게임 프로젝트 관리: 개발 방법론

이유: 일 좀 똑바로 해 보자 = 똑 같은 일을 하더라도 빨리, 또는 좋은 결과물 또는 비용 절감

예산을 잘 지키고, 기간을 잘 지키고 목표한 것을 잘 지키는 것 = 일 잘하는 것

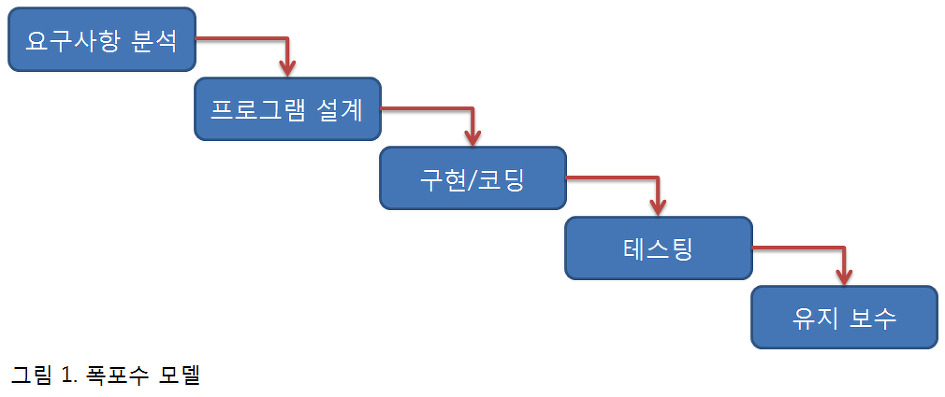
현업 갔을 때 기간 안에 일을 시켰으면, 일단 결과물은 나와야

개발 방법론의 궁극적 목표

1. 스케줄을 지키면서
2. 적정한 예산 범위도 지키고
3. 사용자의 필요와 욕구를 만족시킬 수 있는 결과물을 만들어 내는 것

폭포수 방법론: Cascade, Waterfall (캐주얼 게임 프로젝트 주로 사용)

건축 등의 프로젝트에서 선호되던 방법론을 소프트웨어 개발에 이식



건축 프로젝트와 게임 개발 프로젝트의 결정적 차이

* 차근차근 잘 만들었는데 다 만들고 나니 재미가 없다
* 인간이 건축을 한 역사 vs 인간이 게임을 개발한 역사
* 건축: 역사적으로 수많은 실패 경험, 표준 공정, 프로젝트 기간과 비용 단축시키는 기법, 부품이 있다. Vs 게임은 표준화된 공정이 따로 정해진 게 없다.

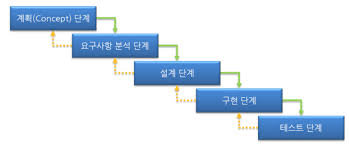
건축은 대략적인 소모 기간, 비용 예측이 정확히 가능하지만, 게임은 작은 것조차 시대의 흐름에 바뀌기 때문에 정확히 예측 불가

폭포수 방법론의 장단점

* 장점: 습득과 이해가 쉽다. (요구 사항을 철저히 분석해서 기획을 다 끝내놓고 그 다음에 설계한 대로 작업 들어가니까) 진행 상황에 대한 정확한 파악 및 관리, 막강한 효율성, 짧고 단순하고 명확한 요구사항을 가진 프로젝트(볼륨이 작은 프로젝트)에 유리(다른 고민 할 필요 없음 🡪 업무 집중도가 높다)
* 단점: 부족한 유연성, 리스크와 상황 변동에 매우 취약, 특히 프로젝트 후반부에 근본적인 문제점 발생 시 대처하기가 매우 힘들다

폭포수 방법론 개선 버전(사시미 모델)

* 각 단계 별 피드백을 통해 각 단계의 전 단계로의 전환을 유연하게 한다
* 단계 중간중간 산출물의 확인 및 검증 활동 강화



나선형 모델: 큰 회사들의 대규모 프로젝트

변화에 취약한 폭포수 모델의 단점을 개선한 모델

시스템 개발 시 생기는 위험을 최소화하기 위한 모델



클라이언트의 요구 사항을 최대한 반영하면서 만들 수 있다. (프로젝트 결과물의 유연성이 매우 올라간다)

요구사항이 불명확하면 계속 끝없이 돌아간다

나선형 모델의 장단점

* 장점: 정확한 요구사항 파악, 위험 부담 감소, 개발 과정에서의 만은 문제점 해결(프로젝트의 근본적인 문제점도 놓치지 않고 손댈 수 있다)
* 단점: 매우 많은 시간이 소요될 위험 부담(이 프로젝트가 10년째 끝나지 않아!)

