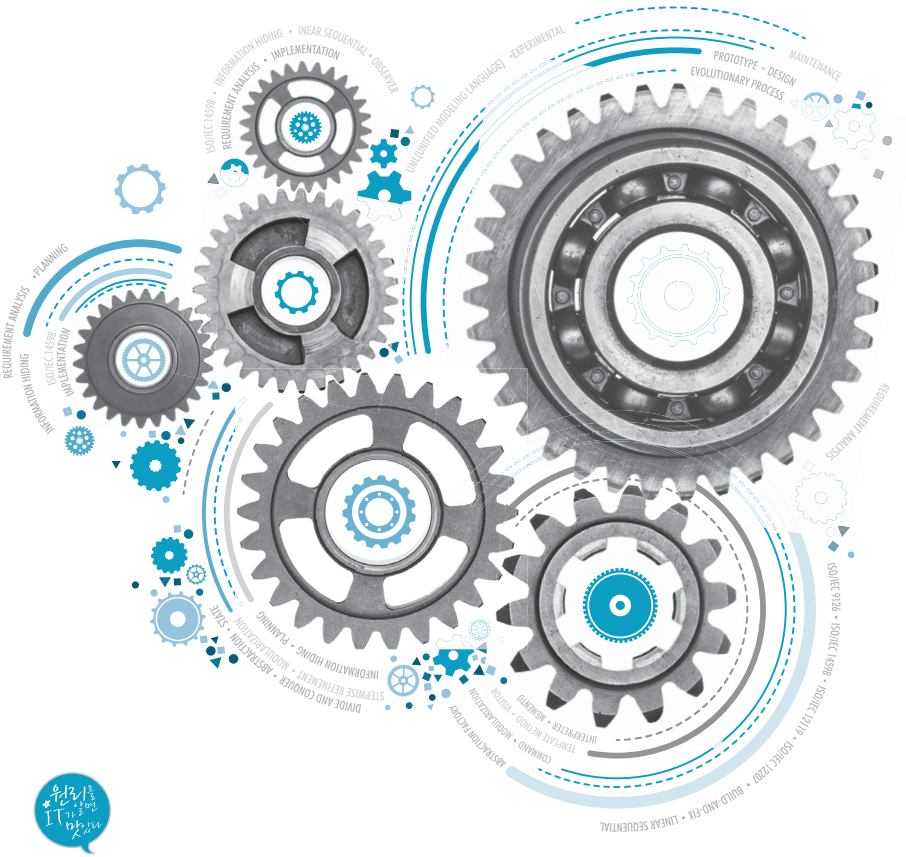

시스템 분석과 설계 개정판

효과적인 비즈니스 정보시스템 개발

허원실 지음

한빛아카데미
Hanbit Academy, Inc.

Chapter 01

시스템 개발 과정의 이해

01 소프트웨어 공학

02 시스템과 시스템 개발자

03 SDLC 모형

04 프로토타입 모형

05 프로젝트 관리

요약

연습문제

학습목표

- 소프트웨어 위기와 소프트웨어 공학의 출현 배경을 이해한다.
- 시스템(소프트웨어) 개발 과정에 참여하는 개발자에 대해 알아본다.
- 시스템 개발 단계를 이해하기 위해 SDLC 모형 및 프로토타입 모형을 학습한다.
- 프로젝트 관리의 개념을 이해하고 절차를 학습한다.

01 소프트웨어 공학

1 소프트웨어 위기

최근 우리는 급속한 과학기술의 발전을 목도하고 있다. 더불어 인류의 미래를 주도할 첨단 산업기술인 6T에 주목하고 있다. 정보기술IT, 생명공학기술BT, 나노기술NT, 환경공학기술ET, 우주항공기술ST, 문화콘텐츠기술CT이 바로 그 주인공이다. 이들은 전통적인 산업기술과 다른 것으로 과거의 어떤 기술보다 인류 생활에 큰 변화를 몰고 올 것이다.

