Javadoc 대상 테스트 요구사항 추출 기법의 정확도 평가

김지웅 홍신
한동대학교 전산전자공학부
{jeewoong, hongshin}@handong.edu

Evaluation of Test Requirement Extraction Techniques for Javadoc Description

Jeewoong Kim Shin Hong
School of Computer Science and Electrical Engineering, Handong Global University

요약

본 논문은 Javadoc과 같은 주석으로부터 테스트 오라클(test oracle)의 요구사항 명세를 추출하는 tComment, Toradocu, Jdoctor 기법을 오픈소스 프로젝트인 JFreeChart 내 총 360개 메소드에 적용하여 테스트 요구사항 기법의 두 가지 정확도 척도, 즉 실제 Javadoc 문서에서 테스트 요구사항을 내포하는 구문을 얼만큼 정확히 인식하는지, 인식한 요구사항 내포 구문으로부터 요구사항 명세를 얼만큼 정확히 추출하는 지 실험적으로 평가하는 사례연구를 수행했다. 실험 평가 결과, 세 기법은 실제 테스트 요구사항 명세를 포함하는 Javadoc 구문 중 43.2%에 대해서 정확하게 요구사항을 추출함을 확인하였다. 반면, 56.8%에 대해서는 요구사항 인식이나 정확한 추출에 실패함을 확인하였는데, 특히 이러한 요구사항에는 추상적, 복합적, 범위표현, 타입검사에 대한 특징이 있음을 구체적으로 파악할 수 있었다.

1. 서론

퍼징(fuzzing)[1], 심볼릭 테스팅(symbolic execution)[2] 등 테스트 입력을 자동으로 생성하는 기술이 실제 소프트웨어 개발 현장에서 널리 적용됨에 따라, 테스트 오라클(test oracle), 즉 테스트 실행 결과를 요구사항에 비추어 자동으로 검사하기 위한 명세의 자동 생성에 대한 기술적 수요가 증가하고 있다. 테스트 오라클 자동 생성 기술은, 높은 수요에도 불구하고, 현재까지 매우 한정적인 기법만 개발된 상황이며, 테스트 입력 생성 기술에 비하여, 발전 속도도 제한적인 수준이다[3]. 테스트 입력 생성 기법의 경우, 기존에 개발되어 온 정적/동적 프로그램 분석 기법을 활용해 테스트 대상 프로그램 자체에 내포된 구조 정보와 의미 정보를 테스트 입력 생성에 효과적으로 활용할 수 있는 반면, 테스트 오라클 생성에는 테스트 대상 코드의 직접적 활용이 어려우며, 테스트 대상 코드와 별개로, 개발자가 의도한 테스트 요구사항을 추정하는 새로운 형태의 분석 기법이 요청되는데 상황이다.

테스트 오라클 자동 생성에 활용도가 높을 것으로 기대되는 새로운 프로그램 분석 기법으로 개발자가 작성한 Javadoc과 같은 자연어 주석으로부터 테스트 요구사항을 추출하는 기법이 개발되고 있다[4][5][6]. 개발자가 자연어로 기술한 코드 주석은 테스트 대상 프로그램 코드와는 별개로 개발자가 의도한 코드의 기능과 요구사항에 대한 고차원적 설명이 내포되어 있다. 또한, 테스트 대상 프로그램에는 결함이 존재하더라도, 주석에는 해당 코드의 정확한 동작에 대한 정보가 존재할 수 있으므로, 테스트 실행을 검사하는 올바른 조건을 발견할 수 있는 정보를 내포하는 경우가 많다. 코드 주석으로부터 테스트 요구사항이 정확하게 파악되고 추출된다면 자동 테스트 생성[7][8], 테스트 케이스 추가 및 향상에 다양하게 활용될 수 있을 것으로 기대된다.

