한국정보과학회
KOREAN INSTITUTE OF INFORMATION SCIENTISTS AND ENGINEERS

제 19 권 제 1 호
Vol. 19 No. 1

SOFTWARE ENGINEERING SOCIETY

소프트웨어공학회 KIPS

2017

제19회 한국 소프트웨어공학 학술대회 논문집

Proceedings of the 19th Korea Conference on Software Engineering (KCSE 2017)

- 일시: 2017년 2월 8일(수) ~ 2월 10일(금)
- 장소: 강원도 평창 한화리조트(휘닉스파크점)

주최: 한국정보과학회, 한국정보처리학회
주관: 한국정보과학회 소프트웨어공학 소사이어티
한국정보처리학회 소프트웨어공학 연구회
한국전자통신연구원

후원: ㈜솔루션링크, ㈜코스콤,
㈜비트컴퓨터, ㈜모아소프트,
소프트웨어공학엑스퍼트그룹㈜, T3Q㈜,
STA 테스팅컨설팅㈜, 슈어소프트테크㈜,
TTA 소프트웨어시험인증연구소
SW 상시모니터링기술연구단
무인자율 및 적응형 소프트웨어센터

	논문 발표 B				
	B1: 분석 및 평가 좌장: 배경민교수 포항공대) 장소: 그랜드홀 2	**B2: 유지보수** 좌장: 이선아 (경상대) 장소: 세미나실 1	**B3: 보안 및 안전성** 좌장: 백종문 (KAIST) 장소: 세미나실 2	**B4: SW 품질 2** 좌장: 이관우 (한성대) 장소: 세미나실 3	
10:50-12:30 (100 분)	추상 도달가능성 그래프 기반 소프트웨어 모델체킹에서의 탐색전략 고려방법 [우수논문] 이낙원, 백종문 (KAIST) A Unified Approach for UML Based Safety Oriented Level Crossing Using FTA and Model Checking [일반논문] Anit Thapaliya, Gihwon Kwon (경기대학교) 감정 분석 기반의 사용자 피드백을 이용한 클라우드 서비스 평가 기법 [우수단편논문] 윤동규, 김웅수, 박준석, 염근혁 (부산대학교) 매쉬업 개발자를 위한 매쉬업 유사도 기반 서비스 추천 방법 [최우수논문] 김현승, 고인영 (KAIST)	코드 가독성 측정을 위한 소프트웨어 특징 목록 [단편논문] 최상철, 김순태 (전북대학교) 객체 지향 프로그램(C++)을 위장한 절차식(C) 패러다임 자동 식별화 구축 [단편논문] 변은영, 손현승, 장우성, 김영철 (홍익대학교), 김영수 (정보통신산업진흥원) N-그램 모델 기반의 코드 변경 추천 시스템 [단편논문] 김태현, 강성원 (KAIST), 이선아 (경상대학교), 금창섭 (한국전자통신연구원) 소프트웨어 유지보수 효율 향상 지원을 위한 의사코드 및 소스코드 양방향 자동 변환 연구 [학부논문] 권혁무, 장현수, 박동민, 서영석 (영남대학교)	모바일 아키텍처를 고려한 애플리케이션 보안 취약점 진단 방안 [단편논문] 김명근, 최은만 (동국대학교) 지능형 지속위협 생애주기를 고려한 보안요구사항 명세 방법 [단편논문] 김승준, 이석원 (아주대학교) 결함 분류체계와 휴먼 에러 분류체계를 적용한 SW 구현 단계 FMEA 수행 체계 [단편논문] 최이수, 한동준, 한혁수 (상명대학교) STPA 를 활용한 ISO 26262 기반의 안전분석 체계수립 [박사논문] 도성룡 (현대오트론), 한혁수 (상명대학교)	안드로이드 버그 분류: 이슈 설명서 기반 버그 담당자 선정 자동화에 대한 연구 [산업체논문] 최성옥, 조재호, 민모란, 민상윤 (KAIST 소프트웨어대학원) 컴포넌트 단위 학습을 이용한 버그 담당자 추천 [단편논문] 조충기, 김영민, 이기성, 이찬근 (중앙대학교) I-BL 기술의 성능 향상을 위한 버그리포트 품질 예측 및 자동 재구성 기술 [우수논문] 김미수, 안준, 이은석 (성균관대학교) 딥 러닝을 이용한 버그 담당자 자동 배정 연구 [우수논문] 이선로, 김혜민, 이찬근 (중앙대학교)	**워크샵**: **SW 상시 모니터링 기술 연구단** 장소: 세미나실 6 (09:00-12:30)
12:30-13:30	**중식**				

