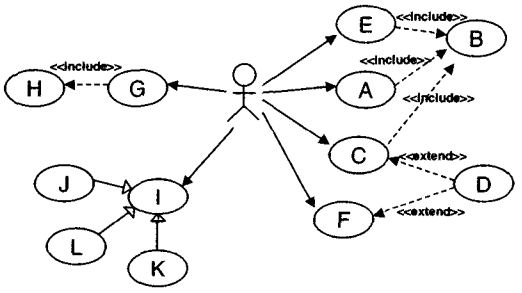

그림 3 의존성 타입 예제

다음은 Use Case간의 명시적인 의존성 타입에 대한 개발 지침이다.

- Include관계일 경우는 Include된 Use Case를 우선 개발한다. 왜냐하면 Include한 Use Case의 기능은 Include된 Use Case를 포함하여 모든 기능이 끝나야 비로소 Include한 Use Case의 기능이 끝났다고 볼 수 있기 때문이다. 즉, 위의 그림 3의 예제에서 B를 E, A, C가 Include하고 있을 때에 B를 먼저 개발하고, G와 H의 관계에서도 마찬가지로 H를 먼저 개발한다.
- Include관계에서 Base Use Case가 많을 경우, Include된 Use Case를 우선 개발한다. 즉, 위의 예제에서 B를 Include하는 Base Use Case가 E, A, C 와 같이 여러 개일 경우, B를 우선 개발한다.
- Base Use Case가 1개일 때와 Base Use Case의 Total Weight가 낮을 경우, Include한 Use Case와 동시 개발한다. Total Weight를 구하는 방법에 대해 단계3에 설명되어 있다. 위의 예제에서 보면, H를 Include하는 Use Case가 G하나이므로 H와 G를 동시에 개발하는 것이 좋다. 또한, B를 Include하는 E, A, C의 Total Weight가 낮다면 이들을 동시에 개발하는 것이 좋다.
- Extend관계일 경우에는 Extend된 Use Case를 Extend한 Use Case다음에 스케줄링 한다. 즉, 위의 예제에서, C와 F를 D가 Extend하고 있을 때에, C와 F를 D보다 우선 개발한다.
- Extend된 Use Case의 Base Use Case가 하나일 경우와 Base Use Case의 Total Weight가 낮을 경우, 이 Use Case들을 동시 개발한다. 위의 예제에서 D의 Base Use Case가 C나 F만 있다면, 이들 둘을 동시에 개발한다. 또한, D의 Base Use Case C와 F의 Total Weight이 낮다면, D와 C와 F를 동시에 개발한다.
- Extend된 Use Case의 Base Use Case가 많을 경우에는 Extend된 Use Case를 먼저 개발한다. 즉, 위의 예제에서 Extend된 Use Case D의 Base Use Case가 C와 F뿐만 아니라 더 여러 개가 있다면 D를 우선 개발한다.
- Generalization관계 시, Parent Use Case가 사실상의 코드로 매핑되지 않기 때문에 따로 작업 계획에 포함하지 않는다. 같은 Parent Use Case를 갖는 Child Use Case는 비슷한 기능을 구현하는 작업이므로 비슷한 시기에 작업을 하도록 한다. 즉, 위의 예제에서 J, L, K를 비슷한 시기에 가능하면 같은 개발자가 개발한다.
- 같은 상위 Use Case를 둔 하위 Use Case들이 여러 개일 경우에는 하위 Use Case를 적절히 배분하여 개발하는 것을 고려한다. 즉, 자원에 대한 제약사항을 고려하여 동일 인력에 의한 동시 개발이 가능한지를 결정한다.

앞서 설명한 Use Case간의 관계들만을 이용하여 전체 Use Case간의 모든 상호의존성을 분석한 것은 아니다. 즉, 앞의 요소들만으로는 작업들간의 순서를 결정지을 수 없다. 따라서, 전체 시스템 기능의 전반적인 순서를 고려해야 한다. Use Case를 이용한 Activity 다이어그램을 작성하여 이를 파악하도록 한다[8][10]. 요구사항 분석 시, Use Case 다이어그램에 추가적으로 Activity 다이어그램을 작성하는 것은 이해를 돕기 위한 유용한 작업이다. 이와 같이 Activity 다이어그램을 이용하여 전체적인 작업순서를 결정하는 방법이 있고, Use Case를 서브 시스템으로 나누어 개발 순서를 결정하는 방법이 있다. 서브 시스템은 서로 상호작용이 많고 비슷한 기능 군이므로 작업시기가 인접해 있는 것이 좋다. 또한 서브 시스템간의 우선순위를 고려하여 스케줄링에 반영한다. 병렬적으로 개발 가능한 Use Case 그룹 간에는 특별히 우선순위를 고려하지 않아도 된다. 위에서 제시된 지침을 기반으

로 다음과 같은 그림 4. Use Case를 이용한 Activity 다이어그램을 작성한다. 아래의 그림은 전체적인 Use Case의 흐름을 파악하기 위한 Activity 다이어그램이다. 이 Activity 다이어그램 작성시 단계 5를 위한 작업으로 주 흐름을 명시한다.

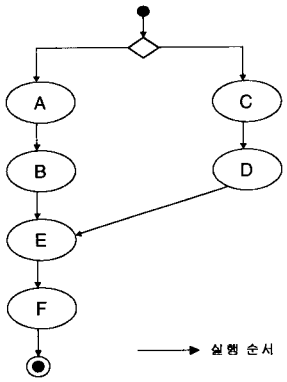

그림 4 Use Case를 이용한 Activity 다이어그램

위의 그림 4. Use Case를 이용한 Activity 다이어그램과 Include, Extend, Generalization의 상호관계를 기반으로 아래의 그림 5. Preliminary PERT 차트를 작성한다. 그림 4의 실행 순서가 그대로 그림 5로 매핑되는 것은 아니다. 매니저는 전체 Activity 다이어그램과 각 Use Case 간의 상호의존성을 기반으로 매니저의 Heuristic을 적용하여 Preliminary PERT 차트를 작성한다. 그림 5는 UC A를 개발한 후, UC B를 개발하고 UC E를 개발함을 나타낸다. 그리고 이와 병렬적으로 UC C를 개발한 후, UC D, UC F를 개발한 뒤에 UC G를 개발한다. 여기서 UC E와 UC F를 개발하고 난 후, UC G개발을 시작한다. 이 Preliminary PERT 차트는 다음 단계 3의 Use Case의 Weight 수치를 이용하여 명확한 검증을 할 수 있다.

