Software Requirements

컴퓨터공학부
200911397 송찬우
200911388 박미관
200911398 신우철

0. Contents

1. THE CONTEXT OF SOFTWARE REQUIREMENTS

- Software requirements는 Software Engineering과 떨어질 수 없는 사이.
- Requirements Engineering(RE)로 알려진 시스템 공학과정에 속해 있다.

- Software Requirements의 변경은 심해질수록 문제를 바로잡는 비용이 증가.

  -> Requirements의 오류를 바로잡는 일은 중요함

- 그러나 Requirements의 변경은 발생하게 되어 있다.

  -> 비즈니스는 진화하며, 새로운 시장이 개척되고, 운용 환경은 수시로 변하기 때문

- Requirements analysis은 요구사항분석가 또는 시스템분석가만의 전문적 역할이 아니다.

  ->비즈니스 분석가 또한 가능하다

- Requirements Engineer는 반드시 업무문제를 이해하고 솔루션의 specification을 전달할 수 있어야 한다.

1.1 Requirements Engineer's role

- Technical skills
- Knowledge with business domain understanding
- "people" skill

1.2 Requirements and Constraints

- Functional / Nonfunctional
  -> Requirements를 구분하기 위한 방법

- Nonfunctional requirements
  ->품질속성/외부 인터페이스와 같은 형태
  -> 외부 인터페이스
  : 다른 소프트웨어 컴포넌트들,하드웨어 장치들, 사용자들과의 인터페이스 등
  -> Requirements의 수행에 관한 속성을 지닌다
  : Reliability, Availability, Security, Safety, Usablitiy

- Emergent Property
  -> 분석하거나 통제하기 어려운 속성
  -> ex) 분산 시스템의 실행
  : 시스템의 디자인, 수요와 네트워크 통신

