ORACLE®

Application Life cycle Management Framework

Principle Consultant
Byungwook Cho C.
(byungwook.cho@oracle.com)

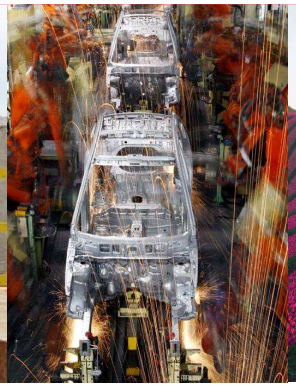
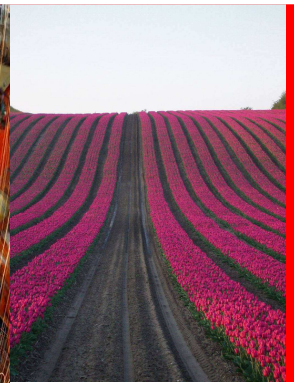
ORACLE PARTNERNETWORK

ICE BREAK

AGENDA

- 소프트웨어 개발의 현실
- ALM
- ALM sample view
- ALM case study
- ALM 구축 전략

소프트웨어 개발의 현실

소프트웨어 개발
우리가 원하는 것

Just
“ 체계화된 관리와 잘 정립된 개발 프로세스
완성도 높은 소프트웨어”

소프트웨어 개발
그러나 현실은?

개발자의 현실

- 밀려드는 요구 사항
- 회의.. 보고서.. 교육..
- 의사소통 문제
- 한 사람에게만 몰리는 일들
- 추적되지 않는 작업들

관리자의 현실

- 개발 진척률은 항상 99.999%
- 항상 밤은 세는데 무슨 일 하지?
- 누구한테 이 일을 줘야 할까?
- 제품 출시는 언제 해야되지?
- 지금 문제점은 무엇일까?
- 그 이슈는 어떻게 진행되고 있지?
- 지금 중요한 일들이 무엇이지?

소프트웨어의 현실

- 짚으로 만든 집
- 개발 완료 후 테스트 단계에서 발생하는 문제
- 코드 변경에 대한 검증 없음
- 빅뱅 방식의 통합

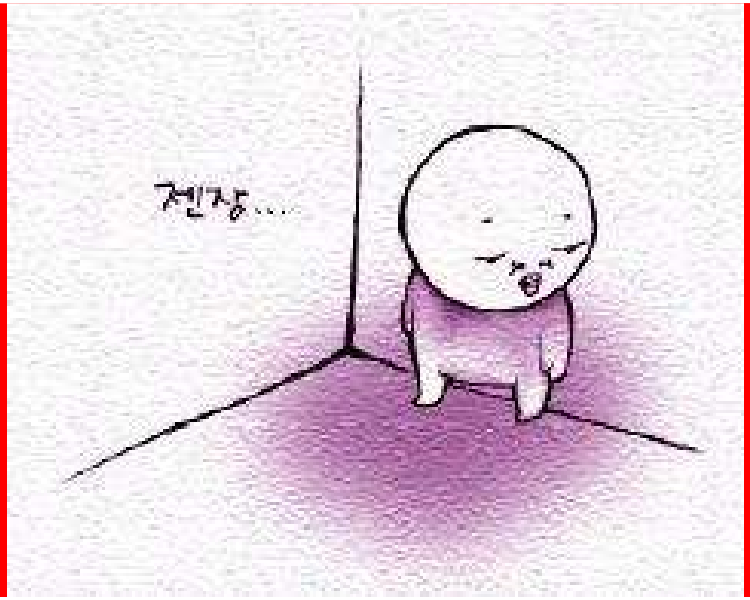
아기 돼지 삼형제
'짚으로 만든 집'

그래서 …

천재들이 만든 각종 방법론과 도구들

- RUP,CBD,CMMI
- **각종 이론들**

여러분의 대답은?

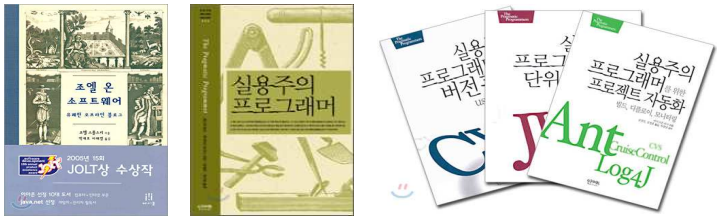


천재들은 소수일 뿐

- 이론은 이론일 뿐
- 좀더 현실적이고 실용적인 방법론 필요

대안은?
실용주의 개발 방법론

- 실용주의 개발 방법론
  - Erich Gamma, Joel Spolsky, Kent beck, Andrew Hunt
  - Iterative & Incremental
  - Agile
- 기존 방법론과의 차이
  - 기존 방법론은 각 단계별로 문제와 변경이 없음을 가정함 (WATER-FALL)
  - 문제가 있음을 인정하고 잦은 테스트로 문제를 빨리 발견하는데 초점을 둠
  - 요구사항 변경이 있음을 가정하고 변화를 수용하는 데 중점
  - 협업과 의사소통에 중점



ALM Application Lifecycle Management

Application Lifecycle Management

- Definition

The coordination of development life-cycle activities, including requirements,modeling,development,build,and testing, through

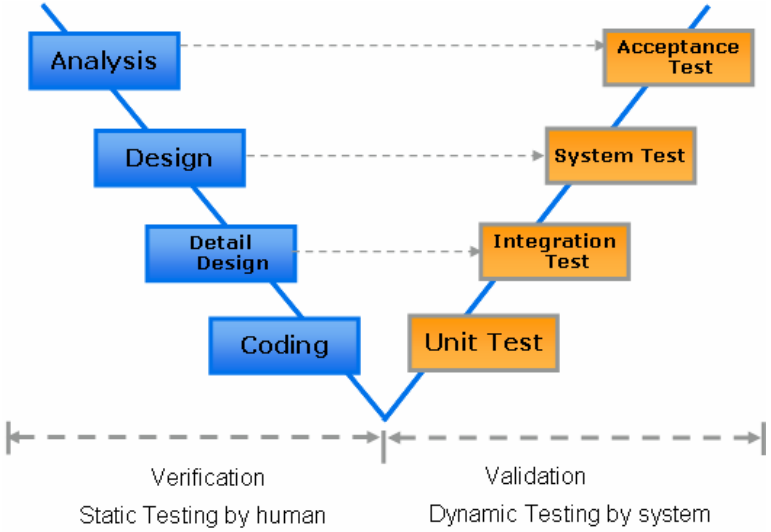
- **Enforcement of processes that span these activities**
- **Management of relationships between development artifacts used of produced by these activities**
- **Reporting on progress of the development effort as a whole**

by Forrester

- ALM
  - 소프트웨어 개발에 관련된 전과정에 대한 관리,통제와 모니터링
  - 개발도구 회사로 부터의 마케팅 메시지?
  - SDLC(Software Development Life Cycle)과 차이? BIZ

Development Process Model

- V Model
  - 전통적인 water-fall model의 발전형
  - Verification & Validation

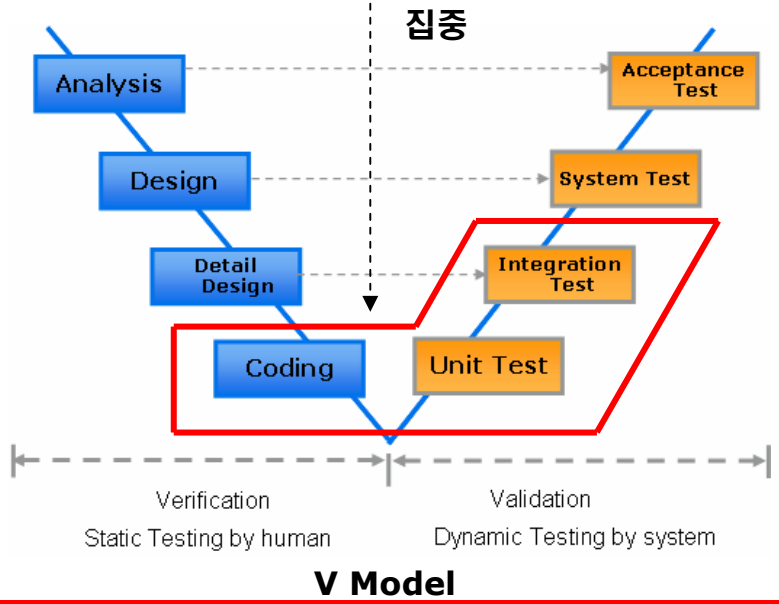


Verification	Validation
To verify the artifacts that has been produced in each development cycle.	Valid & estimate the system
Review & Inspection with **artifacts** from each development step	Testing with **system**
Static testing	**Dynamic** testing

Our Approach

- 실용주의 개발방법론에 근거한 ALM 프로세스
  - 점진적 통합
  - 일일빌드
  - 하위레벨 테스트 강화와 자동화
  - 이슈 추적
- 우리 접근 방법
  - 실용주의 개발방법론 + V Model (SDLC) + Agile 방법론
  - 개발과정과 이슈 기반의 통제에 집중
  - RISK 조기 제거 및 품질 향상을 통한 프로젝트 성공을 목표로 함
  - 이슈 추적을 통해서 작업을 관리 및 통제