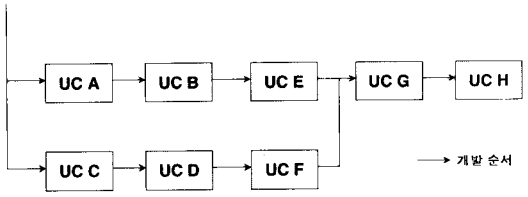

그림 5 Preliminary PERT 차트

4.3 단계 3: 각 Use Case의 특성 규명

단계 3에서는 각 Use Case의 특성을 분석한다. 각 Use Case의 특성에 따라 계획 시에 수행 시기, 수행 기간이 변경된다. 앞의 단계 1의 결과물인 Use Case 리스트를 입력받아 여러 특성 분류 기준을 통해 분석한다. 각 Use Case는 복잡도(Complexity), 완성도(Requirement Completeness), 우선순위(Managerial, Strategical Priority), 위험도(Risk) 등의 기준을 통해 특성을 분류한다[11].

먼저 복잡도는 이 Use Case가 얼마나 복잡한지를 나타낸다. 복잡도의 정도를 High, Medium, Low로 구분하여 Use Case 특성 표에 기입한다. 이때, 각 프로젝트 내에서의 Use Case들의 복잡한 정도를 비교하여 매니저가 Use Case들의 상대적인 복잡도를 분류한다. 복잡도가 낮을수록 위험 요소가 낮기 때문에 개발 시 먼저 개발한다. 둘째로, 각 Use Case가 나타내는 기능이 얼마나 완벽하게 정의되어 있는지 평가한다. Use Case의 기능의 완성도 또한 개발 시기를 결정짓는 요인이 되며 완성도에 대해서도 각 프로젝트의 관리자가 각 프로젝트 내에서의 Use Case 간의 상대적인 완성도를 구분하여 Well-defined, Ambiguous, Ill-defined로 구분하여 Use Case 특성 표에 기입한다. 또한, 관리상, 전략상의 우선순위를 고려한다. 이에 따라 전략상 먼저 개발하면 적당한 Use Case를 구분할 수 있게 된다. 관리자적, 전략적 우선순위 또한 각 프로젝트 내에서 상대적인 평가에 의해 매니저가 다음과 같은 표 3에 기입한다. 위험도는 개발 범위의 변화, 산업계의 변화, 고객과의 대화 사이에서 제대로 반영되지 못한 내용, 개발 환경, 개발자의 경험에 따른 위험을 프로젝트 매니저가 판단하여 다음 표에 그 정도를 기입한다.

표 3 Use Case 특성표

UC ID	특 성			
	복잡도 (Complexity)	완성도 (Completeness)	우선순위 (Priority)	위험도 (Risk)
UC1	H	H	H	H
UC2	M	L	K	M
UC3	L	M	M	L

H : 높음 M : 중간 L : 낮음

표 3의 Use Case 특성 표를 작성한 뒤, 각 Use Case의 특성을 더욱 비교하기 수월하게 메트릭을 적용하여 각 Use Case의 특성 값을 얻을 수 있다. 앞의 단계 2에서 Use Case 특성 표를 기반으로 하여 각 점수마다 Weight을 부여한 후 Total Weight을 계산한다. 먼저 복잡도만을 고려해보면 복잡도가 높을수록 개발 기간을 뒤로 정하는 것이 유리하다. 따라서 복잡도를 Y로 놓고 복잡도 수준이 $Y_h > Y_m > Y_l$ 라고 가정을 하면 각각에 대한 수치적인 값은 $Y_h < Y_s < Y_l$ 의 크기로 부여한다. 즉, 복잡도가 클

수록 작은 값을 부여한다.

또한, 완성도가 높은 Use Case일수록 먼저 개발하기 수월하기 때문에 완성도가 높을수록 큰 값을 부여한다. 즉, 완성도를 S로 놓고 완성도 수준이 $S_h > S_m > S_l$ 라고 가정을 하면 수치적인 값 또한 $S_h > S_m > S_l$ 의 크기로 부여한다. 각 Use Case의 관리자적 측면에서 우선순위가 높은 것은 관리상, 전략상의 수월함을 위해 먼저 개발하는 것이 좋다. 따라서 우선순위 P에 대해 $P_h > P_m > P_l$ 라고 가정을 하면 수치적인 값은 $P_h > P_m > P_l$ 의 크기로 부여하면 된다. 위험도의 관점에서 보면, 위험도가 낮을수록 먼저 개발하는 것이 안전하다. 위험도 R가 $R_h > R_m > R_l$ 와 같다고 가정하면, 수치적인 값은 $R_h < R_m < R_l$ 의 크기로 부여한다.

아래의 표 4와 같이 값을 부여한 후, 각 특성의 중요성에 따라 가중치를 부여하여 계산한다. 즉, a + b + c + d = 100 이라고 가정하고 a > b > c > d 인 값 a, b, c, d를 결정한다. 여기서 a, b, c, d 는 가중치를 나타낸다. 아래의 표 4에 제시된 각 특성의 값들에 대해 완성도, 우선순위, 복잡도, 위험도의 순으로 중요성을 띄고 있다면, 각각의 값에 a, b, c, d를 곱한다. 따라서 UC1의 Total Weight, TW_{uc1} 은 Y_h 5 c + S_h 5 a + P_h 5 b + R_h 5 d이 된다.

표 4 Use Case 특성 가중치 표

UC ID	특 성				
	Complex-ity(Y)	(Com-pleteness(S)	Priori-ty(P)	Risk(R)	Total Weight
UC1	Y_h	S_h	P_h	R_h	TW_{uc1}
UC2	Y_m	S_l	P_l	R_m	TW_{uc2}
UC3	Y_l	S_m	P_m	R_l	TW_{uc3}

위의 Total Weight의 수치는 각 Use Case에 대한 우선순위를 반영한다. 따라서, 앞의 단계 2에서 도출된 Preliminary PERT 차트에 표 4의 Total Weight을 적용하여 Preliminary PERT 차트가 올바르게 유도되었는지 검증하여 정제된 PERT 차트를 작성한다.

