KIPS Tr. Software and Data Eng.
Vol.3, No.3 pp.109~118 pISSN: 2287-5905

http://dx.doi.org/10.3745/KTSDE.2014.3.3.109

Analysis of Energy Efficiency for Code Refactoring Techniques

Jae-jin Park† · Doohwan Kim†† · Jang Eui Hong†††

ABSTRACT

Code refactoring focuses on enhancing the maintainability of software to extend its lifetime. However as software applications were varied and the range of its usage becomes broaden, there are some efforts to improve software qualities like performance or reliability as well as maintainability using code refactoring techniques. Recently, as low-energy software has become one of critical issues in mobile environment, developing energy efficient software through code refactoring becomes an important one. Therefore this paper has its goal to investigate whether the existing refactoring techniques can support energy efficient software generation or not. That is to say, the existing code refactoring techniques can cause the minus of energy efficiency because they did not considered the energy consumption in their refactoring process. This paper experiments and analyzes to check whether the M. Fowler's code refactoring techniques can support the energy efficient software generation or not. Our research result can give to software developer some informations about energy-efficient refactoring techniques, and can support the development of software that has high maintainability and good energy efficiency.

Keywords : Code Refactoring, Software Quality, Energy Consumption

코드 리팩토링 기법의 전력 효율성 분석

박 재 진† · 김 두 환†† · 홍 장 의†††

요 약

코드 리팩토링은 소프트웨어의 수명을 연장하기 위한 목적을 가지고, 유지보수성을 증진하는데 초점이 있다. 그러나 최근 소프트웨어의 유용성이 높아지고 활용 범위가 방대해지면서, 성능 및 신뢰성 등의 다양한 품질 속성을 코드 리팩토링을 통해 향상시키고자 하는 노력이 있었다. 최근 스마트 폰과 같은 모바일 기기에서 저전력 소프트웨어의 중요성이 강조됨에 따라, 전력 효율성을 보장하는 코드 리팩토링 기법들도 필요하게 되었다. 본 연구에서는 코드 리팩토링이 소모 전력의 절감 효과를 가져 올 수 있는지를 확인하고자 하였다. 즉 기존에 제시되었던 코드 리팩토링 기법들이 소모 전력에 대한 충분한 고려가 이루어지지 못했기 때문에 코드의 유지보수성은 향상시키지만, 전력 효율성이 감소하는 결과를 초래할 수 있다는 것이다. 따라서 본 연구에서는 M. Fowler가 개발한 코드 리팩토링 기법들을 대상으로 전력 효율성을 분석한다. 제시된 연구 결과를 통해 개발자들은 어떠한 리팩토링 기법이 전력 효율성을 제공하는지 판단할 수 있으며, 이를 통해 유지보수성이 높은 전력 효율적인 소프트웨어를 개발할 수 있을 것이다.

키워드 : 코드 리팩토링, 소프트웨어 품질, 소모 전력

1. 서 론

최근 모바일 기기와 같은 임베디드 시스템들은 일반적인 시스템에 비해 한정적인 배터리를 사용하는 특징이 존재한다[1]. 따라서 보다 긴 시간동안 서비스를 받기 위해서는 전력을 효율적으로 사용해야 한다. 이러한 전력 소모 절감을 위하여 저전력 하드웨어 개발과 더불어 소프트웨어 전력 분석에 대한 중요성이 강조되고 있다[2, 3]. 또한 스마트폰과 태블릿 PC의 보급으로 전력 효율성이 더욱 중요한 이슈가 되고 있다[4-6].

코드 리팩토링은 보통 소프트웨어의 유지보수성을 향상시키기 위해 사용하는 기법이며, 최근 성능과 안전성과 같은 다양한 품질 속성을 향상시키기 위해 활용되는 특징을 갖고 있다[7, 8]. 최근 이슈가 되고 있는 전력 효율성은 코드 리팩토링에서도 고려되어야 한다. 예를 들어 개발자가 저전력

※ 이 논문은 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보컴퓨팅기술개발사업(2012-0006426)과 기초연구사업(2011-0010396)의 지원을 받아 수행된 것임.
† 정 회 원 : 충북대학교 컴퓨터과학과 석사과정
†† 준 회 원 : 충북대학교 컴퓨터과학과 박사과정
††† 종신회원 : 충북대학교 소프트웨어학과 교수
논문접수 : 2013년 12월 12일
수 정 일 : 1차 2014년 1월 21일
심사완료 : 2014년 1월 21일
* Corresponding Author : Jang Eui Hong(jehong@chungbuk.ac.kr)

소프트웨어의 코드를 대상으로 유지보수성을 향상시키기 위해 코드 리팩토링 기법들을 적용하고자 할 때 상황에 따라 전력 효율성이 오히려 감소되는 결과를 초래할 수 있다. 즉, 개발자는 저전력 소프트웨어를 리팩토링 하고자 할 때, 기존에 널리 알려진 기법들을 직관적으로 적용하기가 쉽지 않은 실정이다. 이러한 문제를 해결하기 위해서는 코드 리팩토링이 갖는 본래의 목적인 유지보수성뿐만 아니라 전력 효율성에 대한 고려도 필요하게 된다[9, 10].

