소프트웨어 프로젝트 관리

: 프로젝트 실패원인 두가지 : 1.성공적인 프로젝트를 수행하는데 필요한 지식을 가지고 잊지 않거나,
2. 프로젝트를 효과적으로 수행하는데 필요한 의지가 없어서 실패.

PART 1. 생존을 위한 사고 방식

1장. 살아남기 위한 방법 : SW PJ에서 살아 남으려면 우선 문명화된 방식으로 PJ를 시작하면 된다(생존권 보장)
-. 생존의 욕구, -생존권, -.생존 진단표, -.생존진단 문제

2장. 소프트웨어 프로젝트의 생존 테스트 : 건강상태를 평가할 수 있다. (RPPRP)
-. 생존 테스트(5 : Requirements,Planning,Project Control,Risk Management,Personnel), -.생존테스트 해석,

3장. 생존의 개념 : 잘 정의된 개발프로세스는 생산적인 업무(지적인 일에) 전념하도록,상류활동에 집중 하도록 함
-."프로세스"의 힘, -.상류와 하류 효과, -.불확실성 고깔

4장. 생존 기술 : 계획 수립이 잘된 프로젝트, 리스크 관리를 잘 해야 프로젝트가 성공할 수 있다.
-. 계획, -.계획 체크포인트 리뷰(PCR), -. 리스크 관리, 프로젝트의 가시성, -.피플웨어, -.사용자 참여,
-.제품의 단순화(minimalism), -.출시 지향

5장. 성공적인 프로젝트란 : 1.납품위주의 계획을 세워라.(단계별 납품방식) 2.누적코딩량:S자곡선이 안 나오면 심각하다. 많으면 ->릴리즈 준비 안됨,적으면->릴리즈할 준비 됐다. 3. 마일스톤 산출물 수준(매일 데모 좋다)
-. 지적 활동 단계, -.프로젝트의 진행, -.계획 단계, -.인력 투입, -.코딩 진척률, -.주요 마일스톤과 산출물

PART 2. 생존 준비 : 본격적인 개발준비, 생존을 위한 준비 ,첫번째 해야할 일 -> 변경 통제
*** 생존을 위한 해야할일 : 1.변경을 통제해라.2.S/W개발계획을 잘 세워라.3.요구사항개발.4.품질보증

6장. 움직이는 표적 맞추기 : 바뀌는 것을 통제하자.1.변경을 통제하고, 2.변경통제하고,계획을 잘 세워야 한다.
-> 1).변경통제, 2).변경위원회 구성, 3).CCB를 열리는 기준, 4).요구사항,소스코드 두가지에 대해서만 확실하게 하자(산출물)
-. 변경통제 절차, -.변경통제의 이점, -.변경통제시 에로 사항, -.변경통제에 대한 신임(committing)

7장. 사전 계획 : SW 개발 계획서포함->사전계획 포함 : 계획수립을 일찍 한다.비전을 정의한다.
최고의사결정권자를 찾는다.목표설정/리스크 관리/효율적인 인력 운용 등에 대한 전략 수립
-. 프로젝트 비전, -.최고 의사결정권자의 지원, -.프로젝트의 과업 목표, -.계획 및 진척도 공개, -.리스크 관리
-. 인력 관리 전략, -.시간 기록, -. 소프트웨어 개발 계획서

8장. 요구사항 개발 : 사용자 인터페이스 프로토타입과 사용자 매뉴얼과 요구 명세서를 여러 버전으로 작성하면 소프트웨어 개념이 더욱 명확해 진다. -> 아키텍처에 기초가 된다 , Context box 설명
-.요구사항 개발 프로세스 개요, -.주요 사용자 식별, -.사용자 인터뷰,
-.간단한 사용자 인터페이스 프로토타이핑 구축, -.스타일 가이드 작성, -.프로토타입의 확정,
-.완전히 확정된 프로토타입을 명세서의 베이스라인처럼 취급하기
-.프로토타입에 근거한 상세한 사용자용 문서 작성, -.사용자용 인터페이스가 아닌 요구사항 문서는 별도 개발

9장. 품질보증 : 테스트 단계에서 하는것이 아니다.품질보증은 테스트,테크니컬 리뷰,프로젝트 계획 등과 같이 프로젝트 초기에 경제적인 방법으로 결함을 발견하여 수정하기 위한 모든 방법을 포함한다.
제품의 완성도 : 가용성,변경용이성,성능,보안,시험용이성,사용용이성 , snow ball->기술부채(Technical debt)
-.품질이 중요한 이유, -.품질보증 계획, -.결함 추적, -.테크니컬 리뷰, -.시스템 테스트, -.베타 테스트
-.품질 보증 대상 산출물, -. 지원 활동

10장. 아키텍처 : 좋은 SW 아키텍처 문서에는 전반적인 프로그램구조,발생 가능성이 높은 변경사항에 대한 아키텍처의 대응 방안,다른 시스템에서 이미 개발 되었거나 사서 쓸 수 있는 컴포넌트가 기술 되어 있다.
표준적인 기능 요소들에 대한 설계 방안 제시,하류에서 발생될 수 있는 잠재적인 비용을 줄여 준다.
아키텍처는 발견,발명,구축 3가지를 다 알아야 한다. 요구사항(비기능/기능)->후보 Architecture driver(20개)->Architecture tactic(driver별 해결 해야 할 방법:ATAM으로 평가)->Architecture view:(tactic 설명):LPAD
-. 아키텍처의 시작, -.좋은 아키텍처의 특징, -.아키텍처 완료 여부 판단, -.소프트웨어 아키텍처 문서

11장. 최종 준비 : 사전 계획(요구사항,아키텍처)을 확장.
1),추정 작업을 할 준비, 2).중요한 기능을 우선해 납품하기 위한 계획을 수립.
-. 프로젝트 추정, -.단계별 납품 계획, -.지속적인 계획활동

PART 3. 단계에 의한 성공 (하류)

1).단계의 계획->마일스톤,2).상세설계->상세설계서(챕터별로 나누고 전체 리뷰),3)구축->.java, .cpp
4).시스템테스트 ->테스트코드,테스트결과서, 5)릴리스->환경세팅,릴리스노트, 6)마감

12장. 단계 계획 수립의 시작 : 해당 단계의 상세설계,코딩,테크니컬 리뷰,테스트,통합 및 작업을 수행하는 방법에 관한 개별 단계 계획을 수립한다. 가장 많은 노력이 필요로 한 작업은 해당 단계의 진척도를 추적하기 위한 **상세 마일스톤 목록을** 작성

-.단계 계획이 필요한 이유, -.단계 계획 개요, -.상세 마일스톤, -.단계 계획과 관리 방식

13장. 상세설계 : 아키텍처 View를 가지고 .java, .cpp를 떨어 뜨릴수 있는 상세설계, 주요Tactic의 component를 상세설계 할 수 있다.
아키텍처 설계를 확장한 것, 상세설계를 리뷰하면 프로젝트의 품질과 비용 측면에서 상당한 효과가 있다.
프로젝트1단계에 대한 상세설계는 아키텍처의 품질을 검증하는 것과 같은 특수한 작업이 필요할 때도 있다.

-.아키텍처 재검토, -.얼마나 상세하게 설계해야 하는가?, -.테크니컬 리뷰, -,상세설계 문서,
-.프로젝트 제1단계에 대한 특별한 고려 사항

14장. 구축 : 구축계획을 수립한다는 것 : 통합 계획
구현계획 ->개발중요활동(1.기능 개발,2.일별 빌드,3.스모크 테스트)
->관리포인트:관리활동 (1.변경통제,2.상세마일스톤, 3.10대 리스크, 4.지표)
: 지표 (1주일 분량으로 정리)->1).코드양(누적코드량:매일),2).결함 해결률:매일, 3).요구사항 완료률:매일)

-. 소스코드의 품질, -.소프트웨어 통합 절차, -.일별 빌드와 스모그 테스트, -. 제1단계의 특수한 고려 사항,
-.진척도 추적, -.변경통제, -.집중도 유지하기, -.그것이 구축될 전부인가

15장. 시스템 테스트 : 시스템 테스트는 구축과 병행하거나 혹은 절반을 남겨 놓고 실시한다.
품질(완성도) : 납품할 만큼의 수준

-.테스트 철학, -.일별 빌드에 대한 테스트 팀의 지원, -.시스템 테스트에 대한 개발자의 지원, -.전략적 품질관리

16장. 소프트웨어 릴리스 : 각 단계를 종료할 때마다 소프트웨어를 릴리스 가능한 상태로 만들어 둔다.
: 릴리스 시기->1).직감(경험을 통해 얻은 감각:과거프로젝트,
2).량->결함량(-. 밀도 , -. 분포 80:20 : 20%에서 집중 나타난다, -.과거 데이터)
3).추세(경향) : -.라인수와 비례하지 않는다.-.난이도에 상관없다. -.결함이 난 모듈에서 또다시 나온다.
: '결함 공개' 그래프에서 수정선과 발견된 선과 만나는 시점에서 릴리스 한다.
-.릴리스를 신중히 다루기, -.릴리스 시기, -.릴리스 체크리스트, -.릴리스 결재 양식

17장. 단계 마감 : 정확한 추정작업을 할 수 있으며, 이 추정은 다음단계 시작을 위한 계획 수립의 기초를 제공.
프로젝트 로그 기록. 현실적인 계획 수립

-.총괄 변경위원회 개최, -.추정 재조정, -.프로젝트 계획에 대한 실적 평가, -.프로젝트를 매체에 저장,
-.소프트웨어 프로젝트 로그 업데이트

PART 4. 임무 완료

18장. 프로젝트 이력 : 소프트웨어 프로젝트 이력 문서에 저장 되어 있는 정보는 향후 프로젝트에 유용하게 쓰일 것.
-.프로젝트 데이터 수집, -.소프트웨어 프로젝트 이력 문서,
-.향후 프로젝트에 사용하기 위한 프로젝트 이력의 결론 준비, -. 소프트웨어 프로젝트 이력의 사본 배포

19장. 생존 안내서
-. NASA의 성공 체크리스트, -. 기존 생존 자료

0 장 : 소프트웨어 프로젝트 관리 들어 가기 전에 .....

