소프트웨어 공학개론

Introduction to Software Engineering

2022-1학기 (Spring)
선문대학교 AI소프트웨어학과

3월 8일(화)

강의 목차

Part I. 개념

- 제1장 대규모 소프트웨어개발 과제
- 연습 문제 #1

Part II. 활용

Part III. 실습

제1장 **대규모 소프트웨어개발 과제**

- 1절 소프트웨어위기
- 2절 대규모 소프트웨어에 수반되는 문제점들
- 3절 문제해결을 위한 대처 방안
- 연습문제 #1

1. 대규모 소프트웨어개발 과제

개요

- 일반적인 시스템은 소프트웨어에 의해 제어되고 있음
  - 컴퓨터의 운영체제
  - 은행의 ATM
- 소프트웨어 내부 코드 라인 규모
  - 수백, 수만 라인 (책에서 언급)
    - 참고: 현재에는 수천만 라인 이상으로서 지속적으로 코드 규모와 개발 인력이 증가하고 있음
- 간단하게 구축할 수 있는 것이 아님

1. 대규모 소프트웨어개발 과제

개요

- 이와 같은 대규모 소프트웨어를 구축할 수 있게 된 배경에는,

  하드웨어의 진화만으로 설명하기 어려운,

  오랜기간동안의 고난의 대규모 소프트웨어개발의 역사가 있음

- 제1장에서 "대규모 소프트웨어개발의 과제"를 생각해 보기 위해서 살펴볼 내용
  - 소프트웨어공학의 역사
  - 소프트웨어위기
  - 소프트웨어위기에서 드러난 문제점 고찰
  - 대규모 소프트웨어개발의 과제와 소프트웨어공학의 대처방안 소개

1.1 소프트웨어위기

소프트웨어공학의 역사 (수작업 → 컴퓨터 기반 업무)

- 1951년에 세계 최초의 상용 컴퓨터 UNIVAC I가 탄생한 이후, 사무계산을 비롯하여 당시에 수작업에 의존했었던 다양한 업무에 컴퓨터가 사용되기 시작 (참고: Bill Gates' Microsoft Excel 1987, GW-Basic)
  - 하드웨어: "노이먼방식" 이라는 내장 프로그램에 의해 제어하는 공통적인 방식을 채용하였고, 논리소자가 진공관, 트랜지스터, IC(집적회로), LSI(대규모집적회로) 등과 같이 진화, 소형화/고속화
  - 소프트웨어: 간단한 사무처리프로그램 및 탄도계산프로그램으로부터 렌즈의 설계, 과학기술계산, 고급언어처리, 운영체제시스템, 좌석예약, 은행온라인시스템, 우주비행프로그램 등, 다양화/대규모화

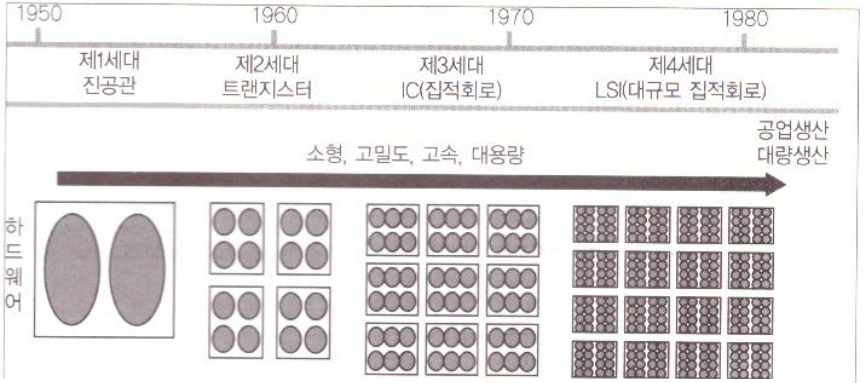


그림 1.1 | 상용 컴퓨터의 진화와 소프트웨어위기

1.1 소프트웨어위기

소프트웨어공학의 역사 (하드웨어의 기술진보가 소프트웨어를 추월)

- 당시에 논리소자를 중심으로 고성능 컴퓨터를 저렴한 가격에 개발하는 하드웨어기술이 눈부신 발전을 이룩한 것에 비해서, 막대한 프로그램은 대량으로 공급하기 위한 소프트웨어기술과 개발체제가 하드웨어기술의 진보를 따라갈 수 없었다.
  - 따라서, 1960년대 후반부터 "소프트웨어위기"라는 위기상황에 빠지게 되었다.

1.1 소프트웨어위기

소프트웨어공학의 역사 (컴퓨터 수요증가 → 소프트웨어위기 발생)

- 소프트웨어위기 상황에 이르게 된 **경위**
  - 1950년대 후반에 진공관을 대신해서 트랜지스터가 사용되기 시작하여, 하드웨어는 소형화/고속화됨
  - 1964년에는 IC(Integrated Circuit)가 사용되기 시작
    - 하드웨어의 소형화/고속화는 더욱 진화 및 대형 프로그램 처리 가능
    - 그 결과, 컴퓨터의 수요가 증가
      - 소프트웨어의 규모가 급속히 확대되었지만, 개발기한에 맞추지 못하고 개발일정이 자주 지연됨
      - 소프트웨어의 품질이 저하 및 사회문제 대두 (참고: Telco의 서비스 일시적/간헐적 장애)
      - 소프트웨어개발과 보수에 필요한 비용 증가 (참고: 연간 인건비 수억원 → 여전히 개발중)
  - 1968년 NATO 주관 국제회의 개최
    - 이러한 상황을 "소프트웨어위기"라고 명명
    - 이를 대처하기 위해 "소프트웨어공학"의 연구개발 분위기가 높아짐

1.1 소프트웨어위기

소프트웨어위기 원인 (서비스의 다양화 요구)

- 고성능 컴퓨터가 널리 사용되기 시작했을 때, "하드웨어위기"가 아닌 "소프트웨어위기"가 발생한 이유
  - 컴퓨터의 수요 증가와 함께 서비스의 다양화 요구
    - 같은 서비스를 제공하는 컴퓨터가 대량으로 필요하게 되었다면,
      - 하드웨어보다는 소프트웨어 쪽이 간단하게 복제품을 만들 수 있으므로, 오히려 하드웨어의 생산이 기한을 맞출 수 없게 될 것이다.
    - 다양한 서비스가 필요한 경우
      - 각 서비스마다 소프트웨어를 설계개발해야 하므로, 소프트웨어의 설계개발기술자가 부족하게 되어 개발이 기한을 맞출 수 없게 되는 상황에 빠져들게 되는 것이다.