컴퓨터로 대표되는 정보기술IT, Information Technology 혁명은 인터넷의 확산과 더불어 인류문명을 새롭게 바꾸어 나가고 있다. 그 중심에 소프트웨어Software가 있다. 그렇다면 소프트웨어의 발전을 이룩한 주인공은 천재적인 프로그래머일까? 아니면 소프트웨어 개발 기술일까?

초기 프로그래머들은 뛰어난 수학자이거나 과학자였다. 아니 자신의 창의적인 아이디어를 이용해 뛰어난 프로그램을 창조해 낸 예술가였는지 모른다. 하지만 소수의 천재들에 의해서만 소프트웨어가 개발된 것은 아니다. 만약 그랬다면 지금과 같이 급격하게 증가하고 있는 소프트웨어의 수요를 충족시킬 수는 없었을 것이다.

실제로 지난 20~30년 동안 우리 사회는 그러한 사실을 경험하였다. 프로그래머의 수요가 늘어났고 대학의 컴퓨터 관련 학과는 인기 상한가를 누렸다. 당연히 컴퓨터 관련 전공자들의 몸값은 뛰었고 수요와 공급의 불균형으로 인한 인력난을 겪었다. 하지만 개발현장에서는 수많은 오해와 시행착오를 경험하는 시기이기도 했다. 컴퓨터는 만능이라는 막연한 기대와, 프로그래머는 원하는 프로그램을 다 만들어낼 수 있을 것이라는 오해 등이 바로 그것이다. 수많은 개발 프로젝트는 실패로 끝나거나 중도에 포기되기도 했다.

개발기간의 지연을 해결해 보려고 추가 인력을 투입했지만 결과는 나아지지 않았다. 오히려 개발비용만 기하급수적으로 증가했다. 이러한 현상을 빗대어 '90 : 90 법칙'이란 용어가 회자되기도 했다. 즉, 개발 공정의 90%가 진행되었다고 말하는 시점부터 실제로 남은 공정에 90%의 시간과 노력이 필요할 것이라는 다소 독설적인 격언이다. 비록 프로젝트가 마무리되어 시스템이 운영된다 해도 끊임없이 제기되는 문제점의 수정과 보완을 위해 추가적인 인력과 비

용을 지속적으로 요구하게 되었다.

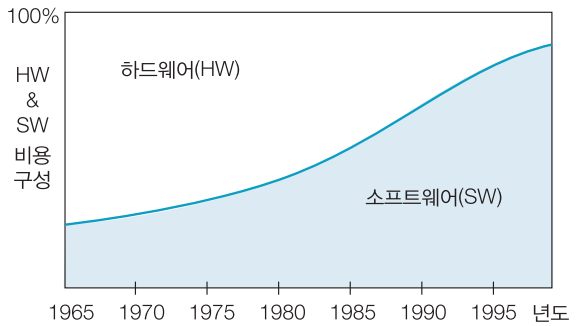

그림 1-1 하드웨어와 소프트웨어 비용 구성률

2 소프트웨어 공학의 출현

이런 값비싼 대가를 치루며 많은 개발자들은 소프트웨어 개발 방법론에 주목하게 되었다. 산업혁명으로 시작된 대량 생산체제가 급격한 수요의 증가에 대응할 수 있는 공급체계를 갖출 수 있게 했다는 사실에 주목하며 소프트웨어의 개발에도 이와 같은 원리를 적용할 필요를 느끼게 된 것이다. 이렇게 해서 탄생한 것이 이른바 소프트웨어 공학Software Engineering이다. 이론적으로는 소프트웨어의 개발도 마치 공장에서 대량으로 필요한 제품을 생산하듯이 자동화된 시스템에 의해 규격화된 제품(소프트웨어)을 생산해 낼 수 있도록 하자는 것이다. 이러한 노력은 이 분야의 선구적인 소프트웨어 공학자들에 의해 시도되었다. 지금도 대학, 연구소, 대규모 컴퓨터 회사 등에 의해 진행되고 있으나 아직 완성된 단계라고 할 수는 없다. 다만 이 과정에서 소프트웨어 개발 방법론이란 개념체계가 형성되었다는 데 의미가 있다. IEEE는 소프트웨어 공학을 다음과 같이 정의한다.