1. 소프트웨어 프로젝트 관리란 ? (기말시험 "SW프로젝트관리_요약" 참조 바람)

-.SW 프로젝트 관리는 3가지(SW, 프로젝트, 관리)로 나눠 이해해야 합니다.

- SW는 원래 복잡한 성격을 가집니다. 프로젝트는 돈을 버는 것과 연결해야 하며 제품을 만드는 공장을 기억해야 합니다. 관리는 힘쓰고 애써야 하는 활동입니다.

1). 소프트웨어란 ? 소프트웨어의 대표적인 속성은 복잡성을 갖고 있다. 요구사항 하나에 대해서 그것을 실현 인지의 한계, 경험의 한계. 다양하게 정의 되어 진다.
소프트웨어의 속성을 이해하고 소프트웨어 프로젝트 관리를 해야 한다.

2). 프로젝트

근거 와 이유 ? 프로젝트는 돈을 벌기 위한 소프트웨어 프로젝트이다. 제품으로서 어떠한 결과물이 나와서 돈을 벌기 위한 것이다. 공급자 -→ 수요자 소프트웨어 완성품 주고, 수요자 -→ 공급자 돈을 준다.
소프트웨어가 제품으로 상품으로서 해석 되어져야 한다.

모든 제품은 공장이 존재한다. 소프트웨어도 공장이 있어야 한다. 소프트웨어 공장은 사람들이다.
소프트웨어 공장의 단면도를 그리면 소프트웨어 공장의 단면도

생산성을 높이기 위해 즉,돈을 벌기 위해서는 공장의 공정 능력에 달려 있다. 공장의 공정능력을 높여주는 줌으로서 생산성을 높인다. 공정능력을 높이는 것은 사람들의 공정능력을 높여준다.상호간 의사소통이 잘되거나,개인이 얼마나 능력이 있나.사람들의 공정능력을 높이면 된다. 공정능력을 높이는 방법은 2가지가 있다.
사람의 능력,커뮤니케이션 첫째, 시작할 때부터 뛰어난 사람들이 있으면 된다.(채용) 그러나 쉽지 않다.
그래서 우리는 일반적인 사람들 간의 협업을 잘 하게 만드는데 초점을 주면 된다. 그것이 관리이다.

3). 관리

잘 적응을 못 할때, 그 역할을 잘 수행하지 못할 때 관리한다.관리를 하지 않으면 프로젝트에 악영향을 준다.
관리를 하지 않으면 안 좋아 진다. 가만히 놔두면 망하는 사람,상황,대상,사건에 대해서 관리라는 것을 쓴다.
이를 소프트웨어 프로젝트에 갖다 붙였다. 돈을 벌어야 되는 제품을 만들어 내는 소프트웨어 프로젝트는 망한다.
가만히 놔두면 자연스럽게 망 할 수밖에 없는 것이 소프트웨어 프로젝트이다. 이러한 것이 소프트웨어 프로젝트의 특성이다. 그래서 소프트웨어 프로젝트를 관리해야 한다. 소프트웨어 프로젝트 관리의 핵심은 가만히 놔두면 망할 수밖에 없는 부패하고 망하고 잘못 될 수 밖에 없는 실패할 수밖에 없는 대상인 소프트웨어 프로젝트를 아주 애를 써서 목적을 달성할 수 있게 만드는 practice와 관계와 의사소통 과 관련된 모든 것을 다루는 것이다.
프로젝트 관리의 방향은 고장의 공정능력을 높이는데 들어간다. Teaming 운영 중에 일이 잘되게 목표를 달성하기 위해 어떻게 커뮤니케이션을 해야 되는지 어떤 방식으로 회복되야 하고 전략을 어떻게 짜야 되는지를 중점적으로 다루는 관리가 소프트웨어 프로젝트 관리이다.

-. 용어 정리 : Joel test -> 12가지 , Sandbox -> 자동화 환경 구성

-. CMMI (Capability Maturity Model Integration)
소프트웨어 개발 및 전산장비 운영 업체들의 업무 능력 및 조직의 성숙도를 평가하기 위한 모델이다.
CMMI는 1~5단계까지 있으며, 5단계가 가장 높은 수준이다.
CMMI는 소프트웨어 개발 및 전산장비 운영 분야의 품질 관련 국제 공인 기준으로 사용되고 있다.
Level 2 : Basic Project Management 가 잘 되어 있어야 한다. 망하지는 않는다. 성공할 가능성이 높다.

-. 소프트웨어에 대한 이해로부터 관리가 시작됩니다 (O . X 문제)

1. 소프트웨어 작업을 성공하려면 유능한 사람을 뽑으십시오.
2. 사람이 일을 잘 할 수 있는 환경(물리적, 문화적)을 만드십시오.
3. 의사소통을 어떻게 하면 잘 하게 할 것인가 고민하십시오.
4. 지적인 일에 집중하고 사무적인 일은 기계가 할 수 있도록 만드십시오.
5. 소프트웨어의 본질인 복잡성을 놓치지 마십시오.
   1). 신속한 f/b
   2). 실시간 알림
   3). 너무 멀리 가서 확인 하지 마십시오.
6. 사무적인 일 20%를 기계에게 맡깁니다.
7. 사람은 지적인 일 80%에 집중하게 합니다.

•SCM(Source Code Management) 시스템이 주는 유익
- 팀이 프로젝트 규모의 ("Undo") 버튼을 갖게 됩니다. 어떤 것도 최종적이지 않고 실수를 쉽게 돌이킬 수 있습니다.
- 여러 명이 동시에 한 파일을 편집할 때 충돌을 조정할 수 있습니다.
- 여러 버전의 소프트웨어를 (추적)할 수 있습니다. 지난 출시 제품의 버그를 고치는 와중에도 다음 출시 제품에 기능을 추가할 수 있습니다.
- 어떤 파일이 변경됐는지 기록할 수 있습니다(언제 누가 고쳤는지도 기록할 수 있습니다).
- 특정 시점의 작업을 (고스란히) 꺼내올 수 있습니다.

•(개인 PC환경) 에서 빌드하는 것을 의미하지 않습니다.
•(개인 PC환경) 은 OS patch 버전, IDE 링크, 라이브러리 버전 등에 따라 빌드되는 현상이 제각각 일 수 있습니다
•수동 빌드는 사람이 실수할 수 있으며, 실수를 만회하기 위한 노력이 만만치 않게 들어갑니다
•프로젝트 초반에 빌드를 (스크립트화) 해야 합니다
•어떤 컴퓨터에서라도 빌드가 되어야 합니다

•자동 빌드는 기계가 빌드를 하므로 여러분은 (지적인 일)에 집중하시면 됩니다.

• 개발할 시스템의 요구사항을 (목록)으로 만드세요
• 목록에 들어가야 할 요소
  -"누가, 언제까지, 무엇을, 얼마나 중요하게, 얼마동안"
•목록관리로 도구로 활용할 수 있는 것
  -화이트보드 -웹기반의 wiki -Trello
•기능 마다 조리에 맞게 (우선순위)를 매긴 포괄적인 목록이 있으면, 여러분의 노력에 의존하는 팀과 관리자, 그리고 다른 팀도 자신감을 가질수 있습니다.
•목록을 사용하여 팀의 기민함이 엄청나게 증가합니다.
 -제품을 기능요구사항으로, 기능요구사항을 목록의 항목으로 분해함으로써 기본적인(설계작업)을 미리 끝내는 효과가 있습니다.

1.여러분이 개발하고 있는 모든 기능을 (화이트보드)에 적어놓으세요. 이렇게 하려면 시간도 제법 걸립니다.
2.각 기능에 (우선순위)를 부여하세요. 이 과정에 적절한 **이해관계자**를 참여시키세요. 이 과정에 **팀 전체를 참가** 시키는게 이상적입니다.
3.모든 기능요구사항을 우선순위에 따라 정렬해서 다시 쓰세요.
4.각 항목에 예측시간을 다세요. -1~2일 사이에 완료할 수 있는 수준으로 분할
•현재의 상위 우선순위 항목들이 완료될 때 까진 누구도 그보다 낮은 우선순위항목을 처리해선 안됩니다.

•모든 사람이 **회의형식(어떤 질문이 오갈 건지 같은 것)**을 알게 하세요.

-. 코드라뷰
•코드리뷰는 소스코드를 비판하고 검사하여 코드를 개선하는 것을 의미합니다
•우리(개발자)는 **코드도 만들어내고**, (결함)**도 만들어 냅니다**.

-. 코드리뷰의 장점
•소프트웨어의 품질을 높입니다
•코더에게 스스로 만든 코드에 대한 책임감을 부여하며(**코드의 공동소유**) 문화확립
•전체 프로젝트의 코딩 스타일에 통일성을 부여합니다.
•지식전파(코드의 핵심부분의 내부작동에 대한지식

•여러분이 계획하는 코드 검토가 어떤 건지 모든 사람이 이해 해야 합니다. (작은 코드블럭 단위)로 (자주)검토 하세요. 몇주씩 검토하지 않고, 수백 또는 수천 줄이나 되는 변경 내역을 쌓아 놓지 마세요.

- 일일 회의
•주간회의: 공지사항, 지난주의 업무, 다음주계획
•일일 회의**: 일정에 심각한 악영향을 주지 않고도 상호작용이나 의사소통을 장려하는 간단한 팀 회의**
•일일 회의 Key point
  -구성원이 무슨 일을 하고 있는지,
  -어떤 문제에 부딪혔는지 간단하게(1인당1~2분)공유
  -회의는 짧게 핵심만 다룹니다
•일일 회의가 필요한 이유
  -의사소통의 실패를 해결하는 가장쉬운 방법은 팀과 보다 자주이야기를 나누는 것입니다.
  -문제 발견시간이 짧아지고, 고칠 **기회가 더 많아**집니다.

-. 코드리뷰 잘 하기 위한 팁
1). 3~4명이 2) 1시간 이하, 3). 한번 할때 200라인 이하, 4). 매일 5). 산출물:5~6Page,
6). 페이지는 라인넘버가 항상 찍혀 있어야 한다

PART 1. 생존을 위한 사고 방식

>. 소프트웨어 프로젝트는 일반적으로 두가지 이유 중 하나로 인해 실패한다.