그러나, 단계 2에서 Use Case간의 상호의존성을 파악하기 위한 Use Case 특성 가중치 표를 작성할 때, 위에서 부여한 특성 값과 다른 값을 부여한다. 다른 값을 부여하는 부분은 복잡도와 위험도에 대한 수치이다. 복잡도를 Y로 놓고 복잡도 수준이 $Y_h > Y_m > Y_l$ 라고 가정을 하면 각각에 대한 수치적인 값은 $Y_h > Y_m > Y_l$의 크기로 부여한다. 즉, 앞에서와는 반대로 복잡도가 높을수록 큰 값을 부여한다. 위험도 R에 대한 정도가 $R_h > R_m > R_l$와 같다고 가정하면, 수치적인 값은 $R_h > R_m > R_l$의 크기로 부여한다. 단계 2에서 사용되는 가중치 표는 이렇게 값을 부여하여 계산한 가중치 표이다. 이렇게 계산된 Total Weight은 앞서 반영한 의미와는 다르게 각 Use Case에 대한 개발 시기가 길다는 것을 의미한다.

4.4 단계 4 : Iteration개수 결정

단계 3의 결과인 정제된 PERT 차트와 과제의 기간, 프로세스 모델을 기반으로 Iteration의 개수를 결정한다. 여기서 결정된 개수에 따라 각 Iteration을 수행하는 동안에 할당된 Use Case에 대한 개발이 이루어지게 되는데 이러한 개발 과정에는 할당된 기능에 대한 설계와 구현(implementation)과 시험 과정이 포함된다[12]. 즉, 각 Iteration에서는 이미 기능적 분석이 완료된 Use Case 모델을 받아서 실행 가능한 프로그램으로 변형할 수 있는 형태로 설계한다. 기능적인 분석에 기반하여 UML의 클래스 다이어그램을 이용한 정적인 분석-설계 그리고 UML의 시퀀스 다이어그램이나 협력 다이어그램을 통한 동적인 분석-설계를 수행한다. 설계를 마친 후에는 설계 결과인 UML의 여러 다이어그램들을 기반으로 자바와 같은 객체지향 언어를 이용하여 프로그램을 구현하는 과정을 거친다. 그리고, 구현된 각 프로그램이 잘 작동되는 지의 여부를 확인하고 오류를 발견하기 위한 시험 과정이 Iteration에 포함된다. 시험은 각 Use Case 별 단위 시험 및 시스템 통합 후의 통합 시험을 수행하게 된다.

본 단계에서 Iteration 개수는 각 프로젝트 상황마다 매니저가 융통성 있게 결정한다. 예를 들어 개발 기간 1년의 과제인 경우, 최소한 2번이나 3번의 Iteration을 통해 개발한다. 반복을 통한 개발은 위험도를 조기에 감지할 수 있어 효과적이다[13]. 즉, 기간 1년의 과제를 반복 없이 한번에 개발할 경우 과제 기간이 끝날 때가 되어서야 발견할 수 있는 위험도를, 여러 번의 반복 개발을 수행할 때에는 첫 번째 Iteration에서 위험도를 감지하여 조기에 대처하여 피해를 줄일 수 있다. 앞의 표 4에서 분석된 결과 중에 복잡도가 높거나 위험도가 높은 Use Case는 여러 Iteration에 걸쳐 개발을 수행한다. 또한, 완성도가 낮은 Use Case는 처음부터 개발하는 것이 아니라, 충분한 이해를 얻은 후에 뒤의 Iteration중에 수행하는 것이 적합하다. 따라서, 복잡도가 높은 Use Case가 많고, 완성도가 낮은 Use Case가 많을수록 Iteration개수를 늘려서 개발을 진행해 나가는 것이 효과적이다.

또한, 프로젝트 스케줄링의 목적 중에 하나는 전체 개발기간을 줄여야 한다는 것이다. 불필요한 반복으로 전체 개발 기간을 불필요하게 늘려서는 안 된다. 각

Iteration당 유용한 리소스를 완전히 이용할 수 있는 Iteration개수로 결정해야 한다. 즉, 전체개발 기간을 줄이기 위해서 Iteration개수를 무한정 늘리거나, 더 이상 개발의 능률을 올리지 못할 정도로 줄여서는 안 된다. 또한, 한 Iteration당 유용한 리소스를 완전히 이용해야 전체 개발 기간을 줄일 수 있다. Iteration개수는 프로젝트 진행 상황에 따라 융통성 있게 조절할 수 있다. 그러나 각 Iteration의 기한은 지키면서 개발을 한다. 그 기한은 초과하는 일이 없도록 하며, 만약 한 Iteration내에 개발할 분량을 전부 개발할 경우에는 반복 횟수를 융통성 있게 조정 가능하다.

다음은 Iteration개수를 결정하는 지침을 제시한 것이다.

- 사용자(클라이언트)가 전달 받고자 하는 중간 prototype 개수를 우선 고려한다.
- 각 Use Case를 개발하는데 필요한 기간을 고려하여 Iteration 개수와 기간을 조정한다.
- 한 Iteration기간을 초과하는 복잡도가 높은 Use Case는 복수개의 Iteration에 걸쳐서 개발할 수 있다.
- 완성도가 낮은 Use Case가 많을수록 Iteration개수를 늘려서 개발을 진행한다.
- 전체적인 개발 기간을 최대한 줄일 수 있는 Iteration개수로 개발을 진행한다.
- 과제 기간을 고려하여 Iteration개수를 결정한다.
- 프로세스 모델을 고려하여 Iteration개수를 결정한다.
- 서브 시스템 개수에 따라 Iteration개수를 결정한다.
- 기본적으로 매니저의 Heuristic을 바탕으로 Iteration개수를 결정한다.

4.5 단계 5: Iteration에 Use Case 할당

핵심적인 기능이라 함은 본 시스템이 실행되기 위한 최소한의 핵심 Use Case를 의미한다. Iteration에 Use Case를 할당하는 방법은 2가지가 있다. 첫 번째 방법은 단계 2의 결과인 Preliminary PERT 차트를 각 팀의 프로젝트 매니저의 주관에 따라 적당히 Iteration 개수로 나누어 할당하는 것이다. 즉, 전체 개발 순서대로 그대로 Iteration을 나누어 개발한다. 첫 번째 Iteration에서는 전체 시스템 요구사항의 한 서브 시스템 중 많은 부분 구현을 하고 다음 Iteration에서는 다른 서브 시스템을 구현하는 형식으로 진행된다.

