소프트웨어 인스펙션 척도의 기준치 비 의존 상대적 데이터 분석

김태현*, 박진희*, 최옥주*, 신주환**, 백종문*
*한국과학기술연구원 전산학과
**국방과학연구소
e-mail :*{thyun, jh_park, okjoo.choi, jbaik}@kaist.ac.kr, **jhshin@add.re.kr

Relative Data Analysis of Software Inspection Metrics without Threshold

Taehyoun Kim*, Jinhee Park*, Okjoo Choi*, Juhwan Shin**, Jongmoon Baik*
*Dept. of Computer Science, KAIST
**Agency for Defense Development

요 약

소프트웨어 개발에 있어 각 단계별 프로세스 활동들에 대한 분석 및 평가는 소프트웨어의 품질을 좌우하는 큰 요인이다. 따라서 많은 소프트웨어 척도들이 소프트웨어 품질을 분석하는데 이용되고 있으며 유사 프로젝트를 통해 설정되는 기준치와 척도 값의 비교가 수행된다. 하지만 기존의 유사 프로젝트를 찾기란 쉽지 않은 일이며 유사 프로젝트를 찾더라도 해당 프로젝트의 개발 환경은 현재 개발 중인 프로젝트의 환경과 다른 경우가 많다. 따라서 본 논문에서는 외적인 기준치에 의존하지 않고 현재 개발 단계의 인스펙션 결과를 분석하는 방법을 제시하도록 한다. 산포도를 이용한 상대적 데이터 분석이 이용되며 국방 도메인에서 개발 중인 프로젝트 내부 31 개의 기능으로부터 수집된 데이터를 통한 사례분석을 수행하도록 한다. 이를 통해 기능들 간 현재 개발 과정의 일관성 유지 여부를 평가하고 다음 개발 단계의 프로세스 활동 강화 여부에 대한 권고 사항을 제시할 수 있다.

1. 서론

소프트웨어는 개발 단계에서의 시스템 분석 및 설계 공정이 소프트웨어의 품질을 좌우한다. 이에 따라 소프트웨어 개발 과정 및 인스펙션에 대한 분석이 중요시 되고 있으며 많은 척도들이 이에 대한 분석을 위해 이용되고 있다. 소프트웨어 척도는 소프트웨어 측정 기술을 기반으로 소프트웨어 생명 주기 동안에 소프트웨어 특징 또는 특성을 객관적이고 과학적인 수치로 정량화 하는 기술이다 [1]. 소프트웨어 척도의 유형은 일반적으로 규모, 복잡도, 품질 척도 등으로 나뉘며 개발 단계별로 다양한 척도가 존재한다 [2-4]. 이 중 기능의 규모 대비 발견된 결함 수를 나타내는 결함 밀도 (Defect Density)는 각 단계의 인스펙션 과정에 대한 분석에 있어 인스펙션을 통해 얼마만큼의 결함을 검출해 내는 가를 확인하는 중요한 척도라 할 수 있다 [2].

하지만 단순히 결함 밀도만을 갖고 인스펙션 단계를 분석하기에는 부족한 점이 존재한다. 결함 밀도가 높다는 것은 인스펙션 단계가 잘 이루어 졌다는 것을 의미할 수 있지만 다른 측면에서는 기능의 품질이 떨어지기 때문에 그만큼 많은 결함이 발견되는 것으로 해석이 가능하다. 또한 결함 밀도가 낮은 경우, 이는 기능의 품질이 높다는 것을 의미하거나 인스펙션이 잘 이루어지지 않았다는 것을 의미할 수 있다. 이렇듯 하나의 척도 만으로는 그 값이 정확한 분석을 나타내기 어렵다. 따라서 더 정확한 분석을 위해 이를 보완하기 위한 다른 척도와의 결합이 필요하다.