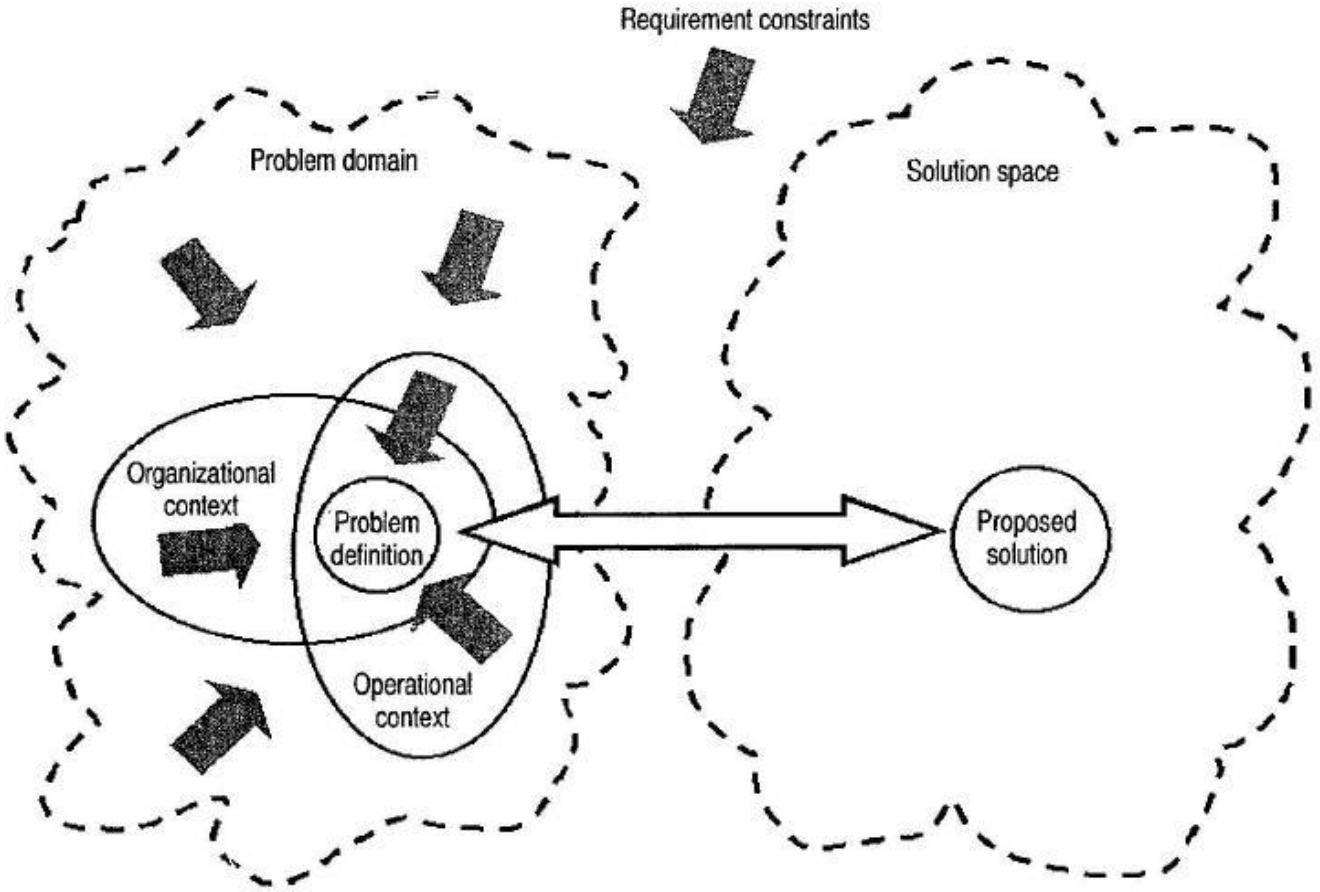

Figure 1. Requirements, constraints, problems, and solutions in RE.

- Level of Requirements

- Highest level is business problem.
  -> Last goal of system

- Next level is system requirements.
  -> Provides ability to User, Stakeholder, Other System
  -> Provides each goal of business

- Requirements Engineer는 전체 시스템의 상위 Requirement에 대한 이해로부터 특정 기능 Requirement을 생성해 낼 수도 있어야 한다.

- Software Requirements

- 소프트웨어 시스템 또는 컴포넌트가 충분히 업무를 수행해야 한다.

- ex) 가정 도난경보시스템.
  -> Software Requirements의 일부분
  -> 다른 Requirement이 센서나 사이렌과 같은 하드웨어 부품의 속성에 관해서 명시하기 때문
  -> 설치 안내와 사용자 메뉴얼에 관한 요구사항도 있을 수 있다

- 즉, 모든 요구사항이 곧 소프트웨어 요구사항이 아니다.

- Constraints

- 합당한 이유를 근거로 개발자들의 선택사항에 제한을 가하는 것으로, 여러 가지 제한이 기술적으로 많은 영향을 끼친다.

- ex) 소음 관련법.

  ->도난경보가 사이렌을 울리는 시간을 제한

2. REQUIREMENTS ENGINEERING PROCESS

- Requirements Engineering(RE) process를 통해 문제에 대한 솔루션을 찾고, 그 솔루션을 명세해야 한다.
- 소프트웨어가 가지고 있는 특성들은 도출되어야 하고, 도출된 Requirements을 분석되어야 함.
- 개발이 계속 됨에 따라, Requirement이 변경되는 것을 제어하기 위해서 Requirements을 관리해야 한다.

- IEEE가 고안한 세가지 표준

- 이 표준에 내재된 프로세스는 시스템을 살피는 것부터 시작한다. 시스템을 살핀다는 것은 다음과 같다.
  -> 시스템이 다루는 근본적인 문제를 이해하는 것
  -> 시스템의 목표를 확인하는 것
  -> 어떻게 그것이 작동하는지 윤곽을 그리는 것

- SRS는 구성요소들을 충족시키는 명확한 요구사항으로 구성되어야 하고, 개발에 착수할 수 있도록 충분히 상세해야 한다.
- 시스템 요구사항 명세로부터 시스템 아키텍쳐의 개발은 소프트웨어(그리고 다른) 구성요소들의 확인을 위해 절대적인 필수 조건이다.
- Requirement의 오류나 불충분히 이해되는 것으로 인해 뒤늦게 Requirement이 변경되는것은 심각한 위험을 초래한다. 따라서, 변경 위험을 최소화 할 수 있는 RE process가 필요.