본 연구는 정부의 재원으로 한국연구재단의 지원을 받아 수행되었음.
(NRF-2017R1C1B1008159, NRF-2017M3C4A7068179)

코드 주석으로부터 테스트 오라클에 활용할 수 있는 요구사항 명세를 자동으로 추출하는 기술은 다음의 두 가지 기능의 정확하고 효과적인 수행이 필요하다: (1) **요구사항 구문 식별**: 어떠한 구문이 테스트 요구사항에 대한 정보를 포함하고 있는지를 확실하게 식별하는 기능과 (2) **요구사항 추출**: 식별된 구문으로부터 해당 요구사항 명세를 정확하게 추출(번역)하는 기능이 필요하다.

코드 주석으로부터 요구사항을 추출하는 기법에 대한 대부분의 연구[4][5][6]는, 각 기술이 집중하는 요구사항 구문 패턴에 한정하여, 해당 패턴의 요구사항에 대한 보다 정확한 추출에 보다 집중하고 있다. 반면, Javadoc과 같은 실제 코드 구문에 어떠한 종류의 요구사항이 존재하며, 현재 제안된 기법들이 이들을 포괄적으로 식별하는 지에 대한 평가는 매우 제한적인 실정이다.

예를 들어, Javadoc을 대상으로 한 요구사항 추출 기법인 tComment[4]와 Toradocu[5]는 파라미터로 `null`이 주어지는 경우와 Exception발생에 대한 요구사항 패턴에 특화되어 있으며, Jdoctor[6] 역시 파라미터의 값 비교 등 제한된 요구사항 패턴에 한정하여 요구사항을 추출한다. 반면, 실제 Javadoc에는 기존 기법이 고려하지 않은 보다 일반적인 형태의 요구사항을 내포하는 구문이 존재한다. 한 가지 예로, 아래의 주석은 JFreeChart의 `XYSeries` 클래스의 `getMinX` 메소드에 대한 주석 중 일부다:

```
/** Returns the smallest x-value in the series,
  * ignoring any Double.NaN values.
  * This method returns Double.NaN if there is
  * no smallest x-value
  * (for example, when the series is empty).
```

이 주석 구문 중 음영으로 표시한 부분은 `getMinX` 메소드가, `series`가 비어 있는 경우, `Double.NaN`을 반환해야 한다는 기능적 요구사항을 표현하고 있으며, 이는 테스트 요구사항으로 직접적으로 활용될 수 있다. 반면, 앞서 언급한

세 가지 기법 모두 메소드 서술 영역(description block)에 작성된 요구사항을 추출하지 못하였으며, 이와 같이 Javadoc에는 다양한 형태로 요구사항이 존재함에도 현재까지 제안된 기술들은 요구사항을 추출하기에는 한계가 존재한다. 현재까지 제안된 기술을 테스트 오라클 생성 기술 요소로 기반하여 새로운 기법을 개발하기 위해서는, 현재까지 제안된 요구사항 추출 기술의 정밀도(precision)는 물론 요구사항 구문 식별 기술의 재현도(recall)에 대한 구체적이고 객관적인 평가가 필요하다.

본 논문은 실제 오픈소스 Java 프로젝트의 Javadoc 주석구문을 대상으로 최근에 개발된 세 가지 기법 tComment, Toradocu, Jdoctor를 적용한 후, 수작업으로 파악한 Javadoc 요구사항 명세에 관련된 구문 중 얼마만큼 정확하고 포괄적으로 탐지하는 지, 또한 추출(번역)된 명세가 정답(ground truth)과 얼마만큼 일치하는 지를 평가하였다. 오픈소스 Java 프로젝트인 JFreeChart의 총 360개 메소드 주석(총 11개 클래스)을 대상으로 한 평가 결과, 총 250개의 요구사항 명세 구문 중, 3개 기법이 총합 111개를 올바르게 식별(탐지)하였으며, 이 중 108개는 정확하게 요구사항이 추출되었다. 본 논문은 3개 기법이 탐지하지 못한 139개 요구사항의 특징(4.1절), 3개 기법이 정확하게 탐지한 111개와 정확하게 추출한 108개 요구사항 구문의 종류를 분석한 결과(3.2절)를 소개한 후, 향후 연구 주제를 논의한다(4.3절).