> 소프트웨어 공학이란 소프트웨어의 개발, 운용, 유지보수 및 파기에 대한 체계적인 접근 방법이다.

공학적 원리란 이론이나 방법론 또는 도구를 적절하게 선택하여 적용시키면서도 기존의 이론이나 방법론으로 해결할 수 없는 문제에 대해서 새로운 이론이나 방법 또는 기술을 찾아내려는 노력을 말한다. 소프트웨어 공학은 이러한 공학적 원리에 의하여 소프트웨어를 개발하

는 것뿐만 아니라 소프트웨어 프로젝트 관리, 도구와 방법론의 개발, 소프트웨어 생산을 지원하는 이론까지 포함하고 있다. 소프트웨어 공학의 의미를 소프트웨어 공학의 목표에 맞추어 정의하면 '품질이 좋은 소프트웨어를 최소한의 비용으로 계획된 일정에 맞추어 개발하는 것'이라고 할 수 있다.

3 소프트웨어 공학의 계층 구조

소프트웨어 공학에서 다루는 주제의 계층 구조는 다음과 같다.

그림 1-2 **소프트웨어 공학에서 다루는 주제의 계층 구조** [01]

■ 도구

도구란 프로그램 개발 과정에서 사용되는 여러 가지 방법을 자동화한 것을 말한다. 예를 들면, 요구분석 단계에서는 시스템 모형을 다이어그램으로 그려 주고 요구사항이 정확한지, 빠짐없이 분석되었는지, 일관성이 있는지 점검한다. 그 밖에도 비용 예측, 설계, 테스팅, 문서화, 유지보수 단계에서 사용하는 도구도 많이 나와 있다.

CASE[Computer Aided Software Engineering]는 소프트웨어 개발 전 단계를 지원하는 대표적인 도구이다. 1980년대 말 개인용 컴퓨터의 성능이 향상되면서 소프트웨어 개발 방법으로 다이어그래밍 도구와 자동 분석 기능을 패키지로 만들어 워크스테이션에 장착시킨 것이다. CASE는 계획, 분석, 설계와 같이 전반부를 지원하는 상위 CASE와 코딩, 테스팅, 유지보수와 같이 후반부를 지원하는 하위 CASE로 구분하기도 한다. CASE를 사용하면 개발 과정의 단계별 상황에 대해 설계자, 프로그래머, 테스터, 계획수립자, 관리자들이 공통의 시각을 공유하여 소프트웨어 개발환경을 증진시킬 수 있다.

■ **방법론**

방법론이란 소프트웨어 개발에 사용되는 기술적인 방법을 정형화시켜 제시하는 것을 말한다. 소프트웨어 개발의 여러 단계, 즉 요구사항 분석 및 설계, 프로그램 코딩, 테스팅, 유지보수 등을 각각 어떻게 할 것인가에 초점을 맞춘다고 할 수 있다. 한 단계에 관해서만 다루는 것을 방법Method이라고 하고 전 단계에 관하여 모색하는 것을 방법론Methodology이라고 구분한다.

참고로 분석 및 설계 단계에서는 주로 다이어그램을 이용하여 필요한 사항과 개발 과정을 단순화 · 추상화시킨다.

방법론은 크게 프로세스 중심 방법론, 자료 중심 방법론, 객체지향 방법론으로 구분한다.

- **프로세스 중심 방법론**

  프로세스 중심 방법론은 1970년대에 제시되어 가장 널리 알려졌는데 자료의 변환과정과 프로세스를 강조하여 프로그램을 개발하는 방법이다. 즉 하나의 문제를 해결하기 위해 프로그램을 작성할 때 그 문제를 풀기 위한 절차를 처음부터 마지막 단계까지 논리적 순서에 맞게 나열하는 과정으로 해법을 찾으려는 방법론이다. 이 과정에서 하나의 처리 단위를 프로세스라고 말하며 단위 프로세스는 그 처리에 필요한 자료를 입력받아 목적에 맞게 그 자료를 변환하여 출력한다는 점에서 자료의 흐름Data Flow, 변환Transaction 등이 중심 개념이라 할 수 있다.

