품질관리 계획서

1. SW 품질확보 방안

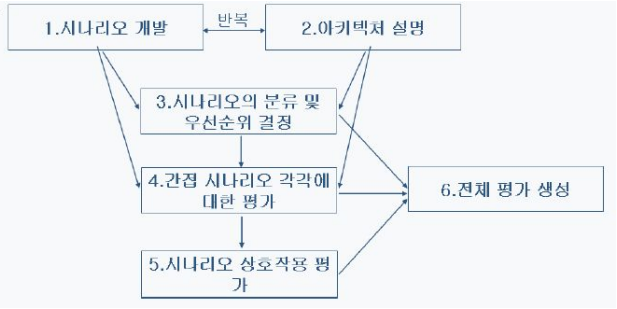
1.1 SW 설계 품질확보 방안

1.1.1 아키텍처 검증 방안

※ 수립된 아키텍처 요구사항이 적합한지에 대한 검증 활동(절차, 기법 및 전략)을 수립방안과 연계하여 기술

초기에 일부 설계만 되어 있을 때 쉽게 평가하기에 적합한 경량급 평가 기법으로 '적합성'에 집중한 ARID기법을 쓰고, 후반에 전체 SW 아키텍처 평가용 기법으로 SAAM을 이용한다.

- ARID
    - 초기에 일부 설계만 되어 있을 때 쉽게 평가하기에 적합한 경량급 평가 기법으로 '적합성'에 집중함
    - SW 아키텍처 설계의 적절성을 판정함
    - Very early mini-evaluation, Discovery Review에 해당
    - ARID의 수행 절차
        - Phase I: 리허설
            - 1. 검토자 확인 2. 설계 브리핑 준비 3. 핵심 시나리오 준비 4. 자료 준비
        - Phase 2: 리뷰
            - 5. ARID 발표 6. 설계 발표 7. 브레인스톰과 시나리오 우선순위 결정 8. 시나리오 응용(최우선 순위, 의사코드 작성) 9. 요약
- SAAM
    - 일반적으로 최초로 문서화된 평가 기법으로 널리 알려짐
    - 변경성, 강건성, 이식성 등 아키텍처를 놓고 일반적으로 거론하면서도 입증할 수 없는 주장들을 검증하기 시작했음
    - 본래는 아키텍처를 변경성 관점에서 분석하기 위해 고안되었으나 실제로는 다른 품질적인 면들을 신속하게 평가하는데 도 유용하다는 것이 입증되었음
    - 간단하고 쉬운 방법으로 아키텍처 평가 경험이 없고 특히 변경성과 기능성에 관심이 있는 경우 아주 좋은 출발점이 됨.
    - 하나의 아키텍처에 대한 평가와 아울러 경쟁관계에 있는 아키텍처들을 비교할 수 있음
    - 아키텍처 문서의 적절한 수준을 결정함
    - 시나리오를 통해 이해관계자자들이 지대한 관심을 갖는 비즈니스 목표를 드러냄
    - 아키텍처가 이 시나리오에 부응할 것인지를 보여줌
    - 이해관계자들이 함께 모여 서로 이해할 수 있는 언어로 아키텍처를 논의할 수 있도록 함
    - SAAM의 입력물 :
        - 아키텍처 설명
        - 품질의 명세화와 평가의 설명적인 수단이 되는 시나리오
    - SAAM의 출력물 :
        - 시나리오와 아키텍처의 맵핑 : 향후에 복잡해질 가능성이 큰 부분과 각 변경에 수반되는 예상 비용을 알려줌
    - SAAM의 수행 절차



1.1.2 설계 검증 방안

※ 설계 산출물에 대한 검증을 방안(예: Peer Review, Inspection 등)을 제시

- 각각의 수행일정은 개발 사이클마다 진행한다.

설계물	방안			
	수행주체	활동내역	기법도구	수행일정
클래스 다이어그램	개발자	클래스 설계 결함율 체크	Technical review	설계 단계
UI설계	이해관계자, 디자인 팀	인터페이스 결함율 체크	Inspection (체크리스트)	프로토타이핑 단계 이후
컴포넌트 설계 (시퀀스 다이어그램)	개발자	컴포넌트 결함율 체크	Inspection or Technical review	설계 단계
유스케이스	이해관계자, 개발자	유스케이스 결함 체크	management review	설계 단계

1.2 SW코드 품질확보 방안

※ SW코드의 품질을 확보하기 위한 방안을 유형별로 구분하여 방법 및 내용, 적용대상, 수행 범위, 사용도구를 기술

※ 서브시스템별 SW코드 품질확보 방안이 다른 경우, 서브시스템별로 구분하여 기술

유형	방법 및 내용	적용대상	범위	도구명
코드 인스펙션 (정적 테스트)	작성한 개발소스 코드를 자동화된 도구를 통해 분석하여 개발 표준에 위배되었거나 잘못 작성된 부분을 수정	전SW	100%	코드마인드
동료 검토	구현 초기 완료이전에 PM주도로 개발자 그룹내에서 workthrough 수행	전SW	필요시	N/A
동적 테스트	보안 취약점을 검출, 모의 해킹, 크래킹 시도 네트워크 충돌 횟수와 길이 초과 패킷에 대한 정보를 수집	보안, 네트워크	필요시	N/A
코드 복잡도 분석	복잡도 적정수준을 설계단계에서 설정, eclipse Metrics plugin을 사용하여 적용한다.	db connection	100%	Eclipse