2. Javadoc으로부터 요구사항을 추출하는 기법

본 연구의 주제가 되는 tComment[4], Toradocu[5], Jdoctor [6]는 Javadoc의 특정 패턴의 구문을 탐지한 후, 특정한 규칙에 따라 요구사항을 추출한다. 본 장에서는 각 기법이 추출하는 Javadoc 구문 대상의 특징과 추출하는 요구사항의 특징, 그리고 동작 방식을 개관한다.

1) tComment. tComment[4]는 메소드에 대한 Javadoc 주석 중 특정 파라미터 P에 대한 `@param` 태그 주석 구문과 특정 Exception E에 대한 `@throws` 주석 구문에 `null`이 포함된 경우를 탐지한 후, 특정 파라미터에 `null`값이 들어올 경우 특정 Exception을 발생(throw) 시켜야 한다는 형태의 요구사항을 "P == null => E" 형태 탬플릿에 맞추어 추출한다. 예를 들어, 한 메소드에 "`@param key the key null not permitted`"와 "`@throws IllegalArgumentException if key is null`"가 주석으로 기재되어 있을 경우, tComment는 "key==null => IllegalArgumentException"을 추출하는데, 이 요구사항 명세는 해당 메소드 호출 시 `key` 파라미터에 `null`값이 올 경우, `IllegalArgumentException`이 반드시 발생해야 한다는 요구사항을 뜻한다. 추가로, `@param` 혹은 `@throws`태그 주석에 "`null`"과 "`not`"이 동시에 나타난 경우에는 "key != null"을 Exception의 조건식으로 하여 요구사항을 추출한다.

2) Toradocu. Toradocu[5]는 메소드에 대한 Javadoc 주석 중 `@throws` 또는 `@exception` 주석 구문에 특정 어휘적(lexical) 패턴이 일치할 때 특정한 패턴의 요구사항을 생성하도록 정의된 규칙에 따라 작동한다. Toradocu가 탐지하는 구문 패턴과 요구사항을 추출하는 템플릿은 다음과 같다:

- Exception E에 대한 `@throws` 혹은 `@exception` 구문에 "X is/are positive" 형태의 표현이 나오고 X에 어휘적으로 직접 대응되거나 어휘적으로 유사한 변수/메소드 X'가 있을 경우 "X' > 0 => E"을 추출함. "X is/are negative" 경우도 대칭적으로 "X' < 0 => E"을 추출함.
- Exception E에 대한 `@throws` 혹은 `@exception` 구문에 "X is/are < 1" 형태의 표현이 나오고 X에 어휘적으로 직접 대응되거나 어휘적으로 유사한 변수/메소드 X'가 있을 경우 "X' < 1 => E"을, "X <= 0"의 경우는 "X' <= 0 => E"을 추출함.
- Exception E에 대한 `@throws` 혹은 `@exception` 구문에 "X is/are true" 형태의 표현이 나오고 X에 어휘적으로 직접 대응되거나 어휘적으로 유사한 변수/메소드 X'가 있을 경우, "X' == true => E"을 추출함. "X is/are false"의 경우는 "X' == false => E", "X is/are null"의 경우는 "X == null => E"를 추출함.

예를 들어, "`@throws IndexOutOfBoundsException if the dataset is empty`"라는 구문이 Javadoc에 있다면, "is empty"와 가장 유사한 이름의 메소드인 "`isEmpty()`"를 대응시켜서 "dataset.isEmpty() == true => E"를 요구사항으로 추출한다.