  -> 약간의 Requirement 변화는 발생할 수 있지만 영향과 크기는 통제되어야 함
- 프로젝트들은 Requirements가 변경될 때, 적소에 그것들을 적용하게 하기 위해서 메카니즘을 가져야 한다.
- 출시 후 소프트웨어는 새로운 Requirement의 충족을 위해서 업데이트 된다.

- RE process model

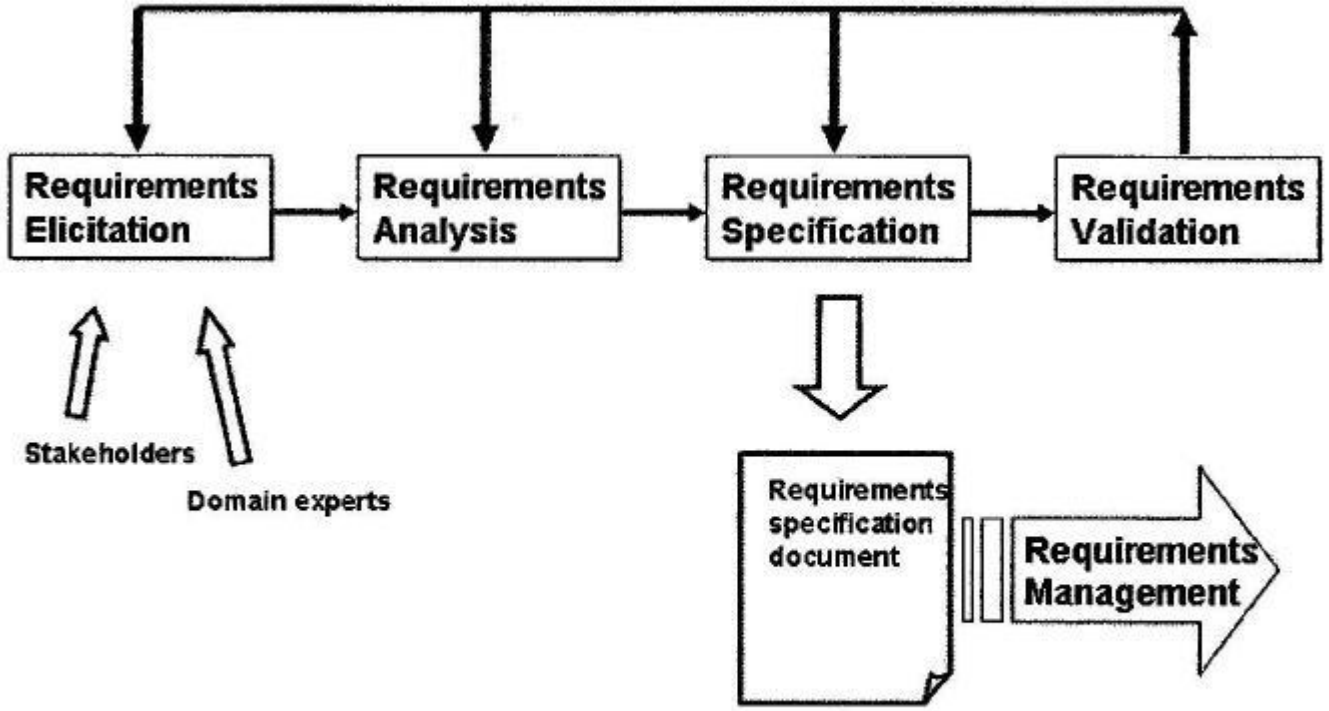

Figure 2. Generic model of the RE process.