Agile 개발 방법론

”프로젝트를 쪼개자“

작고 반복적인 개발 주기를 통해 제품을 개발하는 방법 🡪 요구 사항을 정확히 반영할 수 있다.

폭포수, 나선형 모델에서 발생하는 계약 주체 상호 간의 리스크를 완화하기 위해 발생하는 불필요한 문서 작업 줄여서 개발 자체에 집중할 수 있도록

너희가 원하는 결과물을 만드는 족족 수시로 보여줄게

**목표, 요구 사항, 개발 환경**이 수시로 바뀌는 프로젝트를 위해 고안된 프로젝트 관리 방법

국내외 많은 게임 프로젝트에서 애자일 방법론을 도입하고 있다.

애자일 방법론의 우선 순위

1. 공정과 도구보다 **개인과 상호작용**
2. 포괄적인 문서보다 **작동하는 실제 소프트웨어**
3. 계약 협상보다 **고객과의 협력**
4. 계획을 따르기 보다 **변화에 대응**

애자일 방법론의 기본 프로세스

1. 규모 추정: 어느 정도의 개발 볼륨으로 만들 것이냐, 요구 작업은?
2. 우선 순위 결정: 추정한 규모 내의 일에서 가장 핵심 되는 일의 순위 설정
3. 일정 배치: 무조건 한 일정 안에 끝낸다

애자일 개발 방법의 원칙

1. 애자일 팀은 하나가 되어 일한다: **공동의 목표를 항상 공유**
2. 애자일 팀은 최소 시간 단위로 작업한다: 1주 (또는 1일)
3. 애자일 팀은 최소 시간 단위의 종료 시 **결과를 내놓는다**
4. 애자일 팀은 수시로 프로젝트를 점검한다
5. 애자일 팀은 유저 중심의 설계 및 기능 구현을 한다(문서 중심 X)

애자일 방법론: 스크럼, 스토리, 스프린트

1. 스크럼(작은 개개인이 하나로 모여)

* 애자일 방법론 중 하나로 반복적이고 점진적인 소규모 팀 중심의 개발 방법론. 작은 목표를 주기적으로 개발하고 전달하는 방식

1. 스토리(제품 백로그 / Product Backlog)

* 사용자에게 보여지는 요구 사항들의 구현 목록(예: 캐릭터가 점프하게 해 주세요)

1. 스프린트

* 반복적인 개발 주기를 지칭
* 최소 단위의 완결된 개발 단위(3, 4주가 가장 선호된다)

스토리 구성 항목

1. ID: 스토리의 구분자
2. 이름: 스토리의 짧은 정보
3. 중요도: PM이 생각하는 스토리의 중요도(클수록 중요하다)
4. 추정치: 맨 – 데이(1명이 1일 동안 작업할 수 있는 업무량) 기준으로 설정한 해당 작업의 업무량
5. 데모 방식: 해당 스토리가 제대로 구현되었다는 것을 보여주는 방식
6. 참고: 참고 사항에 대한 간단한 기록

예시)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | 이름 | 중요도 | 추정치 | 데모 방식 | 참고 |
| 1 | 포털 구현 | 30 | 1일 | 포털 스크롤을 사용하여 구현 | 랜덤 위치 지원 |
| 2 | 클래스 전직 | 15 | 2일 | 전직 퀘스트를 통해 클래스 변환 | 클래스 별 퀘스트 구현 |