  - 참고: 하드웨어마다 필요한 소프트웨어가 달라질 수 있음
    - 다양한 CPU 아키텍처 (https://github.com/qemu/qemu/tree/master/target)와 운영체제
      - Intel/AMD x86/x64 기반 IBM PC용 Windows와 서버용 Windows Server
      - PocketPC와 Windows CE
      - PowerPC, ARM기반 Macintosh와 MacOS
      - ARM기반 Cellphone/Tablet용 Android, iOS
      - 다양한 CPU를 지원하는 Linux (CentOS, Fedora, RHEL, Ubuntu 등)

1.1 소프트웨어위기

소프트웨어위기 원인 (정리)

- 하드웨어개발
  - 대량생산에 성공

- 소프트웨어개발
  - 가내수공업적인 형태
  - 다품종/대규모 소프트웨어의 개발요구에 대응 불가

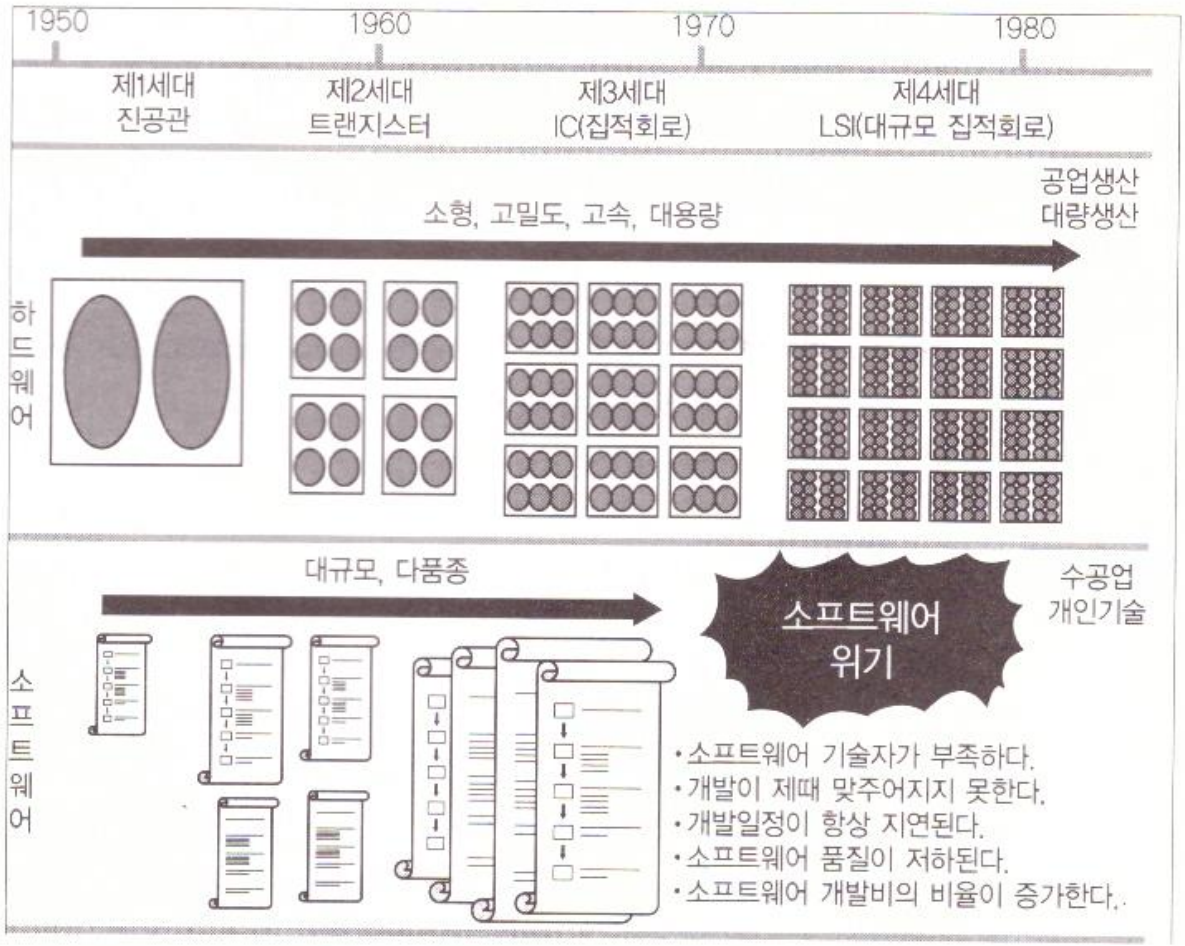


그림 1.1 상용 컴퓨터의 진화와 소프트웨어위기

1.2 대규모 소프트웨어에 수반되는 문제점들

개요

- 소프트웨어의 규모가 커지면, 은행이나 증권거래소에서 소프트웨어 오작동에 의한 시스템다운현상과 같은 사회문제를 발생시키는 경우가 있다.
  - 대규모 소프트웨어의 개발에서 발생하는 문제점들을 이해하기 위해서는,
    프로그램의 규모가 커지면 무엇이 변화하는지를 다양한 측면에서 생각해 볼 필요가 있다.
  - 프로그램의 대규모화에 따라서 확대되는 제반 문제점들
    - (1) 생산물
    - (2) 사람
    - (3) 시간
    - (4) 역할

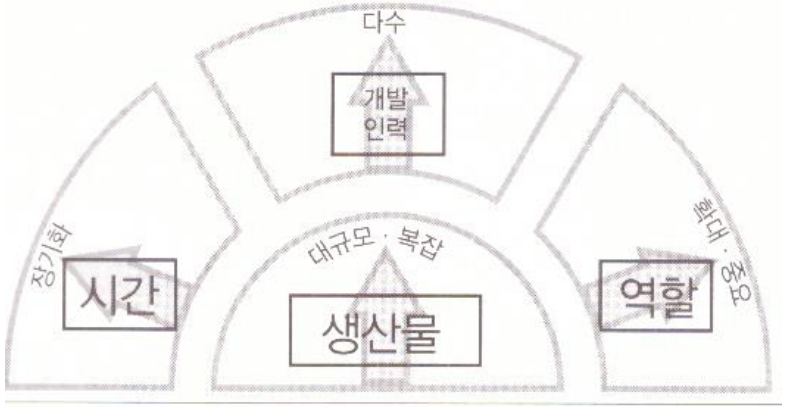


그림 1.2 | 프로그램의 대규모화에 따라서 확대되는 제반 문제점들

1.2 대규모 소프트웨어에 수반되는 문제점들

개요

- 프로그램의 대규모화에 따라서 확대되는 제반 문제점들
  - (1) 생산물
    - 프로그램이 길어짐에 따른 변수 또는 함수의 개수 증가
    - 구현 기능의 증가 및 기능들을 이해하기 위해 필요한 문서 증가
  - (2) 사람
    - 프로그램이 길어짐에 따른 개발 관련 인력 및 상호협력 요구 증가
    - 프로그램의 이용자 증가 (참고: 유지보수성의 어려움)

1.2 대규모 소프트웨어에 수반되는 문제점들

개요

- 프로그램의 대규모화에 따라서 확대되는 제반 문제점들
  - (3) 시간
    - 프로그램이 길어짐에 따른 개발 기간 및 이용 기간 증가
    - 소프트웨어의 생명 주기 증가
      - 프로그램을 개발하려고 생각하기 시작해서 설계개발을 거쳐서 실제로 이용되고, 결국 아무도 사용하지 않게 될 때까지의 기간이 장기화된다.
  - (4) 역할
    - 대규모 프로그램을 실행시킬 수 있게 되면, 컴퓨터를 이용한 활용 범위와 대상 증가
    - 컴퓨터에 대한 사회적 기대 상승, 컴퓨터의 역할 확대, 사회적 중요도 상승
      - 컴퓨터가 사회의 중심에서 사용되게 된다.