따라서 본 논문에서는 M. Folwer[11]가 제시한 코드 리팩토링 기법들에 대해 전력 효율성을 살펴보고자 한다. M. Folwer가 제안하는 68개의 리팩토링 기법은 가장 일반적인 리팩토링 개념을 제안하는 것이며, 또한 적용 가능한 대부분의 기법을 포함하고 있기 때문에 이들에 대한 전력 효율성을 살펴봄으로써, 어떠한 리팩토링 기법이 전력 효율성을 제공하는지 판단할 수 있을 것이다.

이를 위해 본 연구에서는 M. Folwer의 기법들의 적용에 따른 소모 전력량의 변화를 분석하였다. 분석된 결과를 통해 기존에 완성된 소프트웨어의 전력 효율성에 유용한 기법들을 분류하고, 개발자가 소프트웨어의 유지 보수 활동에서 전력 효율성까지 고려할 수 있도록 전력 효율성이 고려된 코드 리팩토링 카테고리를 정의하였다.

본 논문의 2장에서는 관련 연구를 소개하고 3장에서는 실험 설계와 결과에 대해 다룬다. 4장에서는 실험을 통해 얻은 데이터를 바탕으로 특이점에 대해 분석한다. 5장에서는 예제 시스템을 통해 전력 효율성에 대한 효과를 확인하고 6장에서는 결론 및 향후 연구를 기술한다.

2. 관련 연구

코드 리팩토링과 소프트웨어 전력 효율성 사이의 관계를 분석하는 연구들이 진행되고 있다. 대표적으로 Gottschalk[12], Marion[13], Silva [14]의 연구가 있다.

Gottschalk[12]는 다양한 종류의 Energy Code Smell을 소개하였다. Loop Bug와 Dead Code등의 일반적인 Code Smell들을 재조명하여 전력 낭비에 어떠한 영향을 미치고 있는지 설명하고, Restructuring을 통해 전력 낭비의 최소화를 유도하였다. 그러나 이 연구는 Energy Code Smell을 해결할 수 있는 명확한 가이드라인을 제공하고 있지 않기 때문에 해당 분야의 개발 경험이 풍부하지 않은 일반적인 개발자가 활용하는데 어려움이 존재한다.

Marion[13]은 안드로이드 앱을 대상으로 Gottschalk의 연구와 동일한 소모전력 문제를 고려하였다. 전력 낭비를 유발하는 상황에 대해 묘사하고 해당 문제 상황을 해결할 수 있는 코드 리팩토링을 제안했다. 이를 통해 코드 리팩토링이 전력 효율성을 향상시키기에 적합하다는 것을 제시하였다.

기존의 리팩토링 기법들 중 Inline-Method 기법만을 대상으로 전력 효율성에 미치는 영향에 대해 분석한 연구가 존재한다[14]. 해당 기법은 함수 몸체가 함수의 이름만큼이나 명확하여 불필요한 호출을 없애기 위해 사용되는 기법이다. Silva[14]의 연구는 Inline-Method 기법을 반복적으로 적용하면서 성능과 소모 전력량의 변화를 확인하였다. Inline-Method 기법의 적용 횟수가 높아질수록 성능과 전력 효율성이 향상되는 것을 알 수 있음을 보였다. 하지만 다양한 기법들 중 하나의 기법에 대해서만 전력 효율성이 분석되어 개발자의 입장에서 활용도는 매우 낮다고 볼 수 있다.

3. 기법별 전력 소모량 예측

본 논문에서 사용하는 “전력 효율성”은 동일한 서비스를 제공받기 위해 더 적은 전력을 사용하는 정도로 정의하였다[15, 16]. 코드 리팩토링 기법들이 대상 소프트웨어에 적용되기 전과 적용 후, 모두 동일한 기능을 제공한다는 점에 착안하여 소모되는 전력량의 차이만을 고려하고 각각의 기법에 대해 전력 효율성을 평가하였다.

3.1 실험 대상

실험에 사용되는 코드 리팩토링 기법들은 모두 C++ 언어로 작성된 소스코드를 대상으로 실험을 실시했다. 실제 실험에 사용된 기법은 M. Fowler[11]가 제시한 총 68개의 기법들 중 63개이다. 실험에서 제외된 5개의 기법은 아래와 같다.

- Change Unidirectional Association to Bidirectional
- Change Bidirectional Association to Unidirectional
- Replace Type Code with Subclasses
- Replace Type Code with State/Strategy
- Encapsulate Downcast

위의 5가지 기법은 C++ 언어의 특성상 각 기법을 구현하기가 어려우며, 또한 리팩토링 과정에서 사용빈도가 낮아 실험에서 제외하였다.

3.2 실험 환경

본 연구에서는 코드 리팩토링 기법의 적용에 따른 소모 전력량 차이를 확인하기 위해 XEEMU[17]를 사용했다. XEEMU는 소모 전력을 측정하기 위해 개발된 코드기반 전력분석 도구로써, 실측 대비 평균 1.6%의 오차율을 보이는 비교적 정확성이 높은 소모 전력을 분석 도구이다. Table 1은 XEEMU의 타겟 플랫폼 환경을 정리한 것이다.

