테스트 범위 확장 기반 유효 패치 강화

문준현* 이은석**
*성균관대학교 컴퓨터공학과
**성균관대학교 소프트웨어공학과
e-mail : paranocs@gmail.com, leees@skku.edu

Test Coverage Expansion-based Valid Patch Refinement

Junhyeon Moon*, Eunseok Lee**
*Dept. of Computer Engineering, Sungkyunkwan University
**Dept. of Software Engineering, Sungkyunkwan University

요 약

개발자의 부담을 줄이고 소프트웨어의 품질을 향상시키기 위해 APR (Automated Program Repair) 기술을 개선해야 할 필요가 있다. 현재 사용되는 대부분의 APR 기술은 개발자가 수작업으로 작성한 테스트 케이스를 통과하는 유효 패치(Valid Patch)를 만드는 방식을 사용하기 때문에 QoP(Quality of Patch)는 많이 뒤떨어지는 단점을 가지고 있다. 본 논문에서는 테스트 범위가 충분히 높은 테스트 케이스를 자동으로 생성하는 기술을 이용하여 기존의 유효 패치를 강화하는 기술을 제안한다. 제안하는 내용을 Defects4J 의 일부 프로젝트에 적용하여 기존 기법 대비 FL 의 성능은 유지하면서 기존에 찾지 못했던 유효 패치를 추가적으로 찾을 수 있음을 확인하였다.

1. 서론

디버깅은 소프트웨어에 존재하는 결함의 위치를 식별하고 이를 수정하는 작업으로 소프트웨어 개발 비용의 약 50~80%를 차지할 정도로 매우 중요한 작업이다[1,2]. 디버깅 과정은 결함 위치 식별(Fault Localization, FL), 프로그램 수정(Program Repair, PR), 회귀 테스트(Regression Test, RT) 단계로 구성된다. FL과 RT는 여러 연구와 도구들의 개발로 높은 수준의 자동화가 이루어져 있지만 자동 프로그램 수정(Automated Program Repair, APR)은 수준이 매우 낮고 대부분 개발자의 수동 작업에 의존하고 있다. 개발자의 부담을 줄이고 소프트웨어의 품질을 향상시키기 위해 APR기술을 개선해야 할 필요성이 있다.

APR기술을 개선하기 위해 해결해야 하는 여러 문제점 중 테스트케이스 문제를 해결하고자 한다. 테스트케이스들은 개발자의 의도를 대표하는 것뿐이지 모든 실행 가능한 경로들을 커버하지 못한다. APR기술은 오직 주어진 테스트케이스를 모두 성공시키면 유효한 패치(valid patch)라고 판단하므로 타당성(plausible) 문제가 발생한다. 타당성 문제 뿐만 아니라 APR기술에 의해 만들어진 패치의 QoP(Quality of Patch)가 많이 뒤떨어지는 문제도 존재한다.

현재 FL은 개발자가 직접 작성한 테스트 케이스(이하 GivenTC)에 의존적이다. 대표적인 PR 기법인 GenProg[3]의 경우 FL의 결과에 따라 수정 위치(fix location)별로 가중치를 더 주게 된다. 그러므로 FL의 성능이 향상될수록 QoP가 향상될 가능성이 높아진다. 더 좋은 품질의 패치를 만들기 위해 FL의 성능을 향상시키기 위해서는 GivenTC의 테스트 범위가 충분히 넓도록 만들어야 한다. 하지만 이 작업은 전문가의 수작업으로 이루어지기 때문에 시간적, 인적 비용이 많이 소요된다.

본 논문에서는 QoP를 향상시키기 위해 테스트 범위가 충분히 높은 TC를 자동화를 통해 적은 비용으로 만들 수 있는 기법을 이용해 FL의 성능을 향상시키고,이를 통해 생성된 유효 패치를 FL기법의 피드백으로 활용하여 최종적인 유효 패치의의 QoP를 높이는 새로운 APR의 모델을 제안한다.

본 논문의 나머지 구성은 2장에서 관련연구, 3장에서 연구의 접근방법, 4장에서 실험 및 평가, 5장에서 결론을 맺는다.

