Software Inspection 을 위한 컴포넌트 우선순위 선정기법

오석원*, 강동수**, 백두권**
*고려대학교 소프트웨어공학과
**고려대학교 컴퓨터전파통신공학과
e-mail : *ritalove@korea.ac.kr, **{2008010372, baidk}@korea.ac.kr

A Method of Component Priority for Software Inspection

SukWon Oh*, DongSu Kang**, DooKwon Baik *
*Dept, of Software Engineering, Korea University
**Dept. of Computer Science & Engineering, Korea University

요 약

소프트웨어 정밀 검사는 품질 보증 활동 중 결함 제거율이 가장 높은 것으로 알려져 있지만, 비용과 시간이 가장 많이 소요되는 활동이다. 모든 프로젝트는 한정된 비용과 일정 제약을 가지고 있으므로 정형화된 프로세스로 이루어진 소프트웨어 정밀 검사를 프로젝트에서 산출된 모든 컴포넌트에 대하여 수행하는 것은 현실적으로 어렵다. 따라서 소프트웨어 정밀 검사의 효과를 극대화 하기 위한 효율적인 접근 방법이 필요하며, 본 논문에서는 효율적인 소프트웨어 정밀 검사를 수행하기 위하여 요구사항 크기와 복잡도, 설계 복잡도, 코드 복잡도 평가에 기반한 컴포넌트 우선순위 선정 기법을 제안하고 평가한다.

1. 서론

소프트웨어 품질 보증은 개발 중인 소프트웨어가 명세서를 만족하고 소프트웨어의 비용을 지불하는 사람이 기대하는 기능을 인도하도록 보장하는지 점검하는 것이다[1]. 이를 위한 점검과 분석 프로세스를 증명과 검증(V&V) 이라 한다[1]. 증명과 검증 프로세스 내에서 크게 소프트웨어 정밀 검사 혹은 동료 검토와 소프트웨어 시험 등의 시스템 검사 및 분석 방법이 이용될 수 있다. 소프트웨어 정밀 검사는 시스템을 검토하여 결함을 찾기 위한 정적인 증명과 검증 프로세스이다. 결함은 개발 기간 중 가능한 한 빨리 발견하여 제거하는 것이 비용이 적게 든다. 구현 초기 단계에서 결함을 제거하면, 시스템 테스트 또는 제품을 배포한 후 결함을 제거하는 것보다 1/10~1/100정도 비용이 절감된다[2]. 형식적 설계 정밀 검사의 결함 감지 비율은 45~65%이고, 형식적 코드 정밀 검사의 결함 감지 비율은 45%~70%라고 알려져 있으며, 형식적 설계 정밀 검사와 형식적 코드 정밀 검사를 조합하면 전체 시스템에서 70%~85% 이상의 결함을 제거한다[2]. 따라서 소프트웨어 정밀 검사는 결함을 발견하는데 있어서 매우 효율적이며, 테스트에 비해서 경제적이다.

동료 검토는 형식적인 정도(Degree of Formality)에 의해서 여러 가지 방식으로 분류될 수 있으며 Karl E. Wiegers는 여러 가지 검토 방식을 형식적인 정도에 따라 정밀 검사, 팀 리뷰, 워크-쓰루, 짝 프로그래밍, 동료 탁상 검사, 코드 읽기, 비 형식적 검토로 분류한다[3]. 이들 중 정밀 검사는 사전 계획, 사전 검토와 정밀 검사 회의, 재 작업, 확인 등의 활동으로 이루어진 형식적인 프로세스 이므로 검토를 수행하기 위한 비용이 가장 많이 소요되는 동료 검토 방식이다. 조직에서 수행하는 모든 프로젝트는 한정된 비용과 일정 제약사항을 가지고 있기 마련이다. 따라서 결함 감지 비율이 가장 높지만 가장 많은 비용과 시간이 소요되는 정밀 검사를 프로젝트의 모든 컴포넌트에 대하여 수행하기는 어렵다. 그러므로 프로젝트에 주어진 비용과 일정을 초과하지 않으면서 정밀 검사의 효율성을 극대화 하기 위한 전략적인 접근방법이 필요하다.

