http://dx.doi.org/10.3745/KIPSTD.2012.19D.4.293

레퍼런스 흐름에 기반한 디자인 패턴의 확장 지점 식별

김 희 천[†] · 박 찬 진[††] · 김 택 수[†††] · 유 찬 우[††††] · 이 형 원[†††††]

요 약

디자인 패턴은 기존 소프트웨어의 설계 정보를 추상화하는 단위로 사용될 수 있다. 소프트웨어의 기능을 확장하기 위해 설계를 파악하고자 할 때 디자인 패턴의 인스턴스를 파악하는 것 만으로는 충분치 않을 수 있는데, 이는 기능의 확장이 일어나는 지점이 패턴 인스턴스 바깥에 존재할 수 있기 때문이다. 본 논문에서는 디자인 패턴의 핵심적인 구조를 구성하는 과정으로서 레퍼런스 흐름을 정의하였으며, 레퍼런스 흐름을 이용하여 패턴의 확장 지점을 찾아내는 방법을 제시하였다.

키워드 : 레퍼런스 흐름, 디자인 패턴, 확장 지점

Identification of the Extension Points of Design Patterns Based on Reference Flows

Heechern Kim[†] · Chanjin Park[††] · Taeksu Kim[†††] · Chanwoo Yoo[††††] · Hyungwon Lee[†††††]

ABSTRACT

Design patterns is a kind of abstraction that represents design information of software. Sometimes it is not sufficient to identify design pattern instances for extension of software, because the extension points exist outside of the instances. We define reference flows, which is a process of composing an intrinsic structure of design patterns, and suggest an analysis method based on reference flows for identifying the extension points.

Keywords : Reference Flow, Design Pattern, Extension Point

1. 서 론

디자인 패턴은 설계 정보를 담는 일종의 추상이라고 할 수 있다. 따라서 기존 소프트웨어의 디자인 패턴을 찾아냄으로써 소프트웨어의 설계를 보다 잘 이해할 수 있다. 이에 기존의 많은 연구들이 디자인 패턴 식별을 위해 노력해 왔다.

본 논문은 디자인 패턴을 단순히 식별하는 데 그치지 않고, 디자인 패턴과 관련된 패턴 주위의 설계 정보를 추출함으로써, 패턴의 확장 지점을 식별하는 방법을 제시한다. 패턴의 확장 지점이란, 패턴에 새로운 객체를 추가하고자 할 때, 생성된 객체를 넘겨주어야 하는 지점을 말한다. 많은 디자인 패턴이 인터페이스 타입의 객체를 추가하는 것 만으로 기능을 확장할 수 있도록 되어 있기 때문에, 패턴의 확장 지점을 파악하게 되면 소프트웨어의 유지 보수가 한결 용이해지게 된다.

패턴의 확장 지점을 알아내기 위해서는 패턴 인스턴스를 찾아내는 것 만으로는 부족하다. 확장 지점이 인스턴스 바깥쪽에 존재할 수 있기 때문이다. 본 논문에서는 패턴 주위의 설계 정보를 레퍼런스 흐름으로 규정하고, 레퍼런스 흐름을 이용해 확장 지점을 찾아내게 된다.

본 논문은 다음과 같이 이루어져 있다. 먼저, 2장에서 관련연구를 기술한다. 3장에서는 레퍼런스 흐름의 개념을 설명하고 레퍼런스 흐름과 패턴 확장 지점 사이의 상관관계를 설명한다. 4장에서는 정적 분석을 통해 레퍼런스 흐름을 분석하는 방법을 제시한다. 5장에서는 앞서 제시한 방법의 구현을 적용한 사례를 보인다. 마지막으로 6장에서 결론을 맺고 향후 과제에 대해 기술한다.