스프린트 계획

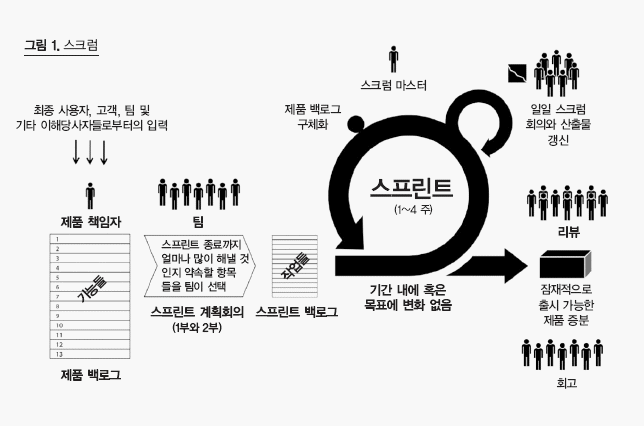
* 스프린트 길이 결정 🡪 스프린트 목표 결정 🡪 스토리를 최소 작업 단위로 분해 🡪 해당 스프린트에 구현할 스토리 고르기
* 작업 속도 결정 시 **집중도** 요소 감안
* 스프린트 진행: **일일 스크럼 회의**를 통해 일일의 업무를 체크하며 진행
* 스프린트 데모: 스프린트 목표의 완료를 보여주는 데모 진행
* 스프린트 회고: 스프린트 완료 후 있었던 일의 회고

추정 속도와 실제 속도의 차이, 스토리 별 만족, 반성 개선 사항 정리

* 스프린트 간 휴식: 전력 질주 시 하지 못했던 작업 진행

스크럼 목표

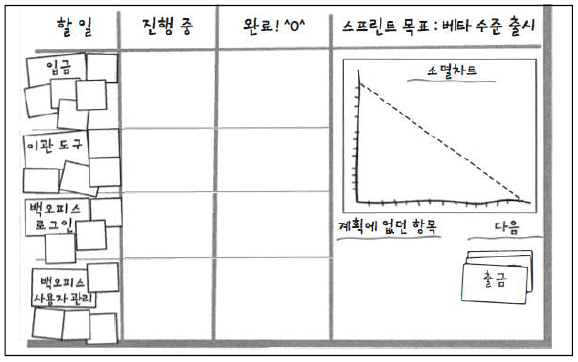
“업무를 잘 나눠 목표와 실제 작업 간의 간극 최소화”



스프린트 백로그

작업 진행을 관리하는 현황 판

* **할 일, 진행중, 완료, 스프린트 목표, 소멸 차트**로 구성



일일 스크럼 회의

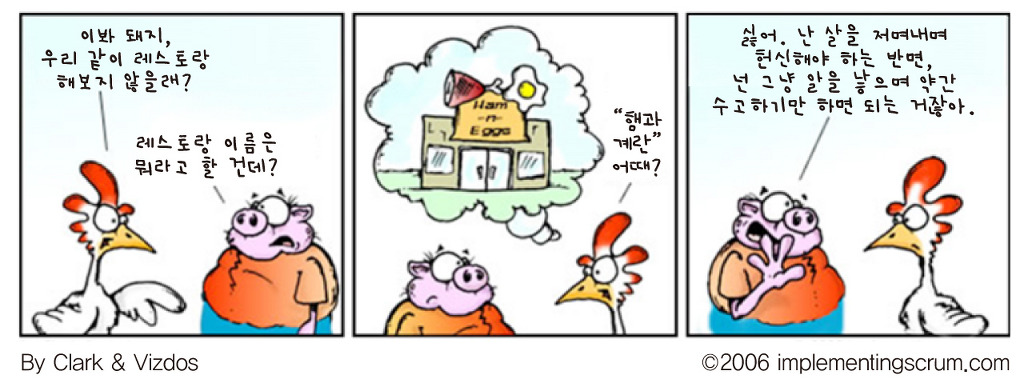
1. 일일 스크럼 회의에서 할 일

* 작업 현황 판 업데이트, 오늘의 업무 체크(한 일, 할 일, 문제점)

1. 매일 일과 시작 전 15분 이내 짧은 미팅으로 진행

주의점

돼지와 닭이 합쳐 레스토랑을 열려고 한다면?