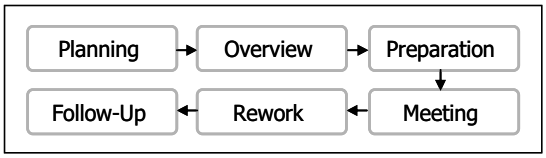
본 논문은 소스코드에 대한 소프트웨어 정밀 검사의 효율성을 극대화하기 위한 컴포넌트 우선순위 선정기법을 제안한다. 우선순위 선정을 위하여 요구분석, 설계, 구현 단계 각각에 대한 지표를 선정하여 종합 평가한다. 요구분석 단계의 지표는 유스케이스 점수(Use Case Points)를 사용하고, 설계 단계의 지표는 설계의 복잡성을 사용하며, 구현 단계의 지표는 함수의 복잡도를 사용한다. 이 세가지 평가 기준을 바탕으로 우선순위 매트릭스를 구성하고 평가 결과를 종합하여 정밀 검사를 위한 컴포넌트 우선순위를 선정한다.

본 논문의 구성은 다음과 같다. 제 2장 에서는 널리 사용되는 정밀 검사 프로세스와 이를 개선하기 위한 관련 연구를 소개하고 제 3장에서는 제안하는 기법에 대하여 설명한다. 제 4장에서는 제안 기법의 프로젝트 사례를 적용한다. 제 5장에서는 결론 및 향후 연구 방향을 제시한다.

2. 관련연구

2.1 Fagan 정밀 검사

Michael E. Fagan에 의해서 개발되었으며 CMM의 KPA 중 동료 검토 에서도 사용되고 있을 정도로 일반적으로 가장 많이 사용되는 프로세스 이다. Fagan 정밀 검사는 (그림 1)과 같이 6단계로 구성된다[3].



(그림 1) Fagan 정밀 검사 프로세스

모든 참석자들에게는 명확하게 정해진 역할이 할당된다. 중개자는 리뷰 대상 코드와 설계, 체크리스트 등 관련 자료를 참석자들에게 사전에 배포하고, 검토자 들은 체크리스트를 바탕으로 결함 목록을 작성하고 정밀 검사 회의에 참석한다. 회의는 결함 해결이 아닌 발견에 중점을 두고 수행되며 회의 후 결함 목록을 취합하여 재 작업을 수행하고 필요할 경우 정밀 검사 회의를 다시 진행한다[2].

대부분의 프로세스와 마찬가지로, Fagan 정밀 검사 프로세스 역시 'what' 만을 정의하고 있다. 따라서 실제 프로젝트에 적용할 때의 현실적인 일정과 비용상의 제약이 고려되어 있지 않다.

2.2 유스케이스 점수 (Use Case Points)

유스케이스 점수는 Gustav Karner에 의해 개발되었으며 소프트웨어의 복잡도와 크기를 측정하기 위하여 유스케이스의 복잡도와 크기를 정량적으로 평가하는 방법이다[4]. 기본 점수(UUCP: Unadjusted Use Case Point)는 유스케이스 모델에서 액터와 유스케이스를 단순, 평균, 복잡으로 분류하여 가중치를 부여한 후 합계를 내는 방식으로 계산된다[4].

$$UUCP = \sum_{i=1}^{6} n_i * W_i$$

보정을 위해 13개의 기술적 요인(TCF)과 8개의 환경적 요인(EF)을 제공한다[4].

$$TCF = C_1 + C_2 \sum_{i=1}^{13} F_i * W_i \quad , \quad EF = C_1 + C_2 \sum_{i=1}^{8} F_i * W_i \quad ,$$

최종 유스케이스 점수는 아래의 공식으로 계산된다[4].

$$UCP = UUCP * TCF * EF$$

유스케이스 모델링은 UML을 이용한 객체지향 분석 설계(OOAD)에서 비즈니스 프로세스와 요구사항을 도출하는데 가장 널리 쓰이는 기법이다[5]. 따라서 유스케이스 점수는 기능 점수(Function Points)에 비하여 객체지향 분석설계를 수행하는 소프트웨어의 크기와 복잡도 분석에 장점을 가진다[5].

2.3 결합도와 응집도

결합도와 응집도는 객체지향 분석 설계(OOAD)를 사용하여 개발된 소프트웨어의 신뢰성과 유지보수성을 나타내는 대표적인 평가 지표이다[6]. 결합도는 클래스들 간의 서로 다른 책임이 얽혀 있어서 상호의존도가 높은 정도를 의미하며, 결합도가 높을수록 클래스의 서로 다른 책임이 복잡하게 얽혀있기 때문에 코드의 가독성이 떨어지고 유지보수가 어려워진다. 좋은 설계는 클래스간의 결합도를 최소화하여 클래스의 독립성을 높인 것을 의미한다. 응집도는 하나의 클래스가 하나의 책임을 완전하게 담당하고 있는 정도를 의미하며 응집도가 높을수록 클래스의 구조는 단순해지며 컴포넌트의 독립성은 높아진다. 클래스의 결합도, 응집도는 객체지향 언어로 개발된 소프트웨어의 복잡도를 측정하는 가장 좋은 방법으로 사용되고 있다[6].