  대부분의 프로그래밍 언어들은 이러한 절차적 처리에 적합하게 만들어졌으며 데이터베이스가 보편화되기까지 이러한 방법론이 주류를 이루었다. 또한 자료 처리를 위해서는 파일시스템을 활용했다. 그러나 이러한 절차적 방법론은 새로운 문제를 해결하기 위해서 또 다른 파일과 프로세스가 필요하기 때문에 소프트웨어의 재사용성이 낮고 파일시스템의 관리가 복잡해지는 등의 단점을 가지고 있다.

- **자료 중심 방법론**

  자료 중심 방법론은 프로그램을 개발할 때 사용할 자료를 규명하고 자료 간의 관계를 분석한 후 자료구조를 정의하고 이를 토대로 프로세스 구조를 고안하는 방법이다. 이러한 방법론의 탄생 배경은 프로세스 중심 방법론에 대한 반성에서부터 출발한다. 즉 처리는 사용자의 요구에 따라 자주 변하는데, 자료는 비교적 변화하지 않는 특성에 착안하여 자료구조를 정의하고 필요한 요구에 따라 해당 자료들을 추출하여 원하는 결과를 출력하는 방식을 택하게 된 것이다.

  프로세스 중심 방법론이 파일시스템에 기반한 절차적 언어(FORTRAN, COBOL, BASIC, PASCAL, C 등)의 프로그래밍 방식이라면 자료 중심 방법론은 데이터베이스에 기반한 쿼리 중

심 언어의 프로그래밍 방식이라고 할 수 있다. 1980년대 후반 이후 대부분의 응용 프로그램이 데이터베이스 기반의 쿼리 중심 언어로 개발되었다.

- **객체지향 방법론**

  객체지향 방법론은 자료와 프로세스를 묶어서 생각하는 방법이다. 자료와 프로세스의 결합을 객체라 부르며 이를 캡슐화 함으로써 좀 더 쉽게 프로세스의 모듈화, 정보 은닉, 코드 재사용의 효율성을 꾀할 수 있다. 앞서 살펴본 프로세스 중심 방법론과 자료 중심 방법론의 장점을 묶어 진화한 방법론이라고 말할 수 있다.

  앞서 1980년대 후반 데이터베이스를 기반으로 한 응용 프로그램 개발이 주류를 이루었다고 설명했다. 그러나 소프트웨어 개발의 획기적 발전의 계기는 객체지향 방법론의 개발과 적용에서 비롯되었다. 1990년대 이후 소프트웨어 개발현장에서는 RAD(Rapid Application Development)라 불리는 비주얼 개발 도구들(Visual Basic, Delphi, PowerBuilder, Visual C++ 등)이 출현하였다. 종전에 개발자들이 수개월에 걸쳐 개발하던 응용 프로그램을 단지 수일 만에 개발할 수 있게 된 것이다. 미리 개발되어 정의된 클래스 객체, 객체들이 단일 목적으로 조합된 컴포넌트 등이 개발자에게 제공되었다. 개발자들은 클래스나 컴포넌트들을 조합하는 것만으로도 훌륭한 응용 프로그램을 손쉽게 개발할 수 있게 되었다.

■ 프로세스

소프트웨어를 개발할 때 사용하는 방법과 도구를 적용할 순서를 정의한 것이 프로세스이다. 프로세스는 소프트웨어 개발에 필요한 작업 이름, 작업 내용, 결과물, 절차나 지시사항 등을 작업 사이의 선후 관계와 더불어 나타낸 것으로 소프트웨어 명세, 소프트웨어 개발, 소프트웨어 검증, 소프트웨어 진화 등의 영역으로 구분하기도 한다.

- **소프트웨어 명세**

  소프트웨어 개발에 앞서 표준화된 소프트웨어 명세 작성이 선행되어야 한다. 이 과정에서는 각종 코드 체계의 정리, 명명(命名) 규칙의 수립, 자료(테이블) 명세, 모듈 명세 등의 작업이 이루어진다.