객체 지향 프로그램(C++)을 위장한 절차식(C) 패러다임 자동 식별화 구축

변은영 O1 손현승 2 장우성 3 김영수 5 김영철 4

홍익대학교 소프트웨어공학연구실

{eybyun[1], son[2], jang[3], bob[4]}@selab.hongik.ac.kr

정보통신산업진흥원 5

ysgold@nipa.kr[5]

Constructing an Automatic system for identifying the faked Procedural-Oriented Paradigm(C) within Object-Oriented Program (C++)

Eun Young Byun, Hyun Seung Son, Woo Sung Jang, Young S. Kim, Young Chul Kim

SE Lab., Hongik University

NIPA

요 약

대규모의 소프트웨어의 증가로 다수 개발자의 협업과 재사용성이 중요한 이슈이다. 바람직하지 않은 코드는 복잡도를 높이고 재사용성을 낮춘다. 이를 해결함으로써 소프트웨어의 품질과 유지보수성을 향상시키고자 한다. 본 논문은 객체 지향 프로그램(C++) 내에서 절차식(C) 패러다임 사용을 최소화하는 방법을 제안한다. 이를 위해 자동 식별 가시화 시스템을 제안하고 식별된 절차식 스타일의 리팩토링을 통해 소프트웨어의 복잡도를 낮추고 재사용성을 높여 품질을 향상시키고자 한다.

1. 서 론

객체 지향 프로그램은 속성과 메소드들의 그룹인 클래스로 구성된다. 하나의 객체는 특정한 작업과 연관된 구조로, 소프트웨어 복잡성을 낮추고 재사용이 가능하다[1,2]. 객체 지향 패러다임은 객체에 대한 충분한 이해가 필요하다. 하지만 대부분의 개발자는 객체 지향 프로그램을 절차식 패러다임으로 구현하는 실수를 한다[3]. 이것이 객체 지향 프로그램을 위장한 절차식 패러다임이다. C++의 경우, C와 이름이 비슷하듯이 유사한 부분이 많다[4]. 하지만 C++과 C는 객체 지향, 절차 지향이라는 결정적인 차이를 갖는다. 이 차이를 이해하지 못한다면 개발자는 C++에서 C패러다임으로 프로그래밍하는 실수를 할 수 있고 이 경우를 개발자가 스스로 인지하기는 쉽지 않다.

기존 연구[5,6]에서 소프트웨어 구조 가시화와 복잡도를 측정하는 툴을 개발하여 C++로 구현된 프로그램을 가시화했다. 그림1은 해당 툴에서 측정한 복잡도를 낮추기 위해 리팩토링을 수행하며 개선한 수치 리스트이다. 복잡도(Coupling)는 정적 분석 도구(CCheck)의 Style, Warning, Performance Violation이 연관성이 있고 이 항목들은 C패러다임을 포함한다. 따라서 리팩토링 과정에서 C패러다임을 사람이 직접 식별하고 수정한다. 수작업으로 수행되기 때문에 종종 식별하지 못하는 경우도 발생했다.

이 논문은 C++을 위장한 C패러다임을 정의하고, 프로그램의 전체 구조와 함께 C패러다임이 나타난 부분을 가시화하는 자동 식별 가시화 시스템을 제안한다. 이 결과를 통해 C패러다임을 자동으로 식별할 수 있다. C패러다임의 리팩토링을 통해 소프트웨어의 복잡도를 낮추면 품질 향상과 함께 객체 지향 특성인 재사용성을 높인다.

본 논문의 구성은 다음과 같다. 2장에서는 C++ 코딩 룰을 기반으로 C++을 위장한 C패러다임을 정의한다. 3장은 자동 식별 가시화 시스템으로 Tool-Chain 구조와 C패러다임 추출 가시화에 대해 기술한다. 마지막으로 4장에서는 결론 및 향후 연구를 언급한다.