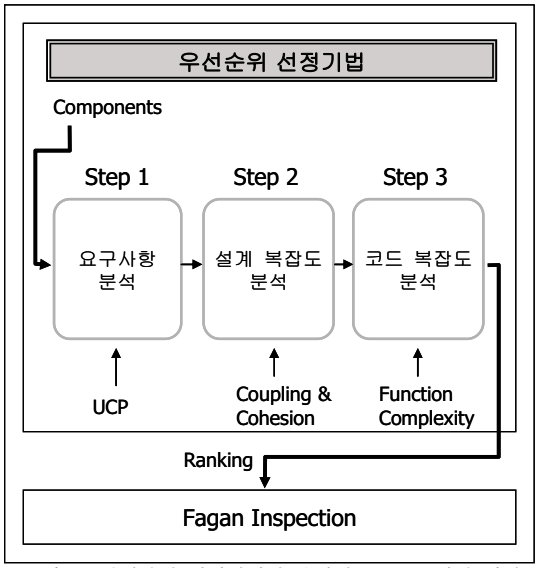
2.4 제어 구조 복잡도 측정

Tom McCabe가 제안한 복잡도 측정 기법으로서 함수 내의 결정점(Decision Points)의 수를 세어 복잡도를 정량적으로 측정하는 방법이다[7]. 함수의 제어 구조에 기반한 방법으로, 함수 내부에 If와 case문, and, or 연산과 루프가 중첩되어 있어 분기될 수 있는 경로가 많을수록 함수의 복잡도는 높아진다. 제어 구조는 소프트웨어 전체의 복잡도에 지대한 영향을 미친다[8]. 제어 구조를 잘못 사용하면 복잡도가 증가하고, 잘 사용하면 복잡도가 감소한다. 제어흐름의 복잡도는 코드의 신뢰성, 오류 발생 확률과 관련이 있기 때문에 중요하다[7].

3. 우선순위 선정기법

3.1 개요

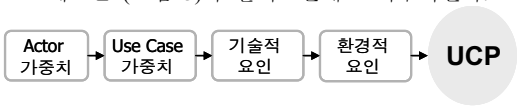
프로젝트는 한정된 일정과 비용 제약을 가지기 때문에 많은 비용과 시간이 소요되는 정밀 검사를 모든 컴포넌트에 대하여 수행할 수는 없다. 따라서 결함을 많이 내포하고 있을 것으로 추정되는 컴포넌트 순으로 우선순위를 부여하고 프로젝트에 주어진 일정과 비용 제약을 고려하여 선정된 우선순위에 따른 정밀 검사를 수행함으로써 정밀 검사에 투입된 리소스 대비 결함 제거율을 높이고, 정밀 검사의 효율성을 극대화할 수 있다. 소프트웨어 개발의 주요 단계는 요구분석, 설계, 구현으로 이루어진다. 본 논문에서 제안하는 컴포넌트 우선순위 선정 기법은 개발 각 단계에서 결함에 가장 큰 영향을 미치는 주요 지표를 선정하여 정량적인 분석을 수행하고 이를 종합적으로 평가하여 정밀 검사를 위한 컴포넌트 우선순위를 선정한다. 우선순위 선정은 정밀 검사 수행 전의 사전 계획 단계이다. 정밀 검사 수행 전에 컴포넌트의 우선순위를 선정하고, 우선순위가 높은 컴포넌트부터 기존의 Fagan 정밀 검사 프로세스를 적용한다. 우선순위 선정 기법은 (그림 2)와 같다.



(그림 2) 우선순위 선정기법이 추가된 Fagan 정밀 검사

3.2 Step 1: 요구사항 분석

결함은 소프트웨어의 크기와 복잡도에 영향을 받으므로 요구사항의 크기와 복잡도를 추정하여 결함밀도를 예측할 수 있다. 우선순위 선정기법에서 요구사항에 대한 크기와 복잡도를 평가하기 위하여 유스케이스 점수를 사용한다. 유스케이스 점수 평가 프로세스는 (그림 3)과 같이 4단계로 이루어진다.



