# 1장. 요구사항 확인

## 1. 소프트웨어 생명 주기

### 1. 소프트웨어 생명 주기(Software Life Cycle)

소프트웨어 생명 주기는 소프트웨어 개발 방법론의 바탕이 되는 것으로, 소프트웨어를 개발하기 위해 정의하고 운용, 유지보수 등의 과정을 각 단계별로 나눈 것이다.

* 소프트웨어 생명 주기는 소프트웨어 개발 단꼐와 각 단계별 주요 활동, 그리고 활동의 결과에 대한 산출물로 표현한다. 소프트웨어 수명 주기로고도 한다.
* 소프트웨어 생명 주기를 표현하는 형태를 소프트웨어 생명 주기 모형이라고 하며, 스프트웨어 프로세스 모형 또는 소프트웨어 공학 패러다임이라고도 한다.
* 개발자는 문제의 유형이나 개발 방법 등에 따라 특정 모형을 선택하여 사용할 수도 있고, 개별적인 모형을 사용할 수도 있다.
* 일반적으로 사용되는 소프트웨어 생명 주기 모형에는 폭포수 모형, 프로토타입 모형, 나선형 모형, 에자일 모형 등이 있다.

cf) : 소프트웨어 공학

소프트웨어 공학의 개념

* 소프트웨어 공학(SE)은 소프트웨어의 위기를 극복하기 위한 방안으로 연구된 학문이며 어러 가지 방법론과 도구, 관리 기법등을 통하여 소프트웨어의 품질과 생산성 향상을 목적으로 한다.
* 소프트웨어 공학은 다음과 같이 여러 형태로 정의할 수 있다.
  + IEEE의 소프트웨어 공학 표준 용어사전
  + 소프트웨어의 개발, 운용, 유지보수, 폐기 처분에 대한 체계적인 접근 방인
  + Fairley
  + 지정된 비용과 기간 내에 소프트웨어를 체계적으로 생선하고 유지보수하는 데 관련된 기술적이고 관리적인 원리
  + Boehm
  + 과학적인 지식을 소프트웨어 설계와 제작에 응용하는 것이며 이를 개발, 운용, 유지보수하는 데 필요한 문서 작성 과정

소프트웨어 공학의 기본 원칙

* 현대적인 프로그래밍 기술을 계속적으로 적용해야 한다.
* 개발된 소프트웨어의 품질이 유지되도록 지속적으로 검증해야 한다.
* 소프트웨어 개발 관련 사항 및 결과에 대한 명확한 기록을 유지해야 한다.

### 2. 폭포수 모형(Waterfall Model)

폭포수 모형은 폭포에서 한번 떨어진 물은 거슬러 올라갈 수 없듯이 소프트웨어 개발도 이전 단계로 돌아갈 수 없다는 전제하에 각 단계를 확실히 매듭짓고 그 결과를 철저하게 검토하여 승인 과정을 거친 후에 다음 단계를 진행하는 개발 방법론이다.

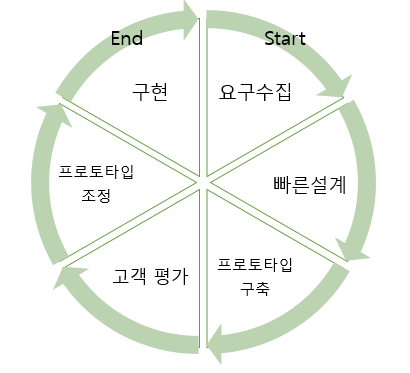
* 폭포수 모형은 소프트웨어 공학에서 가장 오래되고 가장 폭넓게 사용된 전통적인 소프트웨어 생명 주기 모형으로, 고전적 생명 주기 모형이라고도 한다.
* 소프트웨어 개발 과정의 한 단계가 끝나야만 다음 단계로 넘어갈 수 잇는 선형 순차적 모형이다.
* 모형을 적용한 경험과 성공 사례가 많다.
* 제품의 일부가 될 메뉴얼을 작성해야 한다.
* 각 단계가 끝난 후에는 다음 단계를 수행하기 위한 결과물이 명확하게 산출되어야 한다.
* 두 개 이상의 과정이 병행하여 수행되지 않는다.

타당성 검토 → 계획 → 요구 분석 → 설계 → 구현(코딩) → 시험(검사) → 유지보수

### 3. 프로토타입 모형(Prototype Model, 원형 모형)

프로토타입 모형은 사용자의 요구사항을 정확히 파악하기 위해 실제 개발될 소프트웨어에 대한 견본(시제)품을 만들어 최종 결과물을 예측하는 모형이다.

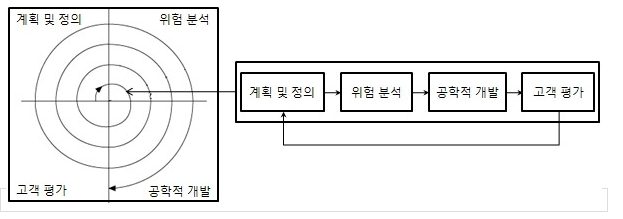
* 시제품은 사용자와 시스템 사이의 인터페이스에 중점을 두어 개발한다.
* 시스템의 일부 혹은 시스템의 모형을 만드는 과정으로서 요구된 소프트웨어를 구현하는데, 이는 추수 구현 단계에서 사용될 골격 코드가 된다.
* 소프트웨어의 개발이 완료된 시점에서 오류가 발견되는 폭포수 모형의 단점을 보완하기 위한 모형이다.