1). 프로젝트 팀이 소프트웨어 프로젝트를 성공적으로 수행하는데 필요한 지식을 가지고 있지 않거나.

(일하는 방법 : 수행하는데 필요한 지식을 관리 / 지원 /기법 공부한다.)

2). 프로젝트를 효과적으로 수행하는데 필요한 의지가 없다.

1장. 살아남기 위한 방법 : SW PJ에서 살아 남으려면 우선 문명화된 방식으로 PJ를 시작하면 된다(생존권 보장)

-. 생존의 욕구, -생존권, -.생존 진단표, -.생존진단 문제

1. 소프트웨어 프로젝트 생존전략이 주는 메시지는
소프트웨어 프로젝트를 얼마나 세밀하게 계획했는지, 그리고 얼마나 신중히 이 계획을 실행했는지에 따라 프로젝트의 성공하거나 실패한다. 프로젝트의 이해 관계자가 성공을 결정짓는 주된 이슈를 이해하고 있다면 이들은 프로젝트가 성공적인 결론에 도달하도록 할 수 있다.

2. 제약 조건 : 3~25명의 팀, 3~18개월의 일정의 규모에 적용하는 것을 권장한다.

3. CMM : Level 2에 있는 KPA에 기초한다.
-. 프로젝트 계획 수림, -. 요구사항 관리, -. 프로젝트 추적과 감독, -. 형상관리, -. 품질관리, -. 협력 업체관리

4. 자아실현 (전문성 유지) ->자긍심(생산성,프로젝트의 중요도)->소속감과 사랑(활력 있는 팀 팀의 역동성)
-> 안전욕구(일정 기능에 대한 개인적인 약속 이행)
-> 생존 욕구(프로젝트가 취소되지 않고,팀도 헤체 되지 않고,만족스러운 물리적 작업 환경, 기타) : 매슬로우 참조

5. 프로젝트 팀이 소프트웨어 프로젝트를 최적화해서 빠듯한 일정 과 정해진 시간에 맞춰 완수하고 현재의 기술 수준을 진일보시키는 등의 모든 일을 동시에 할 수 있게 되기에 앞서, 우선 시간에 맞춰 납품을 할 수 있어야 한다.

6. 소프트웨어 프로젝트에서 살아남으려면 우선 문명화된 방식(Civilized way)으로 프로젝트를 시작하면 된다.
(생존권을 보장해 주는 것을 문명화된 방식 이라고 한다)

7. 고객 관리 장전
-. 프로젝트에 대한 **목표를 세우고** 이를 따르게 할 권리 (고객에게 왜 라고 질문)
-. 소프트웨어 프로젝트에 소요되는 기간과 비용을 알 권리
-. 소프트웨어가 가져야 할 기능을 결정할 권리
..... 나머지 생략

8. 프로젝트 팀의 권리장점
-. 프로젝트의 목표를 알고 우선순위를 명확히 할 권리 : 팀웍 -> 목표에 부합되지 않는 것에 대한 제지하고 목표를 일깨워 줄 수 있는 관계. 생존을 위한 목표에 대한 명확한 인식
-. .... 나머지 생략

2장. 소프트웨어 프로젝트의 생존 테스트 : 건강상태를 평가할 수 있다. (RPPRP)

-. 생존 테스트(5 : Requirements, Planning, Project Control, Risk Management, Personnel), -.생존테스트 해석

1. 생존 테스트 (RPPRP) : 5가지 : 생존할 수 있을까를 알아보는 것

****** 문제 예) 점검 사항으로 해당되지 않는 것은**

-. 요구사항(Requirements)
-. 계획(Planning) : 가장 많다
-. 프로젝트 통제 (Project Control) : 가장 많다.
-. 리스크 관리 (Risk Management)
-. 인력 (Personnel)

문제의 답 정리
계획을 두루뭉실하게 수립해서 통제가 안 된다.
리스크 관리 민감한 10가지 정도를 발굴 하고 그것을 가지고 계획을 수립해야 한다.

3장. 생존의 개념 : 잘 정의된 개발프로세스는 생산적인 업무(지적인 일에) 전념하도록,상류활동에 집중 하도록 함
-."프로세스"의 힘, -.상류와 하류 효과, -.불확실성 고깔

생존의 사고 방식 : 생존을 위해는 이것이 중요한 것이구나 라고 느껴야 한다.

프로젝트 생존이 중요해 – 목표를 정하는 그것을 하기 위해서는 5가지의 점검사항이 중요하다.
요구사항/계획/프로젝트 통제/리스크 관리/인력 이 있다. 내 문제 이다
생존을 하기 위해서는 어떠한 사고 방식으로 접근해야 첫번째 프로세스가 있어야 한다.

< 생존을 위한 프로세스 >
잘 정의된 개발 프로세스는 소프트웨어 프로젝트의 생존에 중요하고,필수적인 요소이다. 이것은 소프트웨어 개발 관련자들이 쓸데 없는 데 신경 쓰지 않고 생산적인 업무에 전념하여 프로젝트를 안정적으로 끝낼 수 있도록 한다.
상류활동에 집중하게 만드는 프로세스가 중요하다.
1.요구사항을 정의하는 프로세스.
2.요구사항을 변경하는 프로세스 .
3. 요구사항,설계,소스코드를 체계적으로 소스코드를 체계적으로 테크니컬 리뷰하는 프로세스.
4. 테스트 계획,리뷰계획,결함 추적 계획하는 프로세스를 만들어야 해.
5. 소프트웨어 컴포넌트를 개발하고 구현하는 순서가 명시된 구현 계획을 (순서가 정의된 계획: 통합하는 순서에 대한 계획)
6. 자동화된 도구로 소스코드를 관리
7. 통제 단위로 잡은 마일스톤이 생길때 마다 비용과 일정 추정을 재검토하는 것

< 프로세스 좋은 점 시나리오 항목 >
-. 변경통제
-. 품질 보증
-. 무분별한 개정(uncontrolled revision) : 형상관리
-. 결함추적
-. 시스템 통합
-. 자동화 소스코드 통제
-. 일정 관리
이것들에 대해서 정하면 된다 . 프로세스가 중요하다.

-. 상류와 하류의 효과
: 상류에 집중한다.
-> 훌륭한 프로젝트 팀은 요구사항과 아키텍처를 주의 깊게 검토하여 상류 쪽 문제를 정정할 수 있는 기회를 스스로 만들어 낸다.

왜 상류에 집중해야 하나 – 상류의 불확실성이 높고 리스크가 앞쪽에 있다는 것이다.
꼬깔 설명 : 상류의 불확성이 높고, 규모추정치를 보면 상세설계 이후는 편차가 거의 없고 앞쪽에 편차가 크다 따라서 앞쪽을 어떻게 정의하고,어떻게 테크니컬 리뷰를 하고,변경하는 것을 어떻게 핸들링 할 것인지를 정해라 ...
이것이 생존의 개념이다.

4장. 생존 기술 : 계획 수립이 잘된 프로젝트, 리스크 관리를 잘 해야 프로젝트가 성고할 수 있다.
-. 계획, -.계획 체크포인트 리뷰(PCR), -. 리스크 관리, 프로젝트의 가시성, -.피플웨어, -.사용자 참여,
-.제품의 단순화(minimalism), -.출시 지향

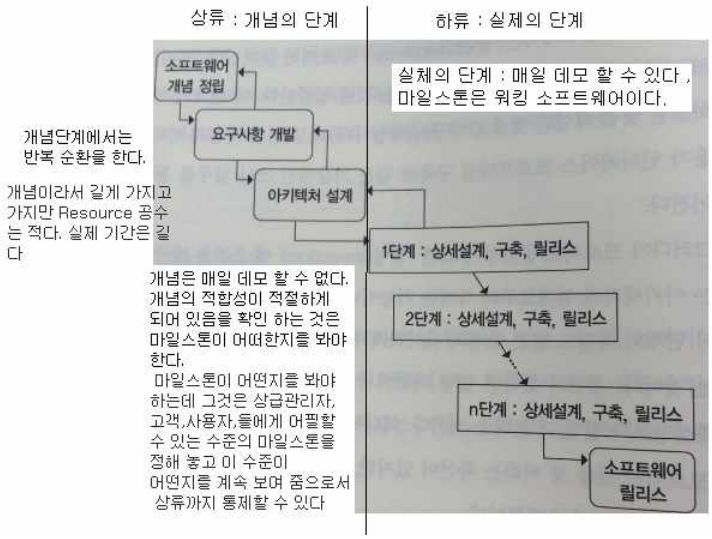
-. 생존의 기술
소프트웨어 프로젝트는 본질적으로 복잡한 것이다. 복잡한데다 계획까지 엉성하다면 절대로 성공 할 수 없다.
소프트웨어 프로젝트는 본질적으로 위험한 것이다. 적극적으로 리스크를 관리하지 않으면 성공하기 힘들다.
리스크 관리를 잘 하려면 사용자를 계속 참여시켜야 한다.