Table 1. Target Platform Spec. of XEEMU

CPU	Intel XScale 80200 processor, 266 to 733 MHz in 66MHz
Architecture	ARM pipelined RISC
RAM	32 MByte Micron SDRAM, 100MHz
Cross Compiler	arm-elf-g++ 4.1.1
Measurement tap	CPU core current, IO current, System peripherals

3.3 실험 설계

실험을 위해 코드 리팩토링의 기법들을 적용할 수 있는 63개의 예제 코드를 준비하였다. 예제 코드들은 M. Fowler가 제시한 예제[11]를 기반으로 작성하였으며, 특히 각 예제 소스 별로 소모 전력의 변화를 없애기 위하여 외부 입력을 받지 않고 동작할 수 있도록 개발하였다. 최초 예제 소스 코드는 Original 코드라고 지칭한다.

준비된 Original 코드에 M. Folwer가 제시한 리팩토링 절차를 적용하여 개선된 예제 소스 코드들을 얻었다. 이들 예제 코드들은 Refactor 코드라 지칭하였으며, 컴파일 및 실행을 위하여 선언문과 같은 기본적 문장을 추가 하여 생성되었다.

각각의 기법별로 M. Fowler의 리팩토링 기법 적용에 따른 소모 전력량을 측정하기 위해 Fig. 1과 같은 방법으로 실험을 수행한다.

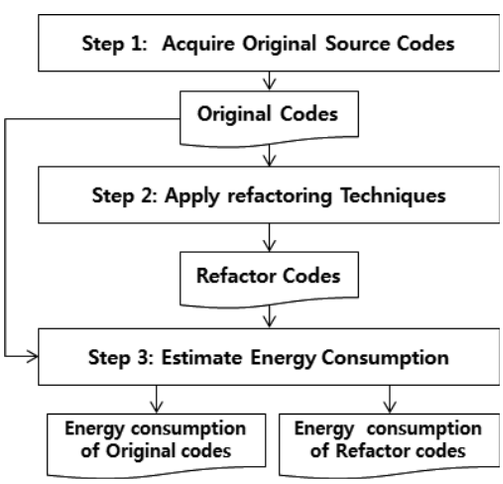


Fig. 1. Estimation Process of Energy Consumption

3.4 실험 결과

M. Folwer의 코드 리팩토링 기법을 적용하여 생성된 Refactor 코드의 LOC의 차이와 측정된 소모 전력량을 Table 2에서 Table 7에 정리하였다. 각각의 표는 M. Folwer가 제안한 카탈로그에 맞추어 구분하였다. 표에 나타난 “효과(Effect)” 항목은 전력 효율성을 의미하는 것으로써, 코드 리팩토링을 적용하였을 때 소모 전력이 감소한다면 전력 효율성이 향상(⬆)되는 것을 의미하며, 소모 전력이 증가한다면 전력 효율성이 감소(⇩)함을 의미한다. 해당 항목의 “−” 기호는 소모 전력량의 변화가 없음을 의미하며, 이는 XEEMU 도구의 출력 결과에 기반한다.

총 63개의 기법 중 26개의 기법이 전력 효율성의 향상을 보였으며, 7개의 기법이 소모 전력의 변화가 없는 것으로 분석되었다. 나머지 30개의 기법은 소모전력 측면에서 효율성이 떨어지는 것으로 나타났다. Fig. 2는 Original 코드와 Refactor 코드의 소모전력 분포를 나타난 것이다.

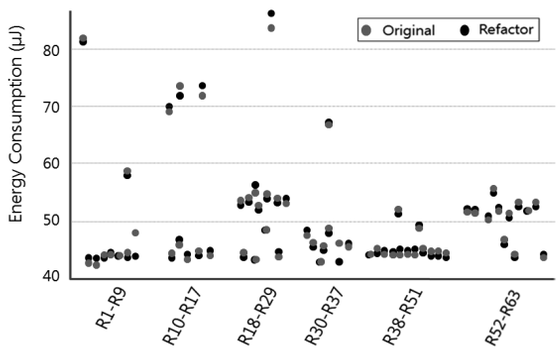


Fig. 2. Energy Consumptions for M. Folwer’s Refactoring Techniques

Fig. 2와 같이 코드 리팩토링 기법들의 소모전력 분포는 데이터 객체화 그룹인 R18-R29이 가장 많은 전력을 소모하는 것으로 나타난다. 각 기법별로 Original 코드와 Refactor 코드의 소모전력의 차이는 Fig. 3에 나타내었다.

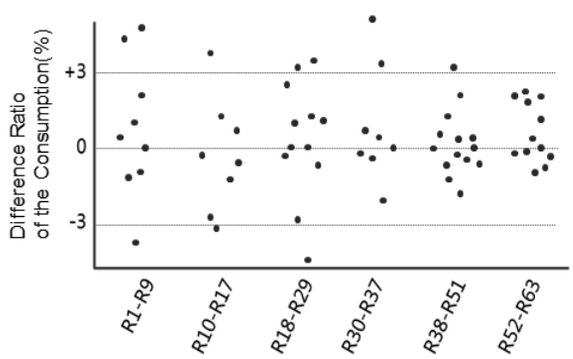


Fig. 3. Distribution of Difference Ratios for Energy Consumption by refactoring technique

Fig. 3에서 전체적으로 객체간의 기능 이동 그룹인 R10-R17이 상대적으로 전력 효율성이 좋아지는 그룹에 해당되는 것으로 판단되었다.