- **소프트웨어 개발**

  소프트웨어 명세를 바탕으로 소프트웨어를 개발하는 과정으로, 다양한 사용자층의 요구에 맞추어 사용자 편의성을 최우선으로 고려하여 개발을 진행한다.

• **소프트웨어 검증**

소프트웨어 개발을 완료한 이후에는 검증 과정을 거쳐 소프트웨어의 품질보증을 하게 된다. 소프트웨어 품질의 중요성이 날로 높아지고 있는 점을 감안할 때 소프트웨어 검증 절차의 중요성 또한 소홀히 할 수 없다.

• **소프트웨어 진화**

처음부터 완벽한 소프트웨어를 개발하는 것이 최선의 목표이지만 사용 환경이 변화함에 따라 소프트웨어도 진화가 필요하다. 소프트웨어 진화란 개발이 완료된 소프트웨어가 새로운 요구에 부응하기 위해 새로운 버전으로 발전하는 것을 의미한다.

프로세스는 소프트웨어 공학의 관리적인 측면을 강조한 것으로 개발팀이나 사용자의 특성에 따라 달라진다. 프로세스를 어떻게 정의하느냐에 따라 소프트웨어의 품질이 달라지는 만큼 프로세스를 최적화하고 향상시키려는 노력은 계속되어야 한다.

■ 품질

소프트웨어 공학에서 가장 중요한 목표는 양질의 소프트웨어를 생산하는 것이다. 양질의 소프트웨어를 한마디로 정의하기란 쉬운 일이 아니다. 당장 소프트웨어를 접하는 관점에 따라 품질을 평가하는 기준이 달라지기 때문이다. 소프트웨어 품질을 평가하는 기준이 많이 있지만 정확성, 유지보수성, 무결성, 사용성 네 가지는 프로젝트팀에게 유용한 지표가 된다.

• **정확성**

정확성은 소프트웨어가 요구하는 기능을 수행하는 정도를 말한다. 정확하게 작동하지 않는 프로그램은 사용자에게 가치가 없다.

• **유지보수성**

소프트웨어 유지보수는 다른 소프트웨어 공학 활동보다 오랜 기간과 많은 노력이 필요하다. 유지보수성은 프로그램에 오류가 발견되면 이를 수정할 수 있고, 프로그램의 환경이 변하면 새로운 환경에 적응시킬 수 있고, 고객의 요구사항이 변경되면 이를 수용할 수 있는 프로그램의 용이성을 말한다.

• **무결성**

소프트웨어 무결성은 해커와 바이러스가 번성하는 요즘 그 중요성이 날로 강조되고 있다. 무결성은 시스템 보안을 위해 공격에 저항하는 시스템의 능력을 말한다. 공격은 소프트웨어의 세 가

지 구성요소(프로그램, 자료, 문서) 모두에 가해질 수 있다. 무결성을 측정하려면 위협과 보안이 정의되어야 하는데, 위협은 특정한 유형의 공격이 주어진 시간 내에 발생하는 확률이고 보안은 특정한 유형의 공격을 물리칠 수 있는 확률이다.

• **사용성**

소프트웨어 제품을 논의할 때마다 나오는 캐치프레이즈는 사용자 편이성User Friendliness일 것이다. 사용자가 쓰기에 편리하지 않으면 프로그램의 수행능력이 아무리 뛰어나도 보편화되기 어려워 실패하기 쉽다. 사용성은 다음 네 가지 특성으로 측정할 수 있다.

- 시스템을 배우는 데 요구되는 물질적, 지적 노력
- 시스템의 사용이 적합한 효율을 갖는 데 걸리는 시간
- 보통 사람이 시스템을 사용할 때 측정되는 생산성의 순수 증가
- 시스템에 대한 사용자의 주관적 평가

Chapter 15

미니 프로젝트 3

01 사례 소개

02 기존 방법론을 활용한 분석

03 UML 도구를 활용한 분석 및 설계

04 기존 방법론과 객체지향 방법론의 비교

학습목표

- 객체지향 방법론을 적용하여 실제 사례를 분석 및 설계해 본다.
- UML 도구의 활용법을 사례를 통해 학습한다.
- 기존 방법론과 객체지향 방법론의 차이점을 이해한다.