### 나선형(Spiral Model, 점진적 모형)

나선형 모형은 보헴이 제안한 것으로, 폭포수 모형과 프로토타입 모형의 장점에 위험 분석 기능을 추가한 모형이다.

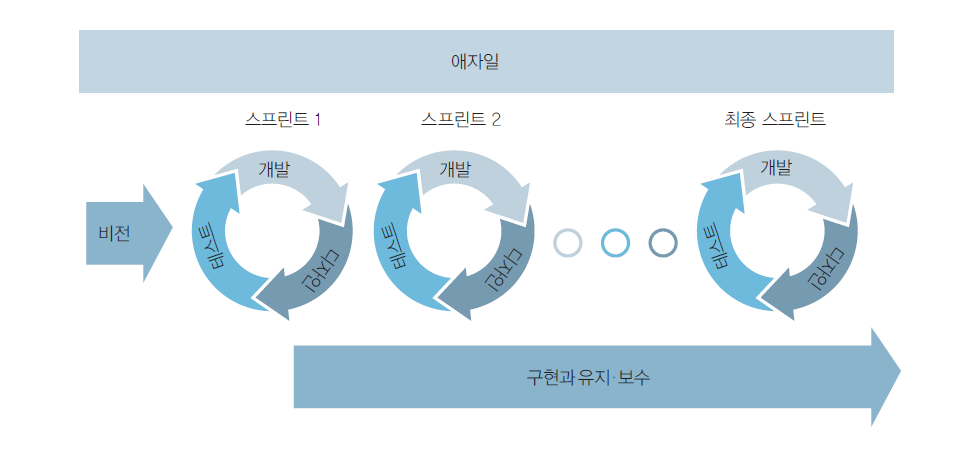
* 나선을 따라 돌듯이 어러 번의 소프트웨어 개발 과정을 거쳐 점진적으로 완벅한 최종 소프트웨어를 개발하는 것으로, 점진적 모형이라고도 한다.
* 소프트웨어를 개발하면서 발생할 수 있는 위험을 관리하고 최소화하는 것을 목적으로 한다.
* 점진적으로 개발 과정이 반복되므로 누락되거나 추가된 요구사항을 첨가할 수 있고, 정밀하며, 유지보수 과정이 필요 없다.



### 5. 애자일 모형(Agile Model)

애자일은 '민첩한'이라는 의미로, 고객의 요구사항 변화에 유연하게 대응할 수 있도록 일정한 주기를 반복하면서 개발과정을 진행한다.

* 애자일 모형은 어느 특정 개발 방법론이 아니라 좋은 것을 빠르고 낭비 없게 만들기 위해 고객과의 소통에 초점을 맞춘 방법론을 통칭한다.
* 애자일 모형은 기업 활동 전반에 걸쳐 사용된다.
* 애자일 모형은 스프린트 또는 이터레이션이라고 불리는 짧은 개발 주기를 반복하며, 반복하는 주기마다 만들어지는 결과물에 대한 고객의 평가와 요구를 수용한다.
* 각 개발주기에는 고객이 요구사항에 우선순위를 부여하여 개발 작업을 진행한다.
* 소규모 프로젝트, 고도로 숙달된 개발자, 급변하는 요구사항에 적합하다.
* 애자일 모형을 기반으로 하는 소프트웨어 개발 모형에는 스크럼, XP, 칸반, Lean, 크리스탈, ASD, 기능 중심 개발, DSDM, DAD 등이 있다.



cf) : 애자일 선언(Agile Manifesto)

2001년 17명의 애자일 전문 개발자가 공통의 관점을 정리해 '애자일 SW 개발 선언문'을 만들었다. 선언문에는 애자일 개발 철학이 담겨있는 4가지 핵심 가치와 애자일 개발을 실무에 적용할 때 기준이 되는 12가지 실행 지침이 담겨있는데, 그 내용은 다음과 같다.

* 애자일 개발 4가지 핵심 가치
  1. 프로세스와 도구보다는 개인과 상호작용에 더 가치를 둔다.
  2. 방대한 문서보다는 실행되는 SW에 더 가치를 둔다.
  3. 계약 협상보다는 고객과 협업에 더 가치를 둔다.
  4. 계획을 따르기 보다는 변화에 반응하는 것에 더 가치를 둔다.
* 애자일 개발 12가지 실행 지침
  1. 유용한 소프트웨어를 빠르고, 지속적으로 제공하여 고객을 만족시킨다.
  2. 개발 막바지라도 요구사항 변경을 적극 수용한다.
  3. 몇 개월이 아닌 몇 주 단위로 실행되는 소프트웨어를 제공한다.
  4. 고객과 개발자가 프로젝트 기간에 함께 일한다.
  5. 개발에 대한 참여 의지가 확실한 사람들로 팀을 구성하고, 필요한 개발 환경과 지원을 제공하며, 일을 잘 끝낼 수 있도록 신뢰한다.
  6. 같은 사무실에서 얼굴을 맞대고 의견을 나눈다.
  7. 개발의 진척도를 확인하는 1차 기준은 작동하는 소프트웨어이다.
  8. 지속 가능한 개발을 장려하고 일정한 속도로 개발을 진행한다.
  9. 기술적 우수성과 좋은 설계에 지속적인 관심을 기울이면 민첩성이 향상된다.
  10. 단순화를 추구한다.
  11. 최상의 아키텍쳐, 명확한 요구사항, 최상의 설계는 자기 스스로 일을 주도하는 조직적인 팀으로부터 나온다.
  12. 더 효과적인 팀이 될 수 있도록 방안을 정기적으로 깊이 고민하고 그에 따라 팀의 행동을 조정한다.