※ 이 논문은 2008년도 강릉원주대학교 교수연구년연구 지원에 의하여 수행되었음.
† 정 회 원 : 한국방송통신대학교 컴퓨터과학과 교수
†† 정 회 원 : LG전자 TV연구소 수석연구원
††† 정 회 원 : 삼성전자 소프트웨어센터 책임연구원
†††† 정 회 원 : 서울대학교 전기컴퓨터공학부 공학박사
††††† 종신회원 : 강릉원주대학교 컴퓨터공학과 교수
논문접수 : 2012년 5월 3일
심사완료 : 2012년 6월 7일
* Corresponding Author : Hyungwon Lee(lhw@gwnu.ac.kr)

2. 관련 연구

기존 연구들은 크게 디자인 패턴을 그 의미에 기반하여 찾는 연구와 코드 형태의 유사도에 기반하여 찾는 연구로 나눌 수 있다.

디자인 패턴을 그 정의에 기반해 분석하지 않고도 코드의 형태적인 특징에만 기반하여 찾는 것이 가능하다[1]. 실제로 프로그램을 그래프로 나타낸 뒤, 그래프를 직접 비교하거나[2][3], 그래프의 성질을 표현하는 메트릭을 이용해[4][5][6][7][8] 패턴 인스턴스를 찾는 연구들이 수행되었다. 이와 같이 의미에 그 기반을 두지 않고 디자인 패턴을 찾는 방법은 그 정확도에 있어서는 대체로 좋은 결과를 보여주지만, 찾아낸 패턴 인스턴스를 의미적으로 설명하기가 어렵다는 단점을 가진다.

의미에 기반하여 패턴을 찾는 연구들은 패턴 인스턴스가 되기 위한 구조적 조건[9][10] 및 동적 조건[11][12]들을 패턴의 정의에 맞게 설정하고, 코드를 분석한 결과가 해당 조건들을 만족시키는지의 여부를 판단하여 패턴을 찾아내게 되며, 조건들을 선택적으로 적용함으로써 유사하지만 조금씩 다른 패턴들을 찾아내기도 한다[13]. 의미 기반의 기존 연구들의 문제는 패턴 인스턴스를 제시하는 데 그쳐, 설계를 이해하는데 필요한 인스턴스 주위의 정보를 제공하는 데는 소홀하다는 것이다. 패턴 확장에 필요한 부분이 인스턴스 바깥에 존재할 경우, 이와 같은 접근 방식은 도움이 되지 못할 수 있다. 본 논문은 패턴 인스턴스 주위의 설계 정보를 제공함으로써 패턴의 확장 지점에 대한 정보를 제공한다는 점에서 기존 연구들과 차이를 가진다.

3. 레퍼런스 흐름 및 확장 지점

GoF의 디자인 패턴[14] 23개 중 18개는 '구현이 아니라 인터페이스 타입에 대해 프로그래밍 하라'는 원리를 따르고 있다. 구현을 직접 사용하지 않고, 인터페이스 타입의 레퍼런스를 사용하게 되면, 레퍼런스에 어떤 객체가 저장되느냐에 따라 소프트웨어의 행동이 바뀌기 때문에, 소프트웨어의 확장을 쉽게 만든다. 이와 같이 인터페이스 타입의 레퍼런스에 저장된 객체를 사용하는 관계를 인터페이스 타입에 대한 '구성 관계'라고 한다. 많은 디자인 패턴이 변경에 강한 소프트웨어를 만들기 위해 이와 같은 관계를 사용하고 있기 때문에, 구성 관계를 중심으로 패턴을 분석할 때 의미적으로 보다 유효한 결과를 얻을 수 있다.

레퍼런스 흐름이란 구성 관계가 형성되는 과정을 가리킨다. 즉, 객체가 생성되어 인터페이스 타입의 레퍼런스에 저장되고, 레퍼런스가 호출되어 사용되기까지 객체의 레퍼런스가 프로그램 내에서 이동한 경로를 레퍼런스 흐름으로 정의할 수 있다. 이 때 경로를 이루는 노드의 단위는 메서드와 필드가 된다. 또한, 경로 상에서 상대적으로 중요한 역할을 가지는 노드들을 다음과 같이 정의할 수 있다.