01 사례 소개

13장과 14장에서 객체지향 방법론의 핵심개념과 UML 도구에 대한 간략한 설명 그리고 작성 사례를 살펴보았다. 이제 지금까지 학습한 내용을 바탕으로 실제 사례를 들어 객체지향 방법론이 어떻게 적용되는지 이해하고, 앞으로 개발에 적용할 수 있는 능력을 키워보자.

1 차계부 앱 개발

이 장에서 다룰 사례는 '차계부 앱'이다. 차계부 앱은 스마트폰을 이용하여 차량운행과 관련된 주유기록, 정비기록, 여행기록 등을 기록하고 조회해 볼 수 있는 프로그램이다. 차량을 소유한 운전자라면 대부분 차계부를 가지고 있을 것이다. 필자도 차를 산 후부터 직접 차계부를 작성해 오고 있다. 차계부에 작성한 내용을 바탕으로 정비기록을 살펴보고 적정한 시점에 필요한 점검을 받을 수 있어 아주 유용하다. 그러던 중 이러한 기록을 스마트폰 앱으로 만들어 활용하면 좋겠다는 생각이 들었고, 직접 앱 개발에 착수하였다. 이 장에서는 그 과정을 바탕으로 설명하고자 한다.

2 업무(기능) 명세 작성

우선 최초로 해야 할 일이 무엇일까? 이 고민은 객체지향에만 국한된 것이 아니다. 어느 방법론으로 시스템을 만들더라도 '시스템으로 무엇을 할 것인가, 그리고 무엇을 하지 않을 것인가'를 결정할 필요가 있다. 이것을 소위 '명세Specification'라고 한다.

명세는 자세히 작성할수록 좋지만, 객체지향 방법론은 시스템 변경에 매우 유연하므로 '분석 단계로는 절대로 되돌아가지 않는다'는 확고부동한 결의로 완고하게 대처할 필요는 없다. 근본적으로 전혀 다른 명세가 아니라면, 최초의 명세는 프로토타입Prototype 정도로 생각해도 된다(실제로 객체지향 개발에서 프로토타입부터 시작하는 경우가 빈번하다). [01]

차계부 앱 작성을 위한 업무 명세는 [그림 15-1]과 같다. 앱 이름은 '스마트 차계부'라고 하겠다.

스마트 차계부 명세

- 스마트 차계부는 차량의 주유기록은 물론 정기적인 점검을 통해 부품이나 소모품 등을 교환한 정비기록을 저장한다.
- 이렇게 저장한 주유기록과 정비기록은 언제든 조회 가능하며, 월별주유현황 등을 그래프로 볼 수도 있다.
- 스마트 차계부는 여행기록을 남기는 기능도 있다. 다시 찾고 싶은 여행지, 맛집, 포토존 등을 지도에 표시하여 보여준다.

그림 15-1 **차계부 명세**

스마트 차계부는 스마트폰용으로 개발하기 때문에 기존의 전통적 방법보다는 객체지향 방법으로 개발하는 것이 적합하다. 대부분의 앱은 스마트폰 운영환경에 맞추어 제공되는 개발 플랫폼을 이용해 만드는데, 필자의 경우 자바와 안드로이드 기반 SDK에서 제공하는 다양한 클래스와 컴포넌트를 활용하여 개발하였다.

다음 2절에서는 기존의 방법론을 활용하여 스마트 차계부를 분석해 보고, 3절에서는 객체지향 방법론을 활용하여 분석 및 설계해 보고자 한다. 구체적인 분석 도구를 이용하여 진행하므로 기존 방법론과 객체지향 방법론의 차이를 보다 잘 이해할 수 있을 것이다.

02 기존 방법론을 활용한 분석

1 구조적 방법론에 의한 분석

기존의 구조적 방법론을 활용해 스마트 차계부 시스템을 분석해 보자.