(그림 3) 유스케이스 점수 평가 프로세스

3.3 Step 2: 설계 복잡도 분석

복잡도 관리는 소프트웨어 개발에서 가장 주요한 기술적인 주제이며, 소프트웨어 설계의 가장 중요한 목표는 복잡도를 최소화 하는 것이다. 따라서 설계 복잡도는 결함과 직접적인 연관성을 가지므로 우선순위 선정기법에서 설계에 대한 복잡도를 평가하기 위하여 클래스의 결합도, 응집도에 대한 분석을 수행한다.

3.4 Step 3: 코드 복잡도 분석

제어구조의 복잡성은 코드의 결함 발생 확률을 높이고, 가독성을 떨어뜨려 유지보수를 어렵게 한다. 따라서 우선순위 선정기법에서 코드에 대한 복잡도를 평가하기 위하여 함수의 제어구조 복잡도에 대한 분석을 수행한다.

3.5 우선순위 선정

3.2~3.4에 기술한 요구사항, 설계, 코드에 대한 분석 결과를 바탕으로 우선순위 매트릭스를 생성하고, 컴포넌트의 우선순위를 선정한다. 우선순위 선정 지표는 <표 1>과 같다.

<표 1> 우선순위 선정 지표

요구사항				설계		구현
액터 가중치	유스케이스 가중치	기술적 요인	환경적 요인	결합도	응집도	복잡도

우선순위 선정 지표 평가에 따른 우선순위 매트릭스는 <표 2>와 같다.

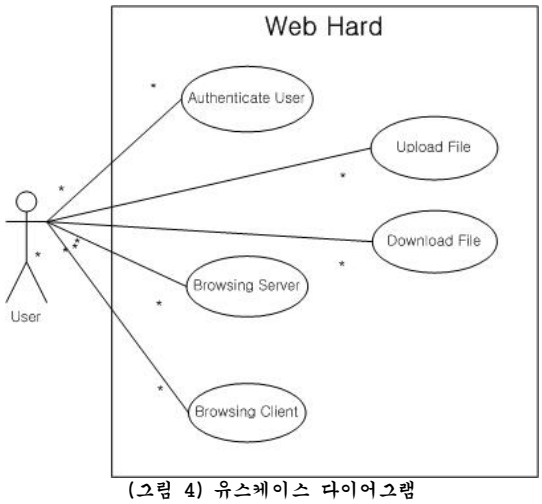
<표 2> 우선순위 매트릭스

구성요소	요구사항	설계	구현	총점	순위
컴포넌트1					
컴포넌트2					

선정된 우선순위를 기준으로 우선순위가 높은 순서대로 정밀 검사를 수행하면 프로젝트의 일정과 비용의 범위 내에서 정밀 검사의 효율성을 극대화 할 수 있다.

4. 사례적용

본 논문에서 제안하는 우선순위 선정기법을 이용한 정밀 검사의 효율성을 검증하기 위하여 웹 하드 시스템 개발에 적용한다. 웹 하드 프로젝트의 전체 시스템은 총 5개의 컴포넌트로 구성되며 각 컴포넌트는 1개의 유스케이스를 구현한다. (그림 4)는 웹 하드 시스템의 유스케이스 다이어그램이다.



(그림 4) 유스케이스 다이어그램

우선 5개의 컴포넌트에 대한 요구사항 평가를 수행한다. 업로드 컴포넌트의 경우 액터 가중치는 GUI를 통해 상호작용하는 사람이므로 3을 부여한다. 유스케이스 가중치는 유스케이스 명세서에 의한 전체 트랜잭션 수가 7 이상이므로 15를 부여한다. 따라서, 조정되지 않은 유스케이스 가중치는 다음과 같다.

UUCP = 3 + 15 = 18

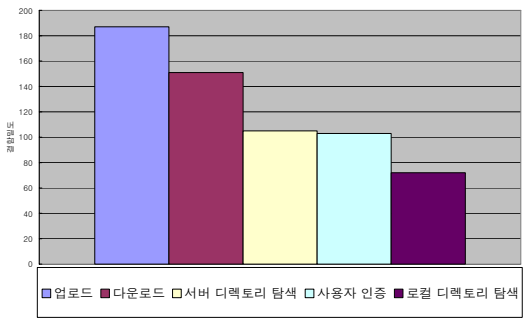
기술적 요인은 13개 인자 각각에 대하여 0~5까지 평가 후 가중치와 상수를 곱하여 합산한다.
TCF = 0.6 + 0.01 * TFactor = 0.6 + (0.01 * 44.0) = 1.04
환경적 요인도 동일하게 8개 인자 각각에 대하여 0~5까지 평가 후 가중치와 상수를 곱하여 합산한다.
EF = 1.4 – 0.03 * EFactor = 1.4 - (0.03 * 13.0) = 1.01
최종적으로, UCP = 18 * 1.04 * 1.01 = **18.9**이다.
위와 같은 방식으로 총 5개의 컴포넌트에 대한 유스케이스 점수를 평가하여 우선순위 매트릭스에 기록한다.