### 6. 폭포수 모형과 애자일의 비교

| 구분 | 폭포수 모형 | 애자일 |
| --- | --- | --- |
| 새로운 요구사항 반영 | 어려움 | 지속적으로 반영 |
| 고객과의 의사소통 | 적음 | 지속적임 |
| 테스트 | 마지막에 모든 기능을 테스트 | 반복되는 일정 주기가 끝날 때마다 테스트 |
| 개발 중심 | 계획, 문서(메뉴얼) | 고객 |

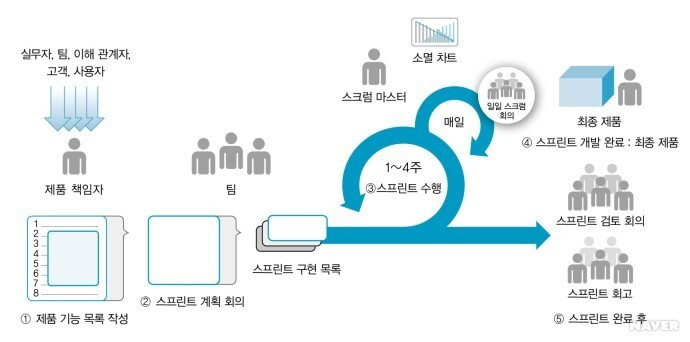
## 2. 스크럼(Scrum) 기법

### 1. 스크럼의 개요

스크럼이란 럭비에서 반칙으로 경기가 중단된 경우 양 팀의 선수들이 럭비공을 가운데 두고 상대팀을 밀치기 위해 서로 대치해 있는 대형을 말한다. 스크럼은 이처럼 팀이 중심이 되어 개발의 효율성을 높인다는 의미가 내포된 용어이다.

* 스크럼은 팀원 스스로가 스크럼 팀을 구성(self-organizing)해야 하며, 개발 작업에 관한 모든 것을 스스로 해결(cross-functional)할 수 있어야 한다.
* 스크럼 팀은 제품 책임자, 스크럼 마스터, 개발팀으로 구성된다.
  + 제품 책임자(PO; Product Owner)
    - 이해관계자들 중 개발될 제품에 대한 이해도가 높고, 요구사항을 책임지고 의사 결정할 사람으로 선정하는데, 주로 개발 의뢰자나 사용자가 담당한다.
    - 이해관계자들의 의견을 종합하여 제품에 대한 요구사항을 작성하는 주체다.
    - 요구사항이 담긴 백로그(Backlog)를 작성하고 백로그에 대한 우선순위를 지정한다.
    - 팀원들이 백로그에 스토리를 추가할 수는 있지만 우선순위를 지정할 수는 없다.
    - 제품에 대한 테스트를 수행하면서 주기적으로 요구사항의 우선순위를 갱신한다.
  + 스크럼 마스터(SM; Scrum Master)
    - 스크럼 팀이 스크럼을 잘 수행할 수 있도록 객관적인 시각에서 조언을 해주는 가이드 역할을 수행한다. 팀원들을 통제하는 것이 목표가 아니다.
    - 일일 스크럼 회의를 주관하여 진행 사항을 점검하고, 개발 과정에서 발생된 장애 요소를 공론화하여 처리한다.
  + 개발팀(DT; Development Team)
    - 제품 책임자와 스크럼 마스터를 제외한 모든 팀원으로, 개발자 외에도 디자이너, 테스터 등 제품 개발을 위해 참여하는 모든 사람이 대상이 된다.
    - 보통 최대 인원은 7 ~ 8명이 적당하다.