1. 계획
PCR (Planning Checkpoint Review)
2. 리스크 관리 : 리스크 관리 민감한 10개 리스트 화
프로젝트 통제 : 일주일 마다 데모를 해라
3. 프로젝트 가시성 : 초기는 개념 ---→ 가시성(가시화 시켜라: 마일스톤(산출물)) : 리스크로 관리 해라
상세마일스톤을 설정해라 (binary milestons)을 만어서 가시성을 확보해라. (Y/N Binary)
4. 피플웨어
5. 사용자 참여 : Ask , Show , Ask
-. 사용자가 원하는 것이 무엇인지를 물어봐라.
-. 개발자들이 구축하고자 하는 것이 무엇인지 보여주라
-. 그런 다음 제품에 대한 사용자의 생각을 물어봐라.
6. 출시 지향
: 제품 릴리즈에 전략을 다한다. 사용자 환경으로 소프트웨어를 데모를 보여줘라

5장. 성공적인 프로젝트란 : 1.납품위주의 계획을 세워라.(단계별 납품방식) 2.누적코딩량:S자곡선이 안 나오면 심각하다. 많으면 ->릴리즈 준비 안됨,적으면->릴리즈할 준비 됐다. 3. 마일스톤 산출물 수준(매일 데모 좋다)

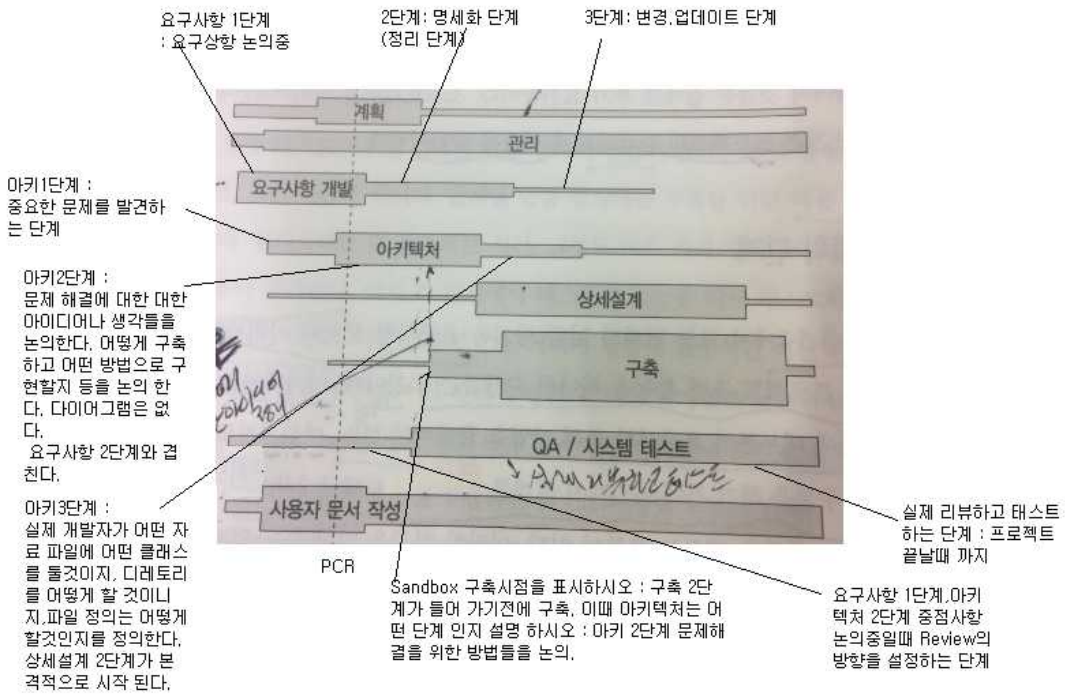
-. 지적 활동 단계, -.프로젝트의 진행, -.계획 단계, -.인력 투입, -.코딩 진척률, -.주요 마일스톤과 산출물

1. 납품위주의 계획을 세워라 (Delivery release): 제때 납품을 해야 생존을 한다.
   왜 단계별(Stage)이냐 : 누적의 개념 A, A+B, A+B+C 순으로 단계별로 릴리즈 해라 (increament)
   : 중복 테스트 시 비용 발생 : 해결하기 위해 -> Sandbox를 만들어라
   -> 중요한 기능을 우선 개발하여 납품한다. (중요한 : 기능,비즈니스,리스크 등등의 난이도를 모두 이야기 한다)
   -> 중요도에 의해서 Delivery Plan을 세워라.
2. 누적 코딩량은 직선이 아니라 S 곡선을 그린다.
   프로젝트 진행할 때 정상적으로 진행될 때 S곡선이 나온다. 많으면->릴리즈 준비가 안됐다. 적으면-> 릴리즈할 준비가 됐다. 남아 있는 Task와 일정 그래프시 완만한 곡선을 그려줘야 한다. 계단형 일때는 통제를 매일 하지 못했거나 통제를 잘 못했거나 Task가 서로 연결되어 되어 있거나 (설계상) 계단형의 문제가 있는 프로젝트이다. Binary Milestone을 잘게 자르는 것이 좋다.
   코딩량을 관찰하면 프로젝트 상태를 알 수 있고.상급 관리자,고객,사요자는 마일스톤 별 산출물의 수준을 보고 프로젝트가 잘 진행되는지 여부를 파악하게 된다.
3. 마일스톤별 산출물
   가장 좋은 방법은 매일데모 하는 것 현실적이지 않다
-. 매일 데모 할 수 있는 단계, 할 수 없는 단계
   제일 좋은 방법은 매일 데모 한다 . 그러나 매일 데모 할 수 있는 단계, 할 수 없는 단계가 있다



*.지식활동 단계
  1). 우리가 중요하게 해결하고자 하는 문제가 들어 있다. **발견**
  2). 그 문제를 어떤 식으로 해결할지에 대한 아이디어,접근 방식을 정리 한다. **발명**
  3). 실제 내가 코딩할때 뭘 주의해야 되는지 Architecture View를 작성 한다. **구현**
    : 사안에 따라 Dynamic View,Process View 등을 별도로 추가 한다.
  * 발견/발명/구현의 교집합이 아키텍처이다.

-. 단계별 납품 방식의 장점
1). 중요한 것을 먼저 할 수 있다.
2). 리스크를 조기에 해결할 수 있다.
3). 문제를 조기에 발견할 수 있다.
4). 커뮤니케이션 비용이 절감한다. / 5). 보다 많은 선택의 여지를 제공한다. / 6). 추정치의 오류를 줄여준다.
7). 유연성과 효율성 간 균형을 맞출 수 있다.
-. 우리 팀이 뽑아낼 수 있는 것이 이것이 모두다라고 할 때를 80%라고 인식한다.



Ⅱ. 생존준비 (하류활동을 시작하기전에 준비할 것)

- 변경통제 - 사전 계획 - 요구사항 개발 - 품질 보증 - 아키텍처 – 최종준비

6장. 움직이는 표적 맞추기 (10/27)

☐ "변경위원회"(CCB)를 설립하여, 사전에 정해진 룰에 따라 변경을 제한하고, 주요 산출물에 대한 변경을 통제한다.

- 바뀌는 것이 생존에 지대한 영향을 준다 ➔ 통제하지 못하니까 어려움에 처함.
- SW 복잡성 특성 때문에 변경되기 마련. 변경되는 것이 문제가 아님. 변경을 받아 수용해야 함. 변경된 것을 내가 모르는 것이 문제. 변경을 차단하라는 것이 아니라 통제하는 것이 목적.

☐ CCB(Change Commit Board) 실효성이 없는 이유?

- 프로젝트는 공학의 영역 뿐 아니라, 사람간의 협력관계도 포함되어있다.
- CCB 구성에 성의가 없다 ➔ 정성있게 하라. 모든 것을 공학으로 풀면 안된다.
- 어떻게든지 확인 받아야 딴소리가 안나온다.
- 룰 : 위원회 여는 기준을 정해야 한다.
- 주요 산출물 : 요구사항, 소스코드 확실하게 룰을 정해라.

✓ 변경자체가 좋고/나쁜 부정적인 것이 아니라, 변경이 됐는지, 왜 됐는지 아는 것이 중요하다. 왜 일어났고, 우리가 받는 영향. 생존을 위한 준비 첫 번째 통제 가능한것과 불가능한 것을 나누자. 통제 할 수 없는 것은 통제 할 수 있는 사람을 찾자. 통제 할 수 있는 것은 통제하자. 중요한 것은 요구사항에 따라(요구사항을 매우 목록, 세부적으로 매우 상세화 해야한다.

✓ 리뷰를 최우선으로 두고 소나큐브, 지라를 이용하여 파악하자(상황인지)

✓ 변경통제 절차

- 요구사항에 위험도, 중요도, 난이도, 우선순위가 있어야 한다.
- 변경했으면 왜? 누가? 언제? 어디를? 까지 알아야 좋다.

❍ (7조 6장) 변경을 통제를 릴리즈 단계 마무리나 개발 도중에 발생하게 되는 소스 리펙토링에 대해서도 철저한 관리를 하는 것이 옳은지요? 물론 해당 리펙토링이 기능 동장에는 전혀 오류나 영향을 주지 않는 전제에서 말입니다. ➔ 리펙토링은 기능성(기능이 원활하게 동작하는 지) TEST 할 수 있는 testcase 확보된 상태에서 수행하자.

7장. 사전계획 (10/27) ⇒ SW 개발 계획서가 나와야 한다

☐ 성공하는 프로젝트는 계획 수립을 일찍 한다. 사전 계획을 수립할 때는 프로젝트 비전을 정의하고(비전이 없으면 wbs에 의해 움직이는 작업군에 불과), 최고 의사결정권자(실세)가 누구 인지 찾아내자. 또한 과업 범위에 대한 목표 설정, 리스크 관리, 인력 운용등에 대한 전략도 세우자.

✓ 비전이 무엇이냐?

- '이것을 왜 하냐?' 의 답이다.
- Beyond Closed Door – 프로젝트 관리 비전 공유
- Goal은 비전으로의 단계 일 뿐이다.
- 목표는 바뀔 수 있다.
- Why? XS ➔ 본질적인 것을 찾음 = 통찰력
- 비전이 없으면? ➔ 기준도 없고, 아무생각이 없다.