2. 관련 연구

1) Fault Localization

APR 기술에 주로 사용되는 Fault Localization 기법 중 Spectrum-Based Fault Localization (SBFL)은 테스트 케이스들 중 실패한 케이스들이 프로그램의 어떤 영역을 지나갔는지에 대한 분석을 기반으로 프로그램에서 의심스러운 영역을 순위별로 나타낸다[4]. 연구되고 있는 SBFL기법은 Tarantula[5], Ochiai[6], Jaccard[7] 등이 있다. SBFL의 결과물인 소스코드의 스테이트먼트(statement)별 의심도 점수는 0(거의 의심되지 않음) 부터 1(가장 의심스러움)로 표현된다.

$$suspO(s) = \frac{failed(s)}{\sqrt{totalfailed \times (failed(s) + passed(s))}}$$

(그림 1) Ochiai 수식

위 수식은 Ochiai의 의심도를 구하는 식이다. 어느 statement *s*의 의심도를 전체 테스트 케이스 중 fail한 개수 *totalfailed*, fail한 테스트 케이스 중 해당 statement를 지나는 케이스의 개수 *failed(s)*, pass한 테스트 케이스 중 해당 statement를 지나는 케이스의 개수 *passed(s)*로 수식이 이루어져 있다.

2) Program Repair

프로그램 수정 기술은 크게 generate-and-validate 기술과 시멘틱기반(semantics-driven) 기술로 구분할 수 있다[4]. generate-and-validate 기술은 프로그램에서 수정되고 탐색 되어야 할 영역을 정의하고 잠재적인 솔루션들을 만들어낸다. 시멘틱기반(semantics-driven) 기술은 프로그램의 행동 분석과 문맥 정보를 통해 문제를 만들어내고, 만들어낸 문제를 정해진 형식으로 정의해서 수정한다. 이 두 기술 모두 테스트케이스의 결과에만 집중한다는 공통점이 있다.

3) Astor

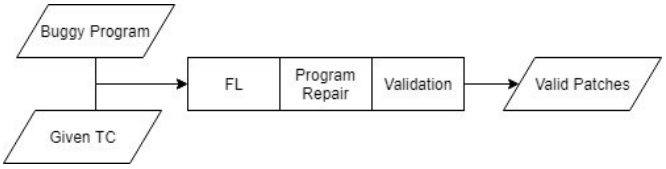
Astor는 Java로 만들어진 자동 프로그램 수정 프레임워크다[8]. 기본적으로 jGenProg[9], jKali[10], jMutRepair[10], DeepRepair[11], Cardumen[12] 같은 generate-and-validate 수정 기법들을 제공한다. 또한 12개의 extension points(EP)를 유저가 수정할 수 있어서 새로운 수정 기법을 시도해 볼 수 있으며 커스터마이징된 FL의 제어가 가능하다.

4) Automated Test Suite Generation

소프트웨어를 테스트하는데 테스트 오라클을 최적화하고, 소프트웨어 모델을 검증하고, 전문 인력의 테스트 비용을 줄이기 위한 목적으로 Automated Test Suite Generation연구가 진행되고 있다. 본 연구에서는 SBST 2017(Searched-Based Software Testing) Unit Testing Tool Competition에서 가장 높은 점수를 얻은 EvoSuite[13]을 사용했다. EvoSuite는 유전알고리즘을 이용해 Java 클래스들에 적용되는 테스트 케이스들을 자동으로 생성하는 탐색 기반 도구이다. EvoSuite는 code coverage를 최대화하면서 JUint형식의 테스트 케이스들을 생성해주기 때문에 자동으로 테스트 케이스를 확장할 수 있다. 하지만 자동으로 생성한 테스트 케이스의 경우 expected output이 존재하지 않으므로 진정한 의미의 테스트 케이스라 볼 수 없다.

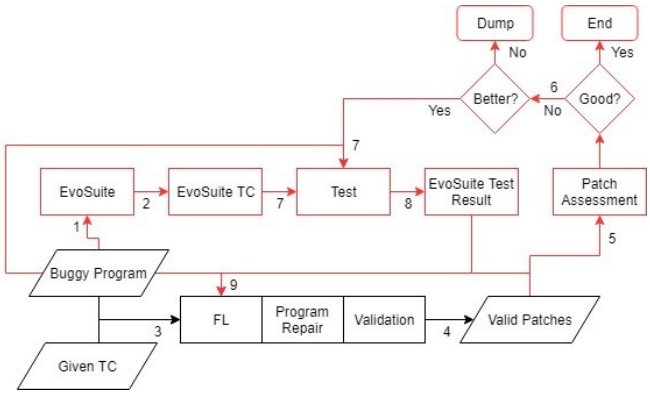
3. 접근 방법

버그 프로그램에 패치를 적용한 후, 사전에 생성한 EvoTC와 테스트를 하면 패치에 의해 변경된 코드를 테스트하는 케이스 중 fail하는 EvoTC가 존재한다. Fail EvoTC를 활용해서 FL을 하면, 상당히 높은 테스트 범위를 만족하는 EvoSuite의 특성상 전보다 좋은 FL 결과를 기대할 수 있다.
EvoSuite이 아무리 높은 테스트 범위를 가진다 해도 오히려 FL 결과가 안 좋아지는 경우가 존재한다. 그로 인해 오히려 QoP가 떨어지는 경우가 생길 수 있고 위 경우를 확인할 수 있는 과정이 필요하다.