Creator	객체를 생성하는 메서드
Store	객체를 저장하는 필드
Injector	객체를 필드에 저장하는 메서드를 호출하는 메서드
Caller	인터페이스 타입의 레퍼런스를 호출하는 메서드

각각의 역할을 GEBNF[15]와 일차 논리를 이용해 기술하면 각각 다음과 같다.

1. $isStore(attr,B) \equiv \exists A,B \in classes \cdot \exists attr \in A.attrs \cdot allAbstract(B) \wedge attr.type = B$

필드 attr의 타입이 인터페이스 타입 B이면 attr은 B의 어떤 하위 타입 객체도 저장할 수 있다. 이 경우에 attr을 B 타입의 Store라고 한다.

2. $isCaller(a, b) \equiv \exists A,B \in classes \cdot \exists a \in A.opers \cdot b \in B.opers \cdot allAbstract(B) \wedge callsHook(a, b)$

메서드 a가 B라는 인터페이스 타입의 레퍼런스에 메시지를 보낼 경우, a는 B 타입에 대한 Caller가 된다. 이 경우에 실제로 호출되는 메서드는 레퍼런스가 어떤 객체를 가리키는가에 따라 달라지게 된다.

3. $isCreator(a,B) \equiv \exists A,B,C \in classes \cdot \exists a \in A.opers \cdot isMakerFor(a,C) \wedge C \in subs(B) \wedge allAbstract(B)$

메서드a가 B 타입의 하위 타입 객체를 생성할 경우, a는 B 타입에 대한 Creator가 된다.

4. $isInjector(a, attr,C) \equiv \exists A,B,C \in classes \cdot \exists a \in A.opers \cdot \exists b \in B.opers \cdot \exists attr \in B.attrs \cdot allAbstract(C) \wedge attr.type = C \wedge calls(a, b) \wedge isSetter(b) \wedge hasInParam(b,C)$

메서드 a가 attr이라는 필드에 객체를 저장하는 셋터(setter) 메서드를 호출하고, attr에 저장되는 객체의 타입이 C의 하위 타입일 때, a는 attr에 C의 하위 타입 객체를 저장하는 Injector라고 정의할 수 있다.

위 역할 정의를 보면 Creator와 Injector는 인터페이스 타입의 하위 타입인 구상 타입과 연관되어 있으며, Store와 Caller는 인터페이스 타입, 즉 추상 타입과 연관되어 있는 것을 볼 수 있다. Store와 Caller는 추상 타입을 다루게 되므로 패턴의 기능 확장 시에도 변하지 않는 추상 계층이 된다. 기능의 확장을 위해서는 추상 계층이 기존에 사용하고 있는 객체 외에 새로운 객체를 생성하여 추상 계층에 넘겨줄 필요가 있다. 추상 계층에 구상 객체를 삽입하는 역할을 하는 것이 Injector이므로, 새로운 기능을 추가하기 위해서는, 객체를 생성하여 Injector로 하여금 Store에 삽입하도록 하면 된다. 따라서 Injector는 기존 패턴에 대한 확장 지점이 된다고 할 수 있다. Injector는 Store의 셋터를 호출하는 메서드로 정의되지만, 이와 같은 기준에 의해 Injector를 단독으로 구하는 것은 유효하지 않은 Injector들을 찾아낼 위험이 있다. 예를 들어, 실제로 Caller에게 객체를 넘겨주지 않는 Store에 대한 Injector는 의미 있는 확장 지점이라고 할 수 없다. 따라서, 의미 있는 확장 지점을 구하기 위해서는

유효한 레퍼런스 흐름 전체를 구한 뒤, 레퍼런스 흐름 상의 Injector를 패턴의 확장 지점으로 보는 것이 바람직하다. 다음 장에서는 Injector를 구하기 위한 레퍼런스 흐름을 정적 분석을 통해 얻어내는 방법을 설명한다.