실용주의 사상



V Model

개발 가상 시나리오

- 개발자의 코딩
  - 개발자 D씨는 아침에 출근해서 이클립스 IDE를 오픈한다.
  - 이클립스 IDE는 이슈추적툴로 부터 D씨에게 할당된 작업들이 리스트업되고, 그중에서 오늘해야 할 작업을 선택하여 PROGRESS로 상태를 변경한다.
  - SCM으로 부터 최신 코드를 UPDATE받고 코딩을 시작한다.
  - 코드를 만들고 테스트 케이스를 작성하여 코드가 제대로 작동함을 확인하고, 커버러지 분석을 통해서 금일 코딩한 내용이 테스트에서 모두 확인 되었는지 체크한다.
  - 오늘 코딩한 내용을 INSPECTION툴을 통해서 잠재적인 문제가 있는지 없는지 검증 받고 NAMING RULE등이 문제 없는지 확인한다.
  - 완료된 내용을 SCM에 작업 번호와 함께 COMMIT한다.
  - 자동 빌드 머신에서 COMMIT된 소스코드를 감지하고 모든 소스를 내려받아서 빌드를 완료한후에 개발 서버에 자동으로 배포하고 테스트를 수행한다.
  - 테스트가 실패한 경우 이전 버전으로 개발 서버를 원복 시키고 모든 개발원과 PM에게 이메일로 테스트 실패 사실을 통보한다.
  - PM은 테스트 실패사실을 이메일로 통지 받고, 이번 빌드에서 변경된 부분을 빌드 자동화 시스템을 통해서 확인하고 빌드 자동화 시스템에 의해서 리포트된 내용에 따라 누가 어느 모듈을 수정했는지를 찾아서 해당 개발자에게 수정을 지시한다.
  - 개발자는 수정을 마친후에 다시 SCM에 소스를 반영하고 빌드 자동화 시스템은 빌드,테스트,커버러지 분석,코드 복잡도 분석,INSPECTION작업을 수행한다.
  - PM은 빌드가 완료된 결과를 통보 받고, 복잡도가 높은 클래스 모듈 10개에 대해서 제대로 테스트가 커버하는지 확인하후에 미비한 부분에 대해서 담당 개발자에게 테스트 보강을 지시한다.

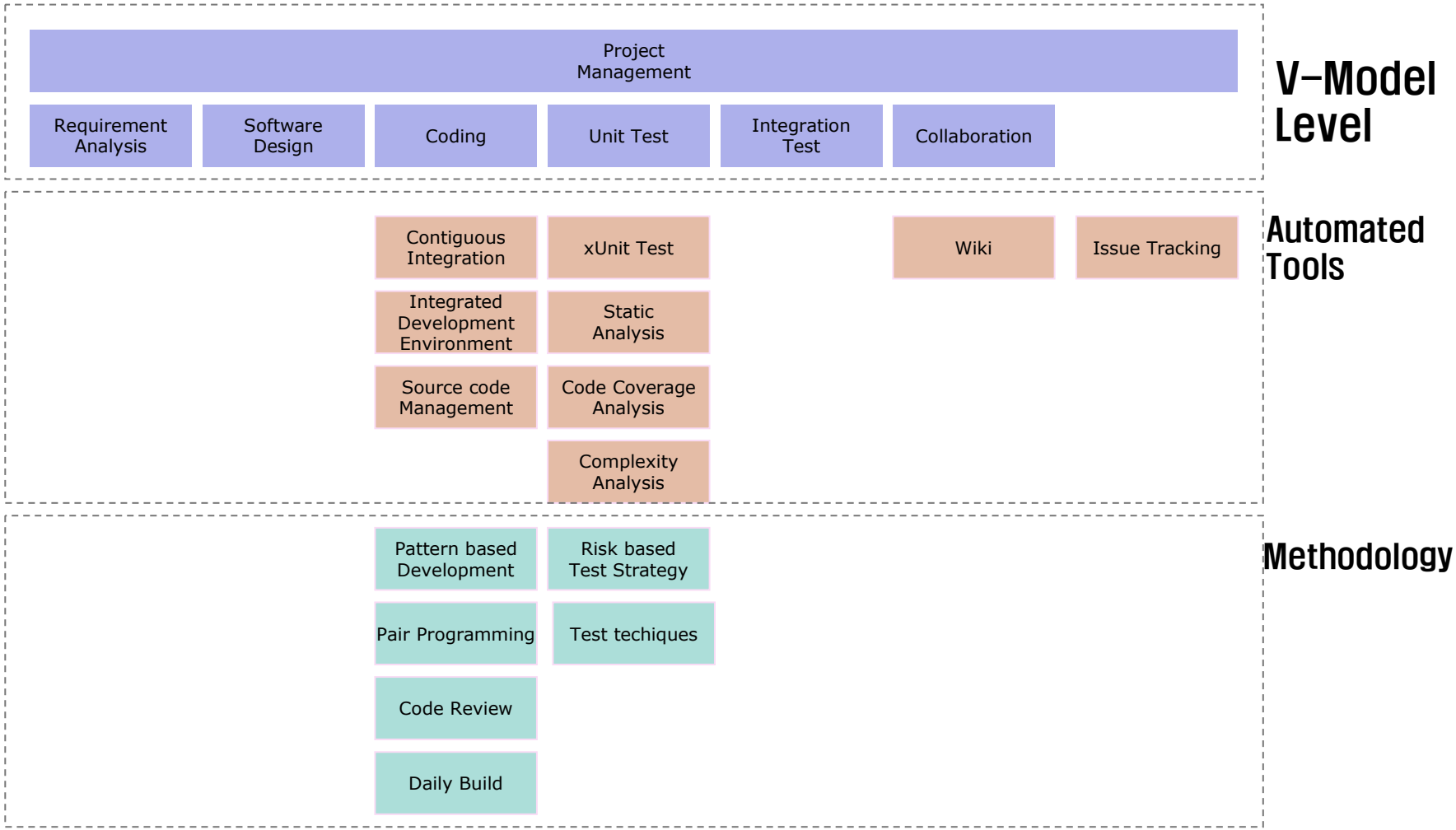
PM 가상 시나리오

- PM의 작업 지시
  - PM P씨는 아침에 출근하여 고객으로 부터 새로운 요건을 받았다.
  - P씨는 요건을 정리하여 내용을 이슈 트랙킹 시스템에 등록하고 심각도와 우선 순위를 지정해서 개발 PL에게 ASSIGN하였다.
  - 개발 PL은 요건의 우선 순위와 긴급도를 보고 누구에게 작업을 지정할것인지 고민한다. 이슈 트랙킹 시스템에서 개발원별로 진행중인 이슈 사항을 보고 개발 난이도에 맞는 사람중에서 가장 할당된 이슈가 적은 사람에게 이슈를 ASSIGN하였다.
  - 개발자는 해당 ISSUE를 받아서 처리한 후 PL에게 다시 ASSIGN한다.
  - 이때 작업에 관련된 메일,전화 통화내용 기타 관련 내용들을 시스템에 모두 LOGGING한다.
  - PL은 작업이 완료된 내용을 검토한후 해당 ISSUE를 CLOSE한다.
  - PM은 자신이 지시한 내용에 대해서 누가 진행하고 있으며 진행현황이 어떻게 되었는지를 이슈를 통해서 추적할 수 있다.
- PM의 현황 관리
  - PM P씨는 1차 오픈 때까지 해결되어야 할 이슈를 이슈 관리 시스템에서 검색한다.
  - 심각도가 높은 이슈와 오픈된지 오래된 이슈를 확인하여 진행이 안되고 있는 이유는 무엇인지 RISK는 무엇인지를 조사하고, RISK에 대한 대비책을 세운다.
  - 만약에 날짜에 비해서 해결되어야 할 이슈가 많을 경우 심각도와 우선순위를 고려하여 2차 오픈으로 이슈를 연기한다.