3) Jdoctor. Jdoctor[6]는, Toradocu와 유사하게, 메소드에 대해 작성된 Javadoc 주석 중 `@param`, `@return`, `@throws` 태그로 작성된 구문에 대해 정의된 구문 탐지 패턴, 요구사항 추출 템플릿에 의해 요구사항을 추출한다. Jdoctor가 사용하는 요구사항 추출 규칙은 Toradocu에 정의된 규칙들과 관련 연구[9][10]에 정의된 규칙을 활용한다. 다만, Jdoctor는, Toradocu 그리고 ALICS[9]와는 달리, 구문 패턴 인식에 어휘적(lexical) 패턴 매칭을 사용하는 대신, 어휘를 Word2Vec 방식으로 임베딩한 후, 구문패턴과의 거리(distance)가 가까운지를 판별하는 방식으로 구문패턴을 탐지한다.

앞서 Toradocu에서 소개한 것 외에 Jdoctor가 관련연구[9][10]로부터 차용하는 추출 규칙은 다음과 같다:

- 태그 구문에 "X is an instance of T"라는 표현이 있고 X에 특정 객체 X'를, T에 특정 클래스 명 T'을 대응할 수 있을 경우"(X' instanceof T') == true" 형태의 요구사항을 추출한다.
- 태그 구문에 "X and Y is smaller than V"가 있고, X와 Y에 멤버 변수 X'과 Y'이 대응되고 상수 V가 대응될 경우, "X' < V && Y' < V"를 요구사항으로 추출한다. "X or Y is smaller than V"도 대응적으로 "X' < V || Y' < V"로 추출한다.
- 태그 구문에 "X is in range [V1, V2]" 혹은 X is between V1 and V2"가 있을 경우 X, V1, V2 각각에 대응되는 변수 X'에 대하여 "V1 <= X' && X' <= V2"를 생성한다.
- 태그 구문에 "X is one of S"라는 표현이 있고 X에 객체 X'가 대응되고 S에 Set 타입 객체 S'이 대응 될 경우, "S'.contains(X') == true"를 요구사항으로 추출한다.
- 태그 구문에 "X is out of bounds of A"라는 표현이 있고 X에 객체 X'가 대응되고 A에 Array 객체 A'이 대응 될 경우, "(0 <= X' && X' <= A'.length) == false"를 추출한다.

이와 같이 현재까지 제시된 문서로부터 요구사항을 추출하는 기법은 메소드 단위로 태그가 명시된 Javadoc 구문에 대해 이미 정의된 특정한 구문 패턴을 탐지한 후 이미 정의된 요구사항 템플릿에 맞는 파라미터를 유추하여 요구사항 명세를 생성하도록 가능함을 알 수 있었다.

3. 실험 설계 및 결과

3.1. 실험 설계

1) 연구 질문. 본 연구에서는 tComment, Toradocu, Jdoctor가 실제 JavaDoc의 테스트 요구사항을 얼마만큼 정확히 추출하는지 확인하기 위하여 각 기법을 실제 오픈소스 프로젝트 JFreeChart에 적용하여 다음 세 가지 연구 질문을 탐구하는 실험을 진행하였다:

- RQ1. Javadoc 주석으로 실제로 테스트 요구사항 명세가 포함된 구문 중 얼마에 대하여 각 기법이 테스트

<표1. 실험에 사용한 기법 적용 대상 프로그램>

Class Name	LOC	# method	# JavaDoc Stmt	# Req Stmt
CategoryLabelPositions	408	11	48 (25)	14 (14)
ChartColor	189	2	9 (4)	3 (3)
DefaultKeyedValues	474	22	81 (40)	25 (24)
GrayPaintScale	237	10	42 (17)	8 (6)
HistogramDataset	514	21	111 (56)	33 (30)
StackedXYAreaRenderer	696	16	101 (52)	12 (10)
Marker	696	37	139 (48)	23 (23)
PiePlot	3722	145	689 (231)	68 (52)
Range	478	23	99 (49)	18 (11)
TimeSeries	1433	58	295 (110)	38 (32)
XYBlockRenderer	448	15	74 (28)	8 (8)
Average	845	32.72	153.45	22.72

요구사항을 추출(구문 패턴이 탐지)하는 가?