기법	설명	적용방법
Extract Method	그룹으로 함께 묶을 수 있는 코드 조각이 있으면 코드의 목적이 잘 드러나도록 메소드의 이름을 지어 별도의 메소드로 추출	새로운 클래스를 만들고 기존의 필드와 메소드를 적용

Move Method	메소드가 자신이 정의된 클래스보다 다른 클래스의 기능을 더 많이 사용하고 있다면, 이 메소드를 가장 많이 사용하고 있는 클래스에 비슷한 몸체를 가진 새로운 메소드 생성후 간단한 위임 아니면 삭제	해당 클래스로 메소드를 옮기고 이전 메소드를 단순 대리자 or 삭제
Rename Method	메소드의 이름이 그 목적을 드러내지 못하고 있다면 메소드의 이름 변경	메소드 이름변경
Inline Method	메소드 몸체가 메소드의 이름 만큼이나 명확할 때는 호출하는 곳에 메소드의 몸체를 넣고 메소드를 삭제	해당 본문을 메소드를 호출하는 호출자 안으로 옮기고 메소드를 삭제
Extract Class	두 개의 클래스가 해야 할 일을 하나의 클래스가 하고 있는 경우 새로운 클래스를 만들어 관련 있는 필드와 메소드를 기존 클래스에서 새로운 클래스로 이동	새로운 클래스를 만들고 이전 클래스에 있던 관련된 필드와 메소드를 새로운 클래스로 옮김
Replace Temp With Query	수식의 결과값을 저장하기 위해서 임시 변수를 사용하고 있다면, 수식을 추출해서 메소드를 만들고, 임시 변수를 참조하는 곳을 찾아 모두 메소드 호출로 교체, 새로 만든 메소드는 다른 메소드에서 사용 가능	그 표현식을 메소드안으로 뽑아내고, 임시변수에 대한 모든 참조를 표현식으로 바꿈
Substitute Algorithm	알고리즘을 보다 명확한 것으로 바꾸고 싶은 경우, 메소드의 몸체를 새로운 알고리즘으로 교체	메소드의 내용을 새로운 알고리즘으로 변경

	Extract Method	**Move Method**	**Rename Method**	**Inline Method**	**Extract Class**	**Replace Temp with Query**	**Substitute Algorithm**

사용전	void print(class thing){
	printBanner();
	print(“name :”+ name);
	print(“price: ”+price);
	}
		class1{
		method();
		}
		class2{
		}
			int add(int x){
			return x+1;
			}
				void addprint(int x,int y){
				print(add(x,y));
				}
				int add(int x,int y){
				return x+y;
				}
					class person{
					name
					age
					grade
					schoolnumber
					}
						int income = salary * monty;
						if(income > 500)
						return income*0.6
						else
						return income*0.3
							string foundperson(string[] people){
							for(int i =0; i<people.length;i++){
							if(people[i].equals(“A”){
							return “A”

사용후	void print(class thing){
	printBanner();
	printDetail(thing);
	}
	void printDetail(class thing){
	print(“name :”+ name);
	print(“price: ”+price);
	}
		class1{
		}
		class2{
		method();
		}
			int addone(int x){
			return x+1;
			}
				void addprint(int x,int y){
				print(x+y);
				}
					class person{
					name
					age
					}
					class student{
					grade
					schoolnumber
					}
						if(income > 500)
						return income()*0.6
						else
						return income()*0.3
						
						int income(){
						return salary * monty;
						}
							string foundperson(string[] people){
							List peoplelist = {“A”,...}
							for(int I =0; i<people.length;i++){
							if(peoplelist.contains(people[i])){
							return people[i];

2. 제품 품질관리 방안

2.1 품질목표 측정 및 검증 방법

※ 품질지표별 측정 메트릭(측정항목/산식)을 구체적으로 제시하되 반복성(Repeatability), 재생성(Reproducibility), 공정성(Impartiality), 객관성(Objectivity)을 확보할 수 있어야 함
※ 신뢰할 만한 측정 및 검증방법(타 시험기관 활용 등 포함)을 제시

세부 시스템	품질지표	품질 목표 측정	메트릭(측정항목/측정산식)	측정 및 검증방법
전체 시스템	표준 순수성	100%	A=X/Y*100 .X= 준수한 표준 요건의 수 .Y= 준수해야 할 표준 요건의 수	OOO 기관 평가 (평가 결과서 제출)
	OO 인증 획득	등급 A	인증여부	

3. 품질관리 수행방안

3.1 프로젝트 품질활동 정의

※ 본 품질관리계획서 1항~4항에서 기술한 품질 활동의 적용방안을 주요 개발단계별로 구분하여 기술하되 활동 수행주체를 포함할 것

단계	품질 관리 활동	수행 주체
착수/계획	품질보증계획서 작성	품질 관리자
	개발표준 정의 및 산출물 양식 작성, 배포	품질 관리자
	개발 방법론 교육	관리자
	개발 방법론 확정	관리자
요구사항 관리	요구사항 검증	개발자, 사용자
	결함/위험/이슈/Action Item 식별, 조치	품질 관리자
	산출물 점검	관리자, 개발자
	프로토 타이핑	개발자(디자인)
분석	요구사항 추적/검증	관리자, 개발자, 사용자
	요구사항 통합 리뷰	분석자, 사용자
	프로세스 / 산출물 점검	관리자, 개발자
설계	설계 산출물 확인	관리자, 개발자
	아키텍처 검증	설계자, 품질 관리자

	설계서에 대한 분석자의 검토 여부 확인	설계자, 분석자