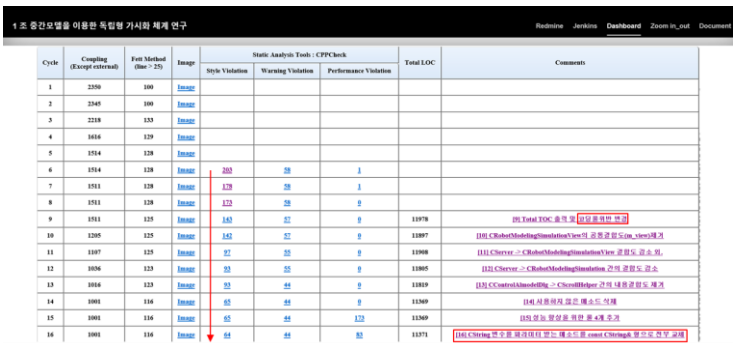

그림1 C++ 프로그램 복잡도 수치 리스트

† 이 논문은 2015년 교육부와 한국연구재단의 지역혁신창의인력양성사업의 지원을 받아 수행된 연구임(NRF-2015H1C1A1035548)

2. C++을 위장한 C패러다임

기존의 C++ 코딩 룰은 프로그램 코딩 시, 가장 바람직하지 않은 경우들을 대상으로 만들어 진다. 이 중에서도 많은 표준들은 C++에서 C패러다임의 사용을 최소화하기 위해서 정의된다[7]. C++을 위장한 C패러다임은 C++ 코딩 룰과 C++에 추가된 라이브러리를 기반으로 정의한다. 표1은 C++를 위장한 C 패러다임 항목이다.

	C 패러다임	C++ 패러다임
input/output	printf(), scanf()	std::cout, std::cin
메모리 할당	malloc(), calloc(), free()	new, delete
cast	(cast형)	static_cast<cast형>
struct	keyword 생략 X typedef사용	keyword 생략
enum	keyword 생략 X	keyword 생략
namespace	X	O
Boolean	X	O

표1 C++을 위장한 C패러다임 항목

◆ input / output

C는 문자열 입출력을 위해서 stdio 헤더파일의 printf(), scanf()함수를 사용한다. C++은 추가된 iostream 헤더파일의 cout, cin을 사용한다. cout, cin은 printf(), scanf()와 다르게 대상이 되는 변수형에 상관 없이 연산자 오버로딩을 통해 자동으로 바꾸어 준다. 따라서 printf(), scanf() 함수를 호출하는 경우, C패러다임으로 판단한다. 그림2는 C/C++ 패러다임 input/output 비교 예시이다.

C Paradigm

```
#include<stdio.h>

class A{
    void func(){
        printf("C Style");
        printf("printf function");
    }
}
```

C++ Paradigm

```
#include<iostream>

class A{
    void func(){
        std::cout << "C++ Style";
        std::cout << "cout function";
    }
}
```

그림2 C/C++ 패러다임 input/output

◆ 메모리 공간 할당

C는 메모리 할당을 위해서 malloc(), calloc(), free()를 사용한다. C++은 new, delete 키워드를 사용한다. C++은 경우에 따라 두 가지 방법을 모두 사용할 수 있지만 객체 생성 시에는 반드시 new와 delete를 사용해야 한다. 따라서 malloc(), calloc(), free()를 호출하는 경우, C 패러다임으로 판단한다. 그림3은 C/C++ 패러다임 메모리 할당 함수 비교 예시이다.

C paradigm

```
#include<iostream>

class A{
    void func(){
        int *i;
        i = (int*)malloc(sizeof(int));
        free(i);
    }
}
```

C++ paradigm

```
#include<iostream>

class A{
    void func(){
        int *i;
        i = new int;
        delete i;
    }
}
```

그림3 C/C++ 패러다임 메모리 할당

◆ cast

C는 적절한 헤더가 없을 시에는 타입에 대한 정보를 찾지 못해 컴파일 후에 오류가 발생한다. C++은 static_cast를 이용해 예상치 못한 오류를 방지할 수 있기 때문에 더 안전하다. 그림4는 C/C++ 패러다임 cast 비교 예시이다.