(그림 2) 기존 APR 모델

기존 APR은 개발자가 직접 작성한 Given TC만을 가지고 Fault Localization, Program Repair, Validation(Regression Test)을 순차적으로 진행한 후 valid patches를 생성하는 과정을 거친다.



(그림 3) 제안하는 새로운 모델

프로그램을 APR에 적용시키기 전 EvoSuite을 이용해 자동으로 생성된 Test Suites인 EvoSuite TC를 만들어 놓는다(1, 2번). 그 후 기존과 동일한 방식으로 프로그램과 GivenTC를 이용하여 valid patches을 만들어낸다(3, 4번). 만들어낸 patch들을 평가해서 correct patch인지 검증한다(5, 6번). correct patch라면 프로세스가 종료된다. 아니라면 해당 패치를 프로그램에 적용한 후 미리 만들어 놓은 EvoSuite TC를 이용한 테스트를 진행한다(7번). 테스트에서 통과하지 못한 케이스가 존재하는 경우 그 케이스를 이용해 FL과 APR과정을 반복한다(9번). 2번째 반복부터 Patch Assessment때 지금 만들어진 패치가 전 과정에서 만든 패치보다 QoP가 높은지 검증하는 과정이 추가된다(6번)
새로운 모델을 통해 테스트 범위를 최대화하는 EvoSuite의 특성상 FL의 결과 개선을 기대할 수 있고 그로 인한 QoP의 상승을 기대할 수 있다. 또한 위 과정은 자동화해서 반복적으로 수행이 가능하다.

<표 1> 실험 결과

Target Program	# of Failed givenTC	Origin FL result	# of failed EvoTC	EvoTC FL result	Error rank diff
Math8	1	2	1	2	0
Math40	1	Not found	5	5	**Not found → 5**
Math49	1	Not found	7	64	**Not found → 64**
Math50	1	1	2	2	-1
Math53	1	2	15	1	**1**
Math70	1	1	2	1	0
Math73	1	1	4	1	0
Math78	1	31	6	14	**17**
Math80	1	Not found	1	Not found	Not found
Math84	2	Not found	1	4	**Not found → 4**
Math85	1	46	3	10	**36**
Math2	1	1	0	x	x
Math5	1	1	0	x	x
Math7	1	8	0	x	x
Math28	1	Not found	0	x	x
Math81	1	Not found	0	x	x
Math82	1	50	0	x	x

4. 실험 및 평가

1) 실험 환경

실험에 이용한 PC 환경은 ubuntu 16.04 (64bit) LTS, JDK 1.8.0_201이다. 사용한 프로그램은 Astor (2018.12.26.), gZoltar(FL), Ochiai, jGenprog(PR), EvoSuite 1.0.6이다. 실험의 Target Program(buggy program)으로는 defects4j (Java8) Math program중 human patch가 존재하고, jGenprog에 의해 패치가 생성되는 프로그램을(17개) 이용했다.

2) 실험 과정

jGenProg 을 이용해 만든 유효 패치에 EvoSuite 이 자동으로 생성한 Test cases 들을 적용했을 때 FL 의 결과가 좋아지는지 검증하는 실험을 진행했다. 먼저 target program 의 EvoSuite TC 를 만들어 낸다. 그리고 Given TC 만을 가지고 APR 을 거쳐서 유효 패치들을 생성한다. 생성한 유효 패치를 target program 에 적용한 후 앞에서 만들어 놓은 EvoSuite TC 를 가지고 테스트와 FL 을 진행해서 GivenTC 만을 가지고 한 FL 과 결과를 비교했다.

3) 실험 결과

FL결과 분석을 위해서 human patch를 이용했다. FL의 결과로 나온 의심도 순위에서 human patch에 해당하는 영역이 몇 순위에 위치하는지를 FL의 성능평가 기준으로 삼았다. 실험 결과에서 not found는 gZoltar에서 의심도가 0.01보다 작은 결과를 리턴하지 않아서 존재한다. Math2부터 Math82까지의 경우 patch를 적용한 후에도 EvoSuite TC가 모두 pass했기 때문에 새로 FL결과를 얻어 낼 수 없었다. 때문에 기존 17개의 target program 중 패치 적용 후 fail한 EvoTC가 존재하는 11개의 프로그램에 대해 실험 결과를 낼 수 있었다.