두 번째는 각 Iteration이 진행될수록 핵심 기능에서 부수적인 기능을 추가하는 형식으로 할당할 수 있다. 두 번째 방법으로 할당할 때에, Iteration 1에서는 먼저 핵심적인 기능을 포함하고 Iteration 2에서는 Iteration 1에 더욱 세부적인 기능을 추가한다. 즉, Iteration 1에 할당된 Use Case만을 개발하여도 고객에게 일차 검증을 받을 시스템을 얻게 된다. 이는 앞의 단계 2에서 중간 산출물로 나온 Activity 다이어그램상에서 메인 흐름으로 표현되는 Use Case들과 단계 4의 Use Case특성 중에 우선순위와 완성도에 가중치를 더 부여하여 계산한 값을 기반으로 한다. 즉, 아래의 표 5를 보면, 완성도의 가중치를 a, 우선순위의 가중치를 b라 하고, a + b = 100라 하면, S(a + S(b를 계산한 값인 TW중에 숫자가 큰 순서대로 Iteration에 우선 할당한다. 각 Iteration에 포함될 Use Case는 앞서 언급한 정보와 프로젝트 매니저의 Heuristic이 적용되어 결정된다.

표 5 핵심적 기능 추출표

UC ID	특 성		
	Completeness(S)	Priority(P)	Total Weight
UC1	S_h	P_h	TWuc1
UC2	S_i	P_i	TWuc2
UC3	S_m	P_m	TWuc3

단계 5에서 정해진 Iteration 개수에 따라 아래의 표 6와 같이 Use Case를 각 Iteration에 할당한 할당표를 작성한다. 본 논문에서는 위험도를 줄이기 위한 방안으로 여러 반복을 통한 점증적인 개발 프로세스를 적용한다. 각 Iteration에 Use Case를 할당 때에는 먼저 고객에게 전달되어야 할 핵심적인 기능을 먼저 포함한다. 각 Iteration 은 그 자체 Iteration만으로 완벽히 실행되어야 한다. 단계 3의 Use Case 특성 표에서 복잡도가 높고 완성도가 낮은 Use Case는 여러 Iteration에 걸쳐 수행한다. 복잡도가 높은 Use Case는 구현이 복잡하기 때문에 한번의 Iteration으로 완벽히 구현하는 것은 어려움이 있다. 여러 번의 반복을 통해 복잡한 개발을 계속하여 정제해 나간다. 또한, 완성도가 낮은 Use Case는 기간을 두어 고객과 매니저와의 충분한 의논을 거쳐 요구사항에 대한 충분한 이해와 완성도를 높여야 하므로 한번의 Iteration이 아닌 여러 Iteration에 걸쳐 개발한다. 모든 상황에 적용되지는 않지만 일반적으로 Use Case 다이어그램에서 기본적인 CRUD (Create, Read,

표 6 Use Case 할당표

	Iteration 1	Iteration 2	Iteration 3
할당된 Use Case	UC1 UC2 UC5	UC3 UC6 UC7	UC4 UC8 UC9 UC10

Update, Delete)의 기능 중, 첫 번째 Iteration에서는 생성관련 Use Case를 개발하는 경우가 많다. 이러한 여러 사항을 고려하여 다음 표 6을 작성한다.

각 Iteration에 Use Case를 할당한 이후에 적합하게 할당되었는가를 검증해야 한다. Use Case를 적합하게 할당하는 것은 중요한 작업이다. 이로 인해 개발 기간이 불필요하게 늘어나 손실을 가져올 수 있기 때문이다. 모든 자원이 공정하고 유용하게 사용되었는지를 검증하기 위한 방안으로 각 Use Case의 개발 비용을 측정해본다. UCi의 추정된 개발 비용을 Cost(UC_i)라 하자. Cost(UC_i)는 여러 방법에 의해 계산될 수 있을 것이다. 각 Iteration에 할당된 Use Case UC_1~UC_i 의 평균 개발 비용은 $\frac{\sum_{i=1}^{n} \infty Cost(UC_i)}{n}$ 이 된다. 각 Iteration의 평균 개발 비용을 구한 뒤에 이를 비교해본다. Iteration간에 평균 개발 비용에 많은 차이가 있다면 이는 Use Case할당이 잘못된 것이다. 특별히 개발 비용이 높은 Iteration이 있다면 이 안에 개발해야 할 Use Case를 상대적으로 개발 비용이 낮은 Iteration에 할당한다.

4.6 단계 6: 유용한 자원과 제약 사항 고려

단계 6에서는 유용한 자원과 여러 제약 사항을 고려한다. 자원에는 사람, 재사용 가능한 소프트웨어 컴포넌트, 하드웨어, 소프트웨어 툴 등이 있다. 매니저는 전체 개발 범위를 평가하여 개발을 완료하기 위해 요구되는 기술을 파악한다[12]. 요구되는 조직 내에서의 직위(예를 들면, 매니저나 선임 소프트웨어 공학자)와 요구되는 전문성(예를 들면, 통신, 데이타 베이스, 클라이언트/서버 기술)을 파악한다. 상대적으로 작은 프로젝트에서는 한 사람이 전 소프트웨어 공학의 단계를 담당할 수도 있다. 프로젝트에서 요구되는 인력자원을 파악한 후, 현재 있는 개발 인력이 지니고 있는 능력이나 전문성을 평가하여 인력 평가표를 작성해도 된다. 각 작업 별로 인력을 배당할 때에 남는 인원이 있어서는 안 된다.

재사용 가능한 소프트웨어 컴포넌트에는 Off-the-shelf 컴포넌트, 부분적으로 구현된 컴포넌트, 이미 구현이 된 엔티티(Entity) 클래스 등이 있다. Off-the-shelf 컴포넌트는 구입 가능한 컴포넌트나 과거에 이미 개발된, 현재의 요구사항을 만족하는 컴포넌트이다. 부분적으로 구현된 컴포넌트는 과거의 프로젝트를 통해 요구사항의 일부분을 만족하는 컴포넌트이다. 또한, 엔티티 클래스를 재사용할 수 있다. 기존에 존재하는 엔티티 클래스를 사용하여 새로운 기능을 구현하는 컨트롤러(Controller) 클래스를 구현할 경우를 말한다. 마지막으로 하드웨어, 소프트웨어 툴 등의 자원을 고려해야 한다. 좋은 소프트웨어를 개발하기 위한 소프트웨어 툴을 지원하는 하드웨어 자원 등에 대한 정보를 파악하고 현재 있는 것과 앞으로 필요한 자원을 고려한다. 이처럼 단계 6에서는 현재 보유하고 있는 인력 자원과 재사용 가능한 컴포넌트 자원과 엔티티 클래스 자원에 대해 파악하여 표를 작성한다.