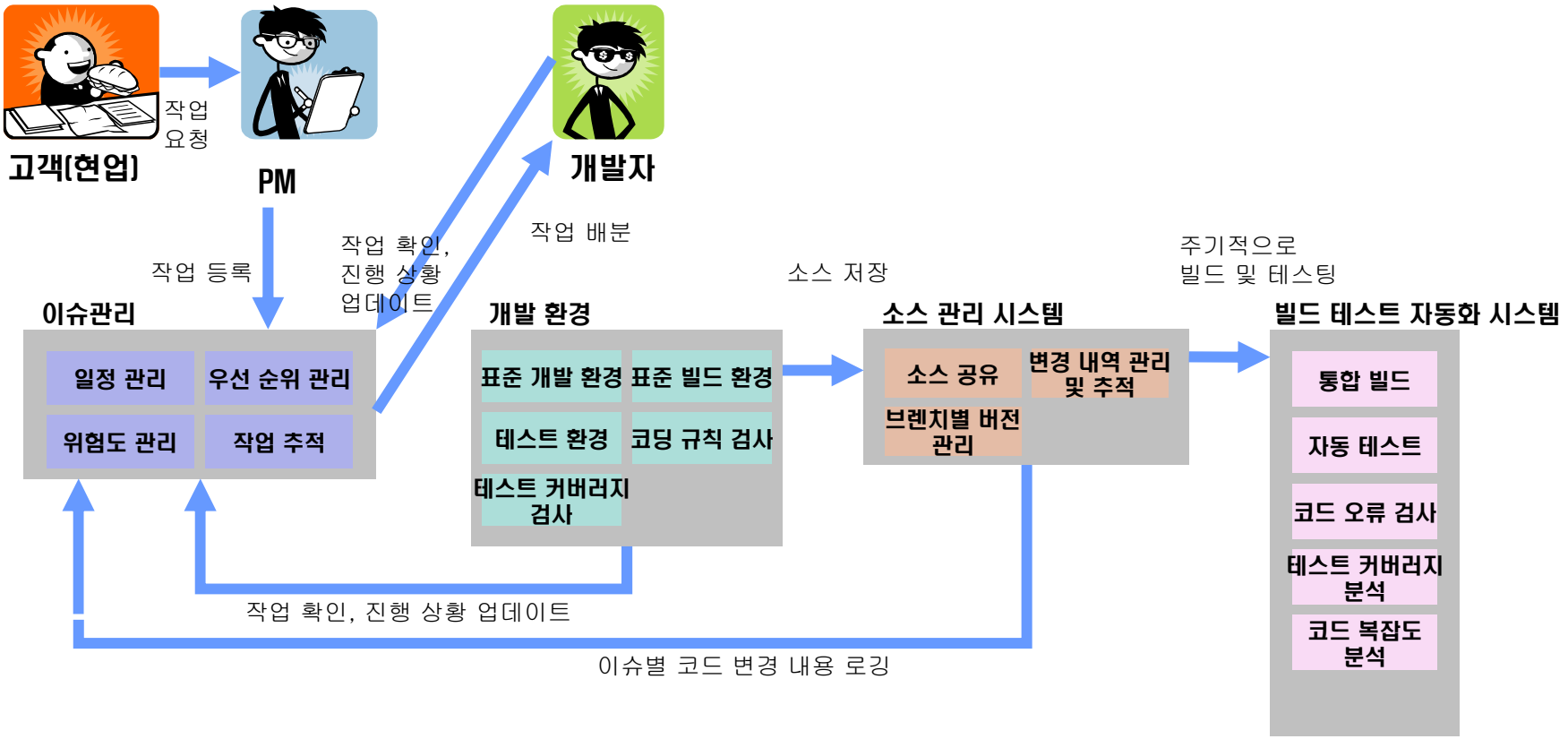
구현 전략

- 매일 1회 이상 통합 빌드
  - BIG BANG식 코드 통합에서 오는 문제점을 조기 제거
  - 코드 변경이나 인터페이스 변경으로 인한 빌드 실패인한 개발 부하를 제거
- 매일 회귀 테스트 수행
  - 매일 테스트를 통해서 소프트웨어의 완성도 증대
  - 변경으로 인한 오류에 대한 조기 검출
  - RISK 기반 접근 전략,커버러지 분석 및 코드 복잡도 분석을 통해서 집중 테스트의 효과성 증대
  - 테스트 자동화를 통한 테스트 효율성 증대
- 오프소스 기반의 구축
  - 상용 ALM → 높은 가격과 많은 기능으로 적용이 어려움
  - 무료로 고객의 수준과 개발 프로세스에 맞는 형태로 ALM을 재구성 가능
  - 널리 알려진 제품 위주 구성으로 추후 타 솔루션과 연동과 확장이 용이함
- 이슈 추적 시스템을 통한 작업의 통제
  - 코드 변경에 대한 추적성 부여

ALM 적용 범위


Project Management
Requirement Analysis
Software Design
Coding
Unit Test
Integration Test
Collaboration
V-Model Level
Contiguous Integration
xUnit Test
Wiki
Issue Tracking
Integrated Development Environment
Static Analysis
Source code Management
Code Coverage Analysis
Complexity Analysis
Automated Tools
Pattern based Development
Risk based Test Strategy
Pair Programming
Test techiques
Code Review
Daily Build
Methodology

ALM PROCESS


고객(현업)
작업
요청
PM
개발자
작업 등록
작업 확인,
진행 상황
업데이트
작업 배분
소스 저장
주기적으로
빌드 및 테스팅
이슈관리
일정 관리
우선 순위 관리
위험도 관리
작업 추적
개발 환경
표준 개발 환경
표준 빌드 환경
테스트 환경
코딩 규칙 검사
테스트 커버리지
검사
소스 관리 시스템
소스 공유
변경 내역 관리
및 추적
브렌치별 버전
관리
빌드 테스트 자동화 시스템
통합 빌드
자동 테스트
코드 오류 검사
테스트 커버리지
분석
코드 복잡도
분석
작업 확인, 진행 상황 업데이트
이슈별 코드 변경 내용 로깅

이슈 관리 시스템

- 일정 관리
  - 이슈 별로 DUE DATE와 주요 MILE STONE을 지정하여 이슈를 기간별로 해결해야 할 작업과 진행 상황을 실시간으로 추적한다.
- 우선 순위와 위험도 관리
  - 작업별로 우선순위와 위험도를 지정하여 RISK 관리와 스케쥴링에 사용한다.
- 작업 관리
  - 인력별로 진행중인 이슈와, 완료된 이슈 추적을 통해서 인력별 작업 부하와 성과를 측정한다.
  - 각 작업에 작업에 관련된 모든 내용을 로깅함으로써 다른 사람이 작업에 진척 상황을 체크하고, 필요한 경우 작업을 타인에게 인수인계 할때 작업을 ASSIGN하여 인수 받는 사람이 작업의 HISTORY를 인식하게 한다.
- 대쉬 보드
  - 현황 구성을 통해서 한눈에 현재 프로젝트의 진행 상황과 위험요소, 자원의 사용률등을 모니터링한다.

개발 환경

- 표준 개발 환경
  - 개발 환경을 통합화하여 신규 개발자도 쉽게 개발 프로세스에 참여할 수 있도록 유도 (IDE+WAS+테스트 환경)
  - CF. 소니 사례 : 압축만 풀면 개발 환경 셋업
- 표준 빌드 환경
  - 개발자 별로 동일한 빌드 스크립트 제공으로 빌드에 소요되는 시간 및 IMPACT를 최소화 (eg. Encoding 문제 etc)
- 테스트 환경
  - 테스트 툴과 환경을 제공하여 개발자가 손쉽게 테스트 케이스를 생성할 수 있게 함
- 코딩 규칙 검사
  - 코딩 규칙을 시스템적으로 검사하여 코드의 가독 품질을 높임
- 테스트 커버러지 검사
  - 테스트 커버러지 분석 기능 제공을 통해서 테스트의 효과성을 높임 (촘촘한 그물망)

소스 관리 시스템

- 소스 공유
  - 팀원간 소스 및 형상 공유
- 변경 내용 추적 및 관리
  - 소스 코드 변경 내용에 대한 추적
- 브렌치별 버전 관리
  - 릴리즈 버전별로 소스코드 브렌치 관리

빌드 자동화 시스템

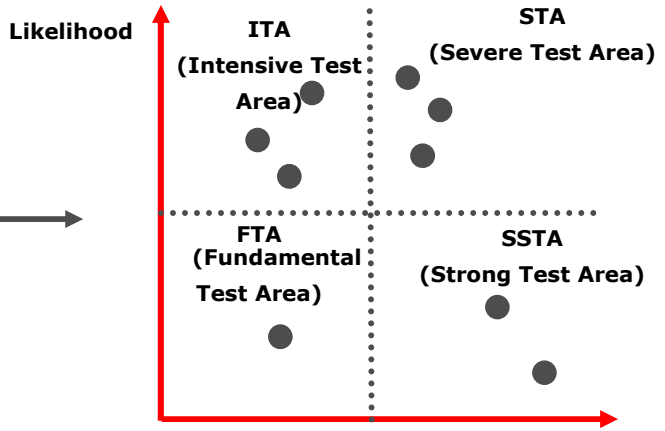
- 통합 빌드
  - 매일 통합 빌드를 통해서 코드 통합에서 오는 오류를 방지
- 자동 테스트
  - 빌드 과정에 UNIT 테스트 및 통합 테스트를 자동화하여 매일 수행함으로써 요건의 충족 여부와 변경에 의한 DEFECT를 조기 검출
- 코드 오류 검사 (CODE INSPECTION)
  - 코딩 RULE을 벗어나는 코딩, 잠재적인 DEFECT를 보유하고 있는 코드를 조기 제거
- 테스트 커버러지 분석
  - 테스트가 전체 시스템의 어느정도 부분을 테스트했는지 매일 자동으로 리포트
- 코드 복잡도 분석
  - 코드 복잡도 분석을 통해서 테스트를 위험도가 높은 코드들을 검출하여 테스트 커버러지와 연계

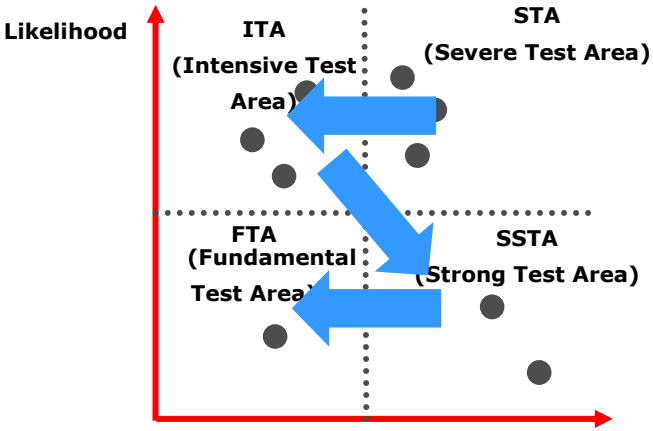
빌드 자동화 시스템

- Risk 기반 테스트 집중

	Likelihood		Impact	
	factor	factor	factor	factor
Risk item				
Risk item				

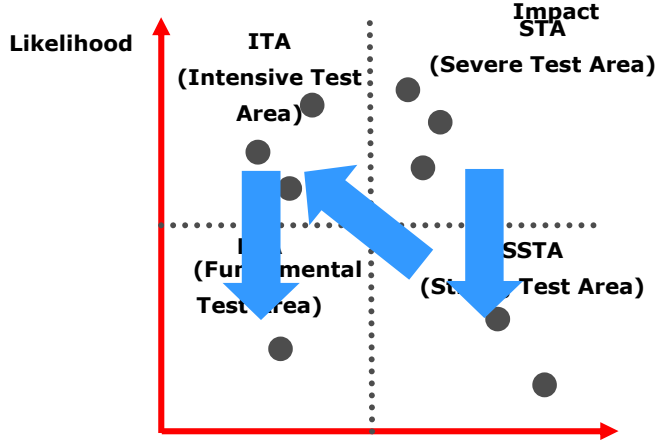
Erik van neneendaal, Risk Based Testing, STAREAST,2006