4. 실험 결과 분석

기본적으로 코드 리팩토링은 유지보수성을 향상시키기 위한 목적으로 사용되며, 절반 정도의 기법들이 모듈을 추출하거나 수정이 용이한 모듈로 대체하고 있기 때문에 전력 효율성에 부정적인 영향을 미치고 있다.

이처럼 직관적으로 코드 리팩토링 기법이 소모 전력에 미치는 영향을 알 수 있는 기법도 존재하지만 그렇지 않은 특별한 기법들 또한 존재한다. 해당 절에서는 이러한 특별한 경우에 대하여 살펴보고자 한다.

4.1 일반적인 경우

실험을 통해 다룬 코드 리팩토링 기법들은 모두 다음에 제시하는 기법 중에서 한개 이상을 적용하고 있다.

- 모듈(변수 포함) 추출 및 병합
- 모듈(변수 포함) 이동
- 모듈(변수 포함) 대체
- 모듈(변수 포함) 추가 및 제거

첫 번째 항목에서 모듈 추출의 경우 새로운 모듈(변수 포함)이 소스 코드에 추가되기 때문에 기존에 없던 함수 호출 및 참조가 발생하여 전력 효율성은 감소한다는 것을 알 수 있고, 반대로 병합은 전력 효율성이 증가 한다는 것을 알 수 있다. 두 번째 항목은 모듈이 적재적소한 곳으로 옮겨지는 것을 나타내는데 이는 불필요한 관계들이 사라지게 되므로 전력 효율성이 증가한다는 것을 알 수 있다.

세 번째 항목은 특정 모듈이나 변수를 보다 유지보수하기 쉽고 이해하기 쉬운 것으로 변경하는 것을 의미한다. 따라서 전력 효율성은 변경 형태에 따라 증가하거나 감소한다. 마지막 항목은 캡슐화를 위해 특정 요소들을 추가 하거나 불필요한 요소들을 제거하는 것이기 때문에 상황에 따라 전력 효율성이 감소하거나 향상된다.

하지만 모듈(변수 포함)의 이동에 따라 모든 기법이 전력 효율성을 향상 시키지는 않는다. R54: Pull Up Constructor Body 기법은 전력 효율성이 감소하는 반면, R53: Pull Up Method 기법은 전력 효율성이 향상되었다. 또한 R9: Substitute Algorithm 기법은 리팩토링에 의해 전력 효율성을 저하시키는 것으로 실험결과가 나타났지만, 실제로 항상 전력 효율성이 저하되는 것은 아니다. 이와 같은 특별한 경우의 분석은 4.2절과 4.3절에서 설명한다.

Table 2. Energy Consumptions for "Composing Method" Techniques

No.	Code Refactoring Techniques	LOC Diff.	Consumption (μJ)		Effect
			Original	Refactor	
R1	Extract Method	+3	83.077	83.622	⇩
R2	Inline Method	-3	44.059	43.980	⬆
R3	Inline Temp	-1	43.599	43.394	⬆
R4	Replace Temp with Query	+2	44.589	44.872	⇩
R5	Introduce Explaining Variable	+4	45.846	45.265	⬆
R6	Split Temporary Variable	0	44.729	44.729	-
R7	Remove Assignments to Parameters	0	43.767	44.216	⇩
R8	Replace Method with Method Object	+10	44.681	48.363	⇩
R9	Substitute Algorithm	-5	58.440	59.893	⇩

Table 3. Energy Consumptions for "Moving Features Between Objects" Techniques

No.	Code Refactoring Techniques	LOC Diff.	Consumption (μJ)		Effect
			Original	Refactor	
R10	Move Method	0	70.066	69.786	⬆
R11	Move Field	0	47.029	45.663	⬆
R12	Extract Class	+23	72.361	74.780	⇩
R13	Inline Class	-23	74.780	72.361	⬆
R14	Hide Delegate	+3	43.125	43.659	⇩
R15	Remove Middle Man	-3	43.659	43.125	⬆
R16	Introduce Foreign Method	+4	44.131	43.916	⬆
R17	Introduce Local Extension	+5	43.916	44.137	⇩

Table 4. Energy Consumptions for "Organizing Data" Techniques

No.	Code Refactoring Techniques	LOC Diff.	Consumption (μJ)		Effect
			Original	Refactor	
R18	Self Encapsulate Field	+6	43.565	44.032	⇩
R19	Replace Data Value with Object	+14	52.241	53.685	⇩
R20	Change Value to Reference	+3	53.685	55.372	⇩
R21	Change Reference to Value	-3	55.372	53.685	⇧
R22	Replace Array with Object	+24	53.884	55.622	⇩
R23	Duplicate Observed Data	+31	52.364	53.063	⇩
R24	Replace Magic Number with Symbolic Constant	+3	43.893	43.893	-
R25	Encapsulate Field	+7	53.970	53.128	⇧
R26	Encapsulate Collection	+8	90.200	84.104	⇧
R27	Replace Record with Data Class	+1	48.759	48.759	-
R28	Replace Type Code with Class	+29	51.533	52.188	⇩
R29	Replace Subclass with Fields	-1	44.590	44.475	⇧