### 2. 스크럼 개발 프로세스



* 제품 백로그(Product Backlog)
  + 제품 개발에 필요한 모든 요구사항(User Story)을 우선순위에 따라 나열한 목록이다.
  + 개발 과정에서 새롭게 도출되는 요구사항으로 인해 지속적으로 업데이트된다.
  + 제품 백로그에 작성된 사용자 스토리를 기반으로 전체 일정 계획인 릴리즈 계획(Release Plan)을 수립한다.
* 스프린트 계획 회의(Sprint Planning Meeting)
  + 제품 백로그 중 이번 스프린트에서 수행할 작업을 대상으로 단기 일정을 수립하는 것이다.
  + 스프린트에서 처리할 요구사항(User Story)을 개발자들이 나눠서 작업할 수 있도록 태스크(Task)라는 작업 단위로 분할한 후 개발자별로 수행할 작업 목록인 스프린트 백로그(Sprint Backlog)를 작성한다.
* 스프린트(Sprint)
  + 실제 개발 작업을 진행하는 과정으로, 보통 2-4주 정도의 기간 내에서 진행한다.
  + 스프린트 백로그에 작성된 태스크를 대상으로 작업 시간(양)을 추정한 후 개발 담당자에게 할당한다.
  + 태스크를 할당할 때는 개발자가 원하는 태스크를 직접 선별하여 담당할 수 있도록 하는 것이 좋다.
  + 개발 담당자에게 할당된 태스크는 보통 할 일(To Do), 진행 중(In Progress), 완료(Done)의 상태를 갖는다.
* 일일 스크럼 회의(Daily Scrum Meeting)
  + 모든 팀원이 매일 약속된 시간에 약 15분 정도의 짧은 시간동안 진행 상황을 점검한다.
  + 회의는 보통 서서 진행하며, 남은 작업 시간은 소멸 차트(Burn-down Chart)에 표시한다.
  + 스크럼 마스터는 발견된 장애 요소를 해결할 수 있도록 도와준다.
* 스프린트 검토 회의(Sprint Review)
  + 부분 또는 전체 완성 제품이 요구사항에 잘 부합되는지 사용자가 포함된 참석자 앞에서 테스팅을 수행한다.
  + 스프린트의 한 주당 한 시간 내에서 진행한다.
  + 제품 책임자(Product Owner)는 개선할 사항에 대한 피드백을 정리한 후 다음 스프린트에 반영할 수 있도록 제품 백로그를 업데이트한다.
* 스프린트 회고(Sprint Retrospective)
  + 스프린트 주기를 되돌아보며 정해놓은 규칙을 잘 준수했는지, 개선할 점은 없는지 등을 확인하고 기록한다.
  + 해당 스프린트가 끝난 시점에서 수행하거나 일정 주기로 수행한다.

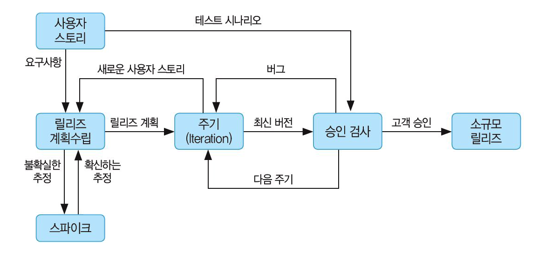
## 3. XP(extreme programming) 기법

### 1. XP(extreme programming)

XP(extreme programming)는 수시로 발생하는 고객의 요구사항에 유연하게 대응하기 위해 고객의 참여와 개발 과정의 반복을 극대화하여 개발 생산성을 향상시키는 방법이다.

* XP는 짧고 반복적인 개발 주기, 단순한 설계, 고객의 적극적인 참여를 통해 소프트웨어를 빠르게 개발하는 것을 목적으로 한다.
* 릴리즈의 기간을 짧게 반복하면서 고객의 요구사항 반영에 대한 가시성을 높인다.
* 릴리즈 테스트마다 고객을 직접 참여시킴으로써 요구한 기능이 제대로 작동하는지 고객이 직접 확인할 수 있다.
* 비교적 소규모 인원의 개발 프로젝트에 효과적이다.
* XP의 5가지 핵심 가치 : 의사소통(Communication), 단순성(Simplicity), 용기(Courage), 존중(Respect), 피드백(Feedback)

### XP 개발 프로세스



* 사용자 스토리(User Story)
  + 고객의 요구사항을 간단한 시나리오로 표현한 것이다.
  + 내용은 기능 단위로 구성하며, 필요한 경우 간단한 테스트 사항(Test Case)도 기재한다.
* 릴리즈 계획 수립(Release Planning)
  + 몇 개의 스토리가 적용되어 부분적으로 기능이 완료된 제품을 제공하는 것을 릴리즈라고 한다.
  + 부분 혹은 전체 개발 완료 시점에 대한 일정을 수립한다.
* 스파이크(Spike)
  + 요구사항의 신뢰성을 높이고 기술 문제에 대한 위험을 감소시키기 위해 별도로 만드는 간단한 프로그램이다.
  + 처리할 문제 외의 다른 조건은 모두 무시하고 작성한다.
* 이터레이션(Iteration)
  + 하나의 릴리즈를 더 세분화 한 단위를 이터레이션(iteration)이라고 한다.
  + 일반적으로 1-3주 정도의 기간으로 진행된다.
  + 이 기간 중에 새로운 스토리가 작성될 수 있으며, 작성된 스토리는 진행 중인 이터레이션 혹은 다음 이터레이션에 포함될 수 있다.
* 승인 검사(Acceptance Test, 인수 테스트)
  + 하나의 이터레이션 안에서 계획된 릴리즈 단위의 부분 완료 제품이 구현되면 수행하는 테스트이다.
  + 사용자 스토리 작성 시 함께 기재한 테스트 사항에 대해 고객이 직접 수행한다.
  + 테스트 과정에서 발견한 오류 사항은 다음 이터레이션에 포함한다.
  + 테스트 이후 새로운 요구사항이 작성되거나 요구사항의 상대적 우선순위가 변경될 수 있다.
  + 테스트가 완료되면 다음 이터레이션을 진행한다.
* 소규모 릴리즈(Small Release)
  + 릴리즈를 소규모로 하게 되면, 고객의 반응을 기능별로 확인할 수 있어, 고객의 요구사항에 좀 더 유연하게 대응할 수 있다.
  + 계획된 릴리즈 기간 동안 진행된 이터레이션이 모두 완료되면 고객에 의한 최종 테스트를 수행한 후 릴리즈, 즉 최종 결과물을 고객에게 전달한다.
  + 릴리즈가 최종 완제품이 아닌 경우 다음 릴리즈 일정에 맞게 개발을 계속 진행한다.