✓ 경영진의 전폭적인 지원 (people ware)

- 필요할 때 필요한 사람이 들어 올 수 있게 해주는 것

✓ 리스크 관리 ➔ 해결책 나열 ➔ 준비

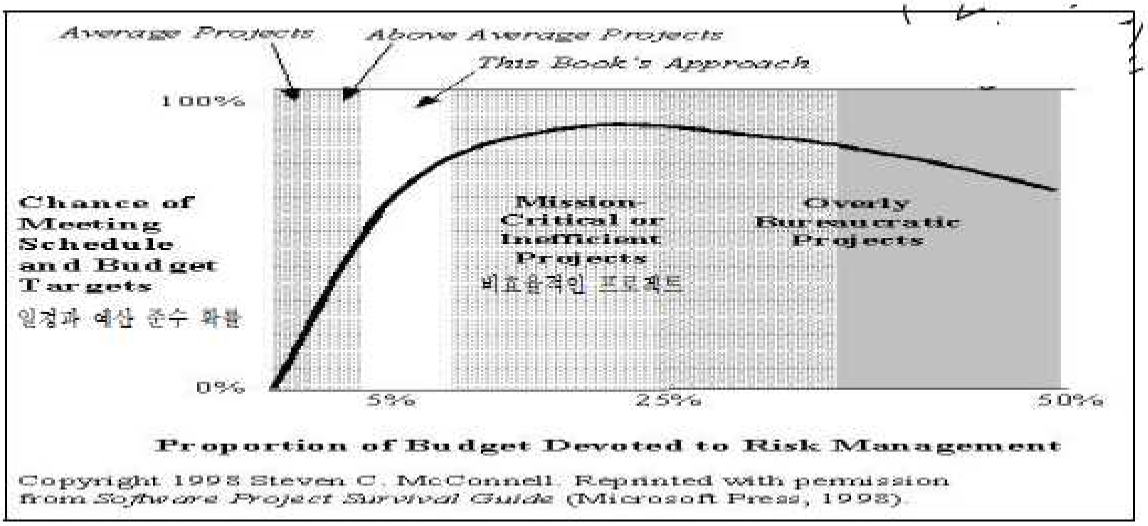
- 준비했는데도 해결이 안된다 ➔ 최고 의사결정권자(executive sponsor)에게 도움을 요청

✓ 계획서

- 프로젝트 비전(나아갈 방향, position)
- 최고결정권자(경영진)
- 과업범위목표설정(프로젝트 목표), 프로젝트 성공의 기준
- 리스크관리(10대 리스크)
- 인력운용 ➔ 자격이 있는 사람을 뽑자. 키워서 쓴다는 이상한 생각은 하지 말자.

✓ 프로젝트의 노력 중 전체비용의 5%만 '리스크 관리'에 투자해 보라. 성공할 확률이 50~75% 정도로 높아짐. (일반

적 25%의 2~3배)



- ❍ (1조 7장) 타사의 프로젝트인 경우는 이미 일정과 비용이 정해진 상태에서 프로젝트가 진행되므로 목표치를 재추정하는 것이 현실에서는 불가능하다. 이때 이로 인한 리스크 관리는 어떤식으로 하면 좋을까요? 오르지 사용자와의 타협과 설득만이 답이 될 수 있을까요? ➔ 재추정 하면된다. 우리가 어떤상황인지 알아야 어필 할 수 있다.
- ❍ (5조 7장) 고참 개발자가 리스크 관리자, 프로젝트 개발 동시에 수행 가능할까? ➔ 리스크 관리자 역할을 수행시켜라. 리스크 관리는 소스코드에만 국한된 것이 아니라, 프로젝트 전반에 대한 관리이다.
- ❍ (6조 7장) 속 썩이는 개발자를 무조건 교체하는것이 최선인가? 잘 못하는 부분이 있으면 상급 개발자를 붙여 서포트 하는 것도 좋은 방법이 아닐까? ➔ 업무 파악을 위해 데일리 미팅으로 해결하자. 계속 말을 안 들으면 자르자.

8장. 요구사항 개발 (11/3)

<요구사항 1. 범위 축소 2. SW 개발 계획 3. 요구사항 개발

☐ 요구사항 개발 준비를 해야한다.
- 무엇을? SW개념을 명확하게 할 것인지. 8장은 그것의 팁이다.

① 사용자 인터페이스 프로토타입 ➔ 사용자가 어떻게 사용할 것인지(사용성)?
② 사용자 매뉴얼 ➔ 사용자에게 의미 있는 기능
③ 요구 명세서 ➔ 사용자에게 드러나는 시스템의 요건들(내부명세)

⇒ 중요도순. 명확화의 대상. 달성하고자 하는 목적. 이것들을 여러 버전으로 작성하면 소프트웨어 개념이 더욱 명확해진다.

✓ 너무 상세하게 작성하지 말자 ➔ 사용자 스토리가 요즘 대세
- product backlog(사용자 요구사항) ➔ 개념적 아키텍처 ➔ 릴리즈 중심 개발 계획
- 개발 직전에 확정 못진 요구사항들 중 결정. 문서화X. 고객·사용자 에게 물어보고 반영하고 하는 방식

✓ 프로토타입 보여주고 끝 방식X
- 보여주고 반드시 질의응답 해야한다 ➔ 상호작용이 필요

✓ 논의과정을 어떻게 끌고 가야하는가? = 체굴을 잘하는 방법? (커뮤니케이션 능력)
- 도메인 지식, 프로토 타입, 자주 보여주기 ➔ 참여를 시키자

① 개방적인 질문 : 질문을 잘 해야한다. 근본을 찾아내야한다. 선택이 아닌 개방적인 질문(처음엔 쉽지 않다. 초반에 폐쇄적인 질문으로 시작해서 why를 물어라)
② 주요 사용자 식별 (사용자 역할 모델링)
  (1) 관심있는 실체 사람들의 이름을 다 나열 ➔ 역할자
  (2) 실체가 관심있는 내용(concern) 작성
  (3) 비슷한 사람, 일 묶어서 역할자 ➔ 사용자, 역할자, concern 여기서 missing 되는 것이 요구사항에서 누락될 가능성이 크다.

✓ 같은 시스템인데 보는 관점(View)에 따라 다르다 ➔ 모으면 명확. Multi View로 작성하다보면 개념이 명확해진다.
✓ 사용자 인터뷰 : 인터뷰가 제일 어렵다 ➔ 대화법

✓ 왜 프로토타입을 보여줘야 할까? 보여야 얘기를 잘 하니까!
✓ 테크닉도 중요하지만 인문학 ➔ 통찰력. 질문을 어떻게 하느냐도 중요하다.
✓ 요구사항 : 공학적 요인 + 인문학적 요인(비중이 더 크다)

9장. 품질보증 (11/3)

□ 기본적인 품질보증

① 테스트, ② 테크니컬 리뷰, ③ 프로젝트 계획 등 과같이 프로젝트 초기에 경제적인 방법으로 결함을 수정하기 위한 모든 방법 포함

- '소프트웨어가 명시적(기능이 돼야하는데, 잘 빨리 되는 것(비기능적)) 암시적 요구사항을 모두 충족하는 정도'

□ 품질의 절대적인 기준은 없다

- 비즈니스의 전략·상황에 따라 다르다. 처한 도메인과 여러 가지 상황에 따라 달라진다.
- 품질 = 제품의 완성도(암시적 요구사항)
- 매번 (목표)달라질 수 있기 때문에, 전략도 달라질 수 있다.
- snow ball ➔ 기술 부채(technical debit)은 완성도 측면에서 복리로 붙는다.

□ 제품의 완성도 (품질의 속성) (가변성보시사)

- 가용성 (고객)
- 변경용이성 (개발자가 편하기위해)
- 성능 (고객)
- 보안 (고객)
- 시험용이성 (개발자가 편하기위해)
- 사용용이성 (고객)

⇒ 뽑아서 품질의 목표를 잡고 전략을 짜야한다.

✓ 결함관리
- 분석 ➔ 어떻게 처리할 것인지 프로세스 확립 + 전략 수립
- 결함 추적은 품질관리 계획에 필수다.

✓ 결함 보고서에 추적되는 정보
- 결함 ID(숫자나 기타 유일한 식별자), 결함에 대한 설명
- **결함이 발생된 과정(reproduce step) ➔ 재현이 가능해야 해결 할 수 있다.**

✓ 테크니컬 리뷰 (ship it)
- 테크니컬 리뷰는 상류 산출물에 대한 결함을 발견하여 수정할 수 있기 때문에, 테스트 만큼이나 비용과 일정 통제에 있어 중요한 작업

✓ 시스템 테스트 : 별도의 조직이 없다면 담당자를 둔다.
- 테크니컬 리뷰와 같은 상류 품질 보증 활동과 하류의 테스트 병행

10장. 아키텍처 (11/3)

□ 아키텍처
- 아키텍처를 보고 발견 - 발명 - 구현을 알 수 있어야 한다.

○ 발견 : 여러 요구사항들 중 기술적으로 난이도 높은 것, 비즈니스 적으로 중요한 것들 ➔ 후보 아키텍처 드라이버(Key) ➔ 아키텍처의 형태에 큰 영향. 그래서 요구사항에서부터 나와야함(난이도 높은 것, 비즈니스 중요한 것은 더 자세히)

○ 발명 : 후보 아키텍처 드라이버 들 중 20개 내외를 결정 ➔ 태틱(tactic)이 있어야 한다. 솔루션(아키텍처 태틱)을 찾음. 하나의 아키텍처 드라이버에 여러개 있을 수 있다.

○ 아키텍처 평가 : ATAM(Architecture Tarde-off Analysis Method)에 따라서 태틱 마다 점수를 매김 ➔ 태틱을 평가

○ 구축 : View Point에 맞춰서 태틱들이 어떻게 SW공학의 용어들로 설명되는지 나타남.
- 정적/동적 , java/.cpp , 구성들 의 형태로 통틀어서 Architecture View
- Allocation View(Development view, Implementation view) – 실제 개발자에게 할당하기 위해서 디렉토리 구조, 패키지 구조가 나와야 함. 프로그래머나 매니저에게 도움을 줌
- 개념적 ➔ Logical View / 성능, 가용성(동적) ➔ Process View : 성능, 가용성 등 runtime 시 확인 할 수 있는 것들

⇒ 4개의 View가 다 필요한 것은 아니지만, Logical View는 개념적인 것이기 때문에 거의 작성한다. 우리가 보고 있는 것은 잡동사니다. Logical View 인데 실제 DB, OS 등이 나옴. 개념적인 것이기 때문에 그것이 무엇인지 정확하게 알 수가 없어야 한다. ➔ 관리의 입장에서, 꼭 View까지 가지 않아도, 아키텍처 드라이버와 태틱으로 개발이 가능하다.

□ 아키텍처는 프로젝트에 대한 기술 구조를 제공
- 전반적인 프로그램 구조(전체적인 static+dynamic view)
- 발생가능성이 높은 변경사항에 대한 아키텍처의 대응방안(관리입장)
- 다른 시스템에서 이미 개발되었거나, 사서 쓸 수 있는 컴포넌트가 기술(개발할 것 – 안할 것 의 상호작용)
- 표준적인 기능(기본적인 기능), 요소들에 대한 설계방안 제시
- 시스템의 요구사항(비기능 속성)에 대하여 아키텍처를 어떻게 수립할지 열거 ➔ 하류에서 발생될 수 있는 잠재적인 비용을 줄여줌.