Table 5. Energy Consumptions for "Simplifying Conditional Expressions" Techniques

No.	Code Refactoring Techniques	LOC Diff.	Consumption (μJ)		Effect
			Original	Refactor	
R30	Decompose Conditional	+11	45.981	46.387	⇩
R31	Consolidate Conditional Expression	+1	44.138	44.368	⇩
R32	Consolidate Duplicate Conditional Fragments	-1	45.600	45.278	⇧
R33	Remove Control Flag	-5	47.853	46.597	⇧
R34	Replace Nested Conditional with Guard Clauses	-7	43.730	43.730	-
R35	Replace Conditional with Polymorphism	+24	56.738	58.530	⇩
R36	Introduce Null Object	+4	65.987	65.758	⇧
R37	Introduce Assertion	+2	43.730	47.009	⇩

Table 6. Energy Consumptions for "Making Method Calls Simpler" Techniques

No.	Code Refactoring Techniques	LOC Diff.	Consumption (μJ)		Effect
			Original	Refactor	
R38	Rename Method	0	43.730	43.730	-
R39	Add Parameter	0	44.222	44.748	⇩
R40	Remove Parameter	0	44.748	44.222	⇧
R41	Separate Query from Modifier	+3	44.552	44.351	⇧
R42	Parameterize Method	-3	45.151	44.278	⇧
R43	Replace Parameter with Explicit Methods	+3	50.075	51.579	⇩
R44	Preserve Whole Object	0	43.571	44.445	⇩
R45	Replace Parameter with Method	-1	44.747	44.148	⇧
R46	Introduce Parameter Object	+9	45.029	44.771	⇧
R47	Remove Setting Method	-3	49.985	49.327	⇧
R48	Hide Method	+1	43.902	43.902	-
R49	Replace Constructor with Factory Method	+9	43.422	43.552	⇩
R50	Replace Error Code with Exception	+11	44.379	44.489	⇩
R51	Replace Exception with Test	-3	43.104	43.389	⇩

Table 7. Energy Consumptions for "Dealing with Generalization" Techniques

No.	Code Refactoring Techniques	LOC Diff.	Consumption (μJ)		Effect
			Original	Refactor	
R52	Pull Up Field	-1	51.781	51.579	⬆
R53	Pull Up Method	-3	51.579	51.456	⬆
R54	Pull Up Constructor Body	+2	49.627	50.368	⇩
R55	Push Down Method	0	51.827	51.827	-
R56	Push Down Field	0	50.164	49.673	⬆
R57	Extract Subclass	+7	51.574	52.204	⇩
R58	Extract Superclass	+5	54.885	55.967	⇩
R59	Extract Interface	+6	51.410	52.212	⇩
R60	Collapse Hierarchy	-6	52.400	51.967	⬆
R61	Form Template Method	+10	45.843	46.585	⇩
R62	Replace Inheritance with Delegation	+5	43.522	43.754	⇩
R63	Replace Delegation with Inheritance	-5	43.754	43.522	⬆

4.2 Substitute Algorithm 기법

Substitute Algorithm 기법은 기존의 알고리즘을 보다 명확한 것으로 바꾸고 싶을 때 사용한다. 하지만 동일한 기능을 제공하는 알고리즘별로 소모되는 전력량이 각기 다르기 때문에[18] Substitute Algorithm 기법의 적용에 따른 에너지 효율성은 저하 또는 향상의 형태를 보일 수 있다. 예를 들면, 정렬(Sorting)을 위한 알고리즘을 버블정렬에서 퀵정렬로 변경하는 경우 전력 효율성은 향상되지만, 버블정렬을 선택정렬로 변경하는 경우 전력 효율성은 감소된다[19].

4.3 Pull Up Method vs. Pull Up Constructor Body

R53: Pull Up Method 기법과 R54: Pull Up Constructor Body 기법은 모두 서브 클래스에 공통적으로 존재하는 함수를 슈퍼 클래스로 옮기는 기법으로 상속 관계를 가지고 있는 소스 코드에서 사용될 수 있다[20]. 두 기법의 근본적인 원리는 동일하지만 전력 효율성에 대한 양상은 상반된다. R53 기법을 적용하게 되면 전력 효율성이 향상되고 R54 기법을 적용하면 전력 효율성이 감소한다.

R54 기법 적용 후에 어셈블리어를 살펴보면 동일한 기능을 수행하는 생성자의 내용을 반복적으로 선언하고 사용하는 반면 R53 기법 적용 후에는 슈퍼클래스에 선언된 내용을 참조만 할 뿐 반복적으로 선언하지 않는 차이가 존재한다. 결국 R53 기법을 적용함으로써 로우(low) 레벨 명령어가 감소하여 전력 효율성이 향상됨을 알 수 있다.

4.4 기법의 전력 효율성

코드 리팩토링 소모 전력 측정 실험 결과와 분석을 통해 3개의 카테고리로 나누어 코드 리팩토링 기법의 전력 효율성을 평가 할 수 있다. 3개의 카테고리는 다음과 같다.