Linda M. Laird 는 프로세스의 효율성을 측정할 수 있는 척도로 기능에 대한 규모 대비 인스펙션 시간을 나타내는 Inspection Intensity 를 정의하고 결함 밀도와의 결합을 통한 분석 방법을 제시하였다 [5]. Inspection Intensity 는 각 인스펙션 단계에서 얼마만큼의 시간을 할애하였는가에 대한 척도로 인스펙션에 대한 효율성을 나타낸다. Linda M. Laird 는 결함 밀도와 Inspection Intensity 두 척도를 이용하여 각 척도의 기준치 충족 여부에 따른 현재 개발 단계에서의 인스펙션 분석 결과를 4 가지 시나리오로 구분 지어 각 시나리오 별 다음 개발 단계에 대한 권고사항을 제시하고 있다. 이 과정에서 이용하는 척도 기준치는 신뢰할 수 있는 기존의 유사 프로젝트를 기반으로 성공적으로 수행되었던 인스펙션 프로세스의 분석 값을 이용하게 된다. 하지만 프로젝트의 성공 여부에 대한 기준이 모호한 경우가 많으며 유사 프로젝트의 성공

프로세스를 찾는다 하더라도 프로젝트의 개발 환경이 현재 개발 중인 프로젝트와 일치하기란 쉽지 않다. 따라서 대부분의 소프트웨어 개발은 기존의 유사 프로젝트에서 성공한 프로세스의 분석 값, 즉 의미 있는 척도 기준치를 찾기가 쉽지 않다 [6].

따라서 본 논문에서는 두 척도간의 결합에 대하여 기존의 성공 프로세스를 기반으로 하는 척도 기준치에 의존하지 않고 상대적으로 현재 개발 단계의 인스펙션 결과를 분석하는 방법을 제시하도록 한다. 이를 통해 기능들 간 현재 개발 과정의 일관성 유지 여부를 평가할 수 있으며 다음 개발 단계에 대한 권고 사항을 제시할 수 있다. 외적인 척도 기준치가 존재하지 않는다는 가정하에 산포도를 이용하여 소프트웨어의 각 기능들에 대한 두 척도간의 상대적 평가를 수행한다. 산포도는 두 변수간의 관계를 알아보기 위해 값을 나타내는 점들을 도표화한 것이다. 이를 통해 두 척도 값에 대한 각 기능간의 상대적인 관계를 나타낼 수 있다. 제안된 방법에서는 산포도를 이용하여 개발하고자 하는 소프트웨어의 각 기능들에 대한 결함 밀도 및 Inspection Intensity 척도의 상대적인 평가를 내리고 이를 통해 현 개발 단계에 대한 인스펙션 결과 분석 및 다음 단계에서의 권고사항을 판단하도록 한다.

본 논문은 다음과 같이 구성된다. 2 장에서는 결함 밀도와 Inspection Intensity 에 대해 간략히 설명한 후 두 척도를 이용한 개발 단계의 인스펙션 결과 분석 방법에 대한 관련 연구를 소개한다. 3 장에서는 논문에서 제시하는 산포도를 이용한 개발 단계의 인스펙션 결과 분석 방법을 설명한다. 4 장에서는 현재 진행중인 실제 국방 소프트웨어의 초기 개발 단계에 대한 수집 데이터를 이용하여 사례연구를 한다. 마지막으로 5 장에서는 결론 및 향후 연구에 대한 기술을 하도록 한다.