(Low Level Test)

- Unit Test
- Integration Test



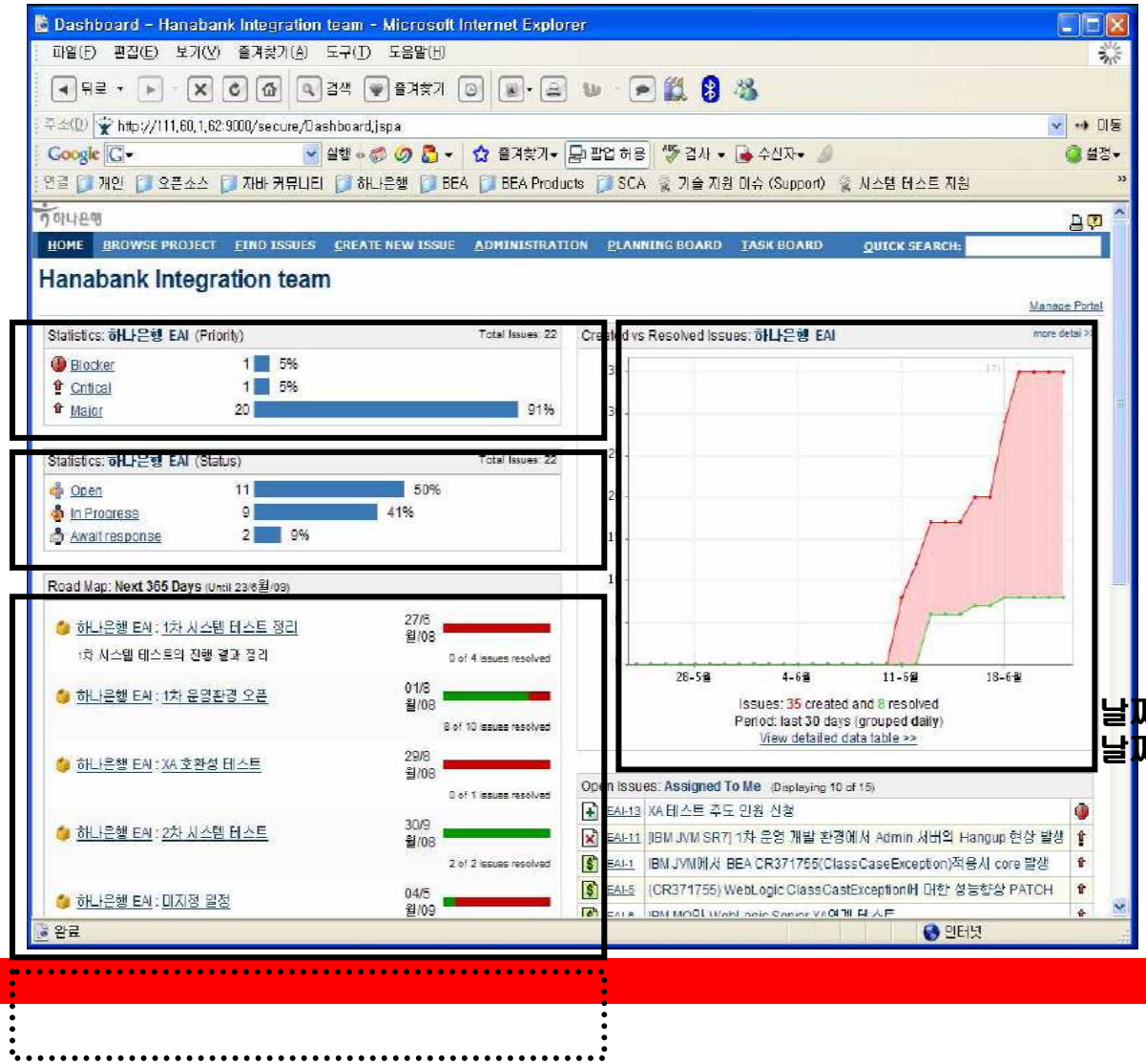
(High Level Test)