C paradigm

```
extern void Fun(Derived*);

class A{
    void Gun(Base* pb){
        Derived* pd = (Base*)pb;
        Fun(pd);
    }
}
```

C++ paradigm

```
extern void Fun(Derived*);

class A{
    void Gun(Base* pb){
        Derived* pd = static_cast<Derived*>pb;
        Fun(pd);
    }
}
```

그림4 C/C++ 패러다임 cast

◆ struct

C++은 C와 다르게 구조체 내에 함수의 선언이 가능하고 구조체 변수의 생성이나 함수 파라미터로 사용 시, struct 키워드를 생략할 수 있다. 따라서 구조체 변수 선언이나 함수 파라미터에서 struct 키워드를 사용한 경우, C 패러다임으로 판단한다. 그림5는 C/C++ 패러다임 struct 비교 예시이다.

C paradigm

```
class A{
    struct cStyle{
        …
    }
    void func(struct cStyle *c){
        struct cStyle s;
        …
    }
}
```

C++ paradigm

```
class A{
    struct cStyle{
        …
        void func(){
            …
        }
    }
    void func(cStyle *c){
        cStyle s;
        …
    }
}
```

그림5 C/C++ 패러다임 struct

◆ enum

struct 키워드와 마찬가지로 C++은 C와 다르게 enum 변수의 생성이나 함수 파라미터로 사용 시, enum 키워드를 생략할 수 있다. 따라서 구조체 변수의 생성이나 함수 파라미터에서 enum 키워드를 사용한 경우, C 패러다임으로 판단한다. 그림6은 C/C++ 패러다임 enum비교 예시이다.

C paradigm

```
class A{
    enum state {
        …
    }
    void func(enum state *s){
        enum state t;
        …
    }
}
```

C++ paradigm

```
class A{
    enum state {
        …
    }
    void func(state *s){
        state t;
        …
    }
}
```

그림6 C/C++ 패러다임 enum

◆ namespace

C++은 C에는 없는 namespace라는 새로운 키워드가 추가되었다. 동일한 이름을 가진 함수들의 충돌을 방지할 수 있다. C는 동일한 기능이지만 이름에 차이를 주기 위해서 앞에 식별할 수 있는 수식어를 붙여 사용한다. 수식어 구분은 대문자나 '_'로 구분한다. 따라서 동일한 기능의 이름 앞에 구분을 위한 단어를 포함하고 있는 메소드들이 있는 경우, C 패러다임으로 판단한다. 그림7은 C/C++ 패러다임 namespace 비교 예시이다.

C paradigm

```
class A{
    void aumoaPrintInfo(void){
        std::cout << "Made by Aumoa";
        return;
    }
    void libPrintInfo(void){
        std::cout << "Made by Lib";
        return;
    }
}
```

C++ paradigm

```
class A{
    namespace Aumoa{
        void PrintInfo(void){
            std::cout << "Made by Aumoa";
            return;
        }
    }
    namespace Aumoa{
        void PrintInfo(void){
            std::cout << "Made by Lib";
            return;
        }
    }
}
```

그림7 C/C++ 패러다임 namespace

◆ Boolean

C++은 C에는 없는 Boolean Type이 있다. C는 int형을 이용해 Boolean Type을 대체했다. 조건 식에 int형 변수만 있는 경우, C 패러다임으로 판단한다. 그림8은 C/C++ 패러다임 Boolean 비교 예시이다.

C paradigm

```
class A{
    void func(){
        int condition=1;

        while(condition){
            …
        }
    }
}
```

C++ paradigm

```
class A{
    void func(){
        bool condition=true;

        while(bool){
            …
        }
    }
}
```

그림8 C/C++ 패러다임 Boolean

3. 자동 식별 가시화 시스템

3.1 Tool-Chain 구성도

C++에서의 C패러다임을 자동 식별 가시화하기 위해서 연구실에서 개발한 xCodeParser와 오픈소스인 SQLite, Graphviz를 이용하여 Tool-Chain을 구현했다. 그림9는 Tool-Chain 구성도로 총 5단계로 나누어진다.

➢ Step1: 대상이 되는 소스 코드를 xCodeParser에 입력한다. 이 논문은 객체 지향 언어인 C++기반의 코드를 입력한다.

➢ Step2: 입력된 코드를 xCodeParser를 사용해 분석한다. 이 때, 각 파일에 대한 분석결과로 ASTM파일들이 생성되며 프로젝트, 클래스, 구조체, 멤버, 변수, 메소드 등에 대한 정보를 트리 형태로 저장한다.