2. 관련 연구

결함 밀도(Defect Density)는 인스펙션 하고자 하는 기능의 규모 대비 발견된 결함 수를 나타내며 이는 다음과 같은 식으로 표현된다 [2].

$$\text{Defect Density} = \frac{Total\ Number\ of\ Defects}{Inspection\ Object\ Size}$$

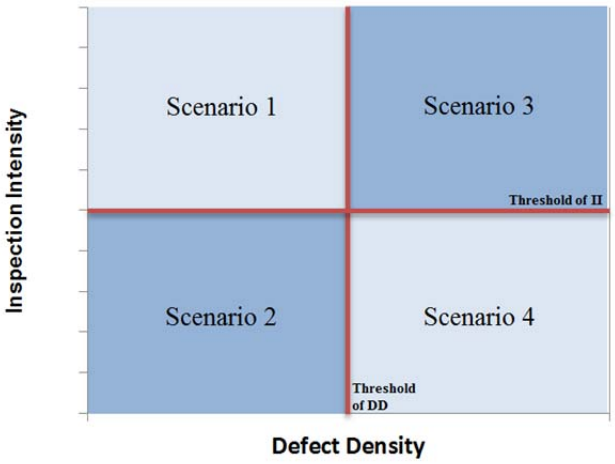
위의 식에서 Total number of Defects 는 인스펙션 과정에서 발견된 총 결함 수를 말한다. Inspection Object Size 는 인스펙션 하고자 하는 기능의 규모를 말하며 SLOC, Function Point, Page 수로 표현될 수 있다.

Inspection Intensity 는 인스펙션 하고자 하는 기능의 규모 대비 인스펙션 수행에 걸린 시간을 나타내며 아래와 같은 식으로 표현된다 [5].

$$\text{Inspection Intensity} = \frac{Inspection\ Times}{Inspection\ Object\ size}$$

위의 식에서 Inspection Times 는 인스펙션 수행에 걸린 총 시간을 말하며 hours, minutes, seconds 단위로 나타낼 수 있다. Inspection Object Size 는 위의 결함 밀도의 식에서 이야기하는 바와 동일하다.

위의 두 척도는 서로 결합하여 현재 인스펙션 단계 수행에 대한 적정성 여부 판단 및 다음단계에 대한 권고사항을 제시하는데 이용이 가능하다. 이에 대한 분석 결과는 각 척도의 기준치 충족여부에 따라 4 가지 시나리오로 구분 지을 수 있다 [5]. (그림 1)은 척도 별 기준치 충족여부에 따라 각각의 상태를 구분 지어놓은 그래프이며 <표 1>은 그래프에서 나타난 각 시나리오 별 조건 및 다음 단계의 권고 사항을 나타낸 것이다.



(그림 1) 기준치에 따른 4 가지 시나리오 구분

<표 1> 4 가지 시나리오에 대한 권고 사항 표

Scenario	조건	권고 사항
Scenario 1	결함 밀도가 기준치 보다 낮고, Inspection Intensity 가 기준치 이상일 경우	software quality 가 기대치 이상으로 좋음. 개발 과정이 잘 진행되고 있음.
Scenario 2	결함 밀도가 기준치 보다 낮고, Inspection Intensity 가 기준치 미만일 경우	결함 발견이 잘 진행되지 않음. Re-inspection 혹은 다음 단계의 detection effort 를 보강할 필요가 있음.
Scenario 3	결함 밀도가 기준치 보다 높고, Inspection Intensity 가 기준치 이상일 경우	현 단계에서 높은 Inspection Intensity 로 인해 많은 결함이 발견된 것으로 예측. Code quality 는 기대치를 만족하나 이후 단계에 대한 관찰이 필요.
Scenario 4	결함 밀도가 기준치 보다 높고, Inspection Intensity 가 기준치 미만일 경우	개발이 잘 이루어지고 있지 않음. 스케쥴 및 예산에 대한 재조정이 필요.

4 가지 시나리오를 통해 개발자는 현재 진행 중인 단계에 대한 개발 과정이 잘 수행되고 있는지 여부를 확인할 수 있다. 하지만 이 시나리오에서는 두 척도에 대해 각각의 기준치 설정 여부를 사전 조건으로 제시하고 있으며 실제 소프트웨어 개발 과정에서는 척도에 대한 기준치를 설정하기란 쉽지 않다.

