韓國컴퓨터情報學會 論文誌
第16卷 第11號, 2011. 11.

2011-16-11-1-7

애자일 방법에서 재사용을 지원하는 태스크 팩토링

김 지 홍*

Task Factoring to support reuse in Agile Methods

Kim-Jihong *

요 약

소프트웨어 재사용과 애자일 방법은 개발 기간의 단축 또는 잦은 요구사항의 변경을 수용하는 방법으로 각각 인식되고 있기 때문에, 이들 기술의 통합에 관한 연구와 관심이 증가하고 있다. 특히, 많은 개발현장에서 애자일 방법의 성공적인 사용 증가에도 불구하고, 애자일 방법에서 더 높은 생산성 향상을 위한 재사용 연구는 부족하였다. 본 연구는 애자일 방법에서의 새로운 유형의 재사용 자산을 식별하고 반복 개발 특성을 이용하여 재사용을 지원하는 태스크 팩토링 기술을 제안하였다. 아울러, 제안된 기술을 비디오 대여 응용에 적용하여 태스크 재사용의 프로토타이핑을 보일 수 있었다.

▸ Keyword : 소프트웨어 재사용, 애자일 방법, 태스크 팩토링, 애자일 재사용, 애자일 개발

Abstract

Since software reuse and agile development methods are seen as ways to shorten development time and accept frequent requirement changes, respectively, there has been growing interest and research on integrating these approaches. But despite the increasing number of software companies which have successfully adopted agile development methods, there has been little research on reuse in agile methods to further improve productivity. In this paper, we identify a new type of reuse asset and propose a task factoring technique by taking advantage of iteration characteristics to support reuse in agile software development methods. In addition, we can apply the proposed technique and show prototyping of task reuse in a video rental application.

▸ Keyword : Software Reuse, Agile Methods, Task Factoring, Agile Reuse, Agile Development

• 제1저자 : 김지홍
• 투고일 : 2011. 07. 18, 심사일 : 2011. 08. 11, 게재확정일 : 2011. 08. 30.
* 경원대학교 컴퓨터공학과(Dept. of Computer Engineering, Kyungwon University)
※ 이 논문은 2011년도 경원대학교 교내연구비 지원에 의한 결과임.(KWU-2011-R249)

I. 서 론

소프트웨어 재사용 기술과 애자일 방법은 개발 기간 단축 또는 생산성 향상을 제공하는 방식으로 인식되고 있기 때문에 이들의 통합을 위한 관심과 연구가 증가하고 있다[1,2,3].

소프트웨어 재사용은 이전에 만든 소프트웨어나 소프트웨어 기술을 새로운 소프트웨어 개발에 사용하여 보다 적은 비용으로 크고 복잡한 소프트웨어를 빠르게 개발하도록 지원해 주는 중요한 기술이다. 그러나 재사용은 사전에 컴포넌트나 문서화 같은 산출물의 준비와 계획이 필요하기 때문에 급변하는 불확실한 환경에서의 투자 및 투자회수에 부담을 가지고 있다.

한편, 전통적인 개발방식을 탈피하여 잦은 요구사항 변경을 수용해야하고 빠른 개발이 필요한 인터넷 시대에 등장한 애자일 방법은 재사용을 포함하는 사전 준비보다는 지금 필요한 것의 개발에 집중하여 빠른 출시를 강조하는 방식이다[4]. 이 방법은 모델이나 문서화 작업은 최소화하고 면대면 의사소통과 코딩에 집중함으로써, 생산성을 향상시키고 개발시간을 절약해준다.

소프트웨어 재사용 기술과 애자일 방법은 서로 다른 접근 방식을 갖고 있지만, 서로 보완하는 방법으로 사용될 수 있다. 그동안의 통합에 관한 연구는 소프트웨어 프로덕트 라인에 애자일 방법의 도입을 위한 전술 또는 전략 방안에 집중 되어있다[5,6,7]. 그러나 익스트림 프로그래밍과 같은 애자일 방법에서의 재사용 연구는 상대적으로 부족하였다. 본 논문은 애자일 방법에서 새로운 형태의 재사용 자산인 태스크를 식별하고 재사용 방안을 제시한다.

본 논문의 구성은 2장에서 기초 연구에 속하는 애자일 방법, 익스트림 프로그래밍 그리고 재사용에 관해 알아본다. 3장에서는 태스크 재사용과 태스크 팩토링 방안을 제안하고, 4장에서는 프로토타이핑을 보인다. 5장에서는 관련연구를 비교하고, 6장에서 논의 및 결론을 기술한다.

II. 관련 연구

1. 애자일 방법과 익스트림 프로그래밍

애자일 방법은 구조화되고 정형화된 전통적 개발방법의 경직성을 개선하여 고객의 요구사항 변화에 민첩하게 대응하면서 고 품질의 소프트웨어를 빠르게 제공하기 위해 등장하였다[6,8]. 이는 애자일 원칙에 동의하는 여러 방법을 통칭하는 용어로서 익스트림 프로그래밍(eXtreme Programming : XP), 스크럼(Scrum), 피처 중심의 개발(Feature Driven Development : FDD) 등이 있다. 이들 가운데 XP가 가장 많이 알려진 방법이며 최근에는 스크럼에 관한 관심이 집중되고 있다[9,10]. 애자일 방법은 1) 상황에 적응적인(adaptive) 방법에 초점, 2) 역할이 아니고 사람에 초점, 3) 자기 적응적인 프로세스 지원의 3가지 사항을 핵심적으로 공유한다[11]. 아울러 다음과 같은 특징을 갖는다[4].

- 짧은 릴리즈와 이터레이션
- 점증적 설계
- 사용자 참여
- 최소의 문서화
- 변경을 수용

특히, 릴리즈(release)를 통하여 고객에게 제품이 배포되고, 각 릴리즈는 여러 이터레이션(iteration)을 통하여 개발하는 방식을 갖는다. 애자일 방법에는 몇 가지 장점과 잠재적인 문제점이 있다[12]. 장점으로는, 변경을 다루는데 매우 유연하고 효율적이며 팀 상호작용을 강조하여 관련 커뮤니티의 가치를 반영하기 때문에 인기가 있다. 또한, 자주 인도되는 산출물은 프로젝트를 지속적으로 검증하여 위험을 줄인다. 그러나 잠재적인 문제도 있다. 예를 들어, 팀 구성원에게는 높은 수준의 기술적 및 대인 관계의 노련함이 필요하다. 최소의 문서화 특징에 따라 발생할 수 있는 문서의 부족은 위험 요소를 불러드릴 수 있다. 사용자 요구 사항은 프로젝트 기간 동안 계속 진화될 수 있기 때문에, 전체적 프로젝트 범위가 크게 변경될 수 있다. 따라서 애자일 방법은 모든 프로젝트를 대상으로 하기보다는 다음 사항들이 있을 때 추천된다[11].

- 예측 불가능하거나 역동적인 요구사항들
- 책임감 있고 의욕이 많은 개발자들
- 이해력과 참여성이 높은 고객들

즉, 애자일 방법은 프로젝트 관련자의 적극적 참여와 집중된 노력을 통해 빠른 개발을 지원하는 장점을 갖지만, 장기적인 관점에서의 인프라 준비와 재사용에 대한 배려는 부족하다.

한편, XP는 고객의 만족과 팀웍을 강조하며 의사소통, 단순함, 피드백, 용기를 핵심적 가치로 하여 프로젝트 활동과 개발에 기본으로 삼고있다. 이 방법은 프로그래밍 중심의 방법으로 비교적 소규모의 개발에 적용하기 쉽고, 구체적인 실천지침(practice)으로 정의되어있다[8,13]. 대표적인 프랙티스에는 점증적 계획 수립, 소규모 릴리즈, 단순 설계, 테스트 주도 개발, 리팩토링, 짝 프로그래밍, 지속적인 통합, 현장의 고객 등이 있다.

2. 사용자 스토리와 태스크 현황판

사용자 스토리는 그림 1과 같이 애자일 또는 XP 프로젝트에서 소프트웨어 사용자나 구매자에게 가치를 줄 수 있는 요구사항을 짧은 문장으로 작성한 것이며, 필요에 따라 우선순위, 난이도, 추정치, 태스크, 테스트를 추가적으로 표현한다[10,14].

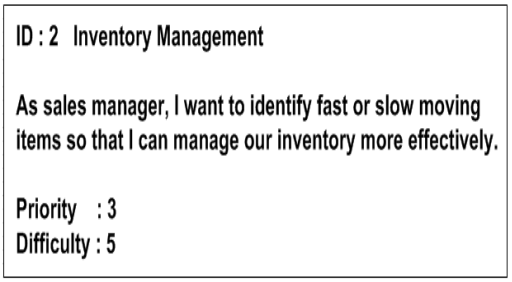

그림 1. 사용자 스토리 견본
Fig. 1. Sample User story

이는 프로젝트 작업양의 추정, 릴리즈 및 이터레이션 계획 수립, 분석, 설계, 코딩 및 테스트와 같은 활동에 기본적으로 사용되는 중요 산출물이다. 스토리는 소프트웨어 프로젝트를 위해 사용자가 작성하며 실질 개발 작업은 여러 개의 작은 태스크(task)로 나누어 수행된다.

사용자 스토리나 태스크는 인덱스 카드나, 메모지에 작성되어 벽면이나 현황판에 부착하여 사용한다. 그림 2의 현황판은 프로젝트에 필요한 스토리와 태스크의 이름과 상황(대기/진행/완료)의 빠른 파악을 용이하게 한다[15,16].

그림 2. 태스크 현황판
Fig. 2. Task board

3. 애자일 이터레이션

애자일 개발에서 실질적 개발 작업은 1-4주 기간의 고정된 타임박스를 갖는 이터레이션에서 수행된다[17,18]. XP의 경우, 이터레이션은 그림 3과 같이 이터레이션 계획, 개발, 신 버전의 산출물 생성으로 이루어진다[19,20].

그림 3. XP의 이터레이션 사이클
Fig. 3. XP iteration cycle

이터레이션 계획에서는 그림 4와 같이 우선순위가 높은 사용자 스토리를 선택하고, 스토리 개발에 필요한 태스크로 분해하고, 태스크마다 개발자 한명이 책임을 서약하고, 개발자는 각자의 작업량을 추정한다.

그림 4. 이터레이션 계획 활동
Fig. 4. Iteration planning activity

4. 소프트웨어 재사용

소프트웨어 재사용(software reuse)이란 소프트웨어 개발에 사용되거나 생성된 각종 경험 및 결과물을 새로운 소프트웨어 개발에 사용하는 것을 말한다[21]. 이전 프로젝트에서 개발된 다양한 산출물과 각종 패턴들을 새로운 소프트웨어 개발에 재사용할 수 있다. 소프트웨어 개발 단계에서 재사용 가능한 자산은 표1과 같다[8,22,23].

표 1. 재사용 가능한 자산
Table 1. Reusable asset

단계	재사용 자산
요구	요구 명세서, 유스케이스, 요구 패턴
분석	분석 명세서, 분석 패턴
설계	설계 명세서, 프레임워크, 설계 패턴, 아키텍처
구현	원시 코드, 컴포넌트, 구현 패턴
테스트	테스트 명세서, 테스트 패턴

재사용은 소프트웨어의 생산성 향상, 소프트웨어 개발 시간 단축, 소프트웨어 개발 및 유지보수 비용 절약의 혜택을 제공한다.

새로운 재사용 접근방식 가운데, 소프트웨어 프로덕트라인 공학(software product line engineering)은 제품군(product family)에 속하는 제품들의 공통점과 가변점을 사

전에 식별하고 준비하여 새로운 응용이 필요할 때 재사용하는 대규모 재사용을 지원하는 방법으로 그림 5와 같이 도메인 공학과 어플리케이션 공학으로 구성된다[24]. 도메인 공학(Domain Engineering)은 도메인 요구공학, 설계, 실현, 테스트 단계를 통하여 유사한 시스템들의 공통점과 가변점을 기반으로 요구사항, 아키텍처, 컴포넌트, 테스트 산출물을 재사용 가능한 자산 형태로 생성한다. 어플리케이션 공학(Application Engineering)은 새로운 응용을 개발하기위하여 도메인 공학에서 개발한 여러 형태의 자산을 재사용하여 새로운 응용을 개발한다. 소프트웨어 프로덕트라인 공학은 높은 재사용성을 제공하지만 장기간의 여러 단계를 거치는 사전 계획과 작업이 필요하다. 특히, 오늘날의 시장 및 기술 환경 변화를 능동적으로 수용하며 빠르게 소프트웨어 자산 생성과 재사용이 더욱 요구되고 있다.

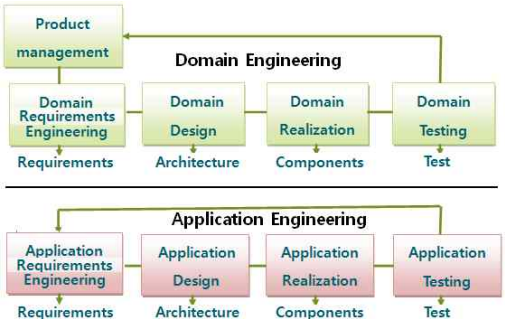

그림 5. 소프트웨어 프로덕트라인 공학 프로세스
Fig. 5. Software Product Line Engineering Process

소프트웨어 재사용의 분류 가운데 기회적 재사용(opportunistic reuse)과 체계적 재사용(systematic reuse)이 있다[22]. 전자는 사전에 재사용을 위한 별도의 노력을 절약하고 새로운 개발 때에는 우연적(accidental)으로 시도하는 재사용을 말한다. 이 방식은 지금 꼭 필요한 업무에만 집중하기 때문에 빠른 개발의 장점을 갖지만 추후의 재사용이 어렵다. 반면에, 후자는 미리 재사용 가능한 산출물을 계획하고 체계적으로 만들어 새로운 개발에 활용하는 의도적(deliberate) 재사용을 말한다. 이는 문서화와 테스트가 잘 준비되어 있기 때문에 재사용하기 쉽고 안정적인 장점이 있다. 그러나 장기간의 계획 및 투자가 필요하며 빠른 변화에 대한 대응의 어려움이 있다. 일반적으로 애자일 방법에서는 기회적 재사용이 많이 추구되고, 소프트웨어 프로덕트라인 공학에서는 체계적 재사용이 채택되고 있지만 이들의 장단점을 보완하는 방식이 기대되고 있다.

III. 재사용을 지원하는 태스크 팩토링

1. 태스크 재사용을 지원하는 개발

애자일 방법의 장점을 유지하며 새로운 형태의 재사용을 지원하는 개발 방안을 제시한다.

1.1 태스크 재사용과 자산

태스크는 사용자 스토리의 구현에 필요한 작업으로, 개발 초기에 결정되고 수행되며 이를 위한 각종 노력과 산출물은 중요한 재사용 자산이다. 태스크 재사용이란 애자일 방법의 이터레이션에서 완료된 태스크를 동일 또는 유사한 새로운 작업에 다시 사용하여 빠른 개발과 비용 절감을 이끄는 방법이다. 부수적으로, 이는 제품 내 동일한 처리 및 사용을 촉진하여 제품의 사용과 유지보수를 용이하게 한다.

한편, 애자일 개발에서 재사용 가능한 태스크 자산은 크게 표 2와 같이 3가지 종류가 있다.

표 2. 재사용 가능한 태스크 자산
Table 2. Reusable task asset

종류	설명
태스크 목록	스토리를 위한 태스크 목록의 재사용
태스크 카드	특정 태스크 카드의 재사용
태스크 산출물	산출물(테스트, 코드, 설계)의 재사용

태스크 목록은 하나의 프로젝트에 요구된 스토리 개발에 필요한 태스크들의 이름을 모아 놓은 리스트이다. 이는 전체적인 태스크 현황 파악과 재사용성 있는 태스크를 빠르게 찾을 수 있는 자산이다. 애자일 개발에서 태스크 목록은 주로 태스크 또는 스토리 현황판, 태스크 카드 모음, 전자 파일이 사용된다.

태스크 카드는 특정 스토리 개발에 필요한 작업을 간단하게 설명한 카드이다. 목록보다 조금 더 자세히 표현된 형태로서 작업의 이해와 재사용성 여부를 빨리 확인하는 데 사용 가능한 자산이다. 애자일 개발에서는 태스크 카드 또는 스토리 카드를 확장하여 사용한다.

태스크 산출물은 특정 태스크를 수행한 결과물이다. 이는 산출물의 종류에 따라 코드의 재사용, 설계의 재사용 또는 테스트의 재사용이 가능한 자산이다.

1.2 태스크 팩토링

태스크는 개발자의 현 개발 프로젝트에 대한 문제와 가용

자원에 대한 인식 그리고 해결방식이 종합적으로 고려되어 결정된 작업이다. 이러한 인식과 접근은 하나의 프로젝트 개발 기간 동안 계속되기 때문에 같거나 비슷한 태스크들이 자주 발생된다. 태스크 팩토링이란 각 이터레이션에 수행될 새로운 작업을 결정할 때, 이전 태스크와 동일 또는 유사성 유무를 간단하게 발견하여 중복 작업의 단축이 가능한 태스크를 결정하는 활동 및 기술이다.

2. 태스크 재사용을 위한 고객화 활동

프로젝트의 제약과 개발 조직의 특성을 반영하여 3.3절의 태스크 팩토링 절차를 효과적으로 수행하도록, 고객화(customizing) 가능한 민첩한 활동들이 지원된다.

2.1 태스크 이름 규칙

이름의 규칙은 태스크의 일차적 식별을 단순화 시켜준다. 애자일팀은 프로젝트 초기에 프로젝트 특성에 맞는 키워드를 사용하여 표 3과 같은 태스크 이름의 규칙을 갖는다.

표 3. 태스크 이름 규칙
Table 3. Task name rule

```
<task_name> := <action> <object>
<action>    := /* registered verb   */
<object>    := /* registered object */
```

기본적으로, 태스크의 이름은 <action>과 <object>가 함께 사용되는 단순한 규칙을 갖는다. <action>은 행동 또는 명령을, <object>는 대상을 나타내는 예약어 또는 등록어가 사용된다. 아울러, 새로운 도메인에 대한 지원이 필요한 경우에는 규칙의 변경이 가능하다.

2.2 단순 태스크형(type) 지원

태스크의 빠른 분류와 간단한 재사용 처리를 위해 형이 지원된다. 태스크가 다루는 작업 대상의 영역을 시스템 내부, 외부, 그리고 중간의 관점으로 나누고 표 4와 같은 형을 지원한다.

- 인터페이스형 : 개발 영역 외부에 있는 사용자 또는 다른 시스템과의 인터페이스를 위한 태스크로서 개발하려는 시스템과 외부 엔티티와의 연결에 관련된 작업들을 인터페이스형 이라 한다. 이 타입은 주로 사용자 인터페이스와 같은 화면 관련 작업이 주로 해당된다.

표 4. 태스크 유형
Table 4. Task type

종류	설명
인터페이스형	인터페이스 관련 태스크
처리형	특정 도메인 객체 관련 태스크
데이터형	외부 데이터 객체 관련 태스크

- 처리형 : 특정 비즈니스 업무의 처리를 위한 태스크로서 도메인 고유 업무에 관련된 작업들을 처리형 이라 한다. 특정 비즈니스 처리를 위한 계산, 가공 또는 판단 작업들이 처리형에 해당된다.
- 데이터형 : 시스템 외부에 있는 데이터베이스 또는 저장소에 관련된 작업을 지원하는 형이다. 데이터의 읽기, 쓰기, 갱신 등에 관련된 작업이 데이터형에 해당된다.

2.3 빠른 공통성 검사

공통점의 빠른 초기 식별을 위해 태스크 목록과 후보 태스크 이름에 나타난 행동과 대상을 기반으로 하는 다음과 같은 검사가 있다.

- 동일 대상 : 이전에 수행한 태스크와 현재 처리 중인 후보 태스크 이름의 규칙 가운데 대상(object)이 동일한지를 검사한다. 같은 객체나, 데이터베이스, 테이블, 모듈을 대상으로 한 태스크들이 해당된다.
- 동일 행동 : 이전 태스크와 후보 태스크 이름 가운데 행동(action)이 동일한지를 검사한다. 태스크들이 다루는 대상은 서로 다르지만 접근 또는 처리 방식이 같은 태스크들이 해당된다. 예를 들면, 도서 대여와 비디오 대여의 경우 대상은 다르지만 대여라는 공통적 방식을 갖는다.
- 동일 작업 : 이전의 스토리를 위해 개발된 태스크이지만 다른 스토리에도 똑같은 작업이 요구되는 행동과 대상이 모두 동일한 태스크인지를 검사한다. 로그인 또는 자주 사용되는 사용자 화면에 관련된 횡단적 기능들의 태스크가 해당된다.

2.4 태스크 할당 메소드

후보 태스크의 재사용성 유무와 크기의 판단에 따라 3가지의 고객화된 태스크 할당 메소드가 지원된다.

- Same() 메소드 : 이전 이터레이션에 수행되었던 태스크와 동일한 작업 요청이 식별되는 경우 수행되는 메소드이다. Same()은 새로운 태스크를 생성하지 않고 이

전에 수행되었던 태스크의 사용을 통하여 가장 높은 재사용을 촉진한다.

- New() 메소드 : 현재 처리 중에 있는 후보 태스크를 위해 재사용 가능한 태스크가 없을 경우, 새로운 작업을 할당하는 메소드이다. 이 메소드는 유사한 태스크이지만 재사용 비용이 높은 경우에도 적용된다.
- Similar() 메소드 : 현 후보 태스크와 이전의 태스크가 명백하게 비슷한 경우의 작업을 위하여 태스크를 생성하는 메소드이다. 이는 새로운 태스크이지만 당장 또는 후속 작업에서의 재사용이 가능한 태스크를 할당한다. Same()보다는 재사용성은 작지만 new() 보다는 큰 크기의 재사용성을 갖는다.

3. 태스크 팩토링 절차

재사용을 지원하는 태스크 팩토링 절차는 그림 6과 같이 제시-발견-결정의 순환적인 스텝으로 이루어지며 개발자 모두가 참가하는 팀원 회의에서 수행된다.

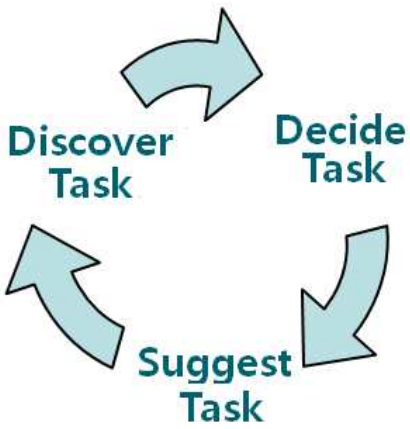

그림 6. 태스크 팩토링 스텝
Fig. 6. Task factoring step

태스크 후보 제시(Suggest)는 요구되는 새로운 스토리를 충분히 이해한 후에, 구현에 필요한 일련의 후보 태스크를 제시하는 활동이다. 팀원은 돌아가면서 개발환경의 자원과 제약사항 및 경험을 고려하여 작업의 후보 이름들을 제시한다. 태스크 발견(Discover)은 제시된 후보 태스크와 동일하거나 비슷한 기존의 태스크가 있는지를 발견하는 활동이다. 태스크 결정(Decide)은 제시한 후보 태스크에 대한 재사용성을 검토하고 최종 태스크를 결정하는 활동이다.

순환적 스텝을 바탕으로 하는 기본적인 태스크 팩토링 알고리즘은 다음 같다.

```
while (more userStory)
{
  get_userStory();
  while (need more task factoring) {
    perform suggest_task(userStory,tempTask);
    perform discover_task(tempTask, oldTask, task_type);
    perform decide_task(task_type, tempTask, newTask);
  }
}
```

하나의 이터레이션에는 기본 스텝인 suggest_task(), discover_task(), decide_task()가 반복적으로 수행된다. 먼저, get_userStory()를 통해 이터레이션에서 처리할 사용자 스토리(userStory)를 선택한다. Suggest_task()에서는 스토리 구현에 필요한 후보 태스크를 브레인스토밍 하여 <action>과 <object>로 이루어지는 단순한 규칙의 이름을 갖는 후보 태스크(tempTask)를 제시한다.

이어서, discover_task()에서는 태스크 유형(task_type)을 식별하고, 기존 태스크(oldTask)와 제시된 후보 태스크(tempTask)의 공통성과 유사성을 검사한다. Decide_task()에서는 태스크 유형과 후보 태스크를 기반으로 최종 태스크(newTask)를 결정한다.

세부적으로, 태스크 팩토링 기본 알고리즘에는 표 5와 같이 3.2절에서 소개된 고객화 활동들이 지원된다.

표 5. 태스크 팩토링 알고리즘(세부 활동)
Table 5. Task factoring algorithm (detailed activity)

주요 스텝	세부 활동
1. suggest_task() 태스크 후보 제시	1. 브레인스토밍 2. 단순 규칙의 태스크 이름 제시
2. discover_task() 태스크 발견	1. 태스크형 검사 2. 공통성 검사
3. decide_task() 태스크 결정	1. 태스크 생성 2. 태스크 카드 작성

애자일 방법에서 태스크의 재사용을 위하여, XP의 이터레이션을 확장한 태스크 팩토링은 그림 7과 같다.

우선순위의 사용자 스토리를 선택한 후, 태스크로 나누고, 본 논문에서 제안하는 태스크 팩토링이 수행된다. 이어서, 서약과 추정이 뒤따른다.

IV. 적용과 프로토타이핑

본 장은 3장에서 제안한 태스크 팩토링 알고리즘을 비디오 대여 응용에 적용하는 프로토타이핑을 보인다.

• 태스크 팩토링과 Same() 메소드의 적용

비디오 대여 응용에서의 태스크 재사용을 위하여, Iteration 1을 마치고 Iteration 2에 수행되는 Story 4(Print BackOrder)에 대한 태스크 팩토링과 Same() 메소드의 적용을 보인다.

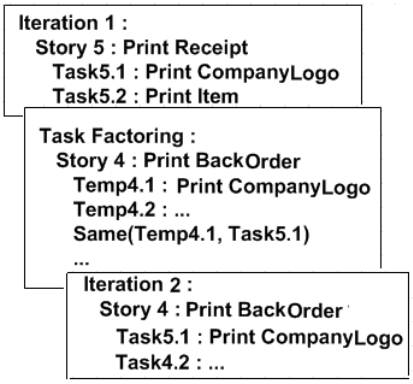

그림 8. 태스크 팩토링(Same() 메소드)
Fig. 8. Task factoring(Same() method)

그림 8의 상위 부분은 이미 수행된 여러 스토리 가운데 Iteration 1에 수행되었던 Story 5(Print Receipt)와 2개의 태스크를 보이고 있다. 그림 중간 부분은 태스크 팩토링 절차를 보이고 있다. 첫 번째 스텝인 태스크 후보 제시 스텝에서는 Story 4에 대한 브레인스토밍 후에, <action>과 <object> 이름의 규칙을 갖는 후보 태스크(Temp4.1)가 제시되었다. 태스크 발견 스텝에서는 Temp4.1과 Task5.1에 대한 태스크형과 공통성 검사가 수행되어 동일 작업으로 식별되었다. 마지막으로, 태스크 결정 스텝에서는 Same(Temp4.1, Task5.1)이 적용되어 기존의 Task5.1이 재사용되고 새로운 태스크는 할당되지 않는다. 태스크 팩토링의 결과는 그림 8의 하단에 나타나있다.

• 태스크 팩토링에서 Similar()와 New() 메소드 적용

Iteration 1,2,3을 끝내고 Iteration 4에 요구되는 Story 1(Report Comment)에 대한 태스크 팩토링과 나머지 2개의 메소드 적용을 보인다. 그림 9의 상단 화면은 Iteration 3에 수행되었던 Story 3(Report Cancel)와 태스크를 보이고 있다.

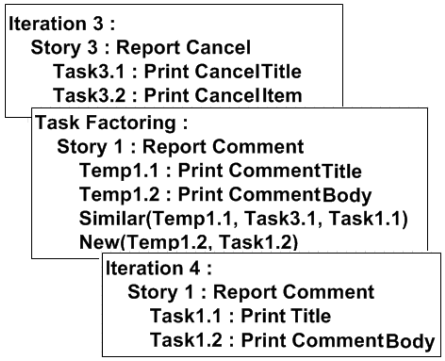

그림 9. 태스크 팩토링(Similar(), New() 메소드)
Fig. 9. Task factoring(Similar(), New() method)

태스크 후보 제시 스텝에서는 Temp1.1과 Temp1.2가 제시되었다(그림 9의 중앙 화면). 태스크 발견 스텝에 이어서, 결정 스텝에서는 Temp1.1에 명백한 유사성을 다루는 Similar(Temp1.1, Task3.1, Task1.1)와 Temp1.2에 New() 메소드가 적용되었다. 그림 9의 하단은 팩토링의 결과로서 재사용 가능한 범용적인 Task1.1(Print Title)과 새로운 Task1.2(Print CommentBody)의 할당을 보여준다.

V. 관련 연구 비교

소프트웨어 재사용 기술과 애자일 방법의 통합을 위한 접근 방식에는, 1) 애자일 방법과 재사용을 접목하려는 연구와 2) 소프트웨어 프로덕트라인 공학에 애자일 방법을 접목하려는 연구를 들 수 있다.

전자의 경우, McCarey는 재사용 기술을 통한 소프트웨어 개발에 애자일 원칙을 사용하는 애자일 재사용을 제안하였다[25,26]. 재사용의 지원이 부족한 애자일 개발환경에서 내용 기반의 필터링 기술을 사용하여 개발자에게 컴포넌트를 추천함으로써, 후 대응적으로 컴포넌트 재사용의 지원을 활성화하는 소프트웨어 개발 방식이다. Atkinson은 XP의 주요 프랙티스인 테스트 주도의 개발(Test Driven Development : TDD)을 이용하여 개발자의 코딩 활동에서 인터페이스 및 시험 정보를 추출하고 재사용 가능한 컴포넌트 제시 방안을 연구하였다[27]. 이 연구는 TDD 특성을 활용하고 있으며 주로 컴포넌트 재사용을 지원하고 있다. 반면에 본 연구는 이터레이션의 특성을 활용하는 태스크의 재사용을 연구하였다.

후자의 경우, Carbon은 재사용 중심의 어플리케이션 공학 프로세스인 PuLSE-I을 기반으로 한 연구를 통하여 프로덕트라인 공학에 애자일 원칙을 포함시키는 것이 더 쉽다고 주장

하였다[28]. 최근에, Hassen은 애자일 원칙을 준수하는 진화적 프로젝트 관리 기법인 Evo의 도입을 통한 통합 프로세스를 제시하였다[5]. Page는 FDD를 사용한 소프트웨어 프로덕트 라인의 구축 방안을 제안했다[29]. 이 연구는 FDD를 바탕으로 소프트웨어 군의 공통점을 추구한데 비하여, 본 연구는 XP 이터레이션을 바탕으로 소프트웨어 재사용을 도입하였다.

한편, 국내 관련 연구에서는 재사용과 애자일 방법의 통합은 강조되지 않고 있다. 최근에는 서비스 지향 또는 관점 지향과 같은 새로운 접근방법과 소프트웨어 프로덕트라인 공학 기술을 결합하는 연구로 확장되고 있다[30,31].

VI. 결 론

본 논문은 애자일 개발에서의 재사용을 지원하기 위하여 반복 개발 특성을 활용하는 태스크 재사용 방법을 연구하였다. 반복되는 이터레이션에서 사용자 스토리를 구현하는데 필요한 작업 결정 시, 간단하고 빠르게 중복 또는 공통점을 식별하고 재사용이 가능한 태스크를 제시하는 태스크 팩토링 방법을 제안하였다.

프로토타이핑을 통해, 태스크 팩토링을 수행하고, 태스크를 재사용하여 중복되는 개발 작업을 줄일 수 있었다. 본 기술은 요구사항이 급변하고 재사용 준비가 어려운 애자일 개발에 재사용의 혜택을 제공할 수 있다.

앞으로의 연구 과제로 태스크 팩토링 기술을 지원하는 소프트웨어 도구의 개발과 애자일 개발에서 프로덕트 라인 아키텍처 생성을 위한 태스크 팩토링 기술의 이용을 다룰 것이다.

참고문헌

[1] Díaz, J., Pérez, J., Alarcón, P. P. and Garbajosa, J., "Agile product line engineering : a systematic literature review", SP&E, Volume 41, Issue 8, pp.921 - 941, July 2011.

[2] Ivonei Freitas da Silva et al., "Agile software product lines : a systematic mapping study", SP&E, Volume 41, Issue 8, pp.899 - 920, July 2011.

[3] Yaser Ghanam and Frank Maurer, "Extreme Product Line Engineering: Managing Variability & Traceability via Executable Specification", Agile Conference, 2009.

[4] Frank Tsui, Orlando Karam, "Essentials of Software Engineering", J&B, pp.108, 2007.

[5] Geir K. Hanssen, "Agile software product line engineering: enabling factors", SP&E, Volume 41, Issue 8, pages 883 - 897, July 2011.

[6] Mohan, K., Ramesh, B., Sugumaran, V., Baruch Coll., "Integrating Software Product Line Engineering and Agile Development", IEEE Software, pp.48-55, May/June 2010.

[7] Yaser Ghanam and Frank Maurer, "Extreme Product Line Engineering - Refactoring for Variability", LNBIP, Volume 48, Part 1, pp.43-57, 2010.

[8] Sommerville, "Software Engineering", 9th ED., Pearson, pp.58-72, 2011.

[9] C Scharff, R Verma, "Scrum to Support Mobile Application Development Projects in a Just-in-time Learning Context", Proceedings International Conference on Software Engineering, pp.25-31, May 2010.

[10] In-Oh Song, Sung-Yul Rhew, Sung-Eun Lee, "A Software Process Certification Model of Small sized Software Development Using Scrum", Journal of The Korea Society of Computer and Information, Vol 16, No 4, pp.215-223, 2011.

[11] Valachich, Gorge, Hoffer, "Essentials of Systems Analysis and design", 4th ED., Pearson, pp426-427, 2009.

[12] Shelly, Rosenblatt, "Systems Analysis and Design", 8th ED., Course Technology, pp145, 513, 2010.

[13] Sang-Hyun Lee, Sang-Joon Lee, "A Study on the Values and Practices of the Extreme Programming for its Adoption", Journal of The Korea Society of Computer and Information, Vol 13, No 7, pp.269-280, 2008.

[14] Mike ohn, User Stories pplied", Addison Wesley, pp.4, 2004.

[15] Henrik Kniberg, Scrum nd P rom he trenches", InfoQ, pp.40, 2007.

[16] Dan Pilone, Russ Miles, "Head First SoftwareDev

lopment", O'Reilly, pp.170, 2007.
[17] Michele Sliger et al., The Software Project Managers Bridge to Agility", Addison Wesley, pp.42, 2008.
[18] James Shore, Shane Warden, "The Art of Agile Development", O'Reilly, pp.41, 2008.
[19] tp://www.extremeprogramming.org/map/ teration.
[20] Pankaj Jalote, "A Concise Introduction to Software Engineering", Springer, pp.30, 2008.
[21] Shari L. Pfleeger, "Software Engineering", 4th ED., Pearson, pp.627, 2010.
[22] Stephen Schach. "Object-Oriented Software Engineering", McGraw Hill, pp.216-227, 2008.
[23] Even-Andre Karlsson, "Software Reuse", Wiley, pp.357, 1995.
[24] Klaus Pohl, van der Linden F., "Software Product Line Engineering", Springer, pp.4-22, 2005.
[25] McCarey et al., "RASCAL: A Recommender Agent for Agile Reuse", Artificial Intelligence Review, Volume 24(3-4), pp.253-276, 2005.
[26] McCarey, F., Cinneide, M., Kushmerick, N., "An Eclipse Plugin to Support Agile Reuse", LNCS Volume 3556, pp.1298-1301, 2005.
[27] Oliver Hummel, Colin Atkinson, "Supporting Agile Reuse Through Extreme Harvesting", LNCS Volume 4536, pp.28-37, 2007.
[28] Ralf. Carbon, M. Lindvall, D. Muthig, "Integrating Product Line Engineering and Agile Methods", 1st International Workshop on APLE'06, 2006.
[29] Richard Paige, Xiaochen Wang, Zoë Stephenson, Philip Brooke, "Towards an Agile Process for Building Software Product Lines", LNCS Volume 4044, pp.198-199, 2006.
[30] Joonseok Park, Mikyeong Moon, Keunhyuk Yeom, "An Approach to realizing a Service with Variability for Service Oriented Applications", Journal of KIISE : Software and Applications, Vol. 38, No. 2, pp.77-85, 2011.
[31] Kwanwoo Lee, "Aspectual Implementation Patterns for Feature-Oriented Product Line Engineering", Journal of The Korea Information Processing Society, Vol 16-D, No 1, pp.93-104, 2009.

저 자 소 개

김 지 홍
1974 : 경희대학교 전자공학과 학사
1982 : California State University (Fullerton) Computer Science 석사
1995 : 경희대학교 전자계산공학과 박사
1982~89 : 미국 Interpac Software, 소프트웨어 엔지니어
89~현재 : 경원대학교 컴퓨터공학과 교수
관심분야 : 소프트웨어공학, UML, 소프트웨어 재사용, 애자일 개발
Email : wiskjh@kyungwon.ac.kr