4. 레퍼런스 흐름 분석 방법

프로그램의 정적 분석을 통해 레퍼런스 흐름을 구하는 과정은 프로그램이 가지고 있는 모든 인터페이스 타입 각각에 대해 다음과 같은 단계를 거쳐 이루어진다.

1. 메서드 내부 경로 분석
2. 메서드 및 필드 간 경로 분석
3. 경로수집 및 요약

각각의 단계에서 이루어지는 분석의 과정을 자세히 설명하면 다음과 같다.

4.1 메서드 내부 경로 분석

각 메서드 별로 인터페이스 타입의 하위 타입 레퍼런스가 들어오고 나가는 경로를 파악하여야 한다. 또한, 레퍼런스가 생성되는 지점 및 레퍼런스에 대한 호출이 일어나는 지점 역시 파악할 필요가 있다. 두 지점이 각각 레퍼런스 흐름의 시작 지점 및 끝 지점이 되기 때문이다. 각각의 지점에 대한 정보를 LocalInfo라고 했을 때 메서드 내부 경로 분석은 관심의 대상이 되는 LocalInfo를 구하는 것으로부터 이루어진다. 레퍼런스 흐름과 연관된 LocalInfo의 종류는 다음과 같다.

LocalInfo ::= In | Out | Creation | Call
In ::= MethodParamIn | FieldIn | InvokeIn
Out ::= ReturnOut | FieldOut | InvokeParamOut
각각의 LocalInfo에 대해 설명하면 다음과 같다.

1. In
메서드 내로 유입되는 객체를 가리키는 지역변수

MethodParamIn	현재분석 대상인 메서드의 파라미터를 담는 지역변수
FieldIn	필드값을 읽어들이는 지역변수
InvokeIn	메서드를 호출하여 리턴된 값을 담는 지역변수

2. Out
메서드 바깥으로 나가는 객체를 가리키는 지역변수

ReturnOut	현재 분석 대상인 메서드의 리턴값을 담는 지역변수
FieldOut	필드에 저장하는 값을 담는 지역변수
InvokeParamOut	메서드를 호출할 때 인자로 넘겨지는 지역 변수

3. Creation
메서드 내에서 객체가 생성될 때, 생성되는 객체를 담는 지역 변수. 레퍼런스 흐름 분석시에 경로의 시작 노드가 된다.

4. Call
메서드를 호출할 때 호출의 대상이 되는 객체를 가리키는 지역 변수. 레퍼런스 흐름 분석시에 경로의 목표 지점이 된다.

레퍼런스 흐름과 연관이 있는 모든 지역 변수를 구한 뒤에는 지역 변수 사이에서 레퍼런스 전달이 있었는지의 여부를 구해야 한다. 이는 지역 변수가 가리킬 수 있는 레퍼런스의 집합을 Points-to Analysis를 통해 구한 뒤, 집합 사이에 공집합이 아닌 교집합이 존재하는가의 여부를 통해 구할 수 있다. 정확한 레퍼런스 집합을 구하는 것은 NP-hard 문제이기 때문에[16], 흐름 무관(flow insensitive) 및 상황 무관(context insensitive) 분석 방식을 사용하는 근사 알고리즘을 통해 레퍼런스 집합을 구해야 한다. 대표적인 알고리즘으로 Andersen의 알고리즘[17]과 Steensgaard의 알고리즘[18]이 존재한다.

4.2 메서드 및 필드 간 경로 분석

메서드 내의 LocalInfo와 LocalInfo 사이에서 레퍼런스가 전달되는 경로를 구한 뒤에는, 메서드와 필드, 그리고 메서드와 메서드 사이에서 레퍼런스가 전달되는 경로를 파악하여야 한다. 이와 같은 경로는 다음과 같은 네 가지 경우로 나뉜다.