3. 산포도를 이용한 분석 방법

이 절에서는 외적 기준치에 의존하지 않고 산포도를 이용하여 두 척도간의 결합에 대한 현재 개발 단계의 인스펙션 결과를 분석하는 방법을 제시하도록 한다. 외부 기준치가 없는 프로젝트의 경우 두 척도간 결합에 대해 4 가지 시나리오의 분석 방법을 적용하기가 어렵다. 본 논문에서 제시하는 산포도를 이용한 상대적 분석 방법은 이러한 문제를 해결하기 위해 프로젝트 내부 기능들 간의 상대적인 기준치를 제시함으로써 시나리오 분석이 가능토록 한다. 이를 통해 각 기능들간 현재 개발 과정의 일관성 평가 및 다음 개발 단계에서의 권고 사항을 도출해 낼 수 있다. 분석을 위해서는 다음과 같은 절차를 거친다.

STEP 1. 데이터를 수집하고 이에 대한 결함 밀도와 Inspection Intensity 에 해당하는 척도 값을 도출한다.
STEP 2. 도출한 척도 값들을 이용하여 산포도를 완성하고 전체 기능에 대한 두 척도 각각의 평균 값을 구하여 상대적 기준치를 정하도록 한다.
STEP 3. 상대적 기준치를 적용한 산포도를 통해 각각의 기능이 어떠한 시나리오에 해당하는지를 확인한다.

분석 결과를 통해 나타난 시나리오는 각 기능이 다른 기능들에 비해 해당 시나리오의 상황에 가까움을 뜻한다. 따라서 이에 대한 <표 2>와 같은 상대적인 재해석이 필요하다.

<표 2> 4 가지 시나리오에 대한 재해석된 권고 사항 표

Scenario	재해석된 권고 사항
Scenario 1	다른 기능들에 비해 Software quality 가 높음을 의미한다.
Scenario 2	다른 기능들에 비해 결함 발견이 잘 이루어지지 않았음을 의미한다.
Scenario 3	다른 기능들에 비해 Code quality 가 만족됨을 의미한다.
Scenario 4	다른 기능들에 비해 개발이 잘 이루어 지지 않고 있음을 의미한다.

재해석된 4 가지 시나리오를 통해 상대적인 기준치에 대하여 각각의 기능이 다른 기능들에 비해 상대적으로 어떠한 시나리오에 해당하는지를 분석할 수 있다. 산포도를 이용한 분석 결과는 각 기능들에 대해 상대적인 분석이기 때문에 각 시나리오에 해당하는 권고사항이 절대적인 분석을 나타내지는 않는다. 하지만 이를 통해 다른 기능에 비해 상대적으로 부족한 부분을 분석하고 다음 개발 단계로 진행됨에 따라 각 기능들은 이전 단계보다 더 나은 개발 및 인스펙션을 수행하도록 권고된다. 또한 이러한 분석 및 보완 작업은 개발 단계를 거치면서 반복적으로 수행된다. 이와 같이 각 단계별 분석이 반복적으로 시행되고 이를 통해 상대적으로 부족한 기능들에 대한 보완이 수행됨에 따라 전체적인 소프트웨어 품질은 점진적으로 증가하고 일관성을 갖게 된다.

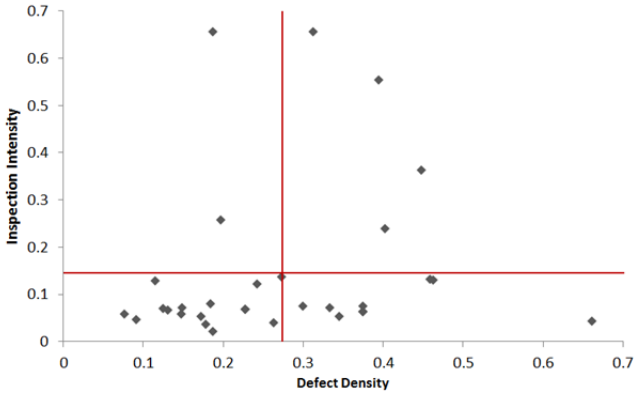
4. 사례 분석

사례 분석에 이용한 데이터는 실제 국방 프로젝트에서 수행하고 있는 소프트웨어에 대한 요구사항 단계의 인스펙션 수집 데이터를 이용하였다. 데이터는 소프트웨어 내부 31 개의 기능에 대한 요구사항 단계의 2 가지 버전 데이터를 이용하도록 한다. 국방 데이터의 특성상 척도에 따른 자세한 데이터 분석 결과는 감추도록 한다.