* 스크럼에서 스크럼 팀, 프로젝트 책임자, 스크럼 마스터를 제외한 모든 인원은 닭이다.

스크럼의 단점

여전히 추정 일정에 의한 오류가 발생

* 게임 개발에는 정량적으로 판단할 수 없는 업무가 자주 발생
* 발생한 오류로 인한 초과 근무가 자주 발생(결국 야근)

스프린트 관리 어려움

* 소규모 조직의 경우 스프린트 기간의 변동이 매우 큼
* 개발 우선 순위 변동의 변화가 심하여 스프린트 변경 작업이라는 업무 부담이 자주 발생

칸반(간판)의 특징

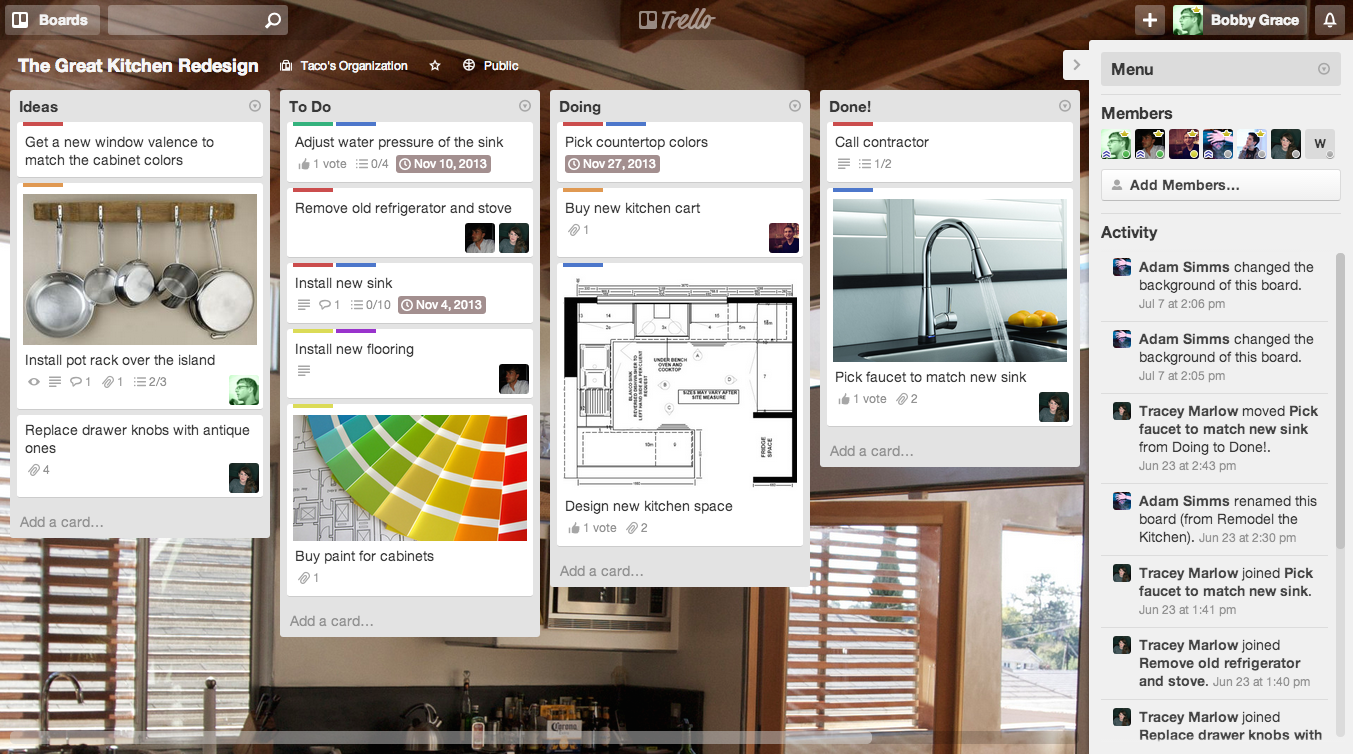
1. 업무 중심: 시간 중심의 개발 관리가 아닌 업무 중심의 개발 관리
2. 업무 시각화: 업무의 흐름을 시각화하여 업무 흐름 공유
3. 업무 제한: 업무의 각 단계마다 동시에 진행 가능한 업무 수 제한