- 전력 효율성이 향상되는 기법
- 전력 효율성이 동일한 기법
- 전력 효율성이 감소하는 기법

전력 효율성이 향상되는 기법들은 Table 8과 같이 정리할 수 있고, 전력 효율성이 동일한 기법은 Table 9와 같이 정리되었다. Table 8과 Table 9에 나타난 기법은 실험한 63개 또는 M. Folwer 가 제안한 총 68개의 리팩토링 기법에 대하여 33개의 기법에 해당된다. 이는 전체의 약 50%에 해당되는 것으로써, 레거시 코드의 리팩토링에 있어서 소모 전력을 고려하는 적절한 기법의 선택이 요구되고 있다.

Table 8. Techniques that improve Energy Efficiency

No.	Code Refactoring Techniques
R2	Inline Method
R3	Inline Temp
R5	Introduce Explaining Variable
R10	Move Method
R11	Move Field
R13	Inline Class
R15	Remove Middle Man
R16	Introduce Foreign Method
R21	Change Reference to Value
R25	Encapsulate Field
R26	Encapsulate Collection
R29	Replace Subclass with Fields
R32	Consolidate Duplicate Conditional Fragments
R33	Remove Control Flag
R36	Introduce Null Object
R40	Remove Parameter
R41	Separate Query from Modifier

R42	Parameterize Method
R45	Replace Parameter with Method
R46	Introduce Parameter Object
R47	Remove Setting Method
R52	Pull Up Field
R53	Pull Up Method
R56	Push Down Field
R60	Collapse Hierarchy
R63	Replace Delegation with Inheritance

Table 9. Techniques that has unchanged Energy Efficiency

No.	Code Refactoring Techniques
R6	Split Temporary Variable
R24	Replace Magic Number with Symbolic Constant
R27	Replace Record with Data Class
R34	Replace Nested conditional with Guard Clauses
R38	Rename Method
R48	Hide Method
R55	Push Down Method

5. 적용 및 토론

5.1 예제 시스템 적용

소모 전력에 효율적인 코드 리팩토링 기법들을 대상으로 예제 시스템에 적용하여 전력 효율성의 변화를 확인하였다. 예제 시스템은 스마트폰 카메라에서 화상 압축 시 사용되는 허프만 알고리즘[21]과 모바일 기기의 보안성을 위해 널리 이용되는 데이터 암호화 알고리즘[22]으로 선정했다. 예제 시스템의 간략한 기술은 Table 10과 같다.

Table 10. Descriptions of Example Systems

Name	Huffman Algorithm	Data Encryption Algorithm
Function	Compressing	Encryption
Applications	Smartphone Camera	Mobile Banking
Language	C++	C++
LOC	861	59
Source Site	ActiveState Code	Cprogramming.com

예제 시스템 중 허프만 알고리즘의 경우 R3: Inline Temp, R40: Remove Parameter, R46: Introduce Parameter Object를 적용할 수 있고 데이터 암호화 알고리즘의 경우 R2: Inline Method, R42: Parameterize Method를 적용할 수 있다. Fig. 4는 데이터 암호화 알고리즘에서 반복적으로 사용되는 전달인자 집합을 객체로 대체하는 R46: Introduce Parameter Object의 적용 예시이다. Fig. 4.(a)는 암호화 알고리즘의 Original 코드를 보여주는 것이며, Fig. 4.(b)는 이에 대한 Refactor 코드를 보여준다.

```
class Queue
{
  ...
  void reheapup(int bottom, int root);
  void reheapdown(int bottom, int root);
  ...
}
```

(a) Original Code

```
class locInfo_int{
public:
  int bottom, root;
  ...
};
class Queue
{
  ...
  void reheapup(locInfo_int param);
  void reheapdown(locInfo_int param);
}
```

(b) Refactor Code

Fig. 4. Original and Refactor Codes of the Huffman algorithm

5.2 적용시스템 소모전력 측정

모바일 기기의 특성상 선정된 예제 시스템들의 전력 소모량이 높을수록 모바일 기기의 가용성을 떨어뜨리기 때문에 전력 효율성을 고려해야 한다.

Table 10의 허프만 알고리즘에 대해 실험의 정확도를 높이기 위해 동일한 입력 파일(9.8KB)을 사용하고 압축된 파일의 크기(1.1KB)가 일치하는 것을 확인했다. 앞서 제시한 3개의 리팩토링 기법에 대한 실험 결과는 Table 11과 같다. Table 11의 결과로부터 R3: Inline Temp 기법을 적용했을 때 에너지 절감 효과가 가장 좋은 것으로 나타났다. 3가지 기법의 적용으로부터 리팩토링의 적용에 의해 대체적으로 0.8%의 절감 효과가 나타나는 것으로 판단되었다.

Table 11. Energy Consumption of Refactored Huffman algorithm

Type		Energy Consumption
Original Code		5098.649 μJ
Refactor Code	Apply R3	5041.133 μJ
	Apply R40	5044.928 μJ
	Apply R46	5096.593 μJ

데이터 암호화 알고리즘 또한 실험의 정확도를 높이기 위해 동일한 입력 값을 사용하여 암호화 및 복호화를 수행했

다. 데이터 암호화 알고리즘에 선정된 2개의 기법을 적용한 결과는 Table 12와 같다. Table 12의 실험 결과로부터 암호화 알고리즘에 적용한 리팩토링 기법이 Original 코드의 소모 전력보다 낮은 전력을 소모하는 것으로 나타났다.