요구사항 단계 버전 1a 에서 각 척도의 상대적 기준치는 <표 3>과 같이 도출되었으며 이를 기준으로 한 산포도의 분석 결과는 (그림 2)와 같다. 이를 통해 첫 번째 분석 결과에서는 상대적으로 시나리오 1 과 3 에 비해 2 와 4 에 많은 기능이 분포되어 있음을 확인할 수 있다. 시나리오 2 와 4 는 결함 발견이 잘 이루어지지 않고 있거나 심지어는 개발 자체에 대한 스케쥴 및 예산 변경이 권고되는 영역이다. 이는 개발 초기 단계의 이른 버전에 대한 데이터이기 때문에 다소 불안한 결과를 나타내는 것으로 분석되며 이를 해결하기 위해 많은 기능에 대한 개발 및 인스펙션의 노력이 필요할 것으로 분석된다. 또한 분포가 어느 한 곳에 밀집되어 있지 않고 방산되어있는 형태를 이루는 것으로 보아 각 기능들에 대한 소프트웨어 품질이 일관성을 띠지 않음을 확인할 수 있다. 위에서 분석된 결과 및 권고사항은 다음 개발 단계에 적용되어 개발 중인 소프트웨어의 품질을 점진적으로 높여 나가게 된다.

<표 3> v1a 의 각 척도 별 상대적 기준치

Metric	상대적 기준치
Defect Density	0.27 fault/Page
Inspection Intensity	0.14 hour/Page

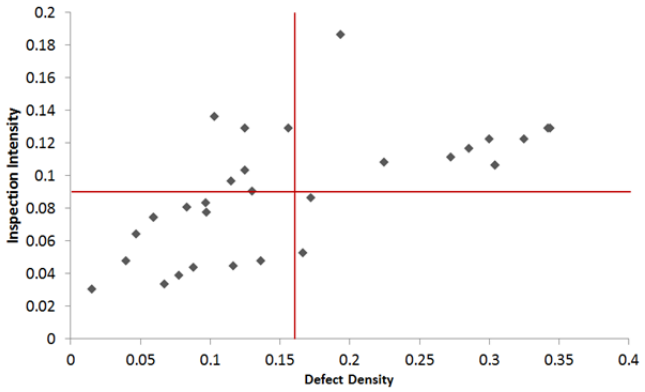


(그림 2) v1a 의 31 개 기능에 대한 척도 값 산포도

요구사항 단계 버전 1.0 에서 각 척도에 대한 상대적 기준치는 <표 4>와 같이 도출되었으며 이에 대한 산포도의 분석 결과는 (그림 3)과 같다. (그림 2)에 나타나는 첫 번째 분석 결과에 비해 시나리오 4 에 해당하는 기능이 많이 줄어들었으며 기능들이 어느 정도 밀집 군을 형성함으로써 각 기능들에 대한 소프트웨어 품질이 일관성을 찾아가고 있음을 알 수 있다. 또한 상대적으로 시나리오 1,2,3 에 기능들이 많이 분포 되어있음을 확인할 수 있는데 이 중 시나리오 2 에 해당하는 기능들은 다른 시나리오에 위치한 기능들에 비해 상대적으로 결함 발견이 잘 이루어지지 않고 있음을 뜻한다. 따라서 시나리오 2 에 해당하는 기능들에 대한 다음 단계의 결함 발견이 보충되어야 함을 알 수 있다.