4.7 단계 7: Revised PERT 차트 작성

단계 7에서는 단계 6까지의 활동을 기반으로 Revised PERT 차트를 작성한다. PERT 차트를 통해 수행해야 할 작업들간의 상호의존성과 흐름을 보여준다. PERT 차트는 작업들간의 순서를 앞의 Use Case간의 순서나 상호의존성을 고려해서 작성한다. 단계 1에서 Use Case 식별을 한 뒤, 단계 2에서 Use Case간의 상호의존성을 분석하여 초기 PERT 차트를 작성한다. 단계 3에서는 각 Use Case의 특성을 분석하여 각 Use Case에 가중치 값을 부여한다. 이 값이 크다는 것은 개발 시에 우선순위가 높다는 것을 나타낸다. 단계 3의 가중치 값을 이용하여 초기 PERT 차트를 검증한다. 단계 4에서는 여러 사항들을 고려하여 Iteration의 개수를 결정한 다음 단계 5에서는 그 앞 단계에서 나온 여러 산출물을 이용하여 각 Iteration에 Use Case를 할당한다. 단계 6에서는 여러 자원에 대한 정보를 수집한 다음 단계 7에서는 단계 6까지의 산출물 중에 Use Case간의 Activity 다이어그램, Preliminary PERT 차트, Use Case할당표, 그리고 유용한 자원 정보를 입력물로 받는다.

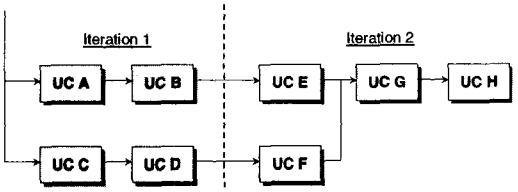

그림 6 Revised PERT 차트

그림 6은 본 논문에서 유도하고자 하는 최종 산출물인 Revise된 PERT 차트다. 꼭 위의 형식을 따르지는 않고 Iteration과 각 Iteration에 포함된 작업의 순서를 명시한다. 병행적으로 개발되는 작업에 대한 명시도 해준다. 위의 그림 6을 보면, Iteration1에서는 UC A와 UC B의 개발과 UC C와 UC D의 개발이 병행적으로 수행됨을 나타낸다. 또한 Iteration2에서는 Iteration1의 네 개의 Use Case 개발을 마친 후에 개발이 진행됨을 나타내고 있다.

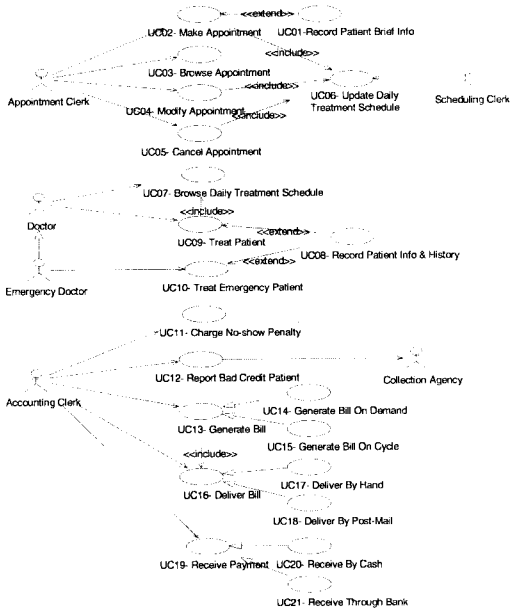

그림 7 병원 Use Case 다이어그램

이는 단계 2의 결과인 Preliminary PERT 차트를 기반으로 단계 5에서 할당된 Use Case 할당표를 이용하여 작성한다. 할당된 Use Case들에 대한 각각의 PERT 차트를 Preliminary PERT 차트를 통해 작성한다. 단계 6에서 조사된 인력 자원, 하드웨어 소프트웨어 자원 등을 고려하여 병렬개발 가능 여부를 분석하여 병렬적으로 개발할 작업을 계획한다.

PERT 차트가 작성과 함께 각 Use Case 별 상세 일정과 계획은 LOC와 FP 기반 추정 방법인 분해 기법이나 COCOMO 모델과 같은 경험적 추정 모형을 이용할 수 있다. 그러나, 이들 추정 방법은 각 기업 마다의 경험적으로 누적된 데이타나 생산성 메트릭이 다르기 때문에 모든 환경에 범용적으로 적용되는 일정 계획 기법에 대한 제시는 어렵다. 각 기업 마다의 다른 경험치가 적용되어 계산된 개발 노력은 태스크별 시작 시간과 종료 시간을 표현하는 Gantt 챠트를 통해 표현할 수 있다[12].

5. 사례 연구

본 논문의 사례 연구에서는 병원 시스템의 요구사항

표 7 병원 Use Case 목록

UC ID	Use Case	설 명
UC01	Record Patient Brief Info	등록되지 않은 환자가 진료예약을 하기 위한 환자의 간략한 정보를 입력 받아 임시 등록한다.
UC02	Make Appointment	진료를 받기 위한 진료 예약을 한다.
UC03	Browse Appointment	현재의 진료 예약 상황을 검색한다.
UC04	Modify Appointment	진료 예약을 변경한다.
UC05	Cancel Appointment	예약을 취소한다.
UC06	Update Daily Treatment Schedule	일일 진료 스케줄을 갱신한다.
UC07	Browse Daily Treatment Schedule	일일 진료 스케줄을 검색한다.
UC08	Record Patient Info & History	임시 등록한 환자가 정식으로 등록하기 위해 진료 시 환자의 상세 정보와 병력 정보를 받아 정식 등록한다.
UC09	Treat Patient	이미 예약되어 있는 환자에 대해서 의사가 진료를 하고 처방을 한 내용에 대한 정보를 저장한다.
UC10	Treat Emergency Patient	응급환자에 대한 진료를 하고 처방을 한 내용에 대한 정보를 저장한다.
UC11	Charge No show Penalty	환자가 예약 취소 없이 예약된 진료 시간에 오지 않았을 경우 벌금을 부과한다.
UC12	Report Bad Credit Patient	만약 환자가 수납 기간 내에 의료비를 납부하지 않을 경우 환자는 신용불량고객으로 신용관리 업체에 통보된다.
UC13	Generate Bill	의료비를 계산하여 청구서를 작성한다.
UC14	Generate Bill On Demand	환자의 청구서 요청이 있을 때에 의료비를 계산한다.
UC15	Generate Bill On Cycle	정기적으로 매월 1일에 의료비를 계산한다.
UC16	Deliver Bill	작성된 청구서를 환자에게 청구한다.
UC17	Deliver By Hand	청구서를 환자에게 직접 청구한다.
UC18	Deliver By Post-Mail	청구서를 환자에게 우편으로 청구한다.
UC19	Receive Payment	의료비를 수납한다.
UC20	Receive By Cash	환자는 병원 수납에서 직접 일시불 납부와 은행을 통한 일시불 및 분할 납부가 있다.
UC21	Receive Through Bank	환자는 청구된 계산서를 은행에서 분할 납부할 수 있다.