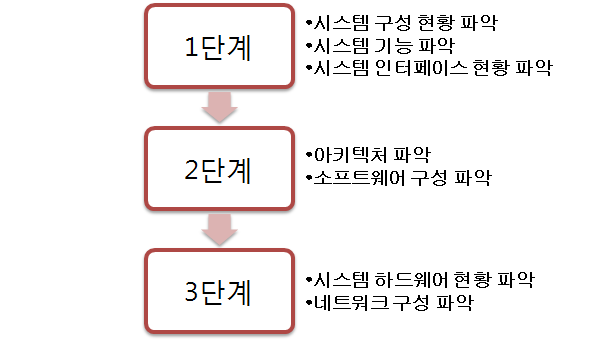
cf) : XP의 주요 실천 방법(Practice)

| 실천 방법 | 내용 |
| --- | --- |
| Pair Programming(짝 프로그래밍) | 다른 사람과 함께 프로그래밍을 수행함으로써 개발에 대한 책임을 공동으로 나눠 갖는 환경을 조성한다. |
| Collective Ownership(공동 코드 소유) | 개발 코드에 대한 권한과 책임을 공동으로 소유한다. |
| Test-Driven Development(테스트 주도 개발) | \* 개발자가 실제 코드를 작성하기 전에 테스트 케이스를 먼저 작성하므로 자신이 무엇을 해야할지 정확히 파악한다. \* 테스트가 지속적으로 진행될 수 있도록 자동화된 테스팅 도구(구조, 프레임워크)를 사용한다. |
| Whole Team(전체 팀) | 개발에 참여하는 모든 구성원(고객 포함)들은 각자 자신의 역할이 있고 그 역할에 대한 책임을 가져야 한다. |
| Continuous Integration(지속적 통합) | 모듈 단위로 나눠서 개발된 코드들은 하나의 작업이 마무리될 때마다 지속적으로 통합한다. |
| Design Improvement(디자인 개선) 또는 Refactoring(리팩토링) | 프로그램 기능의 변경 없이, 단순화, 유연성 강화 등을 통해 시스템을 재구성한다. |
| Small Release(소규모 릴리즈) | 릴리즈 기간을 짧게 반복함으로써 고객의 요구 변화에 신속히 대응할 수 있다. |

## 4. 현행 시스템 파악

### 1. 현행 시스템 파악 절차

새로 개발하려는 시스템의 개발 범위를 명확히 설정하기 위해 현행 시스템의 구성과 제공 기능, 시스템 간의 전달 정보, 사용되는 기술 요소, 소프트웨어, 하드웨어, 그리고 네트워크의 구성 등을 파악한다.

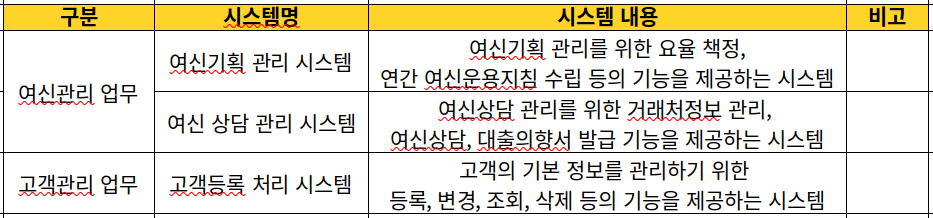


### 2. 시스템 구성 파악

현행 시스템의 구성은 조직의 주요 업무를 담당하는 기간 업무와 이를 지원하는 지원 업무로 구분하여 기술한다.

* 조직 내에 있는 모든 정보시스템의 현황을 파악할 수 있도록 각 업무에 속하는 단위 업무 정보시스템들의 명칭, 주요 기능들을 명시한다.

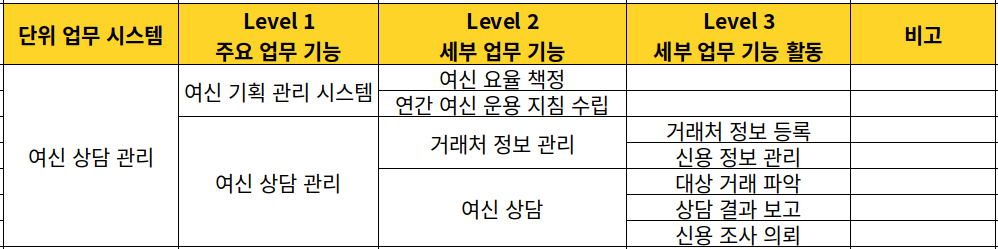
ex) 금융 기관의 여신관리 업무와 고객관리 업무 시스템 현황



### 3. 시스템 기능 파악

현행 시스템의 기능은 단위 업무 시스템이 현재 제공하는 기능들을 주요 기능과 하부 기능, 세부 기능으로 구분하여 계층형으로 표시한다.

ex) 여신상담 관리 시스템의 주요 기능과 하부, 세부 기능



### 4. 시스템 인터페이스 파악

현행 시스템의 인터페이스에는 단위 업무 시스템 간에 주고받는 데이터의 종류, 형식, 프로토콜, 연계 유형, 주기 등을 명시한다.