- RE process는 비즈니스 문제에 대한 적절한 솔루션을 찾고, 명세하기 위한 것이다.
- Requirements는 발견되고, 이해되고, 기록되고, 확인되고, 의사 소통 되고, 관리 되어야 한다.
- RE는 단순히 요구사항을 기록하는 활동이 아니고, 전체 제품 수명 사이클을 지속시키는 연속적인 과정이다.
- Requirement 관리를 배분하고, Specification의 sign-off에 따른 변화에 대처하기 위한 노력이 필요하다.

3. REQUIREMENTS ELICITATION

- The process of discovering the Requirements.
- Not a linear process but requires iteration as information is collected, clarified, corrected, and reformulated.
- Need to find out the requirements actively.
- The stakeholders need to learn about what can be achieved and the relationship between their requirements and those of other stakeholders.

- Requirements have to be *discovered*, whether their sources are human, documentary, or the environmental.

  -> Another way is that requirements that will be documented in t he specification are synthesized from the elicited requirements in formation

- Elicitation the requirements to set project's goal and scope.

3.1 Requirements Sources

- Stakeholder를 식별하는 것이 중요한 첫 번째 단계.
- 프로젝트에서 수수료를 내는 고객에 맞는 시스템을 개발하는 것, 그것은 종종 프로젝트가 개발되어 제품이 시장에 판매하는 것 보다는 대표 Stakeholder를 식별하는데 쉬울 것이다.
- 범위와 Stakeholders의 관점의 한계를 인식하는 것이 중요.
- Requirements Engineer가 Requirement에 대한 걱정을 분산시킴.

  -> 불일치를 해결하고 우선순위를 적용하는 데에 도움을 준다
- Stakeholders는 단지 Requirement의 당사자가 아니다.
- ex) 임베디드 소프트웨어.

- 도메인 전문가의 역할은 이해당사자들의 업무의 암묵적인 지식들을 이끌어내도록 도와주는 것이다.

- 필요조건들을 구현하는 엔지니어들은 아마도 충분한 도메인에 대한 전문지식을 가지고 있는데 만약 그들이 이전에 어플리케이션 도메인 개발의 경험이 있다면 말이다.

3.2 Elicitation Techniques

- Stakeholders가 문제의 영역에서 어떤 역할을 하는지 이해하는 것.
- Requirements Engineer가 필요한 것은 Stakeholders가 일의 역할이 잘 묘사된 전후사정을 알려 주는 방법으로써 정보를 얻어내는 것.

- Identify contents of tasks

- 업무들은 사건들의 나열로 묘사.
  -> Preconditions
  -> Postconditions
  -> Communications of colleagues
  -> Other events that comprise the task

- Elicitation Workshops

- 중재자와 함께 Elicitation Workshop을 개최.
  -> 분석가와 고객 간의 협력을 가능하게 하는 것
  -> 사용자 요구를 파악하고 Requirements Document의 초안을 만들 수 있는 강력한 방법

- ex) Joint Requirements Planning(JRP), Joint Application Development(JAD).

- Analysis of Reliability

- 시스템에 필요한 높은 수준의 신뢰성을 위해.
- 이용 가능한 자원들이 필요한 것이 발견과 Requirement의 신뢰성의 분석에 사용.
- ex) 안전이 중요한 도메인.

4. REQUIREMENTS ANALYSIS

- Stakeholders가 Requirement를 이해하고 문제들이 있는지 검토하여 Requirement을 다듬는 것.

- Analysis는 문제와 필요한 사항에 대한 깊이 이해하고, Requirement의 비호환성, 모순과 같은 문제를 발견하고 해결할 수 있도록 해야 한다.

- Requirements Baseline

- Requirements Baseline는 프로젝트의 목표에 기여하지 않는 Requirement를 포함하지 않으면서도 반드시 완전해야 한다.

  -> 어떻게 문제가 해결될지를 명시하는데 필수적인 것을 잃어버려선 안 된다

4.1 The System Boundary

- 고안된 시스템이 프로젝트 목표에 초점을 맞추도록 돕는다.
- 시스템 경계의 바깥은 시스템에 요구사항을 강제하거나 제한하는 환경을 다룬다.
- 시스템 경계의 안쪽은 고안된 시스템이 해결책을 내는 과정에서의 문제를 다룬다.