- System Test
- Acceptance Test

ALM sample view

고객 관점의 ALM VIEWPOINT

- 이슈 관리 대쉬 보드



작업의 심각도 별로 분류

진행상태별 이슈 수

주요 MILESTONE별 진척율

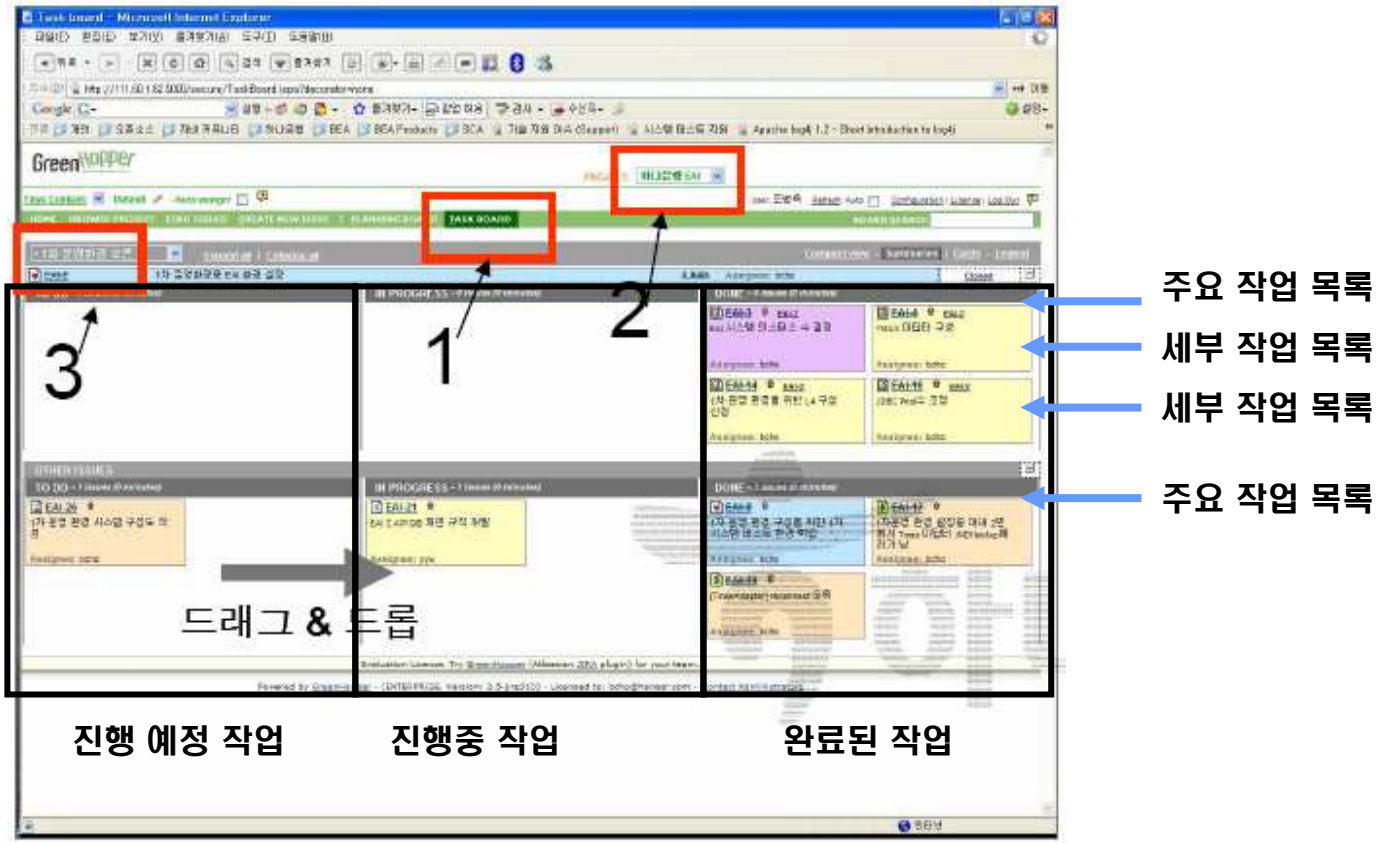
인원별 이슈 할당 상황

날짜별 생성된 이슈
날짜별 종료된 이슈

ORACLE

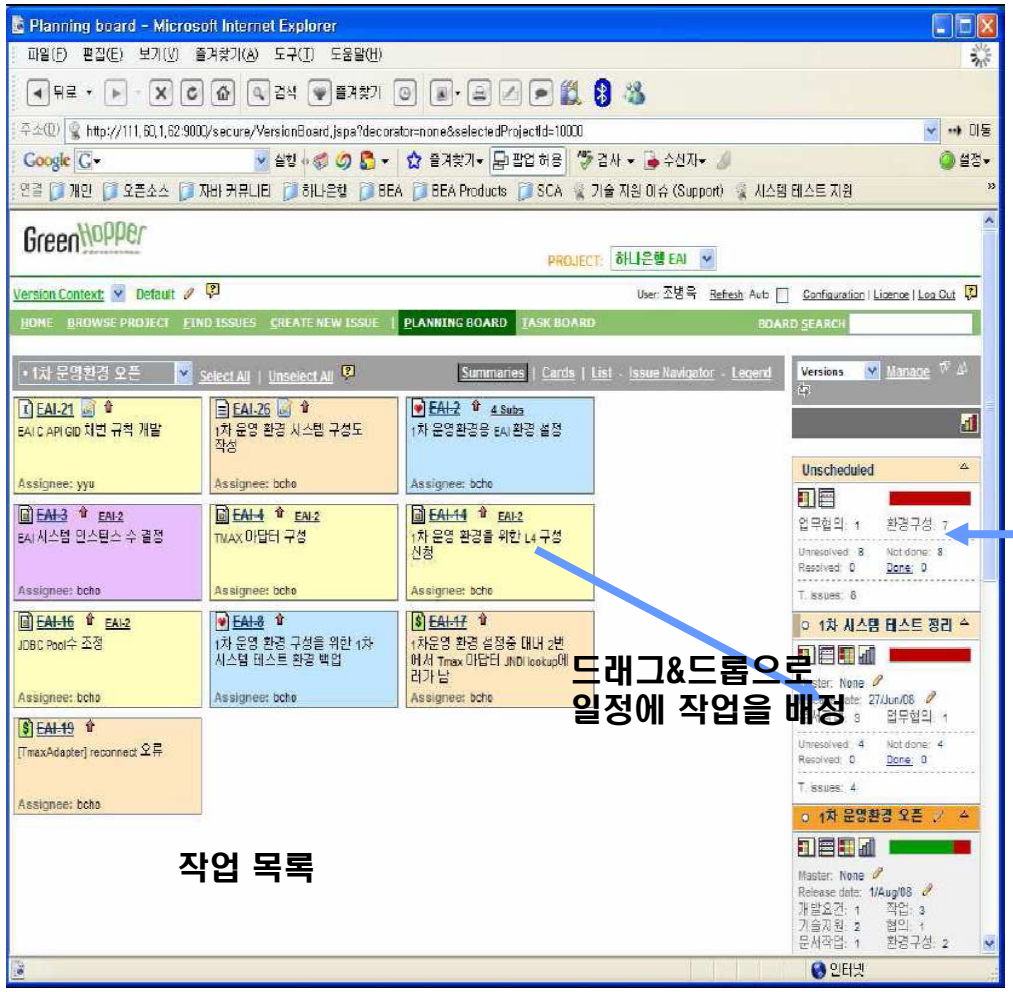
PM 관점의 ALM VIEWPOINT

- 이슈 관리 시스템을 이용한 현황 파악



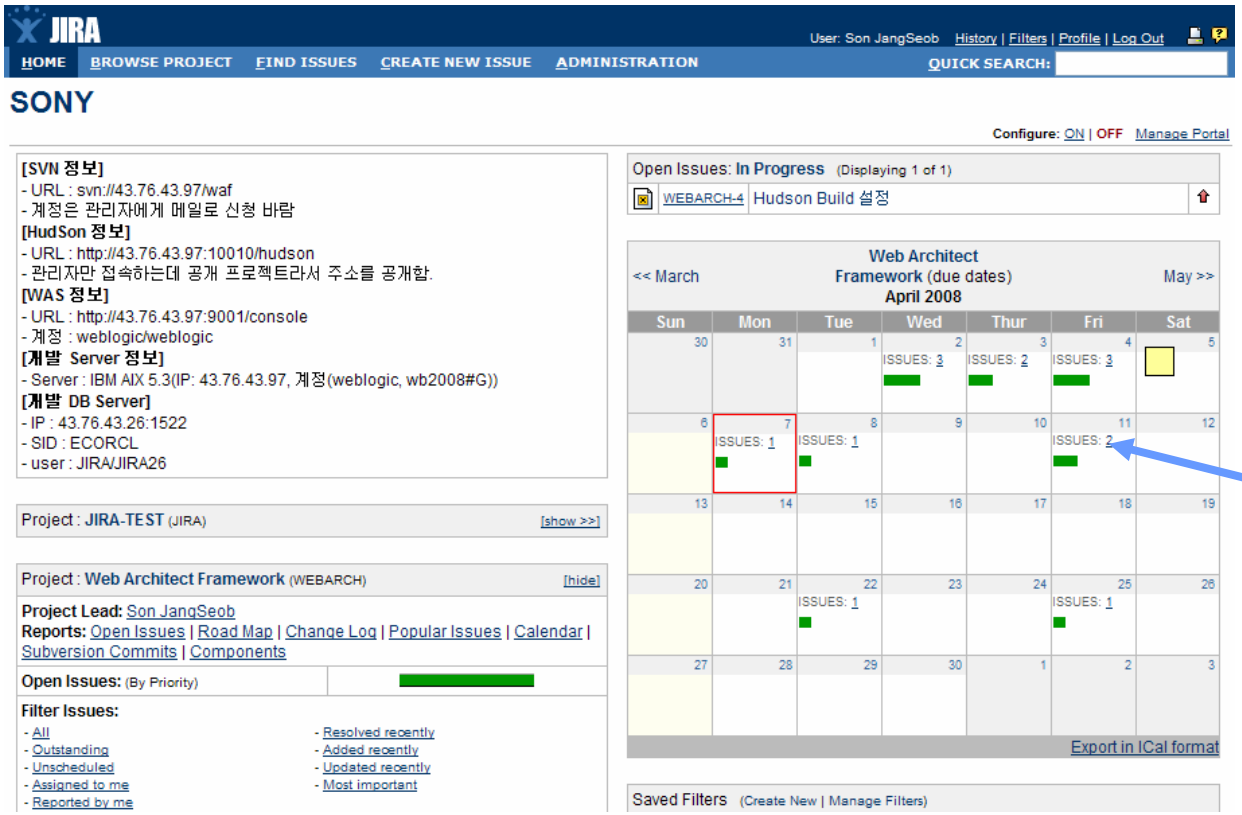
PM 관점의 ALM VIEWPOINT

- 이슈 관리 시스템을 일정 배정



PM 관점의 ALM VIEWPOINT

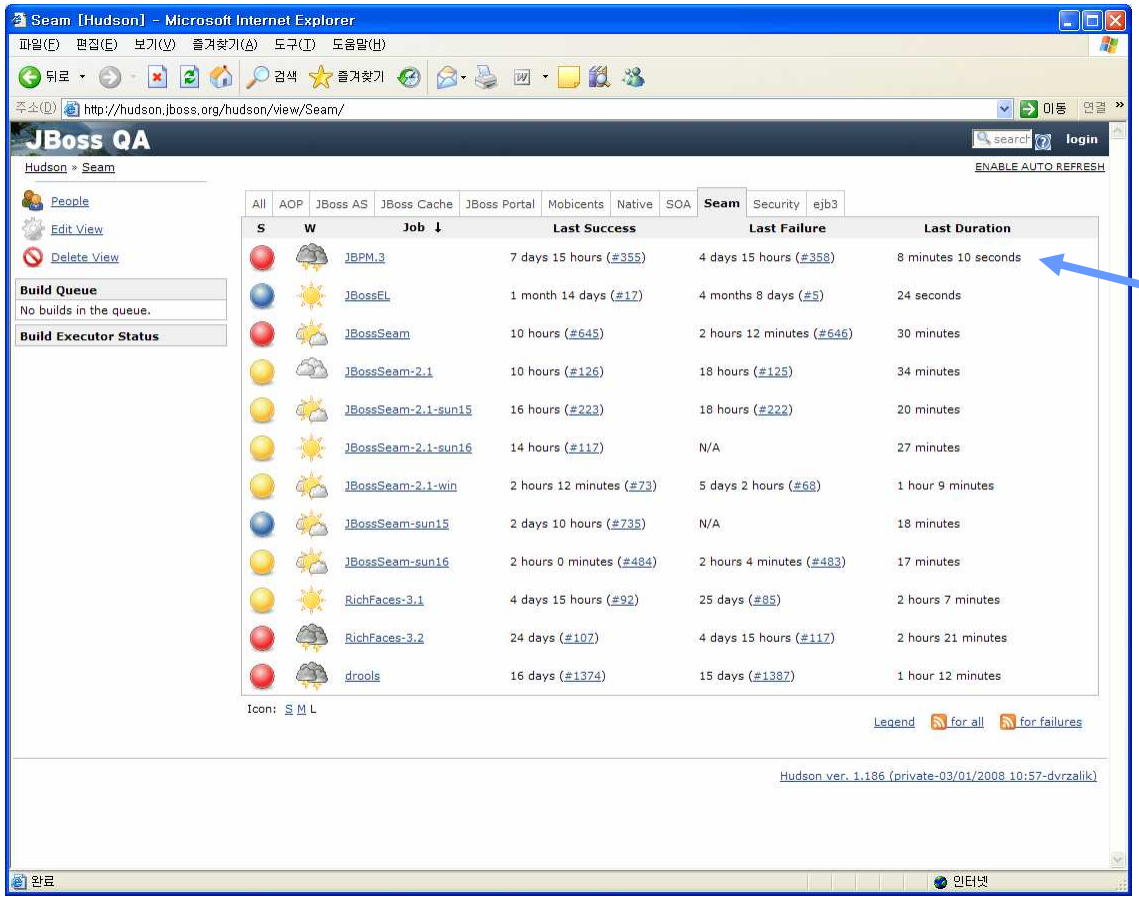
- 주요 마일스톤 별 완료 여부 체크