- RQ2. 각 기법이 테스트 요구사항을 추출한 Javadoc 구문(구문 패턴이 탐지한 구문)은 실제 테스트 요구사항이 기술된 구문인가?
- RQ3. 각 기법이 Javadoc 문서에서 추출한 테스트 요구사항은 실제 요구사항을 올바르게 기술하는 가?

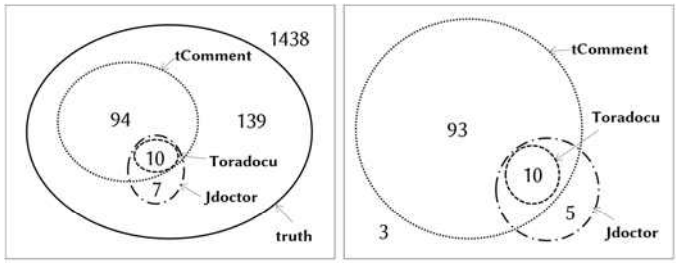
2) 테스트 요구사항 추출 도구. 본 연구에서는 Java 프로그램에 대해 동작하는 기법으로서 tComment, Toradocu, Jdoctor의 공개 버전 도구를 사용했다.

3) 기법 적용 대상 프로그램 및 정답 데이터 준비. 본 실험에서는 그래프, 표 등 데이터를 그래픽으로 표현하는데 사용하는 오픈소스 Java 프로젝트인 JFreeChart 버전 1.0.19 중 11개 클래스의 메소드 별 JavaDoc 문서(총 360개 메소드)를 선택하여 테스트 요구사항 추출 대상 프포그램으로 사용했다. 11개 클래스는 JFreeChart에 존재하는 총 629 클래스 중 JavaDoc 문서가 충실히 작성된 40개를 일차 선별한 후, 서로 특징이 유사한 클래스를 가능한 배제하고 다양한 특징을 갖도록 검토하여 정하였다.

적용 대상으로 정한 11개 클래스의 메소드 별 Javadoc 주석을 수작업으로 검토하여 (1) Javadoc 문서를 구문/문장별로 구분했고, 그 후 (2) 유닛 테스트의 테스트 오라클로 작성이 필요한 테스트 요구사항 구문을 판별하여 RQ1과 RQ2에 대한 실험에 필요한 정답 데이터(ground truth)를 구했다. RQ3의 경우, 정답 데이터를 미리 작성하지 않고, 대신 실험 중 어떤 기법이 특정 구문으로부터 추출한 테스트 요구사항 명세를 추출하였을 때, 수작업으로 검토를 통해 해당 명세가 해당 구문에 내포된 테스트 요구사항의 일부 혹은 전부를 표현하는지 여부를 판별하는 방식으로 진행하였다.

표1은 실험에 사용한 11개 클래스의 이름(Class Name), 코드 라인 수(LOC), 메소드 개수(# method), Javadoc 구문(문장) 수(# JavaDoc stmt)와 그리고 정답 작성 시 파악한 요구사항을 포함하는 Javadoc 구문 수(# Req Stmt)를 나타낸다(총 250개). 괄호 안의 숫자는 Javadoc 주석 중 태그(e.g., `@param`, `@return`, `@throws`)가 있는 주석에 해당하는 수를 나타낸다.

4) 실험 수행 및 측정. 각 기법에 대하여 총 11개 적용 대상 클래스를 각각 입력하여 결과를 얻은 후, 이를 정답과 비교하여 연구 질문에 대한 답을 얻었다[1]. RQ1에 대해서는, 정답에서 실제 테스트 요구사항을 포함하는 것으로 파악한 총



(a) 각 기법이 탐지한 요구사항 포함 구문 (b) 각 기법이 요구사항을 정확히 추출한 구문

250 구문 중 각 기법이 요구사항 명세를 1개 이상 추출한

<그림 1. 세 기법의 Javadoc 주석의 요구사항 탐지 및 추출 결과>

구문의 비율(포괄도; recall)을 측정했다. RQ2에 대해서는 각 기법이 추출한 요구사항 명세 중 정답 데이터로부터 실제 테스트 요구사항을 포함하는 것으로 파악된 구문의 비율(정밀도; precision)을 측정하였다. RQ3에 대해서는 각 기법이 특정 Javadoc 구문으로부터 추출한 테스트 요구사항 중 실제 해당 구문의 요구사항의 일부 혹은 전부를 표현하는 것으로 판별된 요구사항의 비율을 측정하였다.