* 데이터를 어떤 형식으로 주고받는지, 통신규약은 무엇을 사용하는지, 연계 유형은 무엇인지 등을 반드시 고려해야 한다.

ex) 여신상담 관리 시스템의 인터페이스 현황

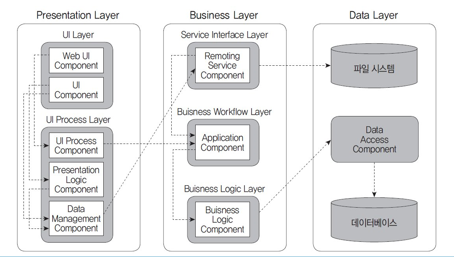


### 5. 아키텍처 구성 파악

현행 시스템의 아키텍처 구성은 기간 업무 수행에 어떠한 기술 요소들이 사용되는지 최상위 수준에서 계층별로 표현한 아키텍처 구성도로 작성한다.

* 아키텍처가 단위 업무 시스템별로 다른 경우에는 가장 핵심이 되는 기간 업무 처리 시스템을 기준으로 표현한다.

ex) 회원 정보 관리 시스템 아키텍처 구성도

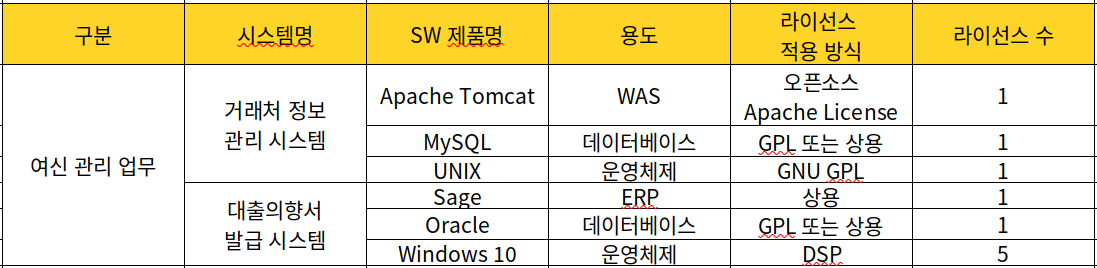


### 6. 소프트웨어 구성 파악

소프트웨어 구성에는 단위 업무 시스템별로 업무 처리를 위해 설치되어 있는 소프트웨어들의 제품명, 용도, 라이선스 적용 방식, 라이언스 수 등을 명시한다.

* 시스템 구축비용 면에서 소프트웨어 비용이 적지 않은 비중을 차지하므로, 상용 소프트웨어의 경우 라이선스 적용 방식의 기준과 보유한 라이선스의 파악이 중요하다.

ex) 단위 업무 시스템별 소프트웨어 현황

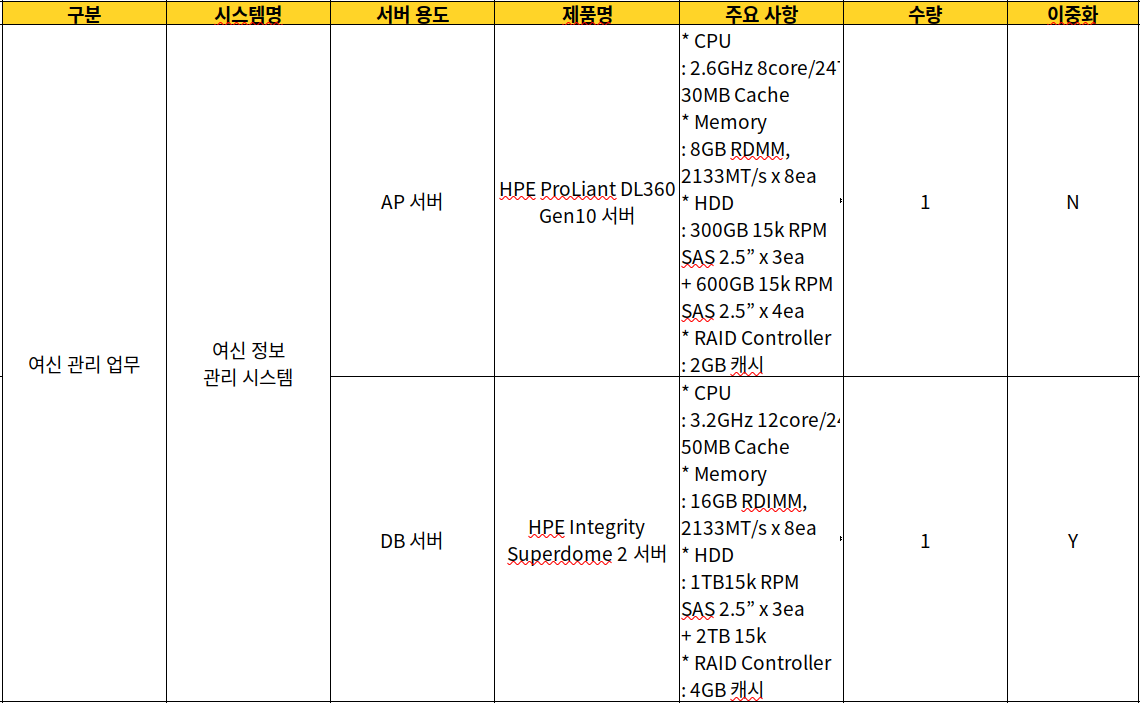


### 7. 하드웨어 구성 파악

하드웨어 구성에는 단위 업무 시스템들이 운용되는 서버의 주요 사항과 수량, 그리고 이중화의 적용 여부를 명시한다.