을 기반으로 하여 4장에서 제시한 프로젝트 스케줄 추출 단계를 적용하여 객체지향 프로젝트의 스케줄링 차트를 도출하는 과정을 살펴보겠다. 다음은 병원의 기능성을 모델링 한 Use Case 다이어그램이다. 본 도메인은 비교적 규모가 작은 도메인으로서 5개의 Actor와 21개의 Use Case가 도출되었다. 그림 7의 다이어그램을 이용하여 앞으로 7단계를 거쳐 PERT 차트를 완성한다.

5.1 단계 1: Use Case 식별

단계 1 에서는 병원 Use Case 다이어그램을 기반으로 Use Case의 기능을 식별한 뒤, 다음의 표 7. 병원 Use Case 목록을 작성한다. 본 사례연구에서 도출된 Use Case들은 Granularity 검증을 한 후에, Granularity가 다른 Use Case들의 정제를 마친 다이어그램이다. 아래의 표에서 개발 시에 포함되지 않는 Use Case는 Generalization관계의 Use Case에서의 Parent Use Case인 UC13, UC16, UC19가 된다. 이 Use Case에 대한 실제적인 코딩 작업은 이루어지지 않는다. 본 단계에서는 Use Case로 명시된 기능을 파악한다.

5.2 단계 2: 상호의존성 분석을 통한 초기 PERT 차트 작성

앞의 표 7을 이용하여 Use Case간의 상호의존성을 분석한다. 다이어그램상에서 명시적으로 표시되는 관계들을 추출하면 아래와 같은 표 8의 결과가 도출된다. 기본적으로 아래 표에서 의존성으로 맺어진 Use Case들은 개발 시기가 인접하도록 계획한다. 밑줄이 그어진 Use Case는 앞의 4.2에서 제시된 지침에 따라 우선 개발이 기대되는 Use Case들을 표시한 것이다.

표 8 병원 도메인의 의존성 추출표

의존성 타입	추출 사항
Include	UC02, UC03, UC04 → UC06
	UC09 → UC07
	UC13 → UC16
Extend	UC01 → UC02
	UC08 → UC09, UC10
Generalization	UC14, UC15 → UC13
	UC17, UC18 → UC16
	UC20, UC21 → UC19

Use Case다이어그램에서 명시적으로 나타나는 의존성을 추출한 다음 전체적인 Use Case간의 실행 흐름을 Activity 다이어그램으로 나타낸다. 다음의 그림 8은 병원 도메인의 Use Case들 간의 실행흐름을 보여주는 Activity 다이어그램이다.

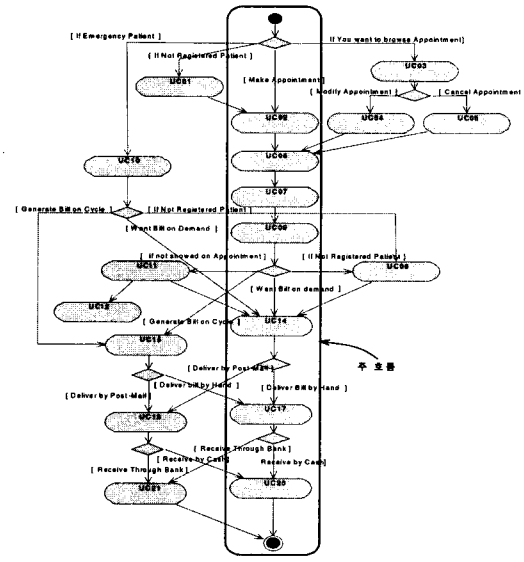

그림 8 병원 도메인의 Use Case 간 Activity 다이어그램

프로젝트 매니저는 앞의 표 8과 그림 8을 바탕으로 4장에서 제시된 사항을 고려하여 병원의 Preliminary PERT 차트를 아래의 그림 9와 같이 작성한다. 아래의 그림에서 UC02, UC04, UC05와 같이 한 열에 계획된 Use Case는 동시 개발을 요구하는 Use Case이다. 즉, 그림 8에서 추출된 관계들이 아래의 차트에서는 동시개발로서 표현된다.

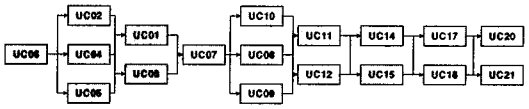

그림 9 Preliminary PERT 차트

5.3 단계 3: 각 Use Case의 특성 규명

단계 3에서는 각 Use Case의 특성을 분석한다. 각, 특성에 대해 정도가 강하면 H, 중간은 M, 약하면 L로서 표기한다. 앞의 4.3에서도 언급했듯이 어떠한 절대적인 기준으로 Use Case의 특성의 정도를 표기한 것이 아니라 매니저의 주관성을 반영하여 각 프로젝트 내에서의 Use Case들 간의 상대적인 분석을 수행한다. 다음 표 9는 병원 도메인의 Use Case를 분석한 표이다.
앞서 분석된 표 9를 기반으로 다음과 같이 가중치를 부여하여 계산된 수치로서 특성을 나타낸다. 본 논문의 사례연구에서는 복잡도의 수치를 Y_h=1, Y_m=3, Y_l=5 로서, 완성

표 9 병원 도메인 Use Case 특성표

UC ID	특 성			
	복잡도 (Complexity)	완성도 (Completeness)	우선순위 (Priority)	위험도 (Risk)
UC1	M	H	M	L
UC2	H	H	H	M
UC3	L	H	M	L
UC4	M	H	M	M
UC5	L	H	L	M
UC6	H	H	H	M
UC7	M	H	H	L
UC8	M	H	M	L
UC9	M	H	H	M
UC10	M	H	H	M
UC11	M	M	M	L
UC12	L	H	M	M
UC13				
UC14	M	H	M	L
UC15	M	H	L	M
UC16				
UC17	L	H	H	L
UC18	L	H	H	L
UC19				
UC20	L	H	M	L
UC21	M	H	M	M

도는 S_h=5, S_m=3, S_l=1로서 우선순위는 P_h=5, P_m=3, P_l=1의 크기로, 위험도는 R_h=1, R_m=3, R_l=5 로 부여한다. 전체의 가중치를 계산하기 위해 각 특성에 부여되는 가중치의 비율은 a=20, b=30, c=35, D=15 이다. 최종적으로 Total Weight, Y 5 a+S 5 b+P 5 c +R 5 d을 계산한다.