- Use case Diagram

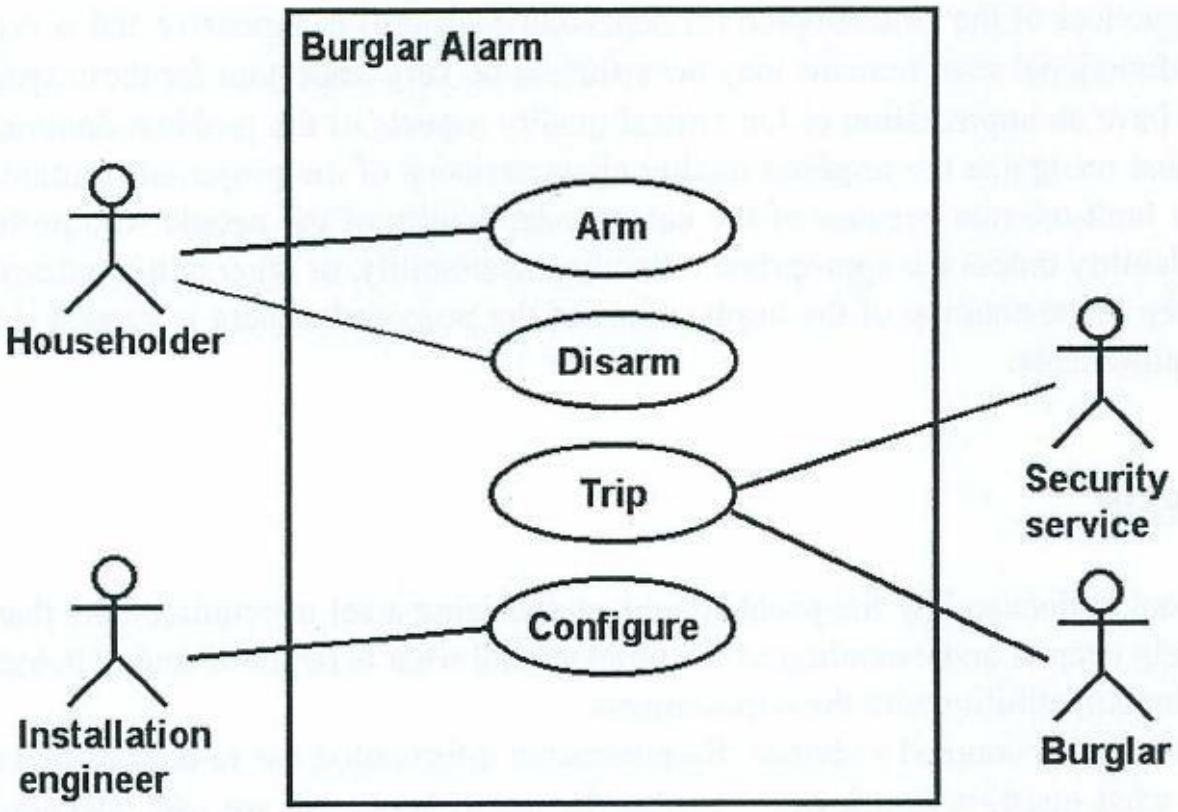

Figure 4. Use case defining the system boundary for a domestic burglar alarm.

- 시스템이 그 환경에서의 작동을 예상하여 시스템 경계를 정의하면, 프로젝트 목적의 도출,가능한 해결책 분석, 시스템 경계 정의와 같은 Concept of Operations 을 알아낼 수 있다.

  -> 미리 프로젝트의 실행가능성의 입증과 프로젝트 참가자들이 목표에 대해 확실히 이해할 수 있도록 돕는다

4.2 Requirements Modeling

- 여러 RE 프로젝트에서, 복잡한 정보를 이해하기 위해 기술자는 모델을 사용한다.
  -> 알맞은 Requirements를 발견하기 위해

- 여러 가지 많은 모델링 표기법이 있다. 일반적으로, 같은 표기법으로 문제와 해결책 모델링 모두에 쓸 수 있다.