3.2. 실험 결과

1) RQ1 결과. 그림 1(a)는 11개 적용 대상 클래스에서 파악된 1688개의 Javadoc 구문 중 정답으로 정의한 요구사항을 포함하는 총 250개 Javadoc 구문(truth 영역)과 각 기법이 요구사항을 추출한(탐지한) 구문의 관계를 나타낸다. 그림 1(a) 각 숫자는 해당 영역에 속한 구문의 개수이다.

그림 1(a)의 결과에서 확인할 수 있듯이, 총 250개 탐지 대상 중 총 139개 구문(55.6%)에 대해서는 세 기법 모두 요구사항을 탐지하지 못했다. 나머지 111개 구문에 대해서는 tComment, Toradocu, Jdoctor 중 하나 이상의 도구가 요구사항을 탐지하였으며, 이 때 tComment는 포괄도 37.6%(=94/250), Toradocu는 포괄도 4%(=10/250), Jdoctor는 포괄도 7%(=17/250)를 보였다.

2) RQ2 결과. 그림 1(a)에서 알 수 있듯, tComment, Toradocu, Jdoctor가 식별한 요구사항 구문은 모두 정답(truth) 영역 내에 있으므로, 세 기법 모두 정밀도 100%이며 잘못된 탐지는 한 건도 없었다. RQ1과 RQ2의 결과를 볼 때, 본 연구의 대상인 세 가지 요구사항 추출 기법은 한정적인 형태의 요구사항 패턴에 집중하여 높은 정밀도를 보임을 알 수 있다.

3) RQ3 결과. 그림 1(b)는 세 기법이 추출한 요구사항(총 111개) 중 실제 Javadoc 구문의 요구사항을 올바르게 표현한 수를 나타낸다. 그림 1(a)와 비교하여 볼 때, tComment가 추출한 104개 요구사항 중 103개가 정확함을 알 수 있다(추출된 요구사항의 정밀도 99%). Toradocu는 추출한 요구사항 10개가 모두 정확하였다(정밀도 100%). Jdoctor의 경우, 추출한 17개 중 15개가 정확하였으며 2개에는 오류가 있었다(정밀도 88%).

4. 논의

4.1. 연구대상 기법이 식별/추출하지 못한 요구사항

1) 모든 기법이 식별(탐지)에 실패한 요구사항. 그림 1(a)에서 알 수 있듯이 총 250개의 요구사항 포함 구문 중 139개는 tComment, Toradocu, Jdoctor 모두 요구사항 탐지는

[1] Jdoctor의 경우, [6]과 공개버전의 설정이 달라, [6]의 설정을 따랐다.

물론 추출에도 실패하였다. 이 139개 요구사항 포함 구문을 분석한 결과 다음과 같이 분류할 수 있었다:

A. 태그(예: `@param`, `@return`, `@throws`)가 아닌 비정형적 주석 구문에 작성된 요구사항 처리에 대한 한계. 표 1에서 나타난 바와 같이 태그로 시작하지 않는 37개 구문에도 테스트 요구사항이 표현되어 있었으나, 세 기법 모두 태그 구문에 한정되어, 이들을 탐지하지 못하였다. 태그 구문에 등장하는 요구사항 구문은 비교적 정형적인 특성을 갖고 있으나, 태그 외에 나타나는 요구사항은 비정형적이므로 이에 대한 포괄적 처리가 부족하다.