* 너무 낮은 WIP 🡪 개발자 유휴 🡪 나쁜 생산성
* 너무 높은 WIP 🡪 태스크가 유휴 🡪 나쁜 리드 타임

1. 흐름 관리: 측정을 통한 시스템 문제점 인식

협업 관리 도구

* 트렐로: 칸반 기반의 협업 관리에 편리한 도구



프로젝트 관리 항목

일 잘하고 있니? 🡪 스케줄을 잘 준수하고 있냐

어떤 목표의 프로젝트를 하고 있는 지 파악

목표를 달성하려면 어떤 것들이 필요하고 지금 뭐가 있는지 파악

목표를 달성하려면 얼만큼의 인력과 시간이 필요한 지

결과물 산출 + 결과물 검토를 통해서 일 잘하고 있는지 파악

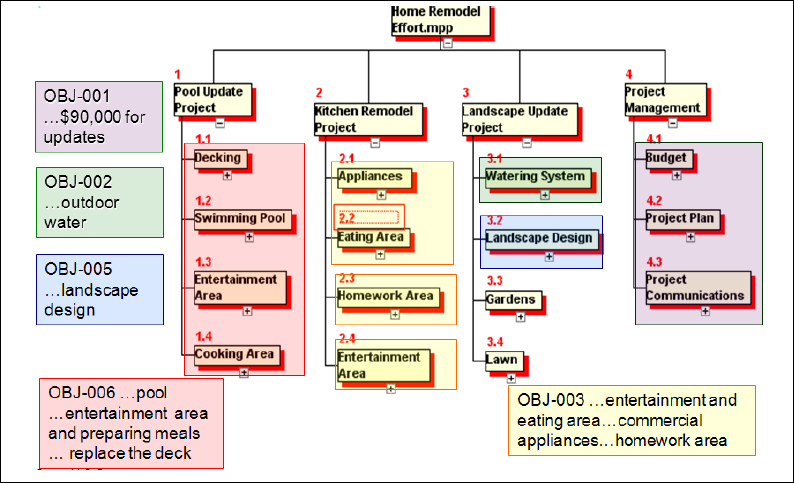
스케줄링의 순서

1. 업무 쪼개기 🡪 한 작업을 하기 위해서 필요한 역할들을 나누는 작업
2. Man – Month(1명이 한 달 동안 일을 할 수 있는 업무량) 추가
3. 배치 🡪 업무들을 어떤 순서로 어디에 속하는 업무인지, 누가 할 수 있는 업무인가 배치
4. 필요 인력 및 기간 산출

업무 쪼개기

WBS (Work Breakdown Structure)