다음은 코드 분석 도구를 사용하여 설계 복잡도와 코드 복잡도를 평가한다. 설계 복잡도는 각 컴포넌트에 포함된 클래스에 대하여 결합도와 응집도를 평가하고 평균값을 구한다. 코드 복잡도는 각 컴포넌트에 포함된 클래스의 메소드에 대하여 제어 구조 복잡도를 평가하고 평균값을 구한다. 평가된 결합도와 응집도, 제어 구조 복잡도 측정 결과를 우선순위 매트릭스에 기록한다. 총 5개 컴포넌트의 평가 결과에 의한 우선순위 매트릭스는 <표 3>과 같이 구성된다.

<표 3> 웹 하드 시스템의 컴포넌트 우선순위 매트릭스

컴포넌트	요구사항	설계	구현	총점	순위
사용자 인증	13.6	14.6	11.0	39.2	4
업로드	**18.9**	16.1	14.0	49.0	1
다운로드	18.5	15.4	13.0	46.9	2
서버 디렉토리 탐색	13.4	14.9	11.6	39.9	3
로컬 디렉토리 탐색	7.8	13.6	10.2	31.6	5

선정된 우선순위를 기준으로 Fagan 정밀 검사를 수행한 결과 정밀 검사에 의하여 발견된 각 컴포넌트의 결함밀도(KLOC당 결함 개수)는 (그림 5)와 같다.



(그림 5) 컴포넌트 별 결함밀도

5. 결론

본 논문에서 제시한 정밀 검사를 위한 컴포넌트 우선순위 선정기법은 시스템을 구성하는 모든 컴포넌트에 대하여 결함밀도가 높을 것으로 예상되는 순서대로 우선순위를 부여하고 실제 프로젝트에서 우선순위에 따라서 정밀 검사를 수행함으로써 주어진 시간과 비용 제약 한 도내에서 정밀 검사의 효율성을 극대화 할 수 있는 방안을 제시하고 있다. 실제 적용 사례에서 보인 바와 같이 우선순위 선정기법에 의해서 평가된 우선순위는 실제 정밀 검사 수행 결과 결함밀도가 높은 컴포넌트의 순서와 일치하는 것을 볼 수 있다.

향후 연구로는 개발 각 단계에서 사용된 평가 지표에 대한 보완과 프로젝트 계획 단계에서 전체 프로젝트의 일정과 비용 대비 정밀 검사 수행에 소요되는 시간과 비용을 산정하여 정밀 검사를 수행할 컴포넌트와 그 외 비형식적인 검토를 수행할 컴포넌트를 구분하는 기준을 제시하고 프로젝트 전체적인 검토 계획을 수립하는 방안에 관한 연구를 진행할 예정이다.

참고문헌

[1] Sommervile, Ian. Software Engineering, 8th Ed. Reading, Mass.: Addison-Wesley, 2007.
[2] Fagan, Michael E. ‘Design and Code Inspection to Reduce Errors in Program Development’, IBM Systems Journal, vol. 15, no. 3, 1976, pp. 182-211.
[3] Wiegers, Karl. Peer Review in Software: A Practical Guide. Boston, MA: Addison Wesley, 2002
[4] Use Case Points - Resource Estimation for Objectory Projects, Gustav Karner, Objective Systems SF AB (copyright owned by Rational Software), 1993
[5] Roy K. Clemmons. ‘Project Estimation With Use Case Points’, The Journal of Defense Software Engineering, 2006 Issue
[6] V. B. Misic, S. Moser, "Measuring Class Coupling and Cohesion : A Formal Metamodel Approach," APSEC'97, pp.31-40, Dec., 1997.
[7] T. J. McCabe, ‘A Complexity Measure’, IEEE Trans. On Software Engineering, Vol. Se-2, No. 4, PP. 308-319, Dec. 1976
[8] Steve McConell, ‘Code Complete’, 2nd Ed. Microsoft Press, 2005.
[9] Wiegers, Karl. ‘More About Software Requirements’, Microsoft Press, 2006.