- Model은 본문의 서술이나 구두로 전해진 문제를 다른 방식으로 묘사.
  -> Requirement 탐구를 돕는다

- Graphic Model은 복잡한 시스템의 속성을 적당한 개념을 사용하여 머릿속에 그려볼 수 있도록 돕는다.

- Z나 CSP와 같은 공식표기법은 분석을 반드시 엄격히 하도록 한다.
- 공식 모델들은 형식에 따른 추론을 다루기 쉽게 하고 일부 입증된 시스템모델을 사용할 수 있게 한다.
- 때때로 시스템의 동적인 행동은 정적인 모델로 충분히 분석되지 못한다.
- 만약 불확실하게 시스템의 동적 행동이 일어날 가능성이 있다면, 요구 과정에서 반드시 미리 시뮬레이션을 하라.

4.3 Derived Requirements

- System Requirements의 비용과 기술적인 영향은 불확실한 경우가 많으며, 그것은 평가나 입증을 어렵게 한다.
  -> 세부적인 새로운 Requirements를 끌어냄으로써 System Requirements를 정교하게 한다

- 모델링은 필요한 세부사항을 알아내기 위해 요구사항을 이끌어내는데 도움이 된다.
  -> ex) 도난경보

- Scenario of Alarm

- Use Case Model에서, 추가적인 Requirements는 경보 시나리오를 쓰는 것으로 자연적으로 나오게 된다.
  -> 전제 조건: 경보가 울리지 않는다
  -> 세대주는 경보 절차를 시작한다
  -> 초읽기가 시작된다
  -> 세대주는 전제를 나간다
  -> 초읽기가 끝난다
  -> 경보 절차가 완료된다
  -> 결과 : 정보가 울린다

- 더 상세한 Requirements 도출을 위해 Requirements 사이의 밀접한 관계를 이해해야 한다.

- Requirements는 소프트웨어 중심적이고 기술적이어야 한다.
  -> ex) 기계적인 잠금 장치의 실용성과 비용에 대한 인식

- Requirement는 Functional Requirement 뿐만 아니라, Nonfunctional Requirement 에서도 도출된다.

- Requirements engineer은 적당한 측정 규준(e.g., 실패에 대한 시간)을 선택하고 어떻게 시스템이 이 측정 규준에 기준하여 기록되야 하는지 명세해야 한다.

- Requirements는 항상 하향식 방식으로 도출되지 않는다.

- 요구사항이 충분히 구체적 일 때, 개발을 시작해야 한다.
- 요구사항 관리의 토대인 요구사항 도출에 관련된 중요한 housekeeping 활동이 있다.
- 프로젝트의 코스에 걸쳐서, 시스템과 추적될 수 있는 도출된 요구사항들 사이의 관계를 허가하는 완벽한 도출 관계가 구성되어야 한다.

4.4 Requirements Attributes

- **Identifier(식별자)**: 모든 Requriements은 그것이 참조되게 하는 유일한 식별자로 부여되어야 한다.
- **Source(근원)**: Requriement이 도출된 곳. 예로는, stakeholder와 그것이 파생된 더 높은 단계의 Requriements
- **Date(날짜)**: Requriement가 공식화 될 때의 날짜.
- **Rationale(근거)**: Requriement의 목적을 나타내는 속성.

- **Type(형태)**: 예를 들어, Requriement이 기능적인지 비기능적인지; 사용자 인터페이스 Requriement인지, 안전한 Requriement인지 등을 나타냄.

- **Priority(우선권)**: 많은 요구들과 제한된 예산이 있을 때, 무엇을 우선시 해야 되는지에 대한 속성

- **Stability(안전성)**: Requirements가 변경될 가능성이 있는 지에 대한 내용

- **Verification procedure(검증 절차)**: Requirement가 만족되었는지를 검증하는 절차

- **Status(상태)**: Requirement가 수명 싸이클에서 현재 위치가 어디인지를 나타냄.