* 프로젝트 전체를 작은 작업 단위로 구성한 **작업 분해도**

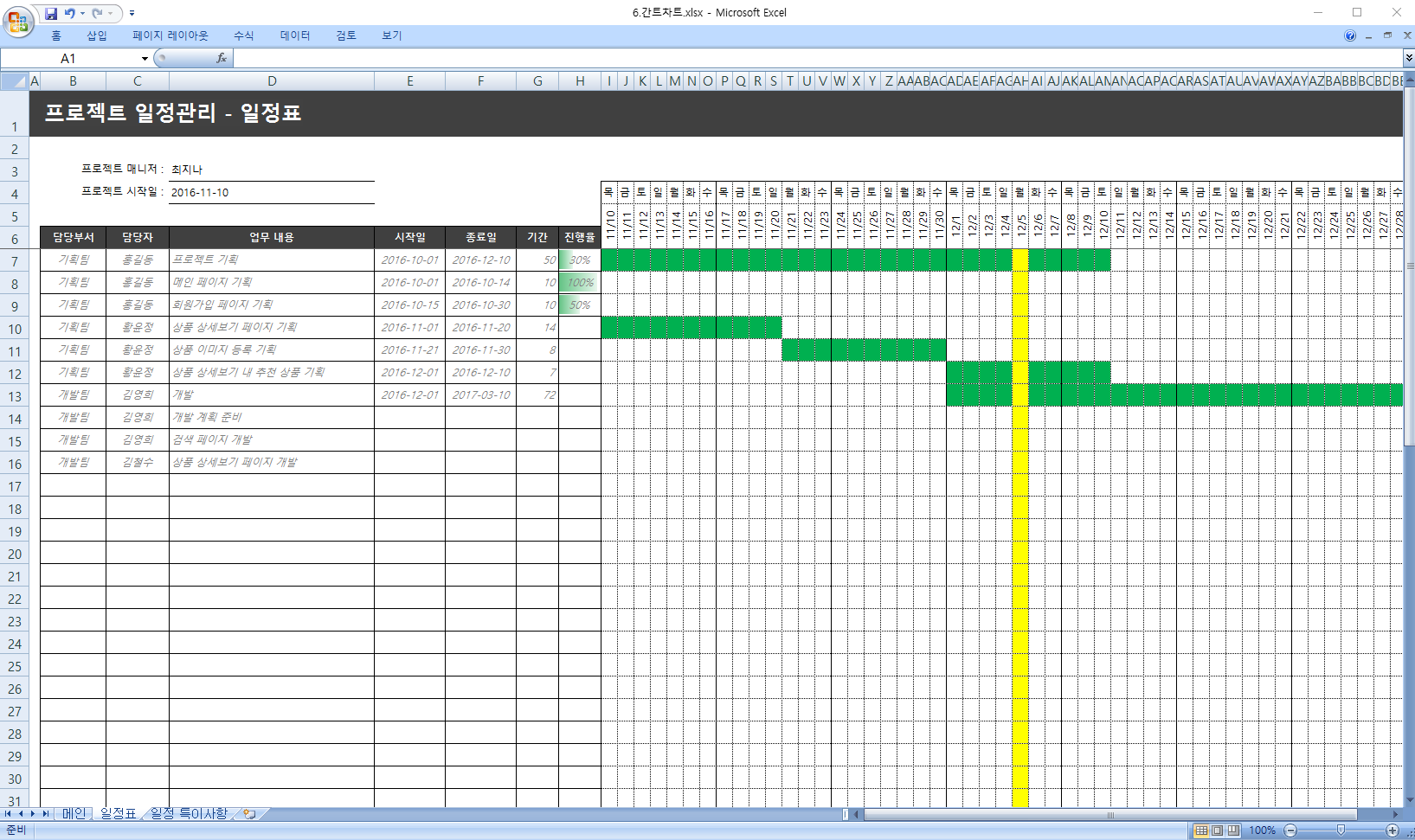


업무 진척도, 누구에게 일을 분배해야 되는지에 대한 파악은 어렵다는 단점

Gantt Chart

**시간의 흐름**에 따라 작업 내역을 직관적으로 판단할 수 있는 스케줄링 차트

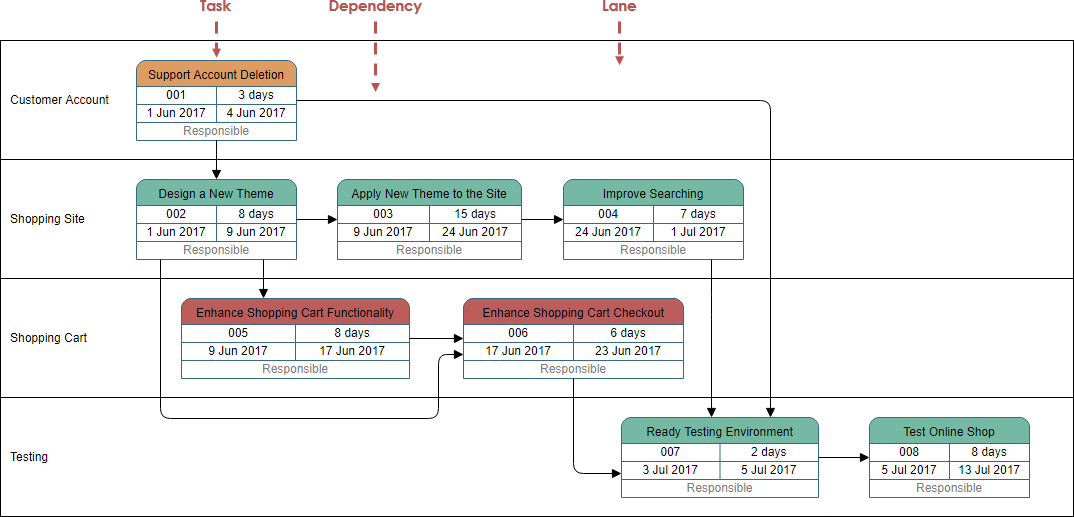
구성: **업무명, 기간, 작업자, 연관 작업**



퍼트 차트

프로젝트 달성에 필요한 전 작업의 내용과 순서, 작업 간 연관을 구조적으로 파악할 수 있는 스케줄링 차트

구성: **과제 명, ES(최대한 빨리 시작했으면 하는 시간), EF(최대한 빨리 끝냈으면 하는 기간), 작업 기간, 작업 담당자, 연관 작업**(필요에 따라 **LS(늦어도 시작해야 되는 시간), LF(데드 라인)** 추가)



너무 디테일하지도 너무 크게 잡지도 말아야 한다

스케줄 관리 Tool

1. MS Project
2. Gantt Project, OpenProj, WBS Schedule Pro
3. 레드 마인(Red Mine)
4. 구글 캘린더
5. Excel (XLGantt 참조)

스케줄 관리 시 주의점

1. 근무시간 = 실제 업무 시간은 아니다 (제외할 시간: 전체 개발 회의, 당사자 간의 회의, 자료 조사, R&D 시간 등)
2. 최 하단 유닛 단위의 업무는 무조건 시간 단위로 환산 되어야 한다
3. 일 단위의 스케줄은 무조건 지킬 것(오늘 일은 오늘 안에)
4. 업무 시간 산정을 방해하는 요소를 최소화 할 것(가급적 계획된 시간에 회의 진행할 것)

* 예고되지 않거나 업무 시간 중간에 위치한 회의