표 10 병원 Use Case 특성 가중치표

UC ID	특 성				
	Com-plexity(Y)	Com-pleteness(S)	Priority(P)	Risk(R)	Total Weight
UC1	3	5	3	5	390
UC2	1	5	5	3	390
UC3	5	5	3	5	430
UC4	3	5	3	3	360
UC5	5	5	1	3	330
UC6	1	5	5	3	430
UC7	3	5	5	5	460
UC8	3	5	3	5	390
UC9	3	5	5	3	430
UC10	3	5	5	3	430
UC11	3	3	3	5	330
UC12	5	5	3	3	400
UC14	3	5	3	5	390
UC15	3	5	1	3	290
UC17	5	5	5	5	500
UC18	5	5	5	5	500
UC20	5	5	3	5	430
UC21	3	5	3	3	360

5.4 단계 4: Iteration 개수 결정

사례 연구에서 주어진 전체 프로젝트 완료 기간은 4개월이었다. 그림 8의 결과에서 보듯이 주 흐름과 그 외에 추가해서 개발할 Use Case의 개수는 비슷하다. 표 9의 Use Case를 분석한 결과를 보면, 상대적으로 복잡도가 높은 Use Case는 2개, 완성도가 낮은 Use Case는 없다. 따라서 4.4의 지침에 따라 많은 반복 개발은 수행하지는 않는다. 매니저는 이 모든 정보를 고려하여 2번의 Iteration을 수행하기로 결정한다. 각각 2번의 Iteration 동안에는 다음 단계 5에서 할당될 Use Case 들에 대한 설계, 구현, 그리고 시험 과정이 포함된다. 본 논문의 범위에 포함되는 내용은 아니지만 사례 연구에서의 설계는 도출된 계획 일정에 맞추어 UML을 이용하여 정적 설계, 동적 설계를 하였으며, 구현은 자바 언어를 이용하여 하였으며, 시험은 블랙 박스 시험으로 각 Use Case에 대한 단위 시험과 통합 후의 통합 시험을 수행하였다.

5.5 단계 5: Iteration에 Use Case할당

Iteration에 Use Case를 할당하기 위해 먼저 핵심적인 기능을 찾는다. Use Case 특성표에서 완성도와 우선순위만을 뽑아 가중치를 부여하여 전체 가중치를 계산하면 다음과 같다. 본 논문의 사례연구에서는 a=30, b=70의 값으로 S 5 a+P 5 b 를 계산하여 전체 가중치를 구하였다. 다음 표에서 계산된 Total Weight 수치가 높을수록 핵심적인 기능임을 나타낸다.

본 논문에서는 단계 4의 첫 번째 방법으로 할당한 후에 그림 8의 주 흐름과 표 11에서 전체 가중치가 높은

표 11 핵심적인 기능 추출표

UC ID	특 성		
	Completeness(S)	Priority(P)	Total Weight
UC01	5	3	360
UC02	5	5	500
UC03	5	3	360
UC04	5	3	360
UC05	5	1	220
UC06	5	5	500
UC07	5	5	500
UC08	5	3	360
UC09	5	5	500
UC010	5	5	500
UC011	3	3	300
UC012	5	3	360
UC014	5	3	360
UC015	5	1	220
UC017	5	5	500
UC018	5	5	500
UC020	5	3	360
UC021	5	3	360

것을 반영하여 할당하였다. 즉, Iteration1에 단계 2에서 주 흐름으로 도출된 Use Case들을 할당하며 위의 표 11에서 Total Weight가 높은 것과 비교해보면 거의 비슷하게 도출된 것을 볼 수 있다. 또한 각 Iteration을 둘로 나눈 것은 병렬적으로 수행하기 위함이다. 이와 같이 할당된 Use Case가 올바르게 할당되었는가를 검사하기 위해 각 Iteration에 할당된 Use Case들의 평균 개발 비용을 계산한다. 즉, Iteration1의 평균 개발 비용과 Iteration2의 평균 개발 비용이 비슷한지 비교하여 차이가 클 경우에 할당된 Use Case를 조정하여 재할당한다.

표 12 Use Case 할당표

	Iteration 2	
할당된 Use Case	UC02 UC06 UC07	UC10 UC09 UC14 UC17 UC18 UC20
	UC04 UC05 UC01 UC03	UC08 UC11 UC12 UC15 UC21

5.6 단계 6: 유용한 자원과 제약 사항 고려

본 프로젝트는 6명이 팀을 이루어 진행하는 비교적 규모가 작은 프로젝트이다. 6명 모두 본 시스템을 구현하기 위해 요구되는 필수적인 전문성은 보유하고 있으며, 각 Iteration의 각 병렬 개발에 3명씩 배당하여 남는 인원은 없도록 한다. 하드웨어자원은 각 개인 당 각자의 개발을 수행할 수 있는 자원은 충분히 있다. 본 프로젝트에서 개발 시에 재사용할 수 있는 Off-the-shelf 컴포넌트나 부분적으로 구현된 컴포넌트, 또한 재사용 가능한 엔티티 클래스는 없어 모든 시스템에 대한 개발을 처음부터 모델링하고 구현해야 한다.

5.7 단계 7: Revised PERT 차트 작성

단계 2의 Use Case간 Activity 다이어그램과 Preliminary PERT 차트, 단계 3의 Use Case 할당표, 단계 6에서 조사된 자원과 제약 사항을 고려하여 다음의 그림 6. Revised PERT 차트를 작성한다. 즉, Use Case 할당표에서 할당된 Use Case들간의 개발 순서를 Use Case간 Activity 다이어그램과 Preliminary PERT 차트를 이용해서 다음과 같이 계획한다.