4.5 Requirement Trade-offs

- Stakeholders가 필요한 것은 Requirements의 Baseline들이 어떤 Requirements 이 목표를 달성할 것이지만 다른조건들은 그러지 못할 것이라는 것을 받아들이고 인정하는 것.

- Stakeholders가 이러한 갈등들을 알게 해야하고 필수적인 균형들을 명쾌하게 포함해야 한다.

- 규모가 큰 프로젝트들에서 유익하고 필수적인 trade-off들은 조직적인 분석방법이 필요하다.
- Stakeholders가 Elicitation Workshop들에 참여해 있다면, 의견 일치를 보는 과정은 쉬워진다.
- 만약 아니라면, Workshop들에서 분명하게 의견 일치를 보는 목표를 잡고, 수긍할만한 기준점을 파생시키는 것이 필수적이다.

5. SOFTWARE REQUIREMENTS SPECIFICATION

- Software Requirements를 기록.

- 소프트웨어로써 필요한 조건들의 설명서(SRS)가 명시하는 것은 소프트웨어의 요소나 하위 시스템들의 Requirement이다.
  -> 소프트웨어 시스템이 제공해야 하는 기능과 능력 그리고 따라야 하는 제약조건을 선언

- Requirements가 영어와 같은 자연어로 쓰여져 있어서 개발자들과 Stakeholders가 이해할 수 있도록 한다.
  -> Z 그리고 CSP가 정확한 묘사가 가능하다

- SRS 관리는 간결하고 정확한 방법으로 그리고 합리적인 해석으로만 Requirements를 적는 데 필요하다.

- Requirement Engineer는 명확하게 가장 일관되게 읽기 쉬우며 설명서의 전체적인 일관성 있는 설명으로 된 형태를 찾아야 한다.

  -> 파생된 원문의 Requirements가 다른 형식적이나 도표로 된 묘사 내용들을 뜻한다

6. REQUIREMENTS VALIDATION

- 요구명세서에서 명세된 요구들은 반드시 근본적인 업무문제를 해결해야 하고 다양한 제한을 정확히 반영해야 한다.
- 단지 개별적인 요구들의 정확함 뿐만 아니라, 전체의 정확함, 완전함, 일관성을 모두 따져야 한다.
- 요구사항은 또한 가독성, 유지성, 일관성 그리고 다른 중요한 특징들을 보장하기 위한 적절한 표준과 지침을 따라야 한다.
- 검증은 항상 요구명세문서나 원고의 마지막에 실시된다.그러나, 요구의 정확성에 관한 비공식적인 입증은 분석과 도출과정 중에도 이뤄진다.

- 명확히 과정의 후반까지 입증을 미루는 것은 어리석다는 것이다.
- 일단 검증되고, 필요한 변경을 하면, 요구명세는 “sign-off"가 된다.
- 그 효과가 디자인과 실행 기반의 문서에서 두드러지면서, 문서와 요구 모두 공식적인 변화와 버전 통제의 대상이 되었다.
- 그러나 꼭 모든 프로젝트가 그러지는 않으며, 때때로 명세의 sign-off 전에 개발을 시작하는 경우도 있다.
- 대부분, 요구는 정적으로 검증된다.
- 일부 복잡하고 동적인 행동이 명세 되는 경우, 요구는 프로토타입이나 시뮬레이션을 사용하여 동적으로 입증될 필요가 있다.

- 그러나 그것들은 보통 비싸고, 훌륭한 프로젝트는 원고명세서류의 발행 이전에 이런 입증의 필요성을 잘 예측한다.
- 요구검토는 모든 경우에서 요구를 검증하는 메커니즘이 될 것이다.
- 검토 패널의 구성은 중요하고 개발자와 관계자의 대리인을 반드시 포함해야 한다.
- 어떤 경우, 이미 개발된 시스템의 요구의 준수여부를 검증해야 할 것이다.
- 안정성과 신뢰성과 같은 일부 비실용적 요구들은 입증이 어렵겠지만, 어떤 유용한 목적을 제공한다면 반드시 입증이 가능해야 한다.