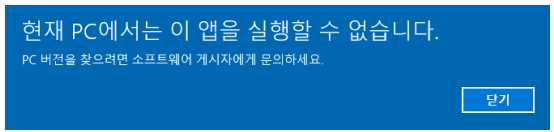
1.2 대규모 소프트웨어에 수반되는 문제점들

목차

1.2 대규모 소프트웨어에 수반되는 문제점들

1.2.1 생산물의 증가에 의해 발생되는 문제점

- (요구사항에 따라 개발되었으므로) 소프트웨어는 취급할 수 있는 프로그램의 크기와 데이터량에 제약 있음
  - 프로그램의 크기 제약 예:
    - 실행가능한 압축해제파일이 4GB를 초과한 경우 (PE32헤더, 2^32=4G)
    - 임베디드 시스템의 경우 업로드할 수 있는 부팅용 커널 이미지의 크기에 제약이 있음
  - 데이터량 제약 예: 1MB 수준의 데이터 파일만을 고려하여 설계된 프로그램에서 1GB 수준의 파일을 불러올 때 응답이 없는 경우가 발생
- 프로그램이 커지면 지금까지 사용해왔던 도구를 (있는 그대로) 이용할 수 없게 되는 경우가 있다. (재이용불가)
- (코드 크기 증가로 인해) 프로그램의 분석에 필요한 시간과 실행시간 증가로 인한 요구된 시간내 종료의 어려움
- (코드 크기 증가로 인해) 실행시간 및 메모리량에 관한 문제는, 하드웨어가 대형화 및 고속화됨에 따라서 경감되어왔지만, 하드웨어의 진보만으로는 대응할 수 없는 문제 존재



EXE파일의 크기가 4GB를 초과한 경우

1.2 대규모 소프트웨어에 수반되는 문제점들

1.2.1 생산물의 증가에 의해 발생되는 문제점

- 코드 가독성 저하
  - 프로그램이 커지면 복잡하게 되고 전체적인 모습이 나빠지게 된다.
- 오류 발견 어려움
  - 상세한 부분까지 파악하기 어려워지므로, 프로그램을 분석해서 오류를 찾아내는 것이 어려워지다.
- 기능간 의존성 증가 (예: 약결합과 밀결합)
  - 프로그램이 커지게 되면 기능도 늘어나서 각 기능들이 서로 영향을 미치는 경우도 발생한다.
    - 여러 프로그램들의 상호관계를 올바르게 유지하고 프로그램과 문서와의 관계를 유지하는 작업 등이 어려워진다.
    - 특히, 프로그램을 수정했을 때 발생하는 영향을 상세하게 인지할 수 있도록, 프로그램과 문서를 파악해 두는 작업이 어렵게 된다.

1.2 대규모 소프트웨어에 수반되는 문제점들

1.2.2 인원 증가에 의해 발생되는 문제점

- 대규모 소프트웨어에 관여하고 있는 사람들
  - 소프트웨어가 필요해서 개발을 의뢰: 소프트웨어 이용자와 구입 고객
  - 개발의뢰에 대응하여 소프트웨어개발에 투입
    - 프로젝트관리자: 개발 지휘
    - 설계담당자: 소프트웨어 설계 담당
    - 개발담당자: 설계결과를 토대로 개발 진행
    - 운용개발자: 소프트웨어를 운용해서 서비스를 제공
    - 유지보수담당자: 변경 및 유지관리를 담당
  - 이 외 담당자들
    - 판매담당자: 소프트웨어를 공급, 판로 개척
    - 기획담당자: 새로운 소프트웨어의 개발과 판매를 기획

그림 1.3 | 관여자: 대규모 소프트웨어에 관여하고 있는 사람들

1.2 대규모 소프트웨어에 수반되는 문제점들

1.2.2 인원 증가에 의해 발생되는 문제점

- 소프트웨어 개발과 그에 따른 성과물에 관련있는 모든 사람들을 관여자(StakeHolder)라고 부른다.
  - 소규모 소프트웨어에서는, 고객과 이용자가 동일 인물인 경우와, 설계담당자, 개발담당자, 유지보수담당자 등이 동일한 경우가 있다.
  - 소프트웨어를 스스로 만들어서 사용하는 경우에는, 관여자들이 모두 동일한 인물이 된다.
- 소프트웨어의 규모가 커지면 관여자 숫자가 증가하여, 관여자들간의 조정이 중요시 됨
  - 관여자들은 각자 관련있는 소프트웨어의 성질에 따라 서로 다른 관점을 갖는다.

표 1.1 | 관여자들의 서로 다른 관점

관여자	관점
이용자	필요한 기능이 있는가/사용하기 수월한가/빠른가/저렴한가/하드웨어에 대한 제약이 적은가
기획담당자	높은 수익이 예상되는가/적용영역이 넓은가/다수의 이용자들에게 납득될 수 있는가
개발담당자	단기간내에 개발가능한가/이전에 개발한 프로그램을 재이용가능한가/자신의 기술을 활용가능한가
운용담당자	조작이 간단한가/서비스 제공할 때 문제가 덜 발생하는가
유지보수담당자	문제가 발생했을 때 원인규명과 복구가 용이한가/변경이 용이한가

1.2 대규모 소프트웨어에 수반되는 문제점들

1.2.2 인원 증가에 의해 발생되는 문제점

- 관여자들은 서로 의사소통을 충분히 수행하지 않으면 오해가 발생됨
  - 앞서 살펴본 담당업무에 따른 관점들의 차이
  - 일상적으로 사용하는 용어들과 경험의 차이

- 같은 개발담당자라고 하더라도 인원수 증가에 따른 담당업무의 세분화로 인한 담당자들간 연계업무 증가
  - 연계업무 수행을 위한 자료 부족 및 애매모호하게 작성된 자료
    - 해당 업무를 충분하게 연계할 수 없음
    - 오해가 발생되었음을 나중에 되늦게 발견

- 개발이 진행된 후에 문제점을 발견한 경우에는, 초기단계로 거슬러 올라가서 다시 수정하게 된다. 이와 같은 수정업무를 "재작업"이라고 부른다.
  - 재작업이 커지면 커질수록, 즉, 문제의 발견이 늦어질수록, 재수정 작업량이 늘어난다.
    - 개발이 지연되어 개발비용이 늘어나서 문제가 심각해진다.

1.2 대규모 소프트웨어에 수반되는 문제점들

1.2.3 개발 기간 장기화에 의해 발생되는 문제점

- 개발기간이 길어지면 고객의 요구사항 및 각 담당자들 등, 주변환경이 변화
  - 이용자와 고객이 당초에 생각하지 않았던 기능을 개발진행 중에 추가해달라고 요청
    - 요구사항을 변경하게 되면 재수정 작업이 필요하며, 재수정작업의 규모에 따라 개발비용 증가
  - 개발담당자가 도중에 교체되는 경우 개발기간이 길어지게 되는 결과를 초래
    - 담당자간 인수인계에 필요한 시간
    - 새로운 담당자가 업무에 익숙해질 때까지의 시간이 필요