<표 4> v1.0 의 각 척도 별 상대적 기준치

Metric	상대적 기준치
Defect Density	0.16 fault/Page
Inspection Intensity	0.09 hour/Page



(그림 3) v1.0 의 31 개 기능에 대한 척도 값 산포도

5. 결론 및 향후 연구

본 논문에서는 각 척도에 대한 기존 유사 프로젝트의 성공 프로세스 기준치가 존재하지 않는다는 가정하에 산포도를 이용하여 개발 단계의 인스펙션을 상대적으로 분석하는 방법을 기술하였다. 기존 유사 프로젝트의 성공 프로세스에 대한 분석 값을 구하는 것은 일반적으로 어려운 것이 사실이며 이에 대한 기준 또한 모호하다. 또한 새로운 유형의 소프트웨어 개발에서는 기존의 유사 프로젝트가 존재하지 않기 때문에 기준치를 잡기가 어렵다. 본 논문에서는 기존의 기준치에 의존하지 않고 프로젝트 내부의 기능들 간의 상대적인 기준치를 적용함으로써 개발 단계가 진행됨에 따라 소프트웨어 품질을 점차적으로 증진시킬 수 있는 분석 방법을 제시하였다. 이 방법을 통해 상대적으로 품질이 낮은 기능들은 점진적인 보완을 수행 함으로써 이전 단계에 비해 모든 기능들에 대한 소프트웨어 품질을 높이고 일관성을 갖도록 할 수 있다.

향후 연구에서는 아직 수집되지 않은 요구사항 이후 단계의 실제 국방 데이터를 이용하여 제안된 방법을 적용해 볼 계획이다. 산포도에 나타난 결과들은 이전 단계의 결과 및 분석에 영향을 받을 수 있기 때문에 향후 많은 단계의 개발이 진행되면 이에 대한 연구도 필요할 것으로 확인된다. 또한, 개발 과정에 대한 공정관리 방법으로 통계적 공정 관리 (Statistical Process control)가 일반적으로 많이 사용된다 [7]. 이는 프로젝트의 특성이 설계사양 혹은 품질 규격과 일치하는지를 통계적으로 측정하고 평가함으로써 공정을 효율적으로 관리해 나가는 방법을 말한다. Nancy Eickelman 는 공정관리에 있어 SPC 의 중요성을 제시하고 이를 위해 필요한 요소들을 정의하였다 [8]. 현재 사례분석에서 이용한 프로젝트의 데이터는 두 버전의 개발 단계만이 존재하기 때문에 SPC 를 이용하기에는 데이터가 다소 부족하다. 추가적인 개발단계가 진행되면 SPC 에 대한 분석을 통해 논문에서 제시하는 분석 방법과의 비교를 수행해볼 계획이다.

Acknowledgement

이 논문은 2010 년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2010-0014375).

참고문헌

[1] Stephen H. Kan, "Metrics and Models in Software Quality Engineering", Pearson Education India, 2003.
[2] IEEE, “Std 982-2, IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software", IEEE Computer Society, 1988.
[3] ISO/IEC, "ISO/IEC 9126-2 Software engineering - Product quality- part2: External metrics", 2002.
[4] ISO/IEC, "ISO/IEC 9126-3 Software engineering - Product quality- part3: Internal metrics", 2002.
[5] L. M. Laird and M. C. Brennan, “Software measurement and estimation: a practical approach”, Wiley-IEEE Computer Society Press, 2006.
[6] S. Herbold, J. Grabowski, and S. Waack, “Calculation and optimization of thresholds for sets of software metrics” Empirical Software Engineering, 2011.
[7] D. E. Coleman, “Statistical Process Control—Theory and Practice”, Technometrics, 1993.
[8] N. Eickelmann and A. Anant, “Statistical process control: what you don’t measure can hurt you!”, Software, IEEE, 2003.