➢ Step3: 데이터베이스 저장 단계는 ASTM 데이터를 원하는 구조로 구조화하여 데이터베이스에 저장한다. ASTM_content, ASTM_link, ASTM_local, ASTM_API 테이블을 생성하고, 이 데이터를 기반으로 C패러다임이 발견된 파일, 멤버와 C패러다임 종류 정보를 구조화하여 CCheck 테이블을 생성한다. 이 테이블로 코드의 어떤 부분에 어떤 C 패러다임이 발견되는지 확인할 수 있다.

➢ Step4: 구조 분석 단계는 데이터베이스에 저장된 정보를 DOT 문법에 맞게 재구성한다.

➢ Step5: 가시화 단계는 재구성한 정보를 Graphviz를 이용해 가시화한다.

3.2 C패러다임 추출 가시화

Tool-Chain을 통해 C++에서 C패러다임을 가시화한 결과는 그림 파일로 추출된다. 소스 코드를 구성하는 모든 클래스, 구조체, cpp파일에 대한 구조 정보를 노드로 가시화하고 컴포넌트(클래스, 구조체, cpp)간의 관계(상속 관계, 호출 관계, 객체 생성 관계)를 화살표로 가시화한다. 그림10은 가시화 노드, 관계의 종류를 나타낸다.