B. 특정한 값(상수)에 관한 요구사항에 대한 한계. 어떤 파라미터나 변수가 특정한 값으로 지정되어 있는 지에 대한 조건, 혹은 특정 변수에 저장되어 있던 값이 변경되었는지 확인하는 요구사항의 경우, 탐지되지 않았다.

C. 값 범위에 대한 다양한 표현으로 기술된 요구사항. 실제 개발자가 작성하는 주석에는 값 범위(range)를 기술하기 위해 특정 상수를 지칭하거나, 세 가지 도구들이 탐지하는 것 외에 다양한 표현(예: max, min)을 사용하여 표현하고 있다. 하지만 이들에 대해 세 기법 모두 요구사항을 탐지하지 못하였다.

D. 복합적 조건을 포함하는 요구사항. 값 범위와 부등식을 동시에 사용하는 복합적 조건의 경우, 세 기법을 통해 요구사항으로 탐지되지 않았다.

E. 절차적 요구사항. 실행 순서를 기술한 요구사항의 경우, 탐지되지 않았다. 이러한 절차적 요구사항은 세 가지 기법이 표현하는 요구사항의 종류(method annotation)으로 표현이 간단하지 않을 것으로 보인다.

F. 이미 등장한 요구사항을 인용하여 기술하는 요구사항. 하나의 Javadoc 구문에 한정되지 않고, 여러 구문에 걸쳐서 있는 요구사항을 인용하거나 대명사로 앞서 기술한 요구사항을 지칭하는 경우, 세 기법으로 탐지가 불가능하였다.

G. 그 외 구현 상 제약점. 총 15개 구문은 `private`메소드의 주석에 존재하였으나, tComment는 `public`, `protected` 메소드에 대해서, Toradocu, Jdoctor는 `public` 메소드에 대해서만 요구사항을 추출하여 이들을 탐지하지 못했다. 또한 타입 정보로 명시되어 있지 않은 런타임 Exception(예: `ClassCastException`)의 발생 조건, 처리 방식에 관련된 요구사항이 올바르게 탐지되지 않았다.

2) 식별하였으나 정확한 추출이 실패한 요구사항. RQ3결과에서 설명한 바와 같이, tComment는 1개 구문에 대해, Jdoctor는 2개 구문에 대해 잘못된 요구사항을 추출한 경우가 있었다. 이를 사례를 관찰한 결과, (1) tComment는 구문에 "not"이라는 단어가 다른 단어에 일부로(예: "annotation") 있는 경우를 잘못 인식해서 1개의 오류가 발생하였으며, (2) Jdoctor는 요구사항 구문에 있는 술어(예: "at least")를 정확히 번역하지 못해 2건의 오류가 발생했다. 즉, 3건 모두, tComment와 Jdoctor의 구문 패턴 종류가 제한적이어서 발생한 오류였다.

4.2. 연구대상 기법 간의 상호 비교

세 기법들이 공통적으로 탐지한 10개의 요구사항은 Javadoc의 `@throws` 주석에 `null`과 관련하여 주어와 서술어가 명시적으로 나타난 경우였다. 반면, `null`과 관련하여 tComment만 고유하게 추출한 경우가 93건 있었는데, (1) 주어 또는 서술어가 생략된 경우, (2) 괄호와 같이 복잡한 형태를 가진 구문의 경우가 대표적이다. Jdoctor만 고유하게 추출한 5개의 요구사항은 `null` 조건 검사 이외의 조건식 형태(예: 대소 비교)를 갖는 요구사항이었다.