* 서버의 이중화는 기간 업무의 서비스 기간, 장애 대응 정책에 따라 필요 여부가 결정된다.
* 현행 시스템에 이중화가 적용된 경우 대부분 새로 구성될 시스템에도 이중화가 필요하므로 이로 인한 비용 증가와 시스템 구축 난이도가 높아질 가능성을 고려해야 한다.

ex) 단위 업무 시스템별 하드웨어 현황

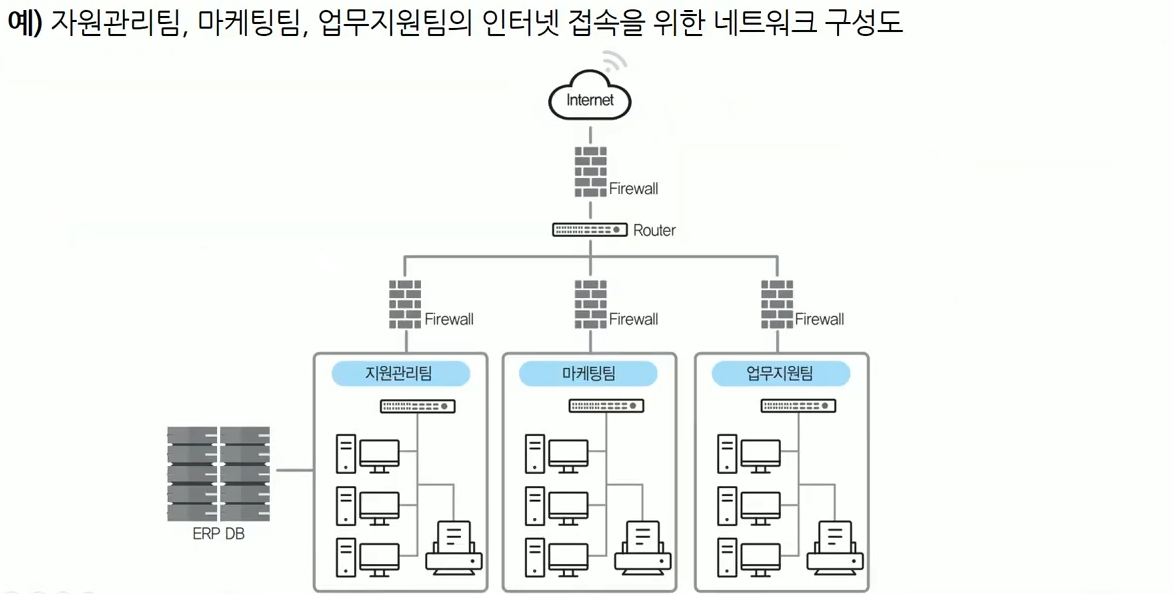


### 8. 네트워크 구성 파악

네트워크 구성은 업무 시스템들의 네트워크 구성을 파악할 수 있도록 서버의 위치, 서버 간의 네트워크 연결 방식을 네트워크 구성도로 작성한다.

* 네트워크 구성도를 통해 서버들의 물리적인 위치 관계를 파악할 수 있고 보안 취약성을 분석하여 적절한 대응을 할 수 있다.
* 네트워크에 장애가 발생한 경우 발생 원인을 찾아 복구하기 위한 용도로 활용될 수 있다.

ex) 자원관리팀, 마케팅팀, 업무 지원팀의 인터넷 접속을 위한 네트워크 구성도



## 5. 개발 기술 환경 파악

### 1. 개발 기술 환경의 정의

개발하고자 하는 소프트웨어와 관련된 운영체제(OS), 데이터베이스 관리 시스템(DBMS), 미들웨어(Middle Ware) 등을 선정할 때 고려해야 할 사항을 기술하고, 오픈 소스 사용 시 주의해야 할 내용을 제시한다.

### 2. 운영체제(OS)

운영체제는 컴퓨터 시스템의 자원들을 효율적으로 관리하며, 사용자가 컴퓨터를 편리하고 효율적으로 사용할 수 있도록 환경을 제공하는 소프트웨어이다.

* 컴퓨터 사용자와 컴퓨터 하드웨어 간의 인터페이스로서 동작하는 시스템 소프트웨어의 일종으로, 다른 응용 프로그램이 유용한 작업을 할 수 있도록 환경을 제공해준다.
* 컴퓨터 운영체제의 종류에는 Windows, UNIX, Linux, Mac OS 등이, 모바일 운영체제에는 iOS, Android 등이 있다.