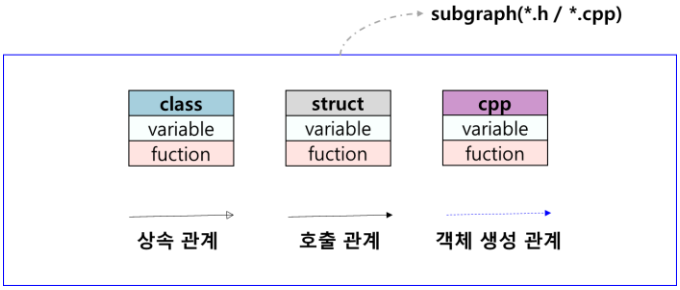

그림10 가시화 노드, 관계의 종류

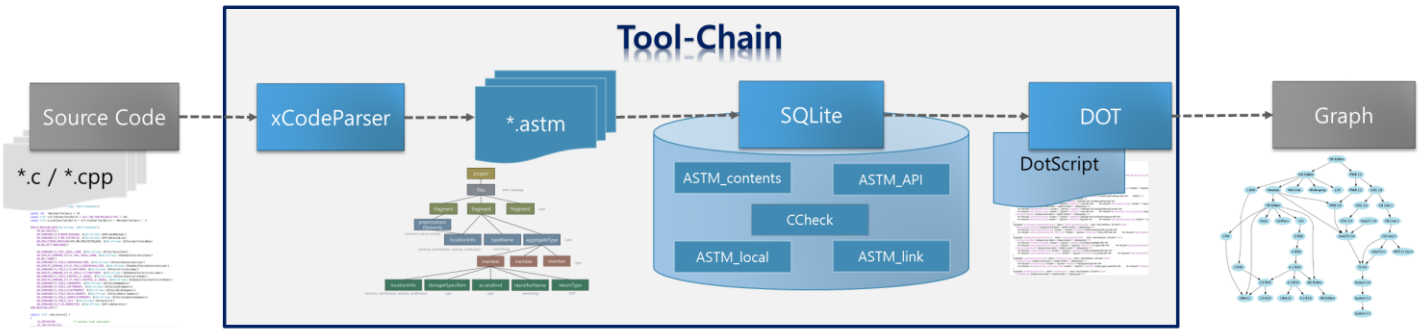

그림9 Tool-Chain 구성도

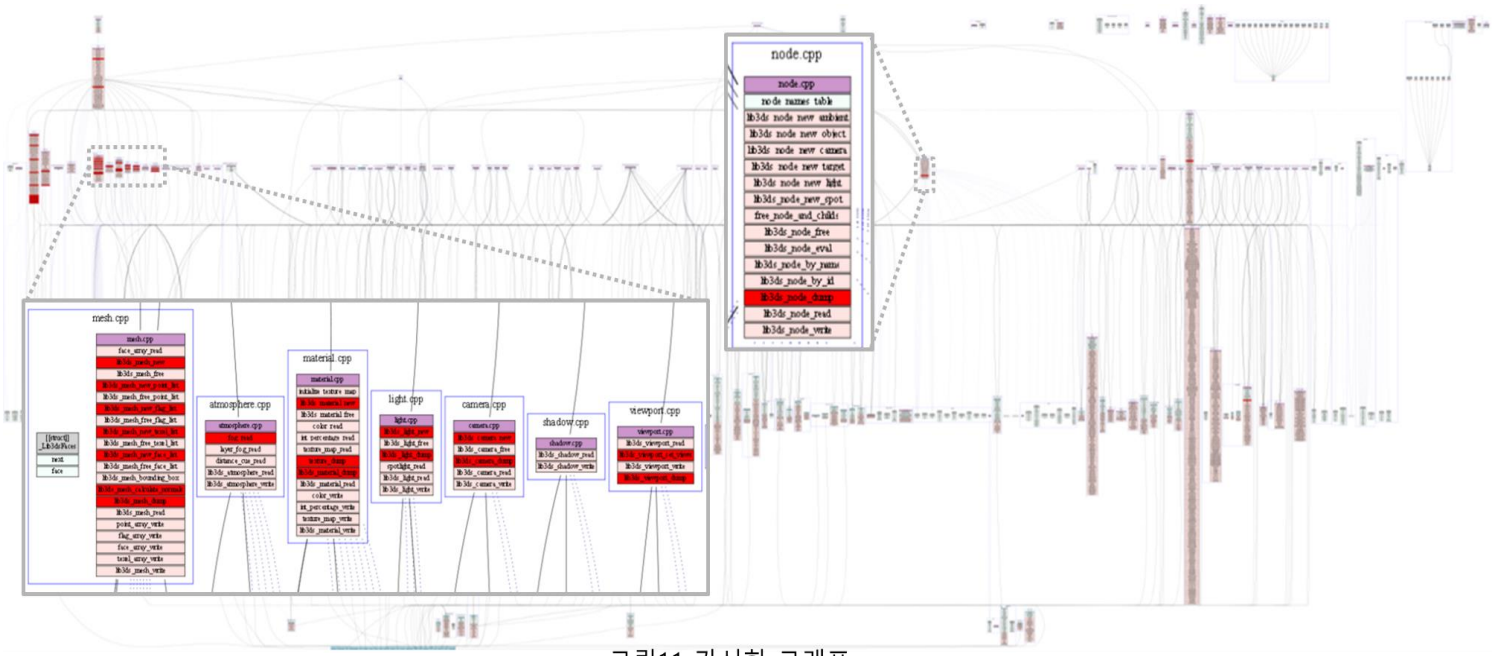

그림11 가시화 그래프

그림11은 RobotModelingSimulation코드를 Tool-Chain을 통해 가시화한 그래프이다. 해당 코드는 크고 복잡한 시스템이지만, 가시화를 통해 모든 컴포넌트들의 구조와 관계를 확인할 수 있다. C패러다임이 발견된 부분을 빨간색으로 표시하여 가시성을 높였다. 그림12는 CCheck 테이블로 어떤 C패러다임이 발견 되는지 그림에 도식화 하지 않았지만 이를 통해 확인 할 수 있다. 대표적으로 그림에서 node.cpp와 mesh.cpp에서 빨간색으로 표시한 부분과 CCheck 테이블에 데이터가 동일하고 C패러다임 종류를 알 수 있다. 해당 부분들을 리팩토링 하면 소프트웨어의 복잡도를 낮출 수 있고 재사용성을 높일 수 있다.