1. 필드로부터 FieldIn으로 레퍼런스가 유입되는 경우
2. FieldOut으로부터 필드로 레퍼런스가 전달되는 경우
3. 한 메서드의 invokeParamOut으로부터 다른 메서드의 methodParamIn으로 레퍼런스가 전달되는 경우
4. 한 메서드의 ReturnOut으로부터 다른 메서드의 InvokeIn으로 레퍼런스가 전달되는 경우

프로시저 간 분석을 통해 위와 같은 경우를 구하게 되면 LocalInfo 및 필드로 이루어진 그래프가 만들어지게 된다.

4.3 경로수집 및 요약

앞의 과정을 통해 만들어진 그래프로부터 레퍼런스 흐름을 수집하게 된다. 그래프에서 사이클을 제거한 뒤, 그래프의 노드 중 Creation을 레퍼런스 흐름의 시작 지점으로, Call을 목표 지점으로 하는 모든 경로를 수집하면, 해당 경로가 곧 레퍼런스 흐름이 된다. 이 때 수집된 레퍼런스 흐름은 지역 변수와 필드를 노드의 단위로 하기 때문에 그 입도(granularity)가 매우 작다고 할 수 있다. 따라서 경우에 따라 수집된 레퍼런스 흐름을 메서드와 필드 단위로 요약하여 추상화할 필요가 있다.

표 1. 상태 패턴 인스턴스의 확장 지점
Table 1. Extension points of state pattern instances

인터페이스 타입	org.jhotdraw.framework.DrawingView
확장 지점	org.jhotdraw.contrib.MDIDesktopPane.internalFrameClosed org.jhotdraw.contrib.DesktopEventService.componentRemoved
인터페이스 타입	org.jhotdraw.framework.Tool
확장 지점	org.jhotdraw.standard.SelectionTool.mouseDown org.jhotdraw.application.DrawApplication.toolDone org.jhotdraw.application.DrawApplication.paletteUserSelected org.jhotdraw.application.DrawApplication.open org.jhotdraw.standard.SelectionTool.mouseUp

5. 구현 및 사례 연구

앞서 제시한 분석 방법을 자바 소프트웨어에 적용하기 위하여 DARE(Design Pattern Analysis based on Reference Flow)라는 도구를 구현하였다. DARE는 soot[19]라는 자바 정적 분석 도구를 이용하여, 프로그램의 모든 인터페이스 타입에 대해 인터페이스 타입의 하위 타입에 해당하는 레퍼런스의 흐름을 추출해 내며, 이를 기반으로 디자인 패턴을 분석한다.

JHotDraw6.0beta1에 대해 DARE를 적용하여 디자인 패턴의 확장 지점을 분석해 보았다. 분석 결과 나온 디자인 패턴의 확장 지점 중 상태 패턴의 예를 기술하면 표 1과 같다. 상태 패턴과 연관된 각각의 인터페이스 타입이 그에 해당하는 확장 지점들을 가지는 것을 볼 수 있다.

상태 패턴의 경우, 패턴 인스턴스의 각 부분이 그림 1이 보여주는 것과 같은 역할을 가진다고 할 수 있다. Context는 Store 및 Caller를 가지게 되며, Injector는 ConcreteState의 메서드가 될 수도 있고, 또는 패턴 인스턴스 외부에 위치할 수도 있다. Injector가 외부에 위치하는 경우에는 ConcreteState로부터 시작되는 호출이 Injector에 대한 호출로 이어지게 된다. 실제로 표 1에 나타난 확장 지점들을 살펴보면 SelectionTool을 제외한 메서드들은 State에 해당하는 인터페이스 타입의 하위 타입이 아니라는 것을 알 수 있다. 즉, 기존 연구의 방법을 통해 상태 패턴의 인스턴스를 찾아내었을 경우, ConcreteState에 해당하는 SelectionTool이 가지고 있는 확장 지점은 찾아낼 수 있지만, 그 외의 다른 확장지점들은 찾아낼 수 없다. 따라서 패턴 인스턴스 외부에 존재하는 확장 지점 역시 찾아낼 수 있다는 것이 기존 연구와의 차이점이라고 할 수 있다.

6. 결론 및 향후 과제

본 논문에서는 디자인 패턴과 연관된 레퍼런스 흐름을 통해 디자인 패턴의 확장 지점을 식별하는 방법을 제시하였다. 기존 프로그램을 정적 분석하여 레퍼런스 흐름을 얻어

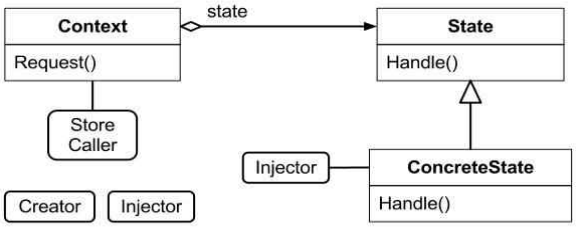


그림 1. 상태 패턴과 관련된 레퍼런스 흐름 상의 역할
Fig. 1. Reference flow roles of state pattern

내는 방법을 기술하였고, 이와 같은 분석을 자바 코드에 대해 수행하는 도구를 구현하여, 실제 소프트웨어의 확장 지점을 추출해 보았다.

논문에서 제시한 방법의 한계는 Points-to Analysis로 얻어내는 레퍼런스의 집합이 정확하지 않기 때문에, 필연적으로 실제로 존재하지 않는 레퍼런스 흐름들이 얻어지게 된다는 것이다. 이와 같이 부정확한 결과들을 걸러내어 분석의 신뢰성을 높이는 방법을 고안할 필요가 있다. 또한, 레퍼런스 흐름 분석을 통해 얻어지는 확장 지점 정보의 양이 방대할 경우, 이와 같은 정보를 파악하기 쉽게 요약하여 보여줄 수 있는 방법 역시 필요할 것이라 생각된다.

참 고 문 헌

[1] P. Tonella and G. Antoniol. Object oriented design pattern inference. Proc. IEEE International Conference on Software Maintenance, pp.230 - 238, 1999.

[2] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. T. Halkidis. Design pattern detection using similarity scoring. IEEE Transactions on Software Engineering, 32:896 - 909, Nov., 2006.

[3] J. Dong, Y. Sun, and Y. Zhao. Design pattern detection by template matching. Proc. ACM Symposium on Applied Computing, pp.765 - 769, 2008.

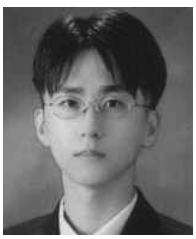
[4] J. Niere, W. Schafer, J. P. Wadsack, L. Wendehals, and J. Welsh. Towards pattern-based design recovery. Proc. 24th

International Conference on Software Engineering, pp.338 - 348, 2002.
[5] J. Niere, J. P. Wadsack, and L. Wendehals. Handling large search space in pattern-based reverse engineering. Proc. 11th IEEE International Workshop on Program Comprehension, pp.274 - 279, 2003.
[6] M. Detten and D. Travkin. An evaluation of the Reclipse tool suite based on the static analysis of JHotDraw. Technical report, Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn, 2010.
[7] Y. Gueheneuc, H. Sahraoui, and F. Zaidi. Fingerprinting design patterns. Proc. 11th Working Conference on Reverse Engineering, pp.172 - 181, 2004.
[8] Y. Gueheneuc, J. Guyomarc'H, and H. Sahraoui. Improving design-pattern identification: a new approach and an exploratory study. Software Quality Control, 18:145 - 174, Mar., 2010.
[9] R. Wuyts. Declarative reasoning about the structure of object-oriented systems. Proc. Technology of Object-Oriented Languages and Systems, pp.112 - 124, 1998.
[10] A. Blewitt, A. Bundy, and I. Stark. Automatic verification of design patterns in Java. Proc. 20th IEEE/ACM International Conference on Automated software engineering, pp.224 - 232, 2005.
[11] N. Shi and R. A. Olsson. Reverse engineering of design patterns from Java source code. Proc. 21st IEEE/ACM International Conference on Automated Software Engineering, pp.123 - 134, 2006.
[12] A. Lucia, V. Deufemia, C. Gravino, and M. Risi. Improving behavioral design pattern detection through model checking. Proc. 14th European Conference on Software Maintenance and Reengineering, pp.176 - 185, 2010.
[13] Y. Gueheneuc and G. Antoniol. Demima: A multilayered approach for design pattern identification. IEEE Transactions on Software Engineering, 34:667 - 684, 2008.
[14] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design patterns: elements of reusable objectoriented software. Addison-Wesley Longman Publishing Co., Inc., 1995.
[15] I. Bayley and H. Zhu. Formal specification of the variants and behavioural features of design patterns. Journal of Systems and Software, 83:209 - 221, Feb., 2010.
[16] V. T. Chakaravarthy. New results on the computability and complexity of points - to analysis. Proc. 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp.115 - 125, 2003.
[17] L. O. Andersen. Program analysis and specialization for the C programming language. Technical report, 1994.
[18] B. Steensgaard. Points-to analysis in almost linear time. Proc. 23rd ACM SIGPLAN SIGACT Symposium on Principles of Programming Languages, pp.32 - 41, 1996.
[19] http://www.sable.mcgill.ca/soot/

김 희 천
e-mail : hckim@knou.ac.kr
1989년 서울대학교 계산통계학과(학사)
1991년 서울대학교 전산과학과(이학석사)
1998년 서울대학교 전산과학과(이학박사)
2008년~2009년 캘리포니아 주립대학 (UCSC) 방문연구원
2004년~현 재 한국방송통신대학교 컴퓨터과학과 교수
관심분야 : 웹 공학, 소프트웨어 테스트, 디자인 패턴

박 찬 진
e-mail : chanjin.park@gmail.com
1994년 서울대학교 계산통계학과
2000년 서울대학교 전산학과(이학석사)
2006년 서울대학교 전기컴퓨터공학부 (공학박사)
2010년 서울시립대 컴퓨터과학과 연구교수
1994년~1998년 LG 소프트웨어 주임연구원
2006년~2009년 LG 전자 CTO 소프트웨어센터 책임연구원
2011년~현 재 LG 전자 TV연구소 수석연구원
관심분야 : 소프트웨어 아키텍처, Reengineering, Testing, 소프트웨어 비즈니스

김 택 수
e-mail : dolicoli@selab.snu.ac.kr
2005년 서울대학교 수리과학부(학사)
2012년 서울대학교 전기컴퓨터공학부 (공학박사)
2012년~현 재 삼성전자 소프트웨어센터 책임연구원
관심분야 : 소프트웨어 테스팅, 소프트웨어 아키텍처, 정적 분석



유 찬 우
e-mail : chanwoo.yoo@gmail.com
2003년 서울대학교 전기컴퓨터공학부/경영학부(학사)
2007년 서울대학교 전기컴퓨터공학부 (석박통합과정 수료)
2012년 서울대학교 전기컴퓨터공학부 (공학박사, 소프트웨어공학 전공)
관심분야 : 소프트웨어 아키텍처 및 디자인 패턴 분석, 메타프로그래밍

이 형 원

e-mail : lhw@gwnu.ac.kr

1987년 서울대학교 계산통계학과(학사)

1990년 서울대학교 계산통계학과 (이학석사)

1995년 서울대학교 계산통계학과 (이학박사)

1998년～1999년 메사추세츠주립대학 방문연구원

2008년～2009년 서울대학교 컴퓨터기술연구소 방문연구원

1993년～현 재 강릉원주대학교 컴퓨터공학과 교수

관심분야 : 소프트웨어 프로세스, 디자인 패턴