* 불쑥 나타나기: 내가 어제 게임을 해봤는데 말이야

프로젝트 관리 항목

체크 포인트와 마일 스톤

1. 체크 포인트

* 개발 중간에 개발 속도와 방향을 점검하는 단계
* 프로토타입, 알파 테스트, 베타 테스트 등

1. 마일 스톤(중요 시점)

* 각 체크포인트의 완성도를 체크하는 **가이드 라인**

마일 스톤이란?

1. 중요한 목표 또는 업무별로 완성해야 되는 시점
2. 시점은 하나만 있는 게 아니다(중요 시점 모두 표기)
3. 프로젝트의 전체적인 그림 또는 일정 파악 시 유용
4. 하나의 마일 스톤이 종료되면 해당 마일 스톤 검증한다

마일 스톤의 검증

1. 마일 스톤은 최대한 정확한 검증이 가능하도록 작성되어야 한다
2. 검증은 테스트 케이스(TC)를 이용하여 진행하는 것이 좋다

* 예시) 회원 가입은 잘 되는가? (X)
* Explorer, Chrome, Firefox, Safari에서 회원가입 폼이 잘 표시되는가? / 사용 가능한 아이디 찾기가 정확히 작동되는가? / 가입자의 휴대폰으로 인증번호가 5초 안에 전달되는가 등 (O)

변경 통제

원래 계획에서 **추가, 삭제, 수정**해야 되는 사항의 관리

타 프로젝트보다 게임 프로젝트는 외부, 내부 환경의 변화에 따라 프로젝트 변경이 불가피한 경우가 많다.

프로젝트 변경의 영향을 받는 **당사자**가 의사 결정에 참여해야 한다.

* 예시: 우리 마케팅 회의에서 하드코어 모드 넣기로 했어. 다음 주까지 작업 끝내!

프로젝트 변경은 **정해진 기간**에 **공식적 절차**에 의해 진행되어야 한다

* 예시: 사장님이 시도 때도 없이 개발실에 들어와 이런 저런 훈수를 두고 간다.