날짜별 완료해야하는
이슈 수와 진행률

PM 관점의 ALM VIEWPOINT

- 빌드 현황 모니터링

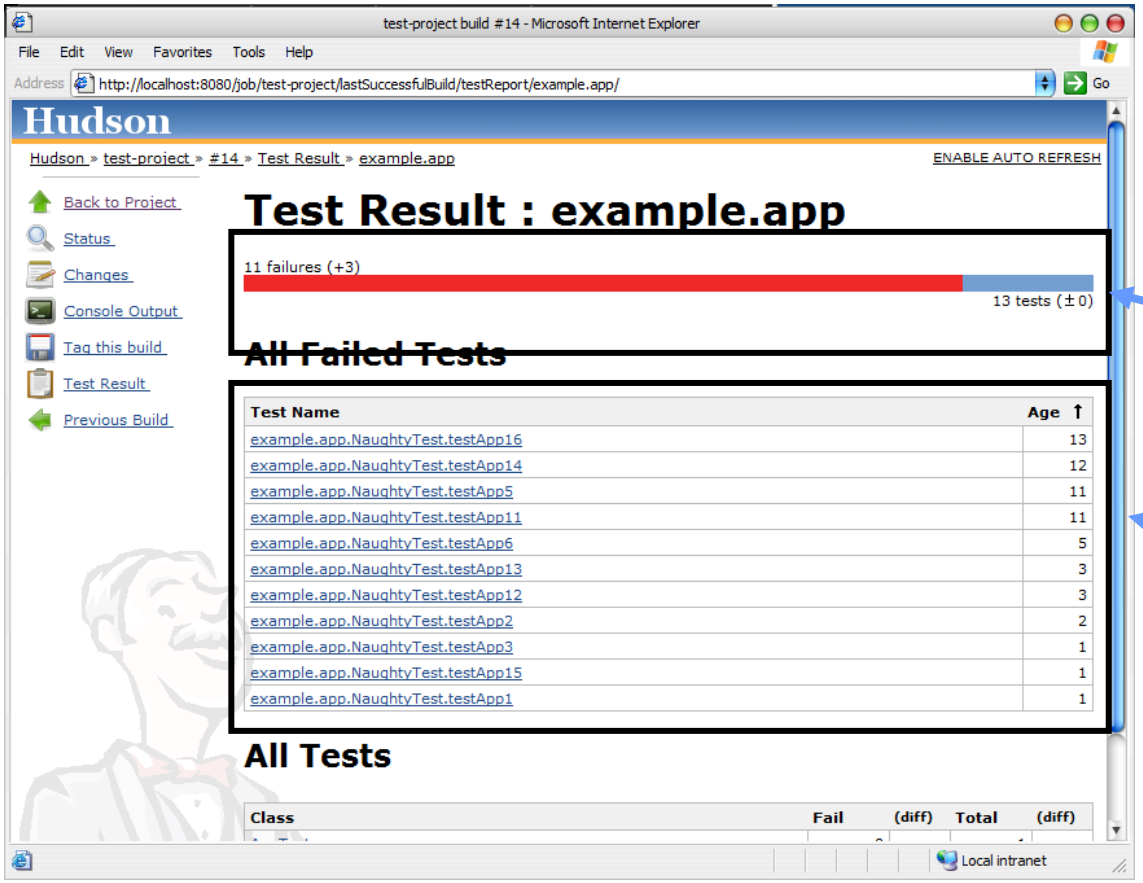


프로젝트별 빌드 성공 실패 여부

최종 빌드가 실패한 경우 붉은색,성공한 경우 파란색, 빌드가 자주 깨지는 경우 흐린 날씨, 빌드가 잘될 수 록 맑은 날씨로 표시됨

PM 관점의 ALM VIEWPOINT

- 테스트 결과 분석

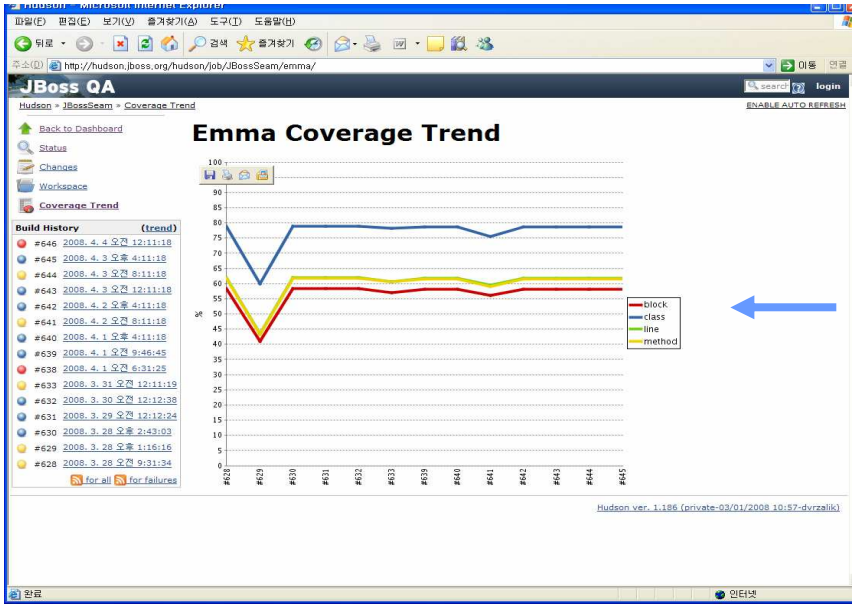


테스트 성공 실패 결과

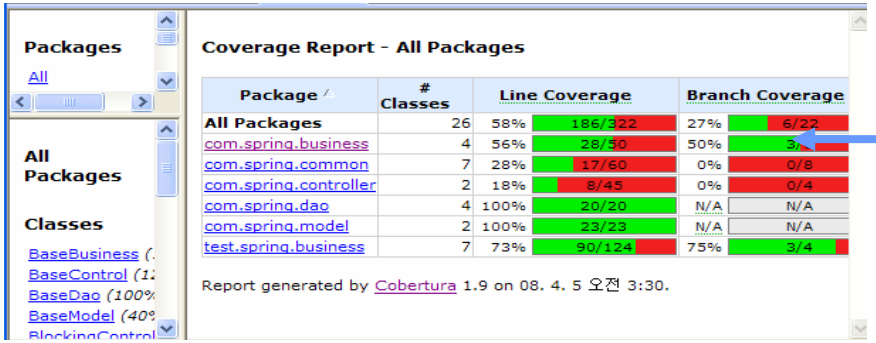
실패한 테스트 케이스

PM 관점의 ALM VIEWPOINT

- 테스트 커버러지 분석



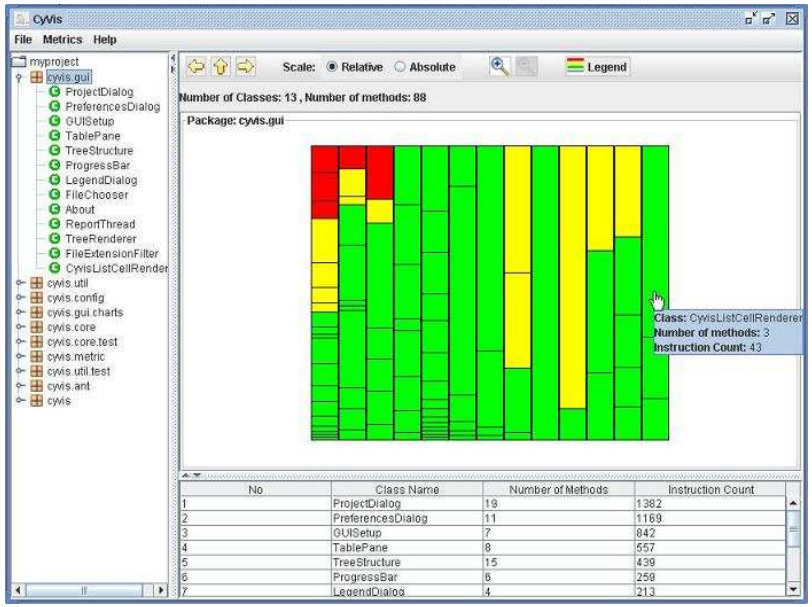
코드상의 전체 라인수,클래스 수,메서드수 들 대비하여 테스트가 수행된 비율