❍ (1조 10장) 동적인 개발프로세스? : 아키텍처가 단계별 납품방식을 지원.
- 아키텍처(드라이버-발견, 태틱-발명, 뷰-구현)
- 태틱에서 risk, trade off 등을 따져서 개발 어렵고 힘든거 + POC 높은 것부터 단계별로 납품하자(가급적 앞에 하자)

❍ (2조 10장) 아키텍처를 수립할 때 없었던 새로운 요구 사항이 추가됨으로 기능 추가 및 성능이나 사용 편의성과 같은 비기능적 요구 사항이 추가되었다. 비기능 요구 사항과 관련된 품질 속성들은 서로 상충되는 경우가 많은데 이로 인해 기존에 수립된 아키텍처의 수정을 어렵게 할 것으로 생각된다. 어떤 방식으로 해결을 해야 하나요? 아키텍처 수립시점에는 가능하다. 시점에 따라 다를 수 있다.

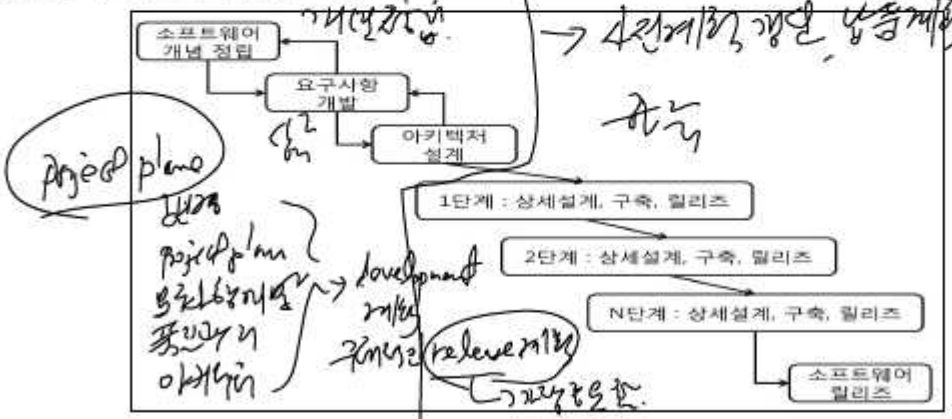
11장. 최종준비 (11/10)

□ Development Plan & Release Plan
- 변경통제, 계획, 품질, 아키텍처 바탕으로 세울 수 있다

⇒ 최종준비 = 사전계획 갱신 + 릴리즈 계획 + 이터레이션 계획(몇번에 나눠서? 릴리즈를 언제할건지? 이번에는 어떤 것을 개발할건지?

□ 요구사항 개발과 아키텍처 이전에 수행되었던 사전 계획을 확장하고 수립한다. 추정 작업을 할 준비를 하며, 가장 중요한 기능을 우선해서 납품하기 위한 계획을 수립한다.

□ 소프트웨어 개념 정립 전에 프로젝트 계획을 세우고, 아키텍처 설계 다음에 개발계획을 가장한 릴리즈 계획을 세운다. ➔ 공식적으로 계획을 2번 세움



❍ (4조 11장) 상용 소프트웨어 추정 도구는 소프트웨어 프로젝트 추정의 객관적 근거로 사용할 수 있으며 추정을 위한 최고의 데이터는 같은 조직이 이전에 완료했던 프로젝트 정보라고 합니다. 하지만 프로젝트는 매번 새로운 환경과 인력으로 진행됩니다. 이런 상황에서 프로젝트 PM과 PL, 개발자들의 역량이 추정에 미치는 영향을 어떻게 고려해야 하나요? 상용 소프트 추정도구를 믿지 말라. 추정은 정말 어렵다. ① 프로젝트 초기의 추정시기 부적절 ② 프로젝트 수행자가 아닌 사람이 추정하는 것은 부적절 함.

✓ 좋은 추정의 특성 ① 지속적·주기적 갱신➔SW불확실성, 현실반영 ② PM,PL이 혼자 하지 않는 추정 ➔ agile(플래닝포커)

✓ 플래닝 포커 : 요구사항 걸리는 시간을 만장일치 할 때 까지 카드에 써서 확인

❍ (5조 11장) 최종 준비 단계에서 실제 상세 설계와 코딩에 대한 일정 수립할 때 참고할만한 중간치 값이 존재하나요? 코딩의 경우, 개인별 활동이므로 개인차가 존재하고, 누가 중간이고 누가 빠른 상태인지 알 수가 없습니다. 어디를 기준으로 코딩 관련된 일정 추정해야하는 것인가요? 혹시 이런 경우에 function point같은 것들이 사용되는 것인가요? 단계를 거듭할수록, 자주, 1단계는 2~4주(상세설계, 구현, 테스트, 회고, 다음준비). 처음 단계에는 추정이 틀리는 것이 당연하다.

Ⅲ. 단계에 의한 성공 ➔ 단계별 납품에 무엇을 해야하나? 실제 실천 practice. 단계 누적 ➔ 성공

12장. 단계 계획 수립의 시작 (11/17)

□ 단계별 계획을 시작할 때는 해당 단계에서 수행될 작업의 상세한 과정에 대한 단계 시작 시의 계획을 수립한다. (상세설계서를 5개의 챕터로 나눠서 더 작은 마일스톤(1일 1마일스톤)으로 나눈다) 프로젝트 진척도를 추적하기 위한 상세 마일스톤 목록을 작성한다. 이를 통해 프로젝트 상태를 쉽게 파악하고, 리스크를 줄일 수 있다.

□ 작은 규모의 프로젝트가 큰 규모의 프로젝트 보다 위험이 적음을 목격했다. ➔ 크고 위험이 많은 프로젝트를 덜 위험한 여러개의 작은 프로젝트를 바꾸는 방법이 바로 단계별 납품 방식이다.

□ 단계별 납품 방식은 개발 팀이 프로젝트를 진행하면서 여러 번 소프트웨어를 릴리즈 가능한 상태로 만들게 한다. 품질 저하의 위험을 줄이고, 상태에 대한 가시성을 높이며, 일정 지연을 예방할 수 있는 방법이다. (제일 좋은 프로젝트 통제 = 매일 데모)

□ 단계별 계획에서 통합 및 릴리즈, 단계 정리 가 가장 중요하다.

□ 상세 마일스톤

- 상세하게 됐다. 안됐다. 를 말 할 수 잇는 정도.
- 목록 : 단계 계획(마일스톤), 상세설계(상세설계서), 구축(java, cpp, cs, jsp..), 시스템테스트(테스트 결과서), 릴리즈 (환경, release note)
- 공통적인 기준 : Done / Not Done
- 쉽게 성취할 수 있는 분량 (주 2회 체크하게 나누는 것이 좋을 것 같다. (데일리 미팅이 중요하다)

기한이 짧은 마일스톤을 사용하면 전체 일정에 영향을 미치는 문제가 발생하고 있는지 즉시 파악할 수 있다. 조기 경보 덕택에 프로젝트 팀은 일정을 조정하거나 기타 방식으로 프로젝트 계획을 조정할 기회를 빨리 가질 수 있다.

13장. 상세설계 (11/17)

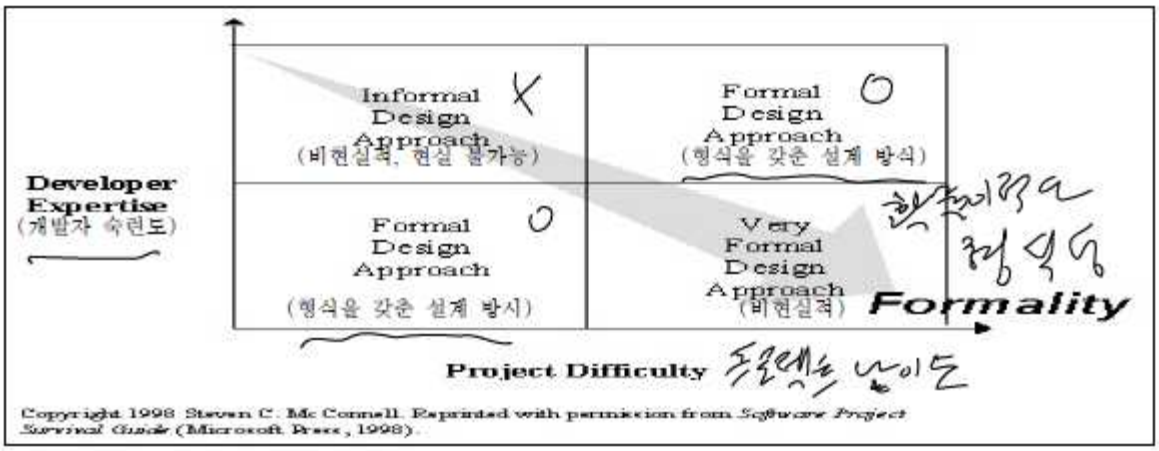
□ 상세설계는 아키텍처 설계를 확장하는 것. 아키텍처와 동일주제를 다루지만 상세하게 처리. 상세설계를 리뷰하면 프로젝트 품질과 비용 측면에서 상당한 효과를 얻을 수 있다.

□ 상세설계 : 아키텍처 view를 변환

- UML, 슈도코드 작성은 거의 코드작성과 다름게 없다
- 주요 태틱의 주요 컴포넌트들은 상세설계(UML, 슈도코드) 할 필요가 있다. 모두 하는 것은 과할 수 있다.

□ 제대로된 아키텍처라면 아직 개발되지 않은 부분에 크게 얽매지 않고도 상세설계를 할 수 있을 것이다. 아키텍처에 따라 작업을 진행하면, 이 작업이 나중에 개발할 부분과도 제대로 동작할 것이라 가정할 수 있다.

□ 얼마나 상세하게 설계해야 하는가?



- 개발자들의 숙련도와 프로젝트의 난이도에 달려있다.
- 상세설계 형식, 프로젝트 난이도, 개발자 숙련도 간의 관계. 개발자의 숙련도가 낮고, 프로젝트가 어려울수록, 상세설계에서 더 많은 형식이 필요하다.(주요컴포넌트에 대해서는)
- 클래스 다이어그램, 시퀀스, state transaction, UML, method 내부 슈도코드, 리턴·파라메터 타입 = 많은형식
- 형식을 갖춘 설계방식을 상세설계 수준으로 가져가는 것이 좋다.
- 이 모듈을 이렇게 짜라는 것이구나 정도가 좋다.