❶ 자료흐름도 작성

[그림 15-1]의 업무 명세를 바탕으로 자료흐름도를 작성하면 다음 그림과 같다. 물론 배경도와 1차 분할도 및 2차 분할도 등으로 세분화해 나가는 것이 좋겠지만 비교적 단순한 사례인 점을 감안하여 1차 분할도에 해당하는 자료흐름도만 작성해 보았다.

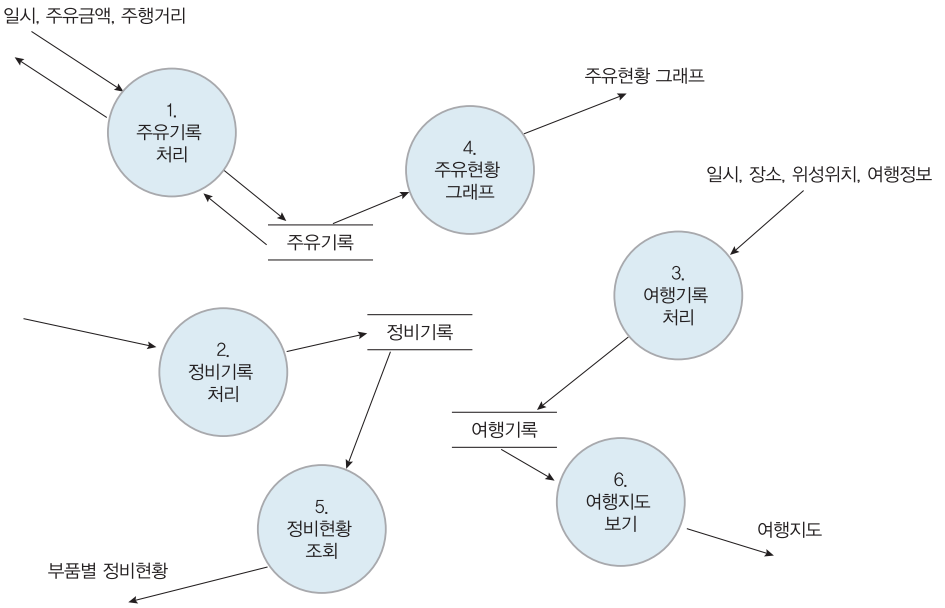

그림 15-2 스마트 차계부의 자료흐름도

❷ 자료사전 작성

- 주유기록 = { 주유일자 + 주유금액 + 주행거리 }
- 정비기록 = { 정비일자 + 부품명 + 정비비용 + 주행거리 }
- 여행기록 = { 여행일자 + 여행장소 + GPS위치 + 여행지정보 }
- 주유현황 그래프 = { 월별 + 주유금액 합계 + 주행거리 합계 }
- 부품별 정비현황 = { 부품명 + { 정비일자 + 정비금액 } }
- 여행지도 = * 여행기록과 동의어로 여행기록을 지도에 표기한 것 *

2 정보공학 방법론에 의한 분석

동일한 사례를 이번에는 정보공학 방법론으로 분석해 보고 그 결과를 기능도Function Chart와 ERDEntity-Relationship Diagram, E-R 다이어그램로 표현해 보자.

❶ 기능도 작성

다음은 스마트 차계부의 기능도를 작성한 것이다.