Table 12. Energy Consumption of Refactored Data encryption algorithm

Type		Energy Consumption
Original Code		102.329 μJ
Refactor Code	Apply R2	101.878 μJ
	Apply R42	101.789 μJ

Table 11과 Table 12의 실험결과로부터 우리는 4장에서 제시한 M Fowler의 63개 코드 리팩토링 기법의 에너지 효율성 분석 결과가 유효하다는 것을 알 수 있다. 비록 적은 양의 전력 절감을 제공하는 것으로 확인되었으나, 해당 기법이 전체 소프트웨어의 여러 부분에서 적용된다면 소모 전력의 절감 효과는 의미 있을 것으로 판단된다. 참고적으로 다수의 기법에 하나의 Original 코드에 적용되는 경우, 에너지 효율성이 좋아진다는 판단을 내리기는 쉽지 않았다. 예를 들면, R3 기법과 R40 기법을 동시에 적용하는 경우는 에너지 효율성이 좋아지지만, R2와 R42를 동시에 적용하는 경우 효율성이 떨어졌다. 따라서 어떤 리팩토링 기법의 조합이 에너지 효율성이 향상시킬 수 있는 가는 상황에 따라 실험을 통해 판단해야 할 것이다.

5.3 전력 효율성을 고려한 코드 리팩토링

소모 전력을 고려한 소프트웨어 개발 시에 개발자들은 코드 리팩토링 기법들의 전력 효율성 분석 정보를 활용하여 코드를 작성할 수 있다. 또한 소프트웨어의 유지보수 활동에서도 이러한 소모전력 분석 정보를 활용하여 코드 리팩토링을 수행할 수 있다.

Table 8과 Table 9에 나열된 기법들은 상대적으로 전력 효율성의 손실이 없는 경우에 해당되는 경우이기 때문에 개발자들이 해당 기법을 선택하는 경우, 소모전력 측면에서 긍정적인 효과를 제공하지만, 그 외의 기법들을 선택하는 경우 기법의 적용전보다 소모 전력이 많아지게 된다. 따라서 기법의 적용이 소모 전력을 높이는 경우, 특히 해당 소프트웨어가 에너지 효율성을 보장해야 하는 경우에 있어서는 기존 기법에 대한 변화가 필요하다. 이러한 변화는 다음과 같은 두 가지 전략으로 대처할 수 있다.

- 전략 1: 해당 기법을 전력 효율성이 좋은 기법으로 대체하여 리팩토링을 수행한다.
- 전략 2: 해당 기법의 리팩토링 절차를 보완하여 소모 전력이 절감할 수 있도록 재정의한다.

전략 1의 경우에 있어서는 M. Folwer의 리팩토링 기법 R17: Introduce Local Extension이 R16: Introduce Foreign Method 기법으로 대체하여 사용될 수 있다. 두 기법 모두 수정할 수 없는 클래스를 대상으로 확장이 필요 할 때 사용할 수 있는 기법이다. 하지만 두 기법의 원리에 차이가 존재하는데, R17의 경우 상속을 이용하고 R16의 경우 wrapper 함수를 이용한다. 즉, 수정할 수 없는 클래스를 활용하는 방법의 대체로 인하여 손쉽게 전력 효율성을 반전 시킬 수 있다.

전략 2의 경우는 좀 더 심도 있는 연구를 수행해야 한다. 기본적으로 R8: Replace Method with Method Object 기법의 경우, 추출이 어려운 긴 함수를 대신하여 동적으로 객체를 생성하고 생성된 객체가 긴 함수가 하는 기능을 대신 수행하도록 한다. 이 과정에서 지역함수의 수를 늘리고, 상대적으로 전력 효율성이 좋은 R1: Extract Method 기법을 수행하여 기존의 R8보다 전력 효율성을 향상시킬 수 있을 것이다.

6. 결론 및 향후 연구

일반적으로 개발자는 소프트웨어의 유지보수성을 향상시키기 위해 코드 리팩토링을 사용하지만 해당 기법들이 전력 효율성에 어떠한 영향을 미치고 있는지 알기가 쉽지 않다.

본 연구에서는 이러한 문제 상황에 대해 개발자가 쉽게 받아들일 수 있도록 M. Folwer가 제시한 63개의 코드 리팩토링 기법을 대상으로 전력 효율성에 대해 어떠한 영향을 미치고 있는지 실험과 분석을 통해 살펴보았다. 분석 결과 전체의 약 50%에 해당하는 33개의 기법이 소모전력 측면에서 우수하다는 것을 알 수 있었다. 분석 결과를 바탕으로 개발자들은 저전력 소프트웨어 개발 시 유지보수성 뿐만 아니라 전력 효율성까지 고려하여 개발할 수 있을 것이다.

추후의 연구는 실험을 통하여 전력 효율성이 저하되는 30개의 기법들에 대하여 효율성이 향상될 수 있도록 리팩토링 기법을 보완하는 방법을 개발하는 것이다. 이러한 연구를 통해 개발자는 전력 효율성이 보장되는 기법을 사용하여 기존 소프트웨어의 유용성을 향상시킬 수 있을 것이다.