병원 도메인에 대한 사례 연구를 통해 Use Case 다이어그램으로부터 완성된 PERT 차트를 도출해내는 과

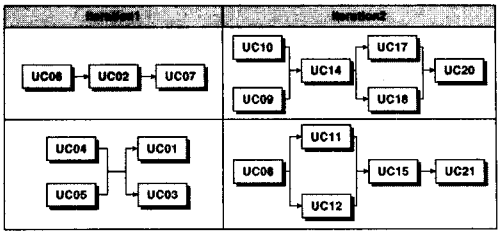

그림 10 Revised PERT 차트

정을 살펴보았다. Use-case라는 단위는 의미 있는 기능의 단위이기 때문에 개발 계획을 Use Case 기반으로 진행하는 것은 합리적이다.

6. 맺음말

객체지향 기술은 소프트웨어를 효율적으로 개발하고 소프트웨어의 품질을 높여 운용의 노력을 최소화하는 소프트웨어 개발 기술이다. 즉, 주어진 문제 영역을 분석하고 시스템을 설계하는 과정이 우리가 일상 생활에서 평소에 사고하는 방식과 유사하고 객체지향의 여러 장치들로 인해 소프트웨어 개발 생산성을 크게 향상시킨다. 그러나 이러한 객체지향 기반의 개발에 적합한 프로젝트 스케줄링 기법이 미약하다.

본 논문에서는 크게 7단계를 거쳐 Use Case 다이어그램으로부터 PERT 차트를 도출해가는 과정을 설명하였다. 먼저 Use Case 다이어그램으로부터 Use Case 목록을 나열하여 Use Case에 대한 식별을 한 뒤, Use Case간의 상호의존성을 고려하여 Preliminary PERT 차트를 작성한다. Use Case의 특성에 대한 분석을 수행한 후, 가중치를 부여하여 수치 값을 얻는다. 그런 후, Iteration 회수를 정하고 각 Iteration에 할당될 작업을 정하여 여러 자원과 여러 제약사항을 고려하여 PERT 차트를 완성한다. 각 Iteration에서는 Use Case에 대한 설계, 구현, 시험 과정을 포함한다. 단계 2, 3, 4의 순서는 각 프로젝트의 매니저가 유연성 있게 결정할 수 있다. Use Case 특성 표를 이용한 가중치 계산은 여러 용도로 사용 가능하다. 특별히, 검사하고자 하는 특성만을 따로 도출하여 계산한다. 즉, 완성도와 우선순위만을 이용해서 단계 5의 핵심적인 기능을 추출할 수 있고, 복잡도와 위험도의 값을 계산하여 각 작업의 기간을 예상할 수 있다.

객체지향 프로젝트를 계획할 때, 특히 대형 프로젝트 진행을 계획하기 위한 기법의 부족으로 인해 그 동안 프로젝트 진행에 어려움이 많았다. 그러나, 본 논문에서 제시한 기법을 사용하면 보다 짧은 기간에 효율적인 개발 계획 차트를 작성할 수 있으며 7단계를 이용하여 체

계적이고 논리적인 흐름으로 PERT 차트를 작성할 수 있다. 사용자가 요구하는 기능을 나타내는 단위인 Use Case를 기반으로 개발하게 되므로 더욱 사용자 지향적인 개발이 가능하며 사용자에게 중간 점검을 받기가 편리해 위험 요소를 줄이는 데에 기여할 수 있다. Use Case 다이어그램 상에 나타난 Use Case간의 상호의존성 분석을 통하여 논리적으로 타당한 Use Case간의 개발 순서를 계획할 수 있다. 또한, Use Case의 특성을 분석하여 가중치를 부여한 값으로 비교적 정확한 우선순위를 알 수 있어 본 논문에서 제안한 기법을 통해 개발 계획을 작성하는 것은 합리적이라고 할 수 있다. 그리고, Iteration 개수를 정하는 지침과 Use Case할당 지침을 이용하여 짧은 기간 동안 자원을 가장 효과적으로 사용한 계획서를 작성할 수 있다.

참 고 문 헌

[1] Bruegge, B., Dutoit, A., *Object-Oriented Software Engineering : Conquering Complex and Changing Systems*, Prentice Hall, 1999.

[2] Pooley, R., Stevens, P., Using *UML : Software Engineering with Objects and Components*, Addison-Wesley, 2000.

[3] Cantor, M., *Object-Oriented Project Management with* UML, WILEY, 1998.

[4] Fayad, M., Tsai, W., Fulghum, M., "Transition to object oriented software development", CACM, Vol. 39, No.2, Feb. 1996.

[5] Cockburn, A., Writing Effective Use Cases, Addison Wesley, 2000.

[6] Wilkie, G., *Object-Oriented Software Engineering : The Professional Developer's Guide*, Addison-Wesley, 1993.

[7] Sommerville, I., *Software Engineering: Sixth Edition*, Addison Wesley, 2000.

[8] Schneider, G., Winters, J., *Applying Use Cases : A Practical Guide, Addison Wesley*, 2001.

[9] Booch, Rumbaugh, Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley, 1999.

[10] Armour, F., Miller *G., Advanced Use Case Modeling*, Addison Wesley, 2000.

[11] Jacobson, I., *Object-Oriented Software Engineering : A Use Case Driven Approach*, Addison-Wesley, 1992.

[12] Pressman, R., *Software Engineering: A Practitioner's Approach*, McGRAW-HILL, 1997.

[13] Cockburn, A., *Surviving Object Oriented Projects*, Addison Wesley, 1998.

허 진 선

2001년 숭실대학교 컴퓨터학부 공학. 2003년 숭실대학교 컴퓨터학과 공학석사 2003년~현재 숭실대학교 컴퓨터학과 박사과정. 관심 분야는 객체지향 개발 방법론, 컴포넌트 개발 방법론, 엔터프라이즈 아키텍쳐

최 시 원

2000년 삼육대학교 컴퓨터과학과 이학사 2002년 숭실대학교 컴퓨터학과 공학석사 2002년~현재 숭실대학교 컴퓨터학과 박사과정 관심 분야는 객체지향 개발 방법론, 컴포넌트 개발 방법론, 소프트웨어 아키텍처, 소프트웨어 품질

김 수 동

1984년 Northeast Missouri State University 전산학 학사. 1988년 The University of Iowa 전산학 석사. 1991년 The University of Iowa 전산학 박사. 1991년~1993년 한국통신 연구개발단 선임연구원. 1993년~1994년 The University of Iowa 교환교수. 1994년~1995년 현대전자 소프트웨어연구소 책임연구원. 1995년 9월~현재 숭실대학교 컴퓨터학부 부교수. 관심분야는 컴포넌트 개발 방법론, 객체지향 개발 방법론, 분산 시스템, 컴포넌트 정형명세, 소프트웨어 시험