11개의 프로그램 중 6개의 FL성능이 상승 했고, 1개의 성능이 하락했으며 5개의 성능에 변화가 없었다. FL성능의 변동이 없거나 하락한 프로그램의 경우는 모두 기존의 FL결과가 매우 좋았다는 공통점이 존재한다. 기존의 FL이 1순위 혹은 2순위로 버그 라인을 찾았을 때 EvoTC를 적용한 FL도 비슷한 수준을 유지한다는 점을 알 수 있다. 또한 기존 FL에서 not found였을 때 EvoTC를 적용하고 나서도 not found로 유지된 결과가 하나 존재한다. 주목할 만한 점은 기존 FL결과가 좋지 못했을 때 많은 경우에서 EvoTC를 적용했을 때 FL의 성능이 크게 향상된 점이다.

5. 결론

본 논문에서는 APR 기술에서 개선해야 할 점으로 개발자에 의해 작성된 테스트 케이스 만으로는 모든 경로를 커버하지 못해서 생기는 테스트 케이스 문제와 APR 기술에 의해 만들어진 패치의 QoP 가 떨어지는 문제를 지적하고 이를 개선하기 위해 Automated Test Suite Generation 과 Patch Assessment 를 이용한 새로운 APR 모델을 제안했다. 실험으로는 Automated Test Suite Generation 기술을 활용했을 때 얼마나 FL 의 성능이 향상되는지 검증했다. 실험 결과로 대부분의 경우 성능이 향상되거나 일정했지만 오히려 낮아지는 경우도 존재한다는 것을 알 수 있었고, 이로 인한 문제는 매번 QoP 를 평가할 수 있다면 해결할 수 있으므로 Patch Assessment 에 대한 연구가 필요하다. 따라서 향후에는 Patch Assessment 에 대한 연구를 진행하고자 한다.

Acknowledgments

이 논문은 2019 년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (2019R1A2C2006411. 교신저자는 이은석임.)

참고문헌

[1] D. Landsberg , H. Chockler , D. Kroening , M. Lewis , (2015) “Evaluation of measures for sta- tistical fault localisation and an optimising scheme”, in: Proceedings of International Conference on Fundamental Approaches to Software Engineering.

[2] X. Xie , T.Y. Chen , F.-C. Kuo , B. Xu , (2013) “A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization”, Transactions on Software Engineering and Methodology (TOSEM). 22(4).

[3] Le Goues, C., Nguyen, T., Forrest, S., & Weimer, W. (2011). “Genprog: A generic method for automatic software repair”, IEEE Transactions on Software Engineering, 38(1).

[4] Gazzola, L., Micucci, D., & Mariani, L. (2017). “Automatic software repair: A survey”, IEEE Transactions on Software Engineering, 45(1).

[5] J.A. Jones , M.J. Harrold , Empirical evaluation of the tarantula automatic fault-localization technique, in: Proceedings of the International Conference on Automated software engineering, 2005 .

[6] A. Rui , P. Zoeteweij , A.J.C. van Gemund , On the accuracy of spectrum-based fault localization, in: Proceedings of the Academic and Industrial Conference Practice and Research Techniques-MUTATION, 2007 .

[7] R. Abreu , P. Zoeteweij , ArjanJ.C. van Gemund , An Evaluation of Similarity Coefficients for Software Fault Localization, in: Proceedings of the International Symposium on Software Testing and Analysis, 2006 .

[8] Matias Martinez, Martin Monperrus (2016), “ASTOR: A Program Repair Library for Java”, in Proceedings of ISSTA, Demonstration Track.

[9] Matias Martinez, Thomas Durieux, Romain Sommerard, Jifeng Xuan, and Martin Monperrus. Automatic repair of real bugs in java: A large-scale experiment on the defects4j dataset. Empirical Software Engineering, pages 1–29, 2016.

[10] Matias Martinez and Martin Monperrus. Astor: A program repair library for java (demo). In Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, pages 441–444, New York, NY, USA, 2016. ACM.

[11] Martin White, Michele Tufano, Matias Martinez, Martin Monperrus, and Denys Poshyvanyk. Sorting and transforming program repair ingredients via deep learning code similarities, 2017.

[12] Matias Martinez and Martin Monperrus. Open-ended exploration of the program repair search space with mined templates: the next 8935 patches for defects4j, 2017.

[13] G. Fraser, (2018) “A Tutorial on Using and Extending the EvoSuite Search-Based Test Generator”, in Proceedings of Search-Based Software Engineering.