14장. 구축 (11/24)

□ 개발자는 꾸준히 시스템에 기능을 추가하고 일별 빌드 및 스모크 테스트를 수행한다. 구축 단계에서 소프트웨어를 더 간결하게 만들고, 소프트웨어에 가해진 변경을 통제하는 방법을 찾는데 더 많은 주의를 기울인다.

- 구축 이전까지 준비가 어마어마 하고, 준비를 잘해야 구축을 수준이 좋다.
- 구축·구현 계획 ➔ 통합의 계획
- ①기능개발, ②일별빌드 ③스모크테스트 ➔ 구축에서 개발입장의 주요활동
- 스모크 테스트 : 가장 기본, 테스트 라고 하기 힘들 정도로 저급 테스트
- ①변경통제, ②상세마일스톤관리, ③10대리스크, ④지표 ➔ 관리 포인트
- 세니티 테스트 : 나온 결함을 이번에 해결했는지 확인.

□ 지표

- 코드의 양(매일 측정해야함)
- 결함 해결률 : 몇 개중에 몇 개 해결이 좋다 (매일측정)
- 요구사항 개발 완료수 : 적절한 단위로 분할해서 분배

➔ 매일매일 측정하면서 프로젝트를 통제하자. (수작업이 힘드니까 도구를 사용하자)

□ 프로젝트 제1단계에서 시스템 골격을 'T' 구조로 개발하기. 시스템의 전체 수평적 구조와 첫 단계의 기능을 뒷받침할 수직적 구조를 개발한다.

- 시스템의 전체 폭과 부분적 수직 구조로 개발해야 함을 의미. T 자 구조라 하는 이유는 사용자 인터페이스를 수평적으로 전부 개발한다.
- POC – 가용성, 성능, 구현 중요한 것은 초기 단계에 개발하자.

❍ (1조 14장) 프로젝트 관리자가 개발자들이 복잡하게 설계 또는 구현하지 않도록 미리 교육하려면 간결성의 기준을 어떻게 잡아주어야 합니까? 코딩 표준만으로 소프트웨어의 복잡도 조절이 가능한가요? : 코드리뷰, 코딩 표준만으로는 SW복잡도 조절 불가능하다

❍ 코딩표준

① Style : 가독성, Layout 위주

② Idiom : 정적분석을 통해서 나오는 결함들. 효과를 볼라면 이것을 담아 체크하는 툴이 있어야 함.

ex) if (j==2) (X) ➔ if (2==j) (O)

❍ (2조 14장) 많은 branch 머지시 문제발생가능성이 높은데 어떻게 관리할까? : 많은 branch를 가져가는 것이 문제. 적절수준의 아키텍처가 안돼서 그런 것임. 공용클래스를 만들어도 확실하게 나누면 문제가 없다. 할당 역할로서의 아키텍처 view가 안되어 있어서다. ex) Facebook – 1소스. 3일 머지.

❍ (7조 14장) 최근 빌드는 단순 컴파일이 아니다. 컴파일+링크+실행파일+자동화 테스트 통과 = 빌드.

15장. 시스템 테스트 (11/24)

□ 시스템 테스트는 구축과 병행하거나 혹은 절반을 남겨 놓고 실시한다

- 피처 추가전에 버그를 픽스하는가? 조엘 테스트 5번

□ 품질 : 납품할 만큼의 수준. 완성도 ∝ 품질

❍ (1조 15장) 시스템 테스트를 진행하다 보면 원래 설계되었던 프로세스대로 테스트를 진행할때는 문제발생이 없으나, 사용자가 테스트하는 과정에서 예기치 못한 문제점이 발견될 때가 많습니다. 이것을 예측하거나 측정하는 방법이 있는지? 혹은 코딩표준과 같은 테스팅 표준이 있는지? : 없다. 예측 불가능. 단위, 시스템, 알파, 베트 테스트 표준이 있다. TMMI(Test Maturity Model Integration). 경험이 많이 쌓여서 회고를 통해 lesson을 찾는 방법이 좋다.

16장. 소프트웨어 릴리즈 (12/1)

☐ 각 단계를 종료할 때마다 SW를 릴리즈 가능한 상태로 만들자(꼭 릴리즈해야하는 것은 아님)

- 간단한 통계적 기법을 사용하면 릴리즈 결정에 도움을 받을 수 있다.

☐ 릴리즈 시기

① 직감 : 경험을 통해 얻은 감각(과거프로젝트를 통한)

② 양 : 결함이 어떠한가를 가지고 판단

③ 트랜드(경향) : 누적그래프, 전체적인 프로젝트의 추세선을 보자

☐ 코드량비 결함수가 충분하지 어떻게 확인?

(1) 밀도를 측정

- 10만 라인 / 결함 1,000개 = 100라인 당 결함 1개
- A(10만 라인 / 결함 1,000개), B(100만 라인 / 결함 1,000개). 비율상 A에 비해 결함이 너무 적다. 덜 찾았다고 유추가능. 어느정도 파악했는지 확인하려면, 그 양이 차야함. 질의 변화가 일어나려면 양이 차야한다.

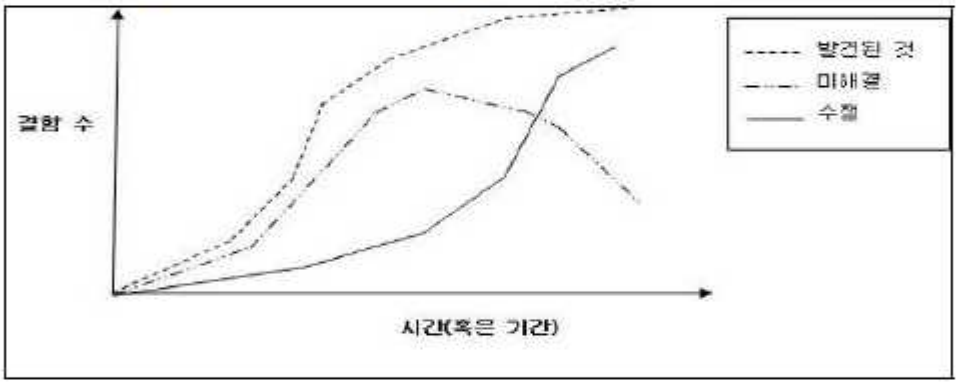
(2) 80:20 분포

- 결함 80%는 특정 20%의 모듈에 몰려있다. (다수의 결함은 소수 모듈에 집중)
- 어느 도메인이나 유사하다. (개발자 역시 20%가 잘하고 80%는 20%를 보조한다)
- 결함 분포를 보자 : reproduce step + 어디 어느 모듈에서 나타났는지?

ex) 파악했는데 골고루다 ➔ 테스트 덜 한 것. 80:20 비슷하게 나옴 ➔ 어느정도 파악 완료한 것 이라 추정 가능.

☐ 복잡도와 결함은 상관없다. 라인수와 결함수는 비례하지 않는다.

⇒ 직감을 형성하기 위해서 양, 트랜드를 보다보면 직감이 생김



☐ 발견된 것. 수정 이 만나는 시점을 예측할 수 있고, 그 시점이 릴리즈 가능 시점이라고 추정할 수 있다.

❍ (2조 16장) 결함 밀도 예측, 결함 풀, 결함 심어 놓기와 같은 예측방법이 실제로 효과가 있는지? : 효과 없다.

❍ (3조 16장) 이번 단계에서 일정을 초과했는데, 다음에 성공과 동시에 전 단계에서 미뤄진 일정을 메꾸기 위해 어떤 전략을 세워야할까? 초기 계획을 세울 때 버퍼를 잡자. 일정 압력(자원 재조정을 통한) ➔ 계획을 팀원들, 개발자들이랑 같이 세워야 현실성이 있다. ①문제의 근원을 찾는다. ②출발점에서 목적지까지의 경로 곳곳에 있는 리스크를 잘 살핀다.

❍ (4조 16장) 릴리스 시기의 결정을 위해 결함 개수, 결함 당 공수 통계, 결함 밀도 예측 등 다양한 기법을 사용하고 있는데요, 측정의 대상이 되는 결함은 기능 요구사항만을 의미하는지 아니면 성능과 같은 비기능 요구사항의 결함도 포함시켜 측정을 해야 하는지? : 완성도의 측면은 품질의 비기능적 요소가 심각하게 관련되어 있다.

17장. 단계 마감 (12/1)

① 총괄 변경위원회 개최

② 추정 재조정

③ 계획에 대한 실적 평가

④ 매체에 저장(SVN)

⑤ 이번 프로젝트 잘했는지 살펴보고 그것을 경험삼아, 다음 프로젝트 대 잘 하자

❍ (1조 17장) "단일 프로젝트내에서는 프로젝트 팀이 마감 시한을 어기게된 근본 원인을 수정하기 어려운 경향이 있다"라고 하는데, 그 이유는 무엇인가요? 이 책이 소개하는 관리방식은 단일 프로젝트들을 위한 것으로 생각되는데 결국 구조적인 원인을 수정하기 어렵다면 이 책에서 요구하는 관리방식이 큰 효과를 보기 어려운것은 아닌지 궁

금합니다. 1) 프로젝트 팀내에서 해결할 수 있는 문제가 아닐 수 있다. 경영진의 지원이나 다른 부서와의 협업등 조직 전체의 유기적이지 못한 지원등이 문제가 될 수 있다는 의미입니다. 2) 여기서 단일 프로젝트 내에서라는 의미가 단계내에서 해결할 수 있는 범위를 벗어 선다고 볼 수 있습니다. 지금 단계 마감이라는 장을 우리가 보고 있죠. 따라서 한단계의 개선으로 끝나는 것이 아니라 애시당초 아키텍처의 누락이나 범위의 문제라면 일정의 지연이 생길 수 있는데 이런 원인들은 해결하기 어렵다는 의미이지요.