리스크 관리

리스크

* 프로젝트 목표에 긍정적(Positive)이거나 부정적인(negative) 영향을 미치는 사건
* 리스크 = 위기 + 기회
* 리스크를 바라볼 때 기회와 위험을 모두 고려

예상되는 리스크를 미리 체크하여 콘트롤 및 대안을 마련하는 것

리스크 체크리스트: 인력

1. 인력 수급 및 변동 전반: 생각보다 좋은 인재가 왔다. 어떡할까
2. 인력 충원: 신규 인력을 충원 했는데 생각보다 역량이 부족해 어떡하지?
3. 인력 퇴사: 사람 한 명 나갔다. 어떡하지?
4. 기타 사고, 소속 변경 등: 팀원 중 한 명 교통사고를 당했다 / 다른 회사로 갔다.

리스크 체크리스트: 요구 사항 변동(게임 같은 경우는 특히 요구 사항 변동이 매우 심함)

1. 사용자(유저, 경영진, 퍼블리셔, 투자자 등)의 다양한 추가 요구
2. 사용자의 변덕

리스크 체크리스트: 주변 환경

1. 개발 툴 및 엔진의 버전 업
2. 경쟁 작 출시
3. 오픈마켓 정책 변경: 구글 플레이스토어에 더 이상 32비트 환경에서 개발된 앱은 못 올린다.
4. 외주 사의 개발 난항 🡪 잘 된 작품을 만든 사람이 없거나 또는 의사소통이 안 되거나

리스크 체크리스트: 비효율적인 업무 환경

1. 노후 된 장비: 저 사양 장비라서 발생하는 문제 또는 노트북이 누구는 있고 누구는 없고
2. 외부 간섭: 사장님이 은근슬쩍 와서 훈수를 두고 가거나 다른 부서 사람이 추가 자료 던지고 가거나
3. 개발 외 업무 부하: 연이은 회의, 흐름을 끊는 다거나, 할 거 다 정의해 놨는데 뜬금포 날리기

리스크 체크리스트: 기타

1. 상위 단계 마일 스톤을 위한 하위 마일 스톤 항목의 변화
2. 자금 상황 = 쓸 수 있는 시간이 줄어든다(어디서부터 포기할 거야)
3. 비현실적 일정: 팀원들의 역량에 대한 평가가 잘 안 이루어졌을 때
4. 팀 내 불화: 결과물 가지고 계속 태클 걸기, 회의에서 감정 소모만 계속 될 때

인력 관리 🡨 사람을 어떻게 관리할 것이냐 에서 출발: 이 사람들이 이런 것들을 적극적으로 활용하고 더 잘 끌어낼 수 있을 것인가 에서 시작

* 게임 프로젝트는 대표적인 **노동 집약** 프로젝트
* 게임 프로젝트에서 사람은 **도구가 아닌 주체**

인력 관리 시 체크해야 될 항목

1. 멤버들의 능력 향상 및 행복한 개발 환경 조성(장비가 주는 효율도 포함)
2. 인력 투입의 적정성: 그 일에 적합한 사람들이 어느 정도 포진되어 있나, 선호도 등
3. 문제 팀원 및 문제 상황에 대한 통제
4. 과제 – 인력 간 지시 체계 및 책임 관리

요구 사항 관리 및 품질 보증

테스트, 버그 관리를 포함한 전반적인 QA 과정(게임은 재미 있는 지, 프로젝트가 우리가 요구하는 것에 적합한 지)

QA는 별도 강의를 통해 학습

좋은 프로젝트 = 좋은 결과물

프로젝트 이력 관리

마일 스톤 또는 프로젝트 완료 후 기간, 사건 등 해당 프로젝트에 대한 이력을 기록

다음 프로젝트 또는 타 팀을 위한 참고 자료로 사용, 이력 관리에는 다음 항목을 포함

* 프로젝트 개요: 프로젝트 명, 일정, 참여 인원 등 기본 정보
* 프로젝트 데이터: 원래 스케줄과 실제 개발 일정 비교, 참여 인력 및 상세 업무 기록, 예산 기록
* 프로젝트 리뷰, 평가 및 반성