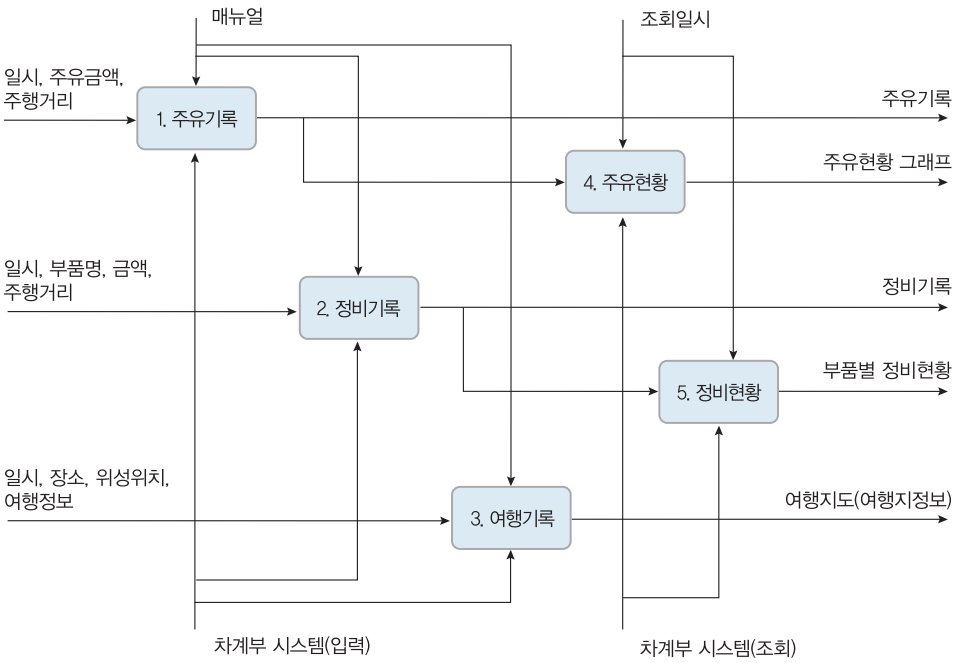

그림 15-3 스마트 차계부의 기능도

❷ ERD 작성

다음은 스마트 차계부에 대한 ERD를 작성한 것이다.

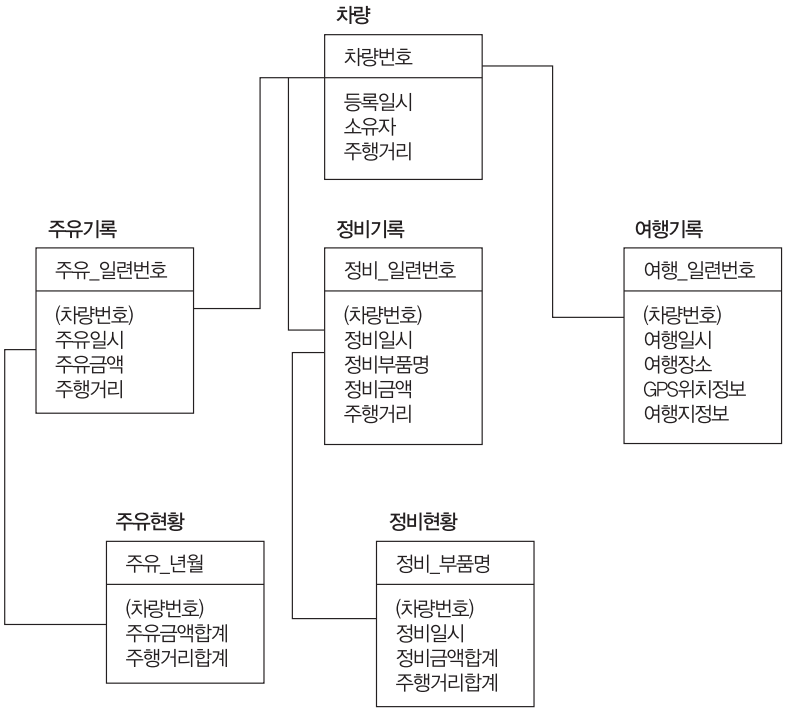

그림 15-4 스마트 차계부의 ERD

우리는 이미 2부와 3부에서 이러한 분석 과정과 그 결과를 활용하여 시스템을 구축하기 위한 절차에 대해 학습한 바 있다. 자료흐름도나 기능도는 시스템의 실행 모듈 단위로 인식되어 프로그램 개발자에게 넘겨질 것이며, 자료사전이나 ERD는 자료저장소 즉, 데이터베이스 설계를 위한 자료로 활용될 것이다.

그렇다면 객체지향 방법론에 의한 분석 및 설계 절차는 기존의 분석 및 설계 절차와 어떻게 다른 것일까? 그 궁금증에 대한 해답을 다음 절에서 다루어 보기로 한다.