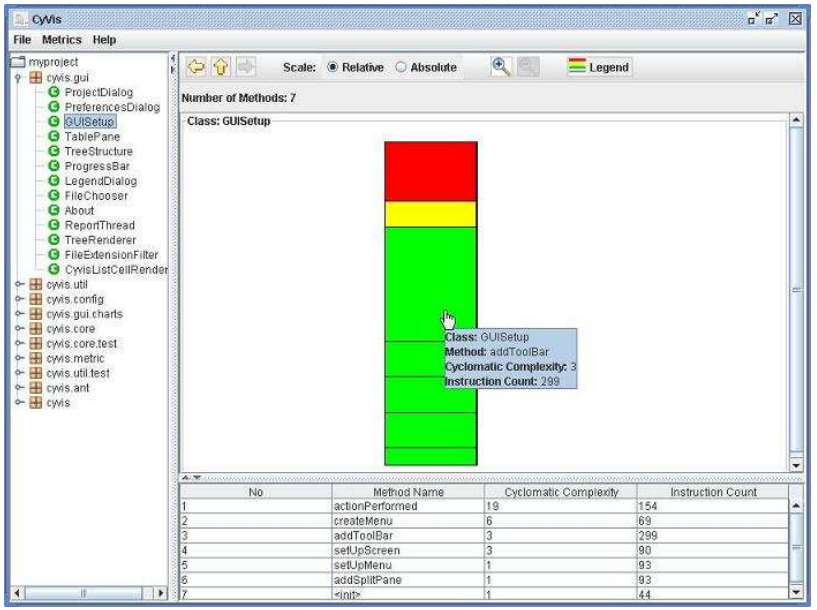
패키지별 Statement Coverage와, Branch Coverage를 리포팅

PM 관점의 ALM VIEWPOINT

- CODE Complexity 분석



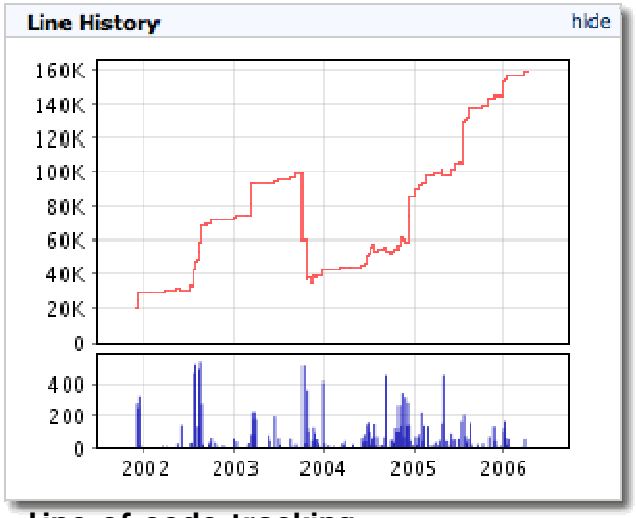
패키지 별로 클래스당 CODE COMPLEXITY를 분석



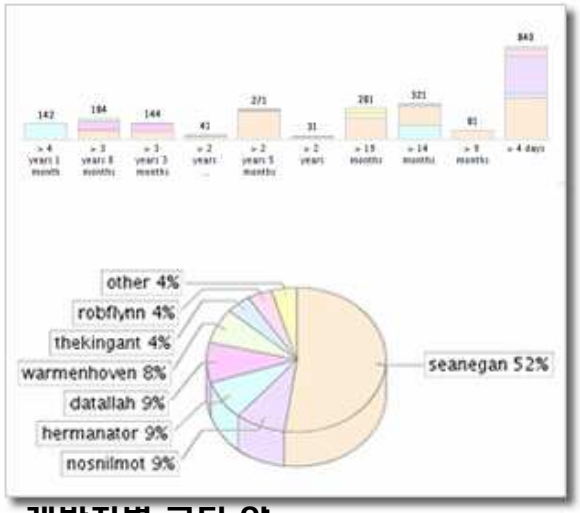
클래스의 메서드당 Cyclomatic Complexity 분석

PM 관점의 ALM VIEWPOINT

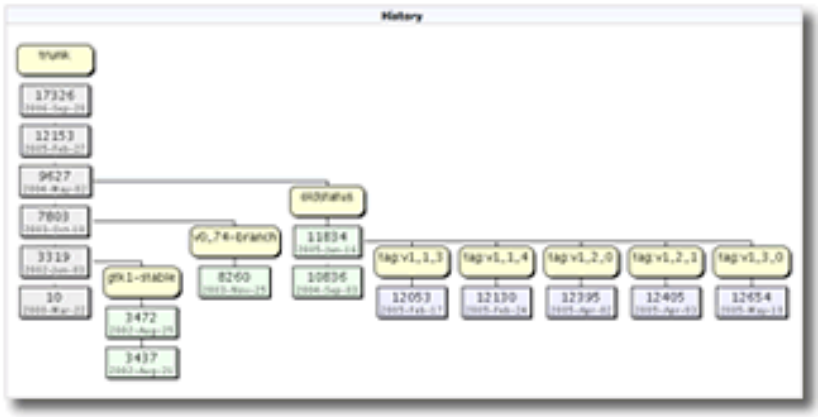
- CODE tracking



Line of code tracking



개발자별 코딩 양

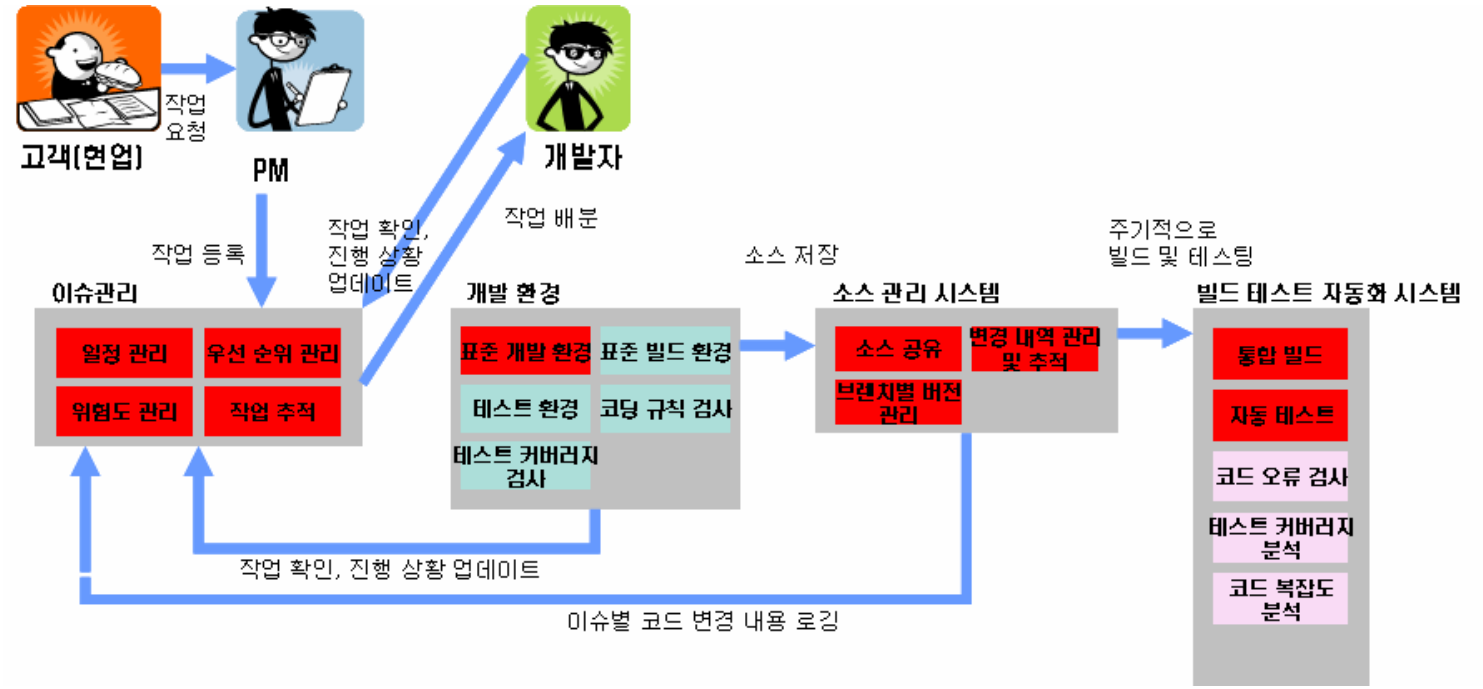


소스 코드의 브랜치별 추적 내용

ALM case study

H 은행 차세대 사례 적용 부문

- 적용 부문



H 은행 차세대 사례 적용 내용

- 적용 내용
  - 적용 환경
    - 15명 정도의 한 프로젝트팀.UI가 없는 시스템
    - 개발 중간 단계에 적용 시작
    - 사례 기반으로 타팀에 전파됨 (아키텍쳐팀,MCA팀)
  - 통합 빌드
    - 10여개의 이클립스 프로젝트로 구성된 빌드 환경
    - 매일 2차례 자동 빌드
    - 통합 과정에서 오는 오류를 완전 제거
  - 소스 관리
    - 프로젝트 중반에 브렌치 정리 – 개발 환경, 1차 운영 환경, 차세대 개발 환경식으로 브렌치를 나눠서 체계적으로 관리
    - 여러 버전의 코드에서 오는 혼잡 제거
  - 이슈 관리
    - 코딩에 관련된 이슈로는 사용하지 않음 팀 운영 전체에 관련된 이슈를 관리
    - 고객의 만족도는 매우 높음
    - 구성 팀원내에서 상당한 효과가 있었음