4.3. 요구사항 추출 기법 발전을 위한 연구방향 제안

본 연구의 실험 결과로 비추어 볼 때, 기존의 요구사항 추출 기법들은 제한된 특정한 패턴에 대해서 요구사항 명세를 명확히 추출하는 방향으로 개발되었으며(예: `null` 관련), 반면 실제 Javadoc 주석에 내포된 요구사항의 50% 정도는 기존 기법이 집중하는 패턴과 맞지 않는 문제가 있음을 알 수 있었다. 이를 개선하기 위해서는 (1) 새로운 요구사항 추출 규칙 정의는 물론 (2) 자연어 처리 기술을 통해 포괄적인 구문 패턴 매칭이 가능하도록 개선하는 연구가 필요함을 확인할 수 있었다. 특별히, 실제 개발자가 사용하는 표현의 다양성을 고려할 때, Javadoc 주석에서 빈번히 발생하는 요구사항에 대한 구문의 패턴을 통계적으로 식별할 수 있도록 데이터 기반의 연구가 요청됨을 확인할 수 있었다.

4.4. 논문에서 다루지 않은 관련 기법

ALICS[9]은 품사(Part-Of-Speech) 태그[11]와 프로그래밍 언어의 특징이 반영된 사전(dictionary)을 정의하여(예: `null`은 명사) Javadoc 구문을 정의된 템플릿에 따라 1차 논리식(First Order Logic)으로 변환한다. ALICS[9]은 `String` 클래스, `Integer` 클래스, `null`값 검사, `@return`, `@throws` 태그에 대하여 미리 정의된 패턴에 따라 1차 논리식을 요구사항으로 추출한다. ALICS[9]은 도구가 공개되었으나, 프로그램 실행 문서 부재로 본 연구에서는 실험 대상으로 포함시키지 못하였다.

도구 미공개로 본 실험에서는 포함하지 못한 관련 연구에는 Zhou 등이 제안한 기법[10]이 있다. 이 기법은 Javadoc 주석 중 `@param`, `@throws` 또는 `@exception` 주석 구문에 작성된 `null`값 허용, `null`값 불허, 타입 검사, 범위 제한에 대한 4가지 종류의 요구사항을 패턴에 따라 추출하여 대상 프로그램의 동작을 테스트한다. 수작업으로 `java.awt`, `javax.swing` 두 패키지의 Javadoc 구문을 분석하여 작성한 패턴을 정의된 규칙에 따라 1차 논리식으로 변환하여 코드와 Javadoc 사이의 일치성을 SMT solver를 사용하여 검사한다.

5. 결론

본 논문은 Javadoc 주석으로부터 테스트 요구사항 명세를 추출하는 tComment, Toradocu, Jdoctor 기법을 오픈소스 프로젝트인 JFreeChart에 적용하여, 이들이 테스트 오라클 생성에 필요한 요구사항 탐지 및 추출 성능을 평가하였다. 평가 결과, 세 기법은 높은 정밀도를 보이나, 포괄도는 최대 41.6% 수준이며 여러 발전 과제가 있음을 관찰할 수 있었다.

참조문헌

[1] V. J. M. Manes et al., The Art, Science and Engineering of Fuzzing; A Survey, IEEE Transactions on Software Engineering, early access, Oct 2019
[2] C. Cadar et al., Symbolic Execution for Software Testing in Practice: Preliminary Assessment, ICSE 2011
[3] E. T. Barr, The Oracle Problem in Software Testing: A Survey, IEEE Transactions on Software Engineering, 41(5), May 2015
[4] S. H. Tan et al., @tComment: Testing Javadoc Comments to Detect Comment-Code Inconsistencies, ICST 2012
[5] A. Goffi et al., Automatic Generation of Oracles for Exceptional Behaviors, ISSTA 2016
[6] A. Blasi et al., Translating Code Comments to Procedure Specification, ISSTA 2018
[7] C. Pacheco et al., Randoop:Feedback-directed Random Testing for Java, OOPSLA 2007
[8] G. Fraser et al., EvoSuite: automatic test suite generation for object-oriented software, FSE 2011
[9] R. Pandita et al., Inferring Method Specifications from Natural Language API Descriptions, ICSE 2011
[10] Y. Zhou et al., Analyzing APIs Documentation and Code to Detect Directive Defects, ICSE 2017
[11] D. Klein et al., Accurate unlexicalized parsing, ACL 2003