### 3. 운영체제 관련 요구사항 식별 시 고려사항

운영체제와 관련된 요구사항 식별 시 다음과 같은 사항을 고려해야 한다.

| 구분 | 내용 |
| --- | --- |
| 가용성 | \* 시스템의 장시간 운영으로 인해 발생할 수 있는 운영체제 고유의 장애 발생 가능성 \* 메모리 누수로 인한 성능 저하 및 재가동 \* 보안상 발견된 허점을 보완하기 위한 지속적인 패치 설치로 인한 재가동 \* 운영체제의 결함 등으로 인한 패치 설치를 위한 재가동 |
| 성능 | \* 대규모 동시 사용자 요청에 대한 처리 \* 대규모 및 대용량 파일 작업에 대한 처리 \* 지원 가능한 메모리 크기(32bit, 64bit) |
| 기술 지원 | \* 제작업체의 안정적인 기술 지원 \* 여러 사용자들 간의 정보 공유 \* 오픈 소스 여부(Linux) |
| 주변 기기 | \* 설치 가능한 하드웨어 \* 여러 주변 기기 지원 여부 |
| 구축 비용 | \* 지원 가능한 하드웨어 비용 \* 설치할 응용 프로그램의 라이선스 정책 및 비용 \* 유지관리 비용 \* 총 소유 비용(TCO) |

### 4. 데이터베이스 관리 시스템(DBMS)

DBMS(DataBase Management System)는 사용자와 데이터베이스 사이에서 사용자의 요구에 따라 정보를 생성해 주고, 데이터베이스를 관리해 주는 소프트웨어이다.

* DBMS는 기존의 파일 시스템이 갖는 데이터의 종속성과 중복성의 문제를 해결하기 위해 제안된 시스템으로, 모든 응용 프로그램들이 데이터베이스를 공용할 수 있도록 관리해 준다.
* DBMS는 데이터베이스의 구성, 접근 방법, 유지관리에 대한 모든 책임을 진다.
* DBMS의 종류에는 Oracle, IBM, DB2, Microsoft SQL Server, MySQL, SQLite, MongoDB, Redis 등이 있다.

### 5. DBMS 관련 요구사항 식별 시 고려사항

DBMS와 관련된 요구사항 식별 시 다음과 같은 사항을 고려해야 한다.

| 구분 | 내용 |
| --- | --- |
| 가용성 | \* 시스템의 장시간 운영으로 인해 발생할 수 있는 운영체제 고유의 장애 발생 가능성 \* DBMS의 결함 등으로 인해 패치 설치를 위한 재가동 \* 백업이나 복구의 편의성 \* DBMS 이중화 및 복제 지원 |
| 성능 | \* 대규모 데이터 처리 성능(분할 테이블 지원 여부) \* 대용량 트랜잭션 처리 성능 \* 튜닝 옵션의 다양한 지원 \* 최소화된 설정과 비용 기반 질의 최적화 지원 |
| 기술 지원 | \* 제작업체의 안정적인 기술 지원 \* 여러 사용자들 간의 정보 공유 \* 오픈 소스 여부 |
| 상호 호환성 | \* 설치 가능한 운영체제의 종류 \* JDBC, ODBC와의 호환 여부 |
| 구축 비용 | \* 라이선스 정책 및 비용 \* 유지관리 비용 \* 총 소유 비용(TCO) |

### 6. 웹 어플리케이션 서버(WAS)

웹 어플리케이션 서번느 정적인 콘텐츠 처리를 하는 웹 서버와 달리 사용자의 요구에 따라 변하는 동적인 콘텐츠를 처리하기 위해 사용되는 미들웨어이다.

* 데이터 접근, 세션 관리, 트랜잭션 관리 등을 위한 라이브러리를 제공한다.
* 주로 데이터베이스 서버와 연동해서 사용한다.
* 웹 어플리케이션 서버의 종류에는 Tomcat, GlassFish, JBoss, Jetty, JEUS, Resin, WebLogic, WebSphere 등이 있다.

### 7. 웹 어플리케이션 서버(WAS) 관련 요구사항 식별 시 고려사항

웹 어플리케이션 서버(WAS)와 관련된 요구사항 식별 시 다음과 같은 사항을 고려해야 한다.

| 구분 | 내용 |
| --- | --- |
| 가용성 | \* 시스템의 장시간 운영으로 인해 발생할 수 있는 고유의 장애 발생 가능성 \* WAS의 결함 등으로 인한 패치 설치를 위한 재가동 \* 안정적인 트랜잭션 처리 \* WAS 이중화 지원 |
| 성능 | \* 대규모 트랜잭션 처리 성능 \* 다양한 설정 옵션 지원 \* 가비지 컬랙션(GC)의 다양한 옵션 |
| 기술 지원 | \* 제조업체의 안정적인 기술 지원 \* 여러 사용자들 간의 정보 공유 \* 오픈 소스 여부 |
| 구축 비용 | \* 라이선스 정책 및 비용 \* 유지관리 비용 \* 총 소유 비용(TCO) |

## 6. 요구사항 정의

## 7. 요구사항 분석

## 8. 요구사항 분석 CASE와 HIPO

## 9. UML(Unified Modeling Language)

## 10. 주요 UML 다이어그램