H 은행 차세대 사례 적용 내용

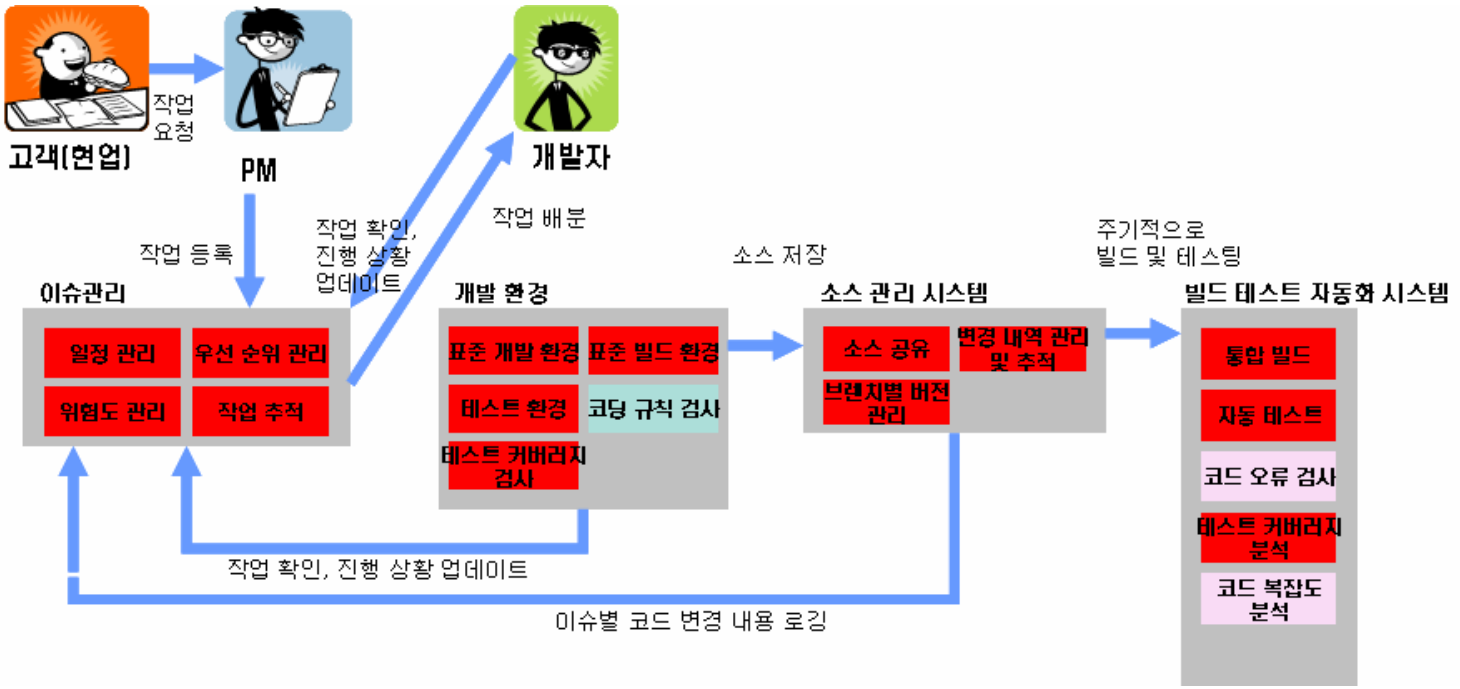
- CODE INSPECTION
  - PMO 조직에서 상용도구 도입 결정
  - 간단한 SYNTAX오류만 검출
  - 도구에 대한 운영 교육과 프로세스없이 사용 결정됨
  - INSPECTION에 대한 룰셋 개발 부재
- 단위 테스트
  - 개발 후기에서 별도의 리소스 계획 없이 단위 테스트 도입
  - 검사 받기 위한 단위 테스트 케이스
- 통합 기능 테스트
  - 기능 위주의 테스트 플랫폼 제작
  - 테스트 환경 구축에 많은 시간이 소요됨 (시뮬레이션 환경)
  - 테스트 환경은 추후에 부하 테스트 환경으로 재활용됨
  - 테스트 과정에서 많은 문제점을 발견 및 해결 할 수 있었음
  - 테스트 환경 전용 HW 부재로, 테스트시마다 환경 설정에 많은 리소스 사용
  - 회귀 테스트 환경 설정 진행중

H 은행 차세대 사례 Lesson learned

- Lesson learned
  - 통합 빌드와 소스 브렌치 정책은 노력대비 효과성이 매우 높음
    - 빌드 스크립트 정리, 정책 수립 및 일일 빌드 셋업
  - 이슈 트랙킹
    - 프로세스 정립 및 코칭이 관건
    - 상위 Manager를 위한 대쉬 보드 구성이 중요
  - ALM 설정에 필요한 리소스 할당과 프로세스 변화에 대한 인식 필요
    - 테스트 자동화 부분에서 별도의 리소스 배정 없이 진행하여 효율성이 매우 떨어짐 , 프로젝트 팀에 부하 , 보이기 위한 테스트가 일부 생성됨
  - 성숙되기 전까지 코칭이 필요
    - 성숙때까지 코칭을 한팀은 smooth하게 ALM 방법론이 적용되고 효과가 극대화됨
    - 코칭 없이 진행한 팀은 TA의 능력에 따라 적용 수준과 효율성이 매우 들쭉 날쭉함

S 외국계 전자 회사 적용 부문

- 적용 부문



S 외국계 전자 회사 적용 내용

- 적용 내용
  - 통합 빌드
    - 빌드과정에서 단위 테스트 및 커버러지 분석 수행
    - HUDSON + ANT
  - 소스 관리
    - 개발,테스트,운영계로 구성된 개발환경
    - 테스트에서 운영으로 배포하는 버전에 대해서 브렌치 관리
    - 웹에서 소스 변경 히스토리 추적 가능 (FishEye)
  - 단위 테스트
    - 비즈니스 메서드 단위로 단위 테스트 케이스를 만들도록 가이드
    - RIA FLEX에 대해서는 단위 테스트 적용 불가
    - 매일 회귀 테스트 진행
  - 이슈 관리
    - 개발자 IDE와 연동
    - 이슈의 오픈과 CLOSE 확인을 현업(고객)에게 ROLE을 할당 고객 참여와 확인 유도
  - 개발환경 관리
    - 개발 서버, 테스트 서버, 운영 서버를 독립적으로 운영
    - 위의 세 환경에 대해서 동일한 디렉토리 구성
    - 개발자 PC에도 서버와 동일한 디렉토리 구성
    - 개발에 필요한 개발자 도구 (IDE,LIB,WAS ETC)을 ZIP으로 패키징하여 배포 → 압축만 풀면 동일한 개발환경

S 외국계 전자 회사 Lesson learned

- Lesson learned
  - 이슈 추적에 있어 고객 참여 유도
  - 파일럿 프로젝트를 통하여 성숙된 모델을 전사 개발팀에 전파
  - 고객의 ALM에 대한 필요성 인식으로 효과적으로 적용된 사례

ALM 솔루션

ALM 솔루션

- 이슈 관리 도구
  - Bugzilla 오픈소스,인스톨 하기 어려움, 버그 추적 위주
  - Mantis 오픈소스,한글화,기능이 빈약
  - Trac 오픈소스,이슈 관리 뿐만 아니라,SCM과 WIKI 포함
  - JIRA 상용, 가능 널리 쓰임
  - Mylyn 오픈소스,이슈관리 도구와 이클립스를 연동하는 플러그인
- 빌드 도구
  - ANT 가장 폭넓게 쓰임
  - MAVEN DEPENDENCY에 대한 관리 기능, 솔루션에 따라 지원 안함
- SCM
  - Subversion
  - CVS
  - Perforce
- Code Inspection
  - PMD
  - Find Bugs

ALM 솔루션

- CI 도구
  - Hudson
  - Cruise Control
  - AntHill
  - Team City
  - Code Beamer
  - Polarion
- 테스트 프레임웍
  - 단위 테스트 — JUnit
  - DB 테스트 — DBUnit
  - IN CONTAINER 테스트 — Cactus,JUnitEE
  - UI 테스트 — Selinium,WattJ
  - 부하 테스트 — Japex,Jperf
  - 웹서비스 — SOAPUI
- 테스트 커버러지 도구
  - EMMA — 아카이브 파일 (WAR,JAR 등 지원 안함)
  - Cobertura — 아카이브 지원함, 동적 Instrumentation 지원 안함
- 코드 COMPEXITY 분석
  - Cyvis

ALM 추천 조합

- 1단계
  - 이슈 관리 JIRA
  - 빌드도구 ANT
  - SCM Subversion
  - CI 도구 Hudson
  - 테스트 프레임웍 Junit + DBUnit + Cactus
  - 테스트 커버러지 Cobertura
- 2단계
  - +테스트 프레임웍 +Seliniuem, +Japex
  - +테스트커버러지 Cobertura
  - +정적분석 FindBug or PMD
- 3단계
  - +Code Complexity 분석 Cyvis
  - +표준 공유 Atlassian Confluence Wiki

ALM 구축 전략

ALM 구축 전략

- Simple & Easy
  - 모든 개발 구성원들이 쉽게 사용할 수 있어야 한다.
- Seamless integration
  - ALM을 구성하는데 필요한 기술과 제품들이 하나의 제품처럼 유기적으로 결합되어야 한다.
- Liquid flow
  - ALM Process가 물 흐르듯이 자연스럽게 연결되어야 한다.
- Effective & Efficient
  - ALM을 도입함으로써 개발 생산성과 소프트웨어의 품질 향상이 이루어져야 한다.
- Control & Maturing
  - ALM 프로세스와 도구가 사용되도록 개발 조직이 통제되어야 한다.
  - ALM 프로세스는 개발조직의 능력에 맞춰서 성숙되어야 한다.
- Step by Step
  - 팀과 리소스, 일정에 맞는 ALM 모델 설정
  - 단계별 발전 모델을 정의하고 단계별로 팀에 ALM을 구축

Thank you

For More Information
http://bcho.tistory.com