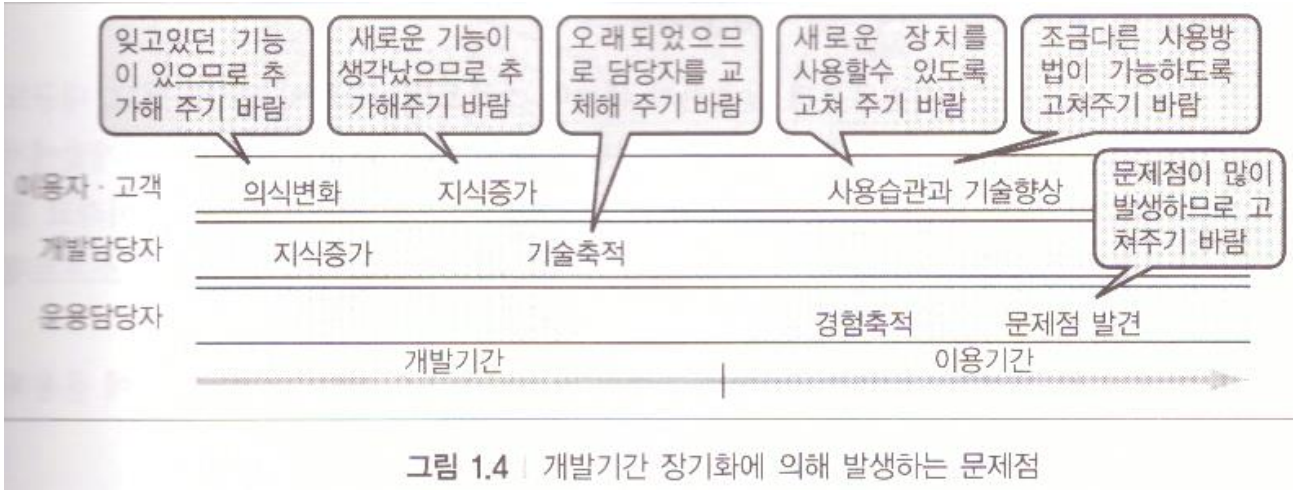


그림 1.4 | 개발기간 장기화에 의해 발생하는 문제점

1.2 대규모 소프트웨어에 수반되는 문제점들

1.2.3 개발 기간 장기화에 의해 발생되는 문제점

- 이용기간이 길어지면, 환경변화에 의해 새로운 요구사항이 발생하여 변경이 필요
- 신기술의 등장에 의해, 구현방식 및 이용형태의 변경이 요구됨
  - 이미 개발되어 이용하고 있는 소프트웨어에 대한 개조 필요
    - 개조에 필요한 자료가 준비되어 있지 않으면, 이용중에 소프트웨어 분석에 막대한 시간이 투입됨
  - 개조요구에 부응하지 못해서 사실상 신규개발이 되어버리는 경우도 발생

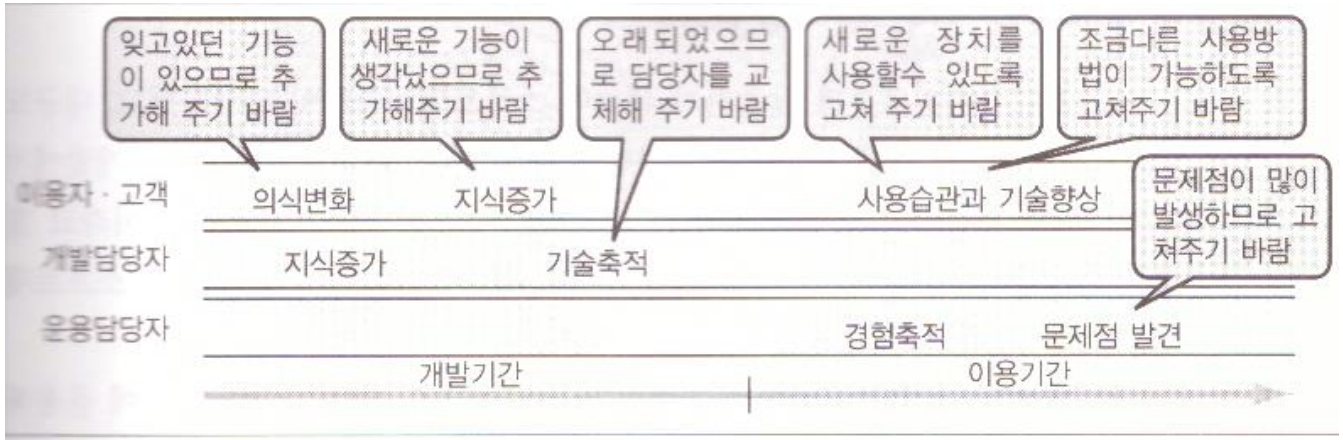


그림 1.4 | 개발기간 장기화에 의해 발생하는 문제점

1.2 대규모 소프트웨어에 수반되는 문제점들

1.2.4 사회적 역할 변화에 의해 발생되는 문제점

- 고성능 하드웨어로 대규모 소프트웨어를 이용할 수 있게 되면, 컴퓨터의 응용영역이 넓어져서 지금까지 수작업에 의존했었던 다양한 업무에 컴퓨터를 이용할 수 있게 됨
  - (일본) 1960년대에는 은행 창구업무 및 철도 좌석예약업무 등과 같은 전국온라인시스템이 구축됨.
    - 컴퓨터가 사회적으로 중요한 역할을 수행하는 환경이 조성되었지만,
    - 작은 문제점도 사회적으로 커다란 영향을 끼치게 됨
      - 소프트웨어의 신뢰성 향상이 필수불가결하게 되었고, 시스템의 품질기준이 높아짐

1.2.4 사회적 역할 변화에 의해 발생되는 문제점

- 소프트웨어의 신뢰성 및 시스템의 품질기준에 부응하기 위하여,
  - 개발기간 및 테스트기간이 길어지면, 개발비용 증가 문제점 발생
  - 시스템 운용을 시작한 이후, 기능 변경 및 확장에 따라 신뢰성 저하 우려로 인해 운용개시후의 수정이 어려움
  - 오류 발생으로 인하여 시스템 전체를 정지할 수 없거나, 정지시키기 위해 허용되는 시간이 지극히 짧은 문제점을 내포하고 있는 시스템도 존재함
    - 예를들면, 은행업무 온라인시스템에서 오류가 발생한 경우에는, 시스템을 정지시키게 되면 경제적으로 혼란을 불러일으킴
    - 오류발생에 의한 영향을 국소화시키거나, 단기간내에 오류를 찾아내서 복구할 수 있도록 개발할 필요가 있음
  - 소프트웨어가 사회활동에 없어서는 안되는 환경하에서는, 사회의 새로운 체제가 도입되거나 새로운 장치가 만들어진 경우에는, 이를 지원하기 위한 소프트웨어를 즉각적으로 개발하여 이용할 수 있도록 요구됨
  - 그 결과, 다양한 종류의 소프트웨어를 신속하게 개발해 달라는 요청이 끊이지 않게 됨

1.3 문제 해결을 위한 대처 방안

개요

- 소프트웨어위기로부터 탈출하기 위하여 소프트웨어공학이 탄생
- 대규모 소프트웨어 개발기술이 등장하여 발전
- 대규모 소프트웨어개발에서 발생되는 문제점들을 해결하기 위한 대처방안이, 곧, "소프트웨어공학"
- IEEE표준화위원회의 용어집(IEEE Std 610.12-1990)
  - 소프트웨어공학 정의
    - 1. 체계적, 학문적, 정량적 기법을 소프트웨어의 개발, 운용, 보수에 응용하는 것, 즉, 소프트웨어에 대한 공학의 적용이다.
    - 2. 위의 (1)에 대한 기법 연구이다.
- 앞에서 살펴본 제반 문제점들에 대한 대처방안 소개
  - 6가지 관점에서 소프트웨어공학의 다양한 기술과 도구들에 공통기반이 되는 개념

1.3 문제 해결을 위한 대처 방안

목차 (6가지 관점)

1.3 문제 해결을 위한 대처 방안

선문대학교

1.3.1 분할통치와 구조화

- 분할통치: 영토가 넓은 큰 나라에 대해 여러 개의 작은 규모의 지역들로 분할하여, 각각의 지역에 권한을 위임한 대리인을 파견하여 통치
  - 소프트웨어의 경우 서로 독립성이 높은 문제들로 분할해서 각 문제들을 해결 필요
- 그대로는 다루기 어려운 커다란 문제를 작은 규모의 문제들로 분할하고, 각각의 작은 문제들에 대한 해를 한데 모아서 원래의 문제에 대한 해를 구하는 방법을 "분할통치법"이라고 부른다.
  - 분할통치법을 토대로하는 문제해결알고리즘과 소프트웨어개발 방법론은 오래전부터 여러가지가 제안되고 있다.