	mem_name	def_type	inout	namespace	bool	struct	enum	malloc	cast
₩RobotModelingSimulation₩lib3ds₩atmosphere.cpp	fog_read	Lib3dsBool				Y			
₩RobotModelingSimulation₩lib3ds₩camera.cpp	lib3ds_camera_new	Lib3dsCamera*						Y	
₩RobotModelingSimulation₩lib3ds₩camera.cpp	lib3ds_camera_dump	void	Y						
₩RobotModelingSimulation₩lib3ds₩chunk.cpp	lib3ds_chunk_debug_dump	void	Y						
₩RobotModelingSimulation₩lib3ds₩chunk.cpp	lib3ds_chunk_unknown	void	Y						
₩RobotModelingSimulation₩lib3ds₩chunk.cpp	lib3ds_chunk_dump_info	void	Y						
₩RobotModelingSimulation₩include₩drawstuff₩drawstuff.h	dsSimulationLoop	void				Y			
₩RobotModelingSimulation₩lib3ds₩file.cpp	lib3ds_file_new	Lib3dsFile*						Y	
₩RobotModelingSimulation₩lib3ds₩file.cpp	dump_instances	void	Y						
₩RobotModelingSimulation₩lib3ds₩light.cpp	lib3ds_light_new	Lib3dsLight*						Y	
₩RobotModelingSimulation₩lib3ds₩light.cpp	lib3ds_light_dump	void	Y						
₩RobotModelingSimulation₩lib3ds₩material.cpp	lib3ds_material_new	Lib3dsMaterial*						Y	
₩RobotModelingSimulation₩lib3ds₩material.cpp	texture_dump	void	Y						
₩RobotModelingSimulation₩lib3ds₩material.cpp	lib3ds_material_dump	void	Y						
₩RobotModelingSimulation₩lib3ds₩matrix.cpp	lib3ds_matrix_dump	void	Y						
₩RobotModelingSimulation₩lib3ds₩mesh.cpp	lib3ds_mesh_new	Lib3dsMesh*						Y	
₩RobotModelingSimulation₩lib3ds₩mesh.cpp	lib3ds_mesh_new_point_list	Lib3dsBool						Y	
₩RobotModelingSimulation₩lib3ds₩mesh.cpp	lib3ds_mesh_new_flag_list	Lib3dsBool						Y	
₩RobotModelingSimulation₩lib3ds₩mesh.cpp	lib3ds_mesh_new_texel_list	Lib3dsBool						Y	
₩RobotModelingSimulation₩lib3ds₩mesh.cpp	lib3ds_mesh_new_face_list	Lib3dsBool						Y	
₩RobotModelingSimulation₩lib3ds₩mesh.cpp	lib3ds_mesh_calculate_normals	void	Y					Y	
₩RobotModelingSimulation₩lib3ds₩mesh.cpp	lib3ds_mesh_dump	void	Y						
₩RobotModelingSimulation₩lib3ds₩node.cpp	lib3ds_node_dump	void	Y						

그림12 CCheck 테이블

4. 결론 및 향후 연구

본 논문은 객체 지향 프로그램을 절차식 패러다임처럼 구현하는 실수로 인한 소프트웨어의 복잡도와 재사용 문제를 해결하기 위해 C++에서의 C패러다임을 자동 식별 가시화 시스템을 언급한다. 추출된 부분을 객체 지향화하는 리팩토링을 통해, 소프트웨어의 복잡도를 낮추고 소프트웨어의 품질을 향상시키면서 객체 지향 언어의 특성인 재사용성을 높일 수 있다. 향후에는 C패러다임을 추가적으로 검출하고 식별한 C패러다임들을 자동으로 리팩토링 해주는 시스템을 구현할 예정이다.

참고문헌

[1] S Wiedenbeck, V Ramalingam, “Novice Comprehension of small programs written in the procedural and object-oriented styles”, International Journal of Human-Computer Studies, Vol.51, No.1, pp. 71-87, 1999

[2] 변은영, 박보경, 장우성, 김영철, “객체 지향 패러다임에서의 코드 재사용을 위한 응집도 레벨 식별 모범 사례”, 한국정보처리학회, Vol. 23, No.2, pp. 475-478, 2016

[3] R Fanta, V Rajlich, “Restructing legacy C code into C++”, ICSM, pp. 77-85, 1999

[4] Herbert Schildt, “C/C++ Programmer’s Reference”, McGraw-Hill, 2000

[5] 강건희, 손현승, 김영수, 박용범, 김영철, “SW 가시화 기반 리펙토링 기법 적용을 통한 정적 코드 복잡도 개선”, 한국정보처리학회, Vol. 21, No.2, pp. 646-649, 2014

[6] 변은영, 박보경, 장우성, 김영철, “가치 있는 모듈 식별을 위한 오픈 소스 기반 소프트웨어 현대화 시스템 구축”, 한국정보과학회, pp. 404-406, 2016

[7] H Sutter, A Alexandrescu, “C++ Coding Standards: 101 Rules, Guidelines, and Best Practices”, Pearson Education, 2004