- (4조 17장) 순수하게 자체 조직으로만 수행되는 프로젝트와 복수의 협력업체와 협업을 통해 진행되는 프로젝트에서 수집된 로그 정보가 이후의 프로젝트에 교훈으로 등일하게 활용될 수 있는지 궁금합니다. : 협력업체와 앞으로 계속 갈 팀으로 보느냐, 아니냐에 따라 다르다. 외주업체라면 관리 평가가 필요할 것.

18장 프로젝트 이력, 19장 생존 안내서

- (1조 18장) 프로젝트 릴리스 이후 프로젝트의 데이터를 수집하여 리뷰를 하고 이력을 문서에 저장하여 향후 프로젝트를 위해 관리하는 것이 좋은 방법이라고 하고 있습니다. 그러나 프로젝트의 각 단계별 이력을 관리하게 되면 프로젝트의 완성도를 더 높일 수 있지 않을까 생각이 되는데, 각 단계별 이력관리의 문제점이나 더욱 효율적인 방안이 있는지? – 왜 수집하고 리뷰할까? 다음 단계·프로젝트에서 개선하기 위해서 .
- (1조 19장) NASA SEL에서 권장하는 성공적인 프로젝트를 위한 일 9가지와 하지 말아야 할 일 8가지를 기술하고 있습니다. 관리적인 측면에서 보면 내용상으로 위에서 권장하는 일들을 완벽하지는 않지만 실천하는 것이 가능해 보입니다. 그러나 새롭게 만들어지는 프로젝트 팀이 아니고, 기존의 훈련이 되어 있지 않은 조직에서 그 조직으로 지속적이 프로젝트를 해야 되는 상황이라면 프로젝트 성공을 위한 시작을 어떠한 일부터 준비해야 하고 현실적으로 가장 중요하게 체크해야 하는 일들은 어떠한 것들이 있습니까? – 리더가 바뀌면 된다.
- (2조 19장) 책에서는 코드 라인수에 따라 페이지수로의 문서작업이 필요하다는 미 국방성 스타일은 피하라고 합니다. 많은 양의 문서가 꼭 성공을 보장한다고 생각되지는 않지만 프로젝트의 규모, 일정 등은 어느 정도 코딩양에 비례하며 실패하지 않기 위한 안전장치로서의 측면으로 문서가 필요하다고 생각되는데요 케이스 바이 케이스로 프로젝트마다 문서의 종류와 양을 결정해야 하는지 아니면 프로젝트 경험에 따라 어느정도의 표준화되어야 한다면 교수님께서 생각하시기에 어느정도의 문서작성이 되어야 적정한 수준이라고 생각하시는지 궁금합니다. – 문서 종류와 양은 전혀 상관없다. 판단기준은 리스크. 리스크(조직역량, 관리측면, 시장변화 등)에 따라 문서타입, 조직문화 등 여러 가지가 변할 수 있다. 리스크 수시 수집 ➔ 리스크가 이슈로 드러나지 않음. 데이터 기반으로 우리가 무엇을 개선해야 할지파악.
- (3조 18장) 향후 프로젝트에 사용하기 위한 프로젝트 이력 결론 준비중 하나로 계획수립 체크리스트를 만들라 고 나와 있습니다. 여기에는 프로젝트 이력에 언급된 주요내용을 포함하고, 할일과 하지 말아야 할일을 포함해야한다고 나와 있는데요. 여기에 추가할 항목이나, 필요 없어도 되는 항목이 있을까요? 그리고 이 체크리스트의 예시를 알 수 있을까요? – 보완하거나, 필요없는 것을 찾으려는 노력을 하자. 피드백이 중요하다.
- (3조 19장) SEL의 소프트웨어 성공을 위해 해야 할 일중에 팀 정신을 키운다 는 내용이 있습니다. 책에는 공동의 비전을 강조하고, 개인의 책임을 명확하게 정의하면서 전체 프로젝트에 대한 책임도 강조한다고 나와있습니다. 앞의 방법만으로 팀 정신을 키우는 것이 가능할까요? 그리고 더 좋은 방법이나 교수님이 추천하는 방법이 있다면 추천해주세요. – 「탁월한 조직이 빠지기 쉬운 5가지 함정법」 에 나와있다.
- (4조 18장) 소프트웨어 프로젝트 이력의 내용을 보면 아주 많은 내용을 작성해야 하고, 이런 이력을 모두 작성하는 것에 대한 시간과 노력 작성자의 역량 또한 중요한 일이라 생각되어 현실과는 조금 거리가 있는 내용이라 생각됩니다. 이 정도까지의 이력관리를 반드시 해야 할 필요가 있을까요? – 리더가 바뀌면 해결된다.
- (4조 19장) 요즘 화두가 되고 있는 빅데이터는, 다양하고 방대한 규모의 데이터가 미래 경쟁력의 우위를 좌우하는 중요한 자원으로 활용될 수 있다는 점에서 주목을 받고 있습니다. 과거와 비교가 안 될 정도의 대규모 데이터를 빠른 시간 안에 분석하는 것이 가능해졌기 때문인데요. 이런 시대에서, 많은 양의 문서를 필요로 하는 미국방성 스타일을 꼭 피해야만 하는지 궁금합니다. – 많은 양의 문서지만 필요없는 내용이면 무슨 의미가 있나?
- (5조 18장) 향후 프로젝트 진행시 참조하기 위한 프로젝트 이력에 대해 신규로 진행할 프로젝트가 유사점이 있더라도, 투입인원의 변화와 고객과의 접점 변화 등 상황 변화에 대해서는 다른 해석이 필요할 것입니다. 이 경우 달라진 변수에 대해 일정, 공수, 시간 데이터 등의 가변적 측정을 객관적으로 보증할 수 있는 데이터 기준이 있을까요? – 가정부터가 잘못
- (5조 19장) 소프트웨어 성공을 위해 하지말아야 할일 중 미치게 될 영향을 평가해 보지 않은 상태에서 수정을 가하지 말라는 부분에 대해 비기능 사항 중 안정성 등 영향 평가가 애매한 부분이 있을 것으로 보입니다. 이 경우

영향평가를 객관적으로 측정할 수 있는 방법에는 어떤것이 있을까요? – 영향평가 못함

❍ (6조 18강) 프로젝트 하다보면 프로젝트 이력을 관리한다는 것 자체가 시간을 많이 소요되는 업무다. 뭔가 시스템적으로 관리해주는 Solution있다면 좋은 것 같은데 추천할만한 것이 있을까? – 지라, 레드마인

❍ (6조 19강) 성공적인 소프트웨어 개발을 위해서는 프로젝트 인력에게 권한을 부여하라고 되어있습니다. 생산성 높은 환경을 조성하고,명확한 책임을 부여하라고 되어있는데, 좀 더 구체적으로 어떤것이 생산성 높은 환경이고, 책임은 어떤식으로 부여하면 효과적인지 알고 싶습니다. – 짧은 구간 결과물로 피드백하자.

> 전 범위가 아래 요약으로 정리가 됩니다. 반드시 읽어 봅시다.(암기)
: 교수님 기말고사 전에 준 자료입니다.

□ SW프로젝트관리 요약

- SW 프로젝트 관리는 3가지(SW, 프로젝트, 관리)로 나눠 이해해야 합니다.
- SW는 원래 복잡한 성격을 가집니다. 프로젝트는 돈을 버는 것과 연결해야 하며 제품을 만드는 공장을 기억해야 합니다. 관리는 힘쓰고 애써야 하는 활동입니다.
- SW 작업을 살펴보니 지적인 작업이 대부분이었고 사무적인 작업은 일부였습니다.
- SW 관리의 포인트는 SW 공장의 공정능력을 높이고 그 공장이 지적인 작업에 집중할 수 있도록 해주는 데 있습니다.
- 지적인 일에 집중하도록 사무적인 일들은 기계(컴퓨터)에게 맡깁니다.
- 그런 활동들이 sandbox, tool chain, infra 환경 등을 본격적인 코딩시작 전에 준비해야 합니다.
- SW공장의 공정능력을 높이기 위해 두가지 방법이 있는데 원래 뛰어난 사람을 데려오던가 아니면 기존 사람들 간의 협업을 잘 하도록 하는 것입니다.
- 원래 뛰어난 사람은 찾기 어려우므로 현실적이지 않은 방법입니다. 그래서 우리는 일반적인 사람들 간의 협업을 잘 하게 만드는데 초점을 주면 됩니다.
- 어떻게 협업을 잘 하게 할까요? 이 답을 우리는 다음 그림에서 답을 찾았습니다.
- Cmmi staged model인데요. Level 2 를 달성하지 않고는 level 3가 될 수 없다입니다.
- 관리가 되지 않으면 표준 프로세스를 정의 할 수 없다. 관리를 한다는 것을 생존전략에서 이렇게 설명합니다.
- 생존과 직접적으로 관련된 항목은 요구사항, 리스크, 인력, 계획, 통제입니다
- 요구사항은 물어보고, 보여주고, 물어보는 방식을 취하는 것이 좋은 것이며 이를 위해 프로토타입을 적극적으로 활용하면 좋습니다. 그러나 품질요구사항도 중요하니 품질속성 시나리오에 의해 작성할 필요가 있습니다.
- 계획은 프로젝트계획과 릴리즈(개발)계획으로 나눠서 세우는 것이 필요합니다. 전체적인 프로젝트의 전략은 상류와 하류로 나눠 상류에 집중할 수 있도록 수립 후 하류에서는 단계별로 납품을 할 수 있도록 합니다.
- 통제는 프로젝트의 가시성을 높이는 것이 중요한데 납품과 상세 마일스톤 체크 방식이 있습니다. 납품은 실제 SW 동작을 보는 것이고 상세 마일스톤이라는 것은 binary milestone이라고 볼 수 있습니다.
- 리스크관리는 10대 리스트 목록을 활용해서 프로젝트가 종료될 때까지 민감하게 리스크를 식별하고 대응 방안을 마련하는 것입니다. 주의할 점은 늘 목록에 살아있는 리스크가 10개 유지하도록 노력하는 것입니다.
- 인력은 준비되지 않은 인력을 서둘러 꾸려서 하기 보다는 오히려 인력이 준비될 때 까지 그냥 진행하는 것이 현명한 방법입니다. 프로젝트관리자는 일반 개발자와 기술적인 논의를 할 수 있는 수준의 follow up을 하고 있어야 합니다.