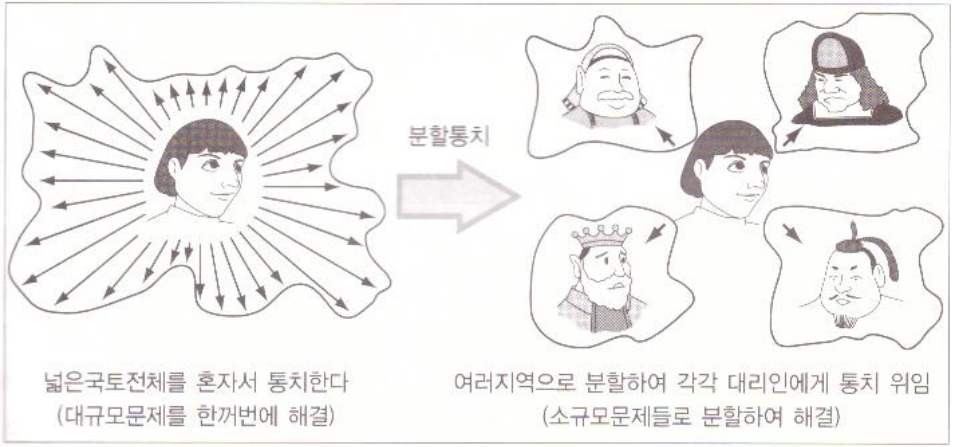


그림 1.5 | 분할통치

1.3 문제 해결을 위한 대처 방안

1.3.1 분할통치와 구조화

- 서로 얽혀있는 복잡한 구조
  - 얽혀있는 것을 풀어서 상세하게 나누어서 생각해보면 구조를 이해할 수 있게 됨
  - 대규모 문제를 여러개의 작은 요소들로 분할한 후에, 각각의 요소들 사이의 관계를 이해하기 쉽게 정리하는 것을 "구조화"라고 부른다.

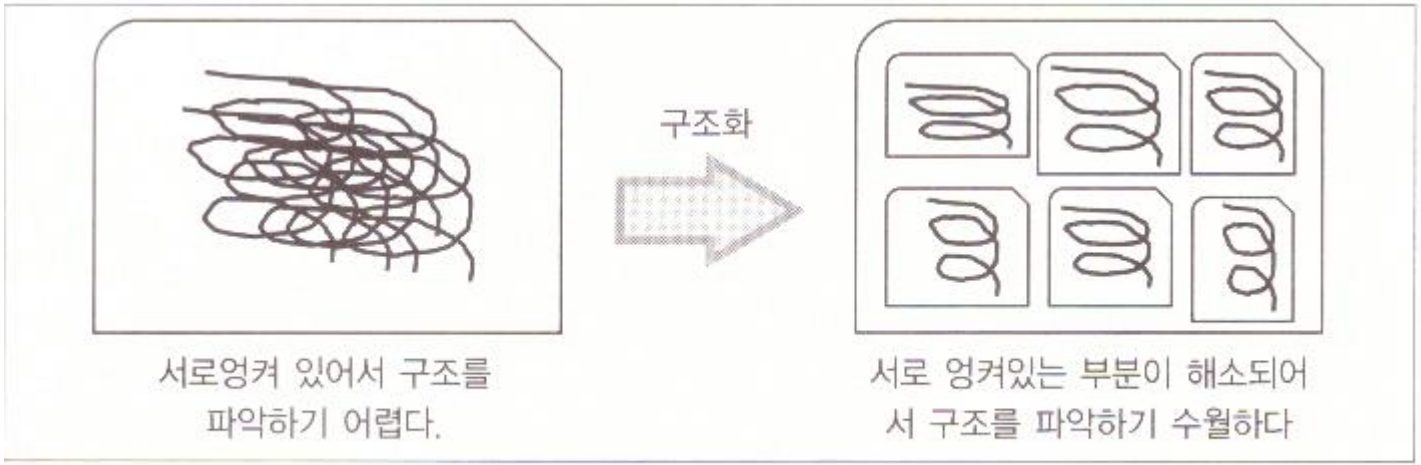


그림 1.6 | 구조화

1.3 문제 해결을 위한 대처 방안

1.3.2 추상화와 모델링

- 분할통치 및 구조화와 함께, 대규모의 복잡한 문제를 이해하기 쉽고 전체적으로 파악하기 수월하도록 지원하는 개념으로서 "추상화"가 있다.
  - 서로다른 모양을 갖는 다양한 집들에 대해서, 지붕, 벽, 현관, 창문 등과 같이 여러 가지 요소들로 나누어보면 공통적인 부분이 있음을 알 수 있다.
  - 추상화란, 대상이 되는 사물로부터 본질적이지 않은 부분은 제거하고, 몇가지 사물에 공통적인 부분을 추출하는 것이다.
  - 소프트웨어에 대한 추상화에서는, 검토해야할 항목들에 관련있는 주요한 부분을 추출해서 표현한다. 이와같이 표현한 것을 "모델(Model)"이라고 부르며, 모델을 작성하는 작업을 "모델링(Modeling)"이라고 부른다.

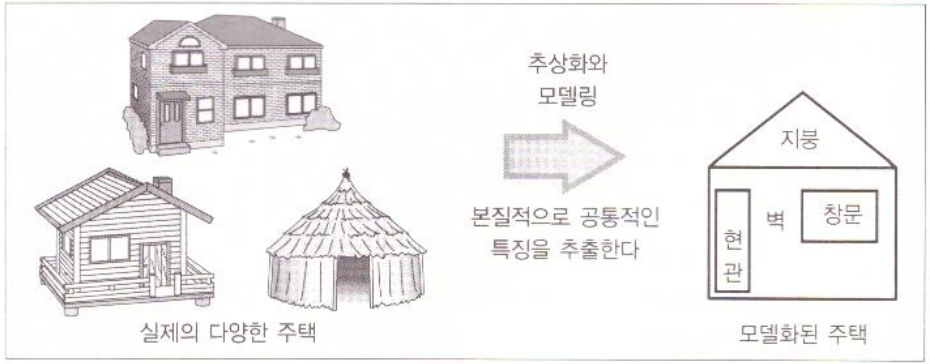


그림 1.7 | 추상화와 모델링

1.3.2 추상화와 모델링

- 모델은 각 항목에 대해서 소프트웨어가 갖는 성질을 추출하여 가시화해 준다.
  - 즉, 검토하려는 항목만을 모델로서 도식화하게 되는 이점
    - 대규모의 복잡한 시스템이라고 하더라도, 복잡함에 얽매이지 않고, 모델을 토대로 수월하게 검토를 진행할 수 있게 된다.
  - 각각의 검토항목마다 해당 소프트웨어의 모델을 작성할 필요
    - "어떤 소프트웨어를 개발해야 하는가"
    - "어떻게 소프트웨어를 개발해야 하는가"
    - "이용자가 어떻게 이용할 수 있도록 해야하는가"
  - 각 관점마다 모델을 작성 필요
    - 같은 항목이더라도, 관점이 서로 다른 경우에는 서로 이해와 비교를 수월하게 하기 위함
  - 소프트웨어개발의 각 공정에서의 다양한 관점에서 모델을 작성하는 기법이 제안되어 개발되고 있다.