Reference

[1] Hong,Jang-Eui and Kim,Doo-Hwan, “Task Extraction from Software Design Models to Improve Energy Efficiency of Embedded Software,” The KIPS Transactions: PartD, Vol.18-D, No.1, pp.45-56, 2011.

[2] H.Jun, et al., “Modelling and Analysis of Power Consumption for Component-Based Embedded Software,” Proc. of EUC Workshop, pp.795-804, 2006.

[3] E. Senn, et al., “SoftExplorer: Estimating and Optimizing the Power and Energy Consumption of a C Program for DSP Application,”EURASIP Jounal on Applied Signal Processing, Vol.16, pp.2641-2654, 2005.

[4] Kim, Doo-Hwan and Hong, Jang-Eui, “Energy Component Library for Power Consumption Analysis of Embedded Software,” The KIPS Transactions:PartD, Vol.16-D, No.6, pp.871-880, 2009.

[5] Kim, Jong-Phil, et al., “Estimating power consumption of mobile embedded software based on behavioral model.” Consumer Electronics (ICCE), 2010 Digest of Technical Papers International Conference on. IEEE, 2010.

[6] Höpfner, et al., “Energy Awareness Needs a Rethinking in Software Development.” ICSOFT (2), 2011.

[7] Yejin Kwon, et al., “Performance-based Refactoring: Identifying & Extracting Move-method Region,” Journal of KIISE: Software and Applications, Vol.40, No.10, pp.567-574, 2013.

[8] Jae-Jin Park, Jang-Eui Hong, “An Approach to improve software safety by Code refactoring,” Proc. of Korea Computer Congress, 2013.

[9] Jelschen, Jan, et al. “Towards Applying Reengineering Services to Energy-Efficient Applications.” Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on. IEEE, 2012.

[10] Vetro, Antonio, et al. “Definition, Implementation and Validation of Energy Code Smells: an Exploratory Study on an Embedded System.” ENERGY 2013, The Third International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies, 2013.

[11] Fowler, Martin. Refactoring: improving the design of existing code. Addison-Wesley Professional, 1999.

[12] Gottschalk, Marion, et al., “Removing Energy Code Smells with Reengineering Services.” GI-Jahrestagung, 2012.

[13] Marion, Gottschalk, Jan Jelschen, and Andreas Winter. “Energy-Efficient Code by Refac- toring.” 15. Workshop Software-Reengineering, 2013.

[14] Wellisson G. P. da Silva, et al., “Evaluation of the impact of code refactoring on embedded software efficiency.” I Workshop de Sistemas Embarcados, 2010.

[15] Oyedepo, Sunday Olayinka. “Efficient energy utilization as a tool for sustainable development in Nigeria.” International Journal of Energy and Environmental Engineering 3.1, 2012, pp.1-12.

[16] Marc Rosen, et al., “MA: Towards energy sustainability: a quest of global proportion. Forum of Public Policy online: A Journal of the Oxford Round Table, Summer, 2008.

[17] Z. Herczeg, D. Schmidt, and et al., “Eergy simulation of embedded XScale systems with XEEMU”, Journal of Embedded Computing, pp.209-219, August, 2009.

[18] Steigerwald, B., et al., “Writing Energy-Efficient Software.” Energy Aware Computing, Intel Corporation, 2011.

[19] Bunse, Christian, et al., “Exploring the energy consumption of data sorting algorithms in embedded and mobile environments.” Mobile Data Management: Systems, Services and Middleware, 2009. MDM'09. Tenth International Conference on. IEEE, 2009.

[20] Van Rysselberghe, et al., “Detecting move operations in versioning information.” Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on. IEEE, 2006.

[21] FB36, Huffman Data Compression (C++ recipe) [Internet], http://code.activestate.com/recipes/577480-huffman-data-compression/

[22] Goron350, Encryption and Decryption Program in C++ using a class [Internet], http://cboard.cprogramming.com/cplusplus-programming/66232-encryption-decryption-program-cplusplus-using-class.html

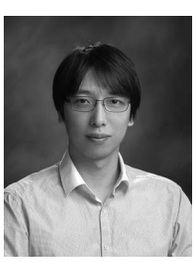
박 재 진

e-mail : jjpark@selab.cbnu.ac.kr

2012년 충북대학교 컴퓨터공학부(학사)

2012년～현 재 충북대학교 컴퓨터과학과 석사과정

관심분야 : 소프트웨어 공학, 소프트웨어 품질, 리팩토링



김 두 환

e-mail : dhkim@selab.cbnu.ac.kr

2007년 충북대학교 컴퓨터공학과(학사)

2009년 충북대학교 전자계산학과(석사)

2010년～현 재 충북대학교 컴퓨터과학과 박사과정

관심분야 : 소프트웨어 아키텍처, 임베디드 소프트웨어 모델링, 임베디드 소프트웨어 품질공학, 저전력 소프트웨어

홍 장 의

e-mail : jehong@chungbuk.ac.kr

2001년 KAIST 전산학과(박사)

2002년 국방과학연구소 선임연구원

2004년 (주)솔루션링크 기술연구소장

2004년~현 재 충북대학교 소프트웨어 학과 교수

관심분야 : 소프트웨어 공학, 소프트웨어 품질, 소프트웨어 프로세스, 임베디드 소프트웨어, 저전력 소프트웨어