7. REQUIREMENTS MANAGEMENTS

- Weiger는 4가지로 구성된 요구관리를 규정하였다
  -> 변경 관리, 버전관리, 요구사항추적, 상태추적

- 요구관리는 중요하지만 종종 도외시된다.

- 1990년대 초반 SW-CMM(소프트웨어 역량 성숙도 모델)에 의해 산업 요구관리가 크게 증가.
  -> 모든 개발조직이 공식적인 요구관리를 위하여 최소한의 효과적인 통제와 운영을 원했기 때문

요구관리의 큰 부분을 이룰 정리작업을 쉽게 해주는 요구관리 도구가 주목을 받을 것.

7.1 Change Control

- SRS이 끝난 이후에라도 변경(요구의 생성,소멸, 새로운 요구의 출현,요구의 우선사항변경)은 발생한다.

-> 변화가 관리 없이 발생하지 못하도록 하는 것이 중요

- Requirements Creep

- 소프트웨어 프로젝트의 유명한 현상.
- 통제되지 않거나 갑자기 생긴 요구변화를 관리하기 불가능한 프로젝트 계획을 만든다면, 필연적으로 시스템은 늦어지고 예산초과가 발생한다는 것.

- 요구변경 요청에서 제안된 변경의 장점과 단점을 신중히 평가되어야 한다.
- 평가의 결과는 변화에 대한 동의/반대 의견으로 바로 결정될 수도 있고 어쩌면 이후의 소프트웨어 배포 때까지 미뤄질 수도 있다.
- 변경관리에는 비용과 이익 그리고 변경요청을 심사할 패널을 선정하는데 필요한 정보를 얻기 위한 공식적인 프로세스가 있어야 한다.
- 프로젝트 스케쥴과 비용 변경과 같은 부분은 패널에 의해 고려되어야만 한다.

7.2 Version Control

- 변경된 날짜, 사람, 이유를 포함하고 있어야 하면서, 요구사항 문서에 기록되어야 한다.
- 이런 버전 문서에는 번호를 붙이는 시스템이 필요하다.
- 문서에 기록된 내용들은 개발 팀과 의사소통 되고, 관리 도구를 통해 데이터베이스에 통합되어 새로운 버전으로 만들어 진다.

7.3 Requirements Tracing

- 개별적인 요구사항과 다른 시스템 요소 간의 종속관계와 그것들의 논리적인 연결을 문서로 정리하는 것.

- 요구사항이 변경되는 것을 관리하고 상태 추적, 좋은 요구사항 관리를 위해 필요함.

- 요구사항들은 양방으로 추적 가능해야 한다.

->전 방위 추적: 요구사항이 아래 단계에 어떤 영향을 미치는지 확인하기 위해

->후 방위 추적: 요소에서 어느 항목이 왜 만들어졌는지 알기 위해서

7.4 Status Tracking

- 요구사항의 수행과 처리에 있어서 요구사항 상태에 대한 정보를 확인하는 것이다.
- 요구사항이 승인됐는지, 구현됐는지, 미결인지, 삭제됐는지, 연기됐는지 등에 관한 정보가 있다.
- 프로젝트 진행 과정을 추적하고, 요구사항의 수행과 스케줄링을 위해서 요구사항의 상태에 대한 기록을 관리하는 것은 중요하다.

8. SUMMARY

- 소프트웨어 요구사항은 실제 비즈니스 문제에서 발생.
- RE는 어떤 시스템이나 소프트웨어 공학 프로세스에서 결정적인 부분이며, 성공적인 프로젝트를 위해 기초적인 것이다. RE가 제대로 되지 않으면, 프로젝트는 실패할 것이다.
- XP와 같은 개발 방법이 있지만, 소프트웨어 명세에 정의된 기록으로 SRS를 도출해내고 관리하기 위해서 인정된 RE process가 필요하다.
- 성공적인 프로젝트를 위해서 요구사항들을 잘 도출해내고, 분석하고, 명세하고, 유효화하고 관리하기 위한 요구가 필요하다.
- 소프트웨어 전문가들과 그들의 관리자들이 RE에 대한 기본 내용들을 인지하는것이 필수적이다.