1.3.3 요구사항분석

- 개발하려는 시스템에는 어떠한 제약사항이 있는지, 그리고 어떤 서비스를 제공해야 하는지를 규정한 것을 개발시스템에 대한 "요구사항"이라고 부른다.
- 시스템을 개발하기 전에, 요구사항이 명쾌하고 상세하게 결정되어 있고 문서로서 엄밀하게 기술되어 있다면, "대규모 소프트웨어에 수반되는 문제점들"의 대부분을 완화하는데 도움
  - 실제로 요구사항을 명쾌하고 상세하게, 그리고 엄밀하게 기술하는 것은 쉽지 않다.
  - 어떤 시스템을 개발해야 하는지를 생각하더라도 자연스럽게 요구사항들을 기술할 수 있는 것은 아니다.
    - 요구사항을 만들어내기 위해서는, 시스템의 서비스와 제약사항을 찾아내서 분석하고 문서화하여 검토하는 등의 일련의 작업들을 긴밀하게 수행할 필요가 있다. 이와같은 작업을 체계적으로 수행하는 기술을 "요구사항분석"이라고 부른다.
    - 소프트웨어개발프로젝트의 실패의 주요 원인 중 하나는 요구사항이 충분히 기술되지 않은 것임
    - 구조화 분석 등 체계적인 작업에 의해 요구사항을 만들어내는 기법과 도구들이 개발됨

1.3 문제 해결을 위한 대처 방안

1.3.3 요구사항분석

- 1980년대까지
  - 1. 시스템의 목적(무엇을 만들어야 되는가) 분석
  - 2. (1)의 결과를 토대로 요구사항 규정 개념 등장

- 이후 요구사항 규정
  - 아래 3가지에 대한 요구사항 수집하여 조기에 규정
    - 1. 기본기능(무엇을 만들것인가)
    - 2. 기본구조(어떤 구조로 만들것인가)
      - 즉, 소프트웨어 아키텍처
      - 품질특성에 미치는 영향 커짐
    - 3. 기본조작(어떻게 사용할 것인가)
      - "GUI(Graphical User Interface)가 진화
      - 이용자의 사용편의성이 중시됨

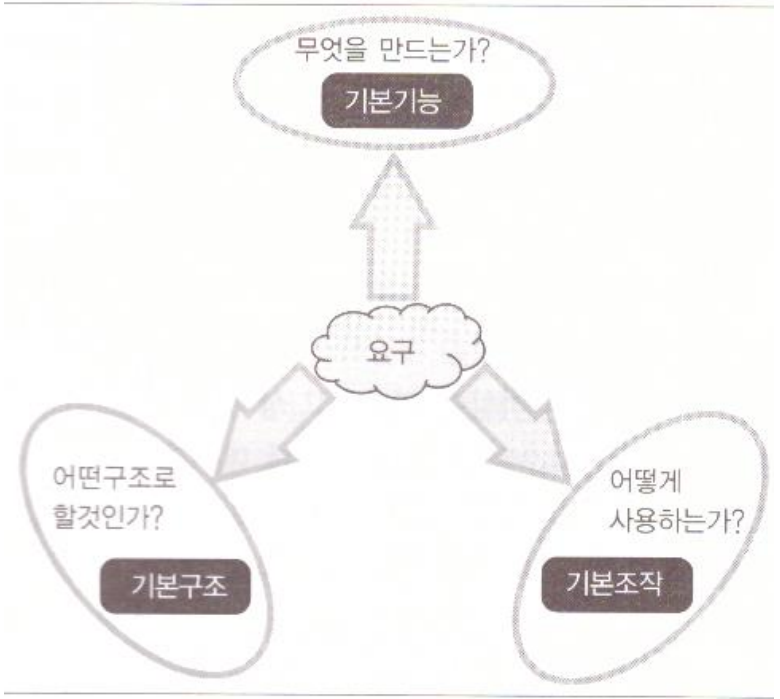


그림 1.8 | 요구사항을 수집하여 조기에 규정해야하는 사항들

1.3 문제 해결을 위한 대처 방안

1.3.4 추적가능성(Traceability)

- 소프트웨어의 설계단계
  - 다양한 문제들에 대해서 설계상의 판단 수행
    - "무엇을 만들어야 하는가"
    - "어떻게 만들어야 하는가"
  - 설계판단에서는, 문제에 대한 해답으로서 여러 방식들을 나열 및 비교 후 최선의 방식 선택 (trade-off 고려)
    - 성능을 중시할 것인지
    - 확장성을 중시할 것인지
  - 설계방침과 제약조건을 토대로, 적절한 판단 필요
  - 일관성있는 시스템 개발을 위해 문서화 및 추적 확인 필요
    - "어떤 판단을 내렸는가"
    - "왜 그런 방식을 선택했는가"
    - "누가 요청했는가"
    - "누가 결정했는가"

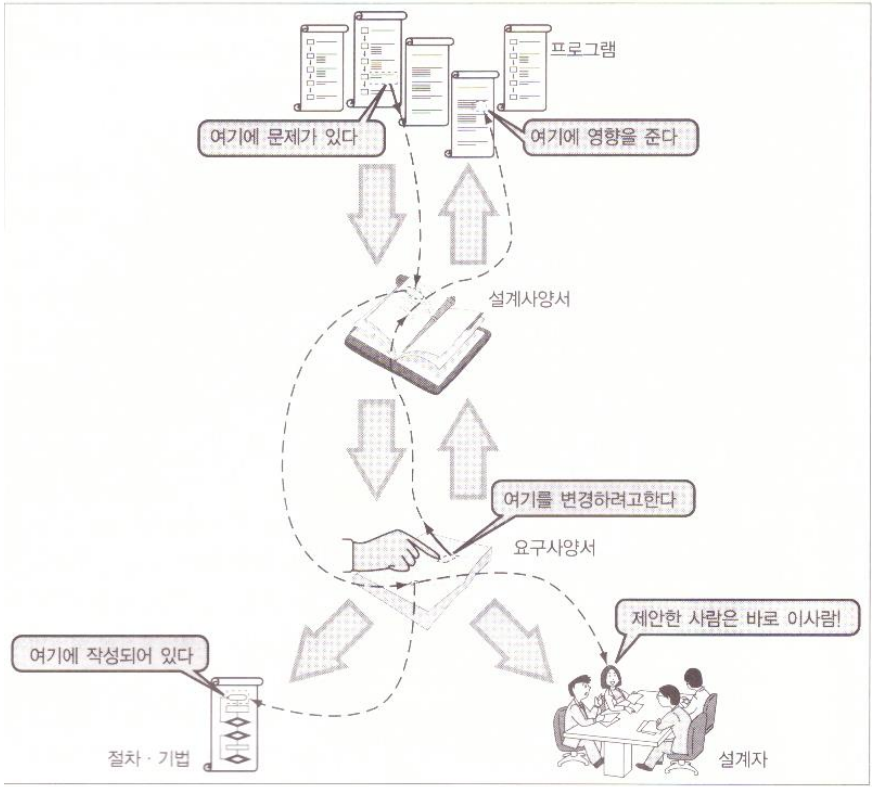


그림 1.9 | 추적가능성(Traceability)

1.3.4 추적가능성(Traceability)

- 설계판단을 추적할 수 있도록 하는 것을 "추적가능성(Traceability)" 또는 "추적성"이라고 부른다.
- 추적가능하다(추적가능성이 있다)는 것의 의미
  - 프로그램의 오류를 발견했을 때, 아래 과정을 통해 추적할 수 있다는 것을 의미
    - 프로그램으로부터 설계사양서 찾기
    - 설계사양서 상의 해당부분으로부터 요구사양서 찾기
    - 요구사양서를 규정한 절차, 기법, 설계자 등을 추적
    - 요구사양서 변경시 변경사항에 대응되는 설계사양서 및 프로그램 추적

1.3.4 추적가능성(Traceability)

- 추적가능성 확보 방안
  - 의사결정시 항상 확인작업 수행, 이후 재확인 필요
  - 설계판단 결과 기록시 해석의 차이가 발생하지 않도록 엄밀하게 서술 필요
  - 각 이해관계자들의 관점에서의 소프트웨어 파악 필요
  - 요구사항분석 결과의 엄밀성 및 도식화 기법 개발
  - 설계단계의 각 공정에서 도표와 문서를 사용한 기술 사양서 개발
  - 요구사항관리기술 개발
    - 개발관여자들 중에서 누가 해당 요구사항을 추출했는지
    - 어떤 요구사항의 토대가 되고 있는 요구사항은 어느 것인지
    - 요구사항에 대응하는 설계결과는 어느것인지
- 개발기간 및 이용기간이 장기화되고 요구사항이 변화되는 경우에 수월한 대응에 도움

1.3 문제 해결을 위한 대처 방안

1.3.5 경험의 축적과 재이용

- 일반적인 업무의 매뉴얼화
  - 패밀리 레스토랑에서는 고객에 대한 대응업무 등과 같은 대부분의 업무가 매뉴얼화되어있다.
  - 업무들이 매뉴얼화되어 있으면, 초보자라도 단시간내에 해당 업무를 익혀서 일정한 서비스 제공 가능

- 소프트웨어개발 관점
  - 다양한 사람들과 관계를 유지하면서 수많은 지적활동을 수행하므로 매뉴얼화하는 것이 어렵다.
    - 예: KDDI 4G LTE/5G 가상화 솔루션 설치 매뉴얼
  - 성공한 소프트웨어 개발프로젝트의 진척방법 조사
  - 다수의 성공 프로젝트의 공통점 추출 후 효과적인 방법론으로 정리 필요

1.3 문제 해결을 위한 대처 방안

1.3.5 경험의 축적과 재이용

- 소프트웨어개발 경험은 공통화/축적/집약화되어 다음번 개발에 재이용된다.
  - 소프트웨어개발 전체 또는 일부 작업들에 대해, 작업을 진행하기 위한 방법론 제안 및 도구 개발
    - 제안된 방법론이 실제 효과가 입증되면
      - 방법론을 이용하는 프로젝트가 증가됨
      - 같은 조직내에서 공통적으로 사용하게 됨
      - 많은 조직들에서 효과가 입증된 방법론 → 보다 많은 조직에서 이용되도록 개량화/표준화

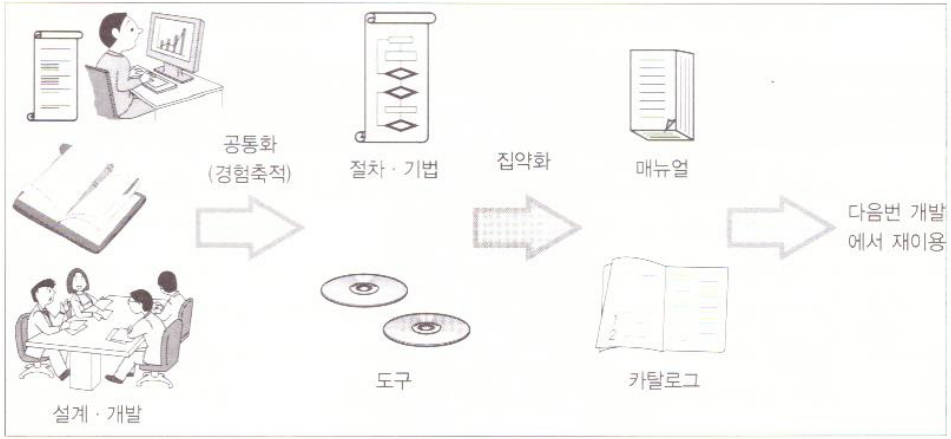


그림 1.10 | 경험축적과 재이용

1.3 문제 해결을 위한 대처 방안

1.3.5 경험의 축적과 재이용

- 국제적인 표준화 진행
  - 경험적으로 우수성이 확인된 기법이나 성과물들을 널리 그리고 수월하게 사용할 수 있도록 하기 위함
- 표준화에 의해 기법과 도구들이 공통화
  - 중간결과를 공유하며 공통적인 인식을 얻어낼 수 있게 되어 많은 담당자들이 협력 및 개발 수월
- 집약화에 의해 기법과 성과물들이 매뉴얼 및 카탈로그 등으로 정리
- 각각의 이용방법과 제약사항 등이 명시 → 다음번 개발에서 카탈로그에서 선택 → 재이용

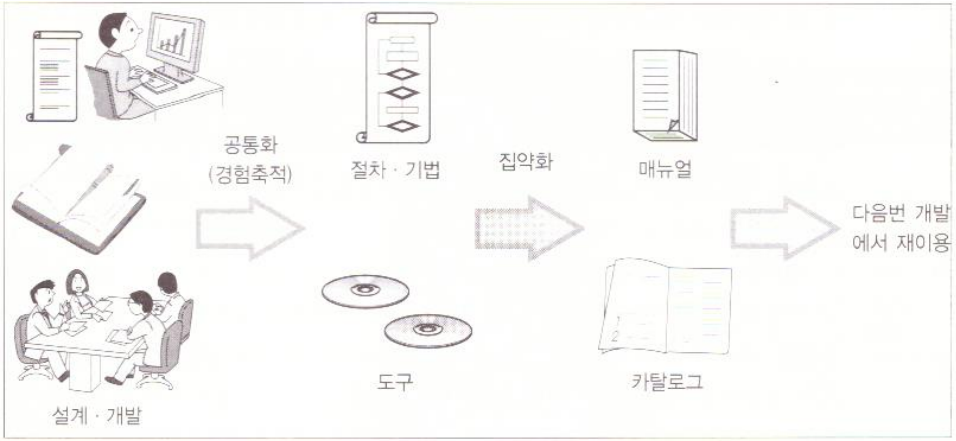


그림 1.10 | 경험축적과 재이용

1.3.6 체계적인 평가와 관리

- 프로그램의 테스트작업 필요
  - 여러 관여자들의 협력에 의해 대규모 소프트웨어를 구축시
    - 혼자서 소규모 프로그램을 만드는 경우에서처럼 디버그 방법을 고민하는 것으로는 충분하지 않음
    - 자동적으로 오류를 검출 어려움
    - 올바른 프로그램을 만들어내는 것은 어려움
    - 아무렇게나 프로그램의 동작 테스트를 계속 수행한다면 시간을 허비할 뿐 오류를 줄이기 어려움

- 프로그램을 체계적으로 조사해서 효율적으로 오류를 줄이기 위한 방법론과 구체적인 기법 필요
  - 프로그램의 타당성 검증 기술
  - 프로그램을 체계적으로 테스트하는 기술
  - 구체적인 도구 개발

1.3 문제 해결을 위한 대처 방안

1.3.6 체계적인 평가와 관리

- 프로젝트 관리의 중요한 요소
  - 개발 소프트웨어의 비용산정
    - 정해진 날짜까지 고품질의 소프트웨어를 개발하기 위함
  - 개발 추진방법의 타당성 검토
    - 세밀한 계획을 수립할 필요가 있다.
  - 프로젝트의 체계적인 진행상황 조사 기술 필요
    - 개발시작후에는 계획대로 진행되고 있는지를 확인
    - 계획보다 지연되는 문제 등이 발생되지 않도록 사전에 예견해서 대책 시행 및 조치 필요

생각해봅시다

- ★ "Stakeholder" 란 무엇인지 설명하시오. 또한, StakeHolder가 증가할 경우에 발생되는 문제점은? (19페이지)
- ★ StakeHolder가 증가한 경우에 발생하는 문제점에 대해서, 추상화와 모델링의 효과는? (28페이지)
- ★ 소프트웨어의 개발기간과 이용기간의 장기화에 의해 발생되는 문제점에 대해서, 추적(가능)성의 효과는?
  (21,22,34페이지)
- ★ 소프트웨어개발 초기단계에서 요구사항을 명확하게 규정해 둘 필요가 있는 이유는? (30페이지)
- ★ "무엇을 만들것인가" 뿐만 아니라 "어떤 구조로 할 것인가" 그리고 "어떻게 사용할 것인가"에 대해서 조기에 규정해 두어야 하는 이유는? (31페이지)

연습문제 #1

객관식 문제

- 1. "StakeHolder"는 증가하더라도 문제가 발생되지 않는다. 오히려, StakeHolder가 감소될 때 문제가 발생된다. **( )**
  (1) 그렇다 (2) 그렇지 않다 (3) 정답없음

- 2. "StakeHolder"가 증가한 경우에 발생하는 문제점을 완화하는 방법인 것을 모두 고르시오. **( )**
  (1) 가상화 (2) 모델분리 (3) 추상화 (4) 모델링 (5) 정답없음

- 3. 분할통치 및 구조화와 함께, 대규모의 복잡한 문제를 이해하기 쉽고 전체적으로 수월하도록 지원하는 개념으로 알맞은 것을 고르시오. **( )**
  (1) 분할화 (2) 상세화 (3) 고도화 (4) 추상화 (5) 정답없음

- 4. 대규모 소프트웨어에 수반되는 문제점이 아닌 것을 고르시오. **( )**
  (1) 생산물의 증가 (2) 사람의 증가 (3) 개발기간 장기화 (4) 사회적 역할 변화 (5) 정답없음

연습문제 #1

객관식 문제

- 5. 대규모 문제를 여러 개의 작은 요소들로 분할한 후에, 각각의 요소들 사이의 관계를 이해하기 쉽게 정리하는 것을 ( )라고 부른다. 괄호안에 알맞은 단어인 것을 고르시오.
  (1) 관계화 (2) 구조화 (3) 시각화 (4) 요소화 (5) 정답없음

- 6. 개발하려는 시스템에는 어떠한 제약사항이 있는지, 그리고 어떤 서비스를 제공해야 하는지를 규정한 것을 개발시스템에 대한 ( )이라고 부른다. 괄호안에 알맞은 단어인 것을 고르시오.
  (1) 제약사항 (2) 규정사항 (3) 요구사항 (4) 개발사항 (5) 정답없음

- 7. 요구사항을 만들어내기 위해서는, 시스템의 서비스와 제약사항을 찾아내서 분석하고 문서화하여 검토하는 등의 일련의 작업들을 긴밀하게 수행할 필요가 있다. 이와 같은 작업을 체계적으로 수행하는 기술을 ( )이라고 부른다. 괄호안에 알맞은 단어인 것을 고르시오.
  (1) 제약사항 분석 (2) 개발사항분석 (3) 기술사항 분석 (4) 요구사항 분석 (5) 정답없음

- 8. 설계판단을 추적할 수 있도록 하는 것을 ( )이라고 부르며, 대규모 소프트웨어에서 관여자들이 많을 경우에 특히 중요하게 된다. 괄호안에 알맞은 단어인 것을 고르시오.
  (1) 설계성 (2) 판단성 (3) 역추적성 (4) 추적성 (5) 정답없음

연습문제 #1

객관식 문제 (답안지)

- 1. (2) → 모호한 문제. 보기로서 제시된 문항만으로는 답안을 제시하기 어려우므로 1번 유형의 문제는 출제되기 어려움
- 2. (3, 4) → 추상화, 모델링
- 3. (4) → 추상화
- 4. (5) → 생산물의 증가, 사람의 증가, 개발기간 장기화, 사회적 역할 변화 등 모두 발생되는 문제점임
- 5. (2) → 구조화
- 6. (3) → 요구사항
- 7. (4) → 요구사항 분석
- 8. (4) → 추적성

연습문제 #1

주관식 문제 (괄호안에 알맞은 단어를 입력하시오.)

- 1. 소프트웨어 개발과 그에 따른 성과물에 관련있는 모든 사람들을 ( )라고 부른다.
- 2. 개발이 진행된 후에 문제점을 발견한 경우에는, 초기단계로 거슬러 올라가서 다시 수정하게 된다. 이와 같은 수정업무를 ( )이라고 부른다.
- 3. 그대로 다루기 어려운 커다란 문제를 작은 규모의 문제들로 분할하고, 각각의 작은 문제들에 대한 해를 한데 모아서 원래의 문제에 대한 해를 구하는 방법을 ( )이라고 부른다.
- 4. 대규모 문제를 여러 개의 작은 요소들로 분할한 후에, 각각의 요소들 사이의 관계를 이해하기 쉽게 정리하는 것을 ( )라고 부른다.
- 5. ( )는 대상이 되는 사물로부터 본질적이지 않은 부분은 제거하고, 몇 가지 사물에 공통적인 부분을 추출하는 것이다.

연습문제 #1

주관식 문제 (괄호안에 알맞은 단어를 입력하시오.)

- 6. 소프트웨어에 대한 추상화에서는, 검토해야할 항목들에 관련있는 주요한 부분을 추출해서 표현한다. 이와같이 표현한 것을 (A)( )이라고 부르며, 모델을 작성하는 작업을 (B)( )이라고 부른다.

- 7. 개발하려는 시스템에는 어떠한 제약사항이 있는지, 그리고 어떤 서비스를 제공해야 하는지를 규정한 것을 개발시스템에 대한 ( )이라고 부른다.

- 8. 요구사항을 만들어내기 위해서는, 시스템의 서비스와 제약사항을 찾아내서 분석하고 문서화하여 검토하는 등의 일련의 작업들을 긴밀하게 수행할 필요가 있다. 이와 같은 작업을 체계적으로 수행하는 기술을 ( )이라고 부른다.

- 9.설계판단을 추적할 수 있도록 하는 것을 ( )이라고 부르며, 대규모 소프트웨어에서 관여자들이 많을 경우에 특히 중요하게 된다.

연습문제 #1

선문대학교

주관식 문제 (답안지)

- ❑ 1. 관여자 (Stakeholder)
- ❑ 2. 재작업
- ❑ 3. 분할통치법
- ❑ 4. 구조화
- ❑ 5. 추상화
- ❑ 6(A). 모델
- ❑ 6(B). 모델링
- ❑ 7. 요구사항
- ❑ 8. 요구사항분석
- ❑ 9. 추적가능성 또는 추적성 (Traceability)

Talktime

Do you have any questions?