디자인 씽킹 메카니즘과 소프트웨어공학 접목에 관한 연구

서채연*, 김장환*, 박보경*, 장우성*, 손현승*, 김영철*
*홍익대학교 소프트웨어공학연구실
{*chyun,janghwan,park,jang,sonh,bob}@selab.hongik..ac.kr

A Study on mapping Software engineering with Design thinking mechanism

ChaeYun Seo*, Janghwan Kim*, Bokyung Park*, Woo sung Jang*, Hyun Seung Son*, R. Young Chul Kim*
*SE Lab, Dept of Software and Communications Engineering, Hongik University

요 약

4차 산업혁명시대가 도래함으로써 수많은 영역에 다양한 소프트웨어(SW)가 필요할 것이며, 특히 비전공자 및 기초 전공자들의 창의적 사고 기반 SW에 대한 이해가 요구된다. 하지만 문제는 창의적 사고 기반의 SW에 대한 정의 및 아이디어가 부족하다. 우리는 교육 영역에서의 창의적 사고 방법 및 주요 쟁점들을 비전공자들에게 강의한 경험이 있지만 실질적으로 창의적 사고기법을 통해서 소프트웨어로 구현하는 것에 큰 어려움을 겪고 있다. 따라서 이러한 점을 개선하는 창의적 사고 기법과 소프트웨어 공학기법을 접목인 디자인 씽킹 메카니즘과 소프트웨어공학 접목을 제안한다.

1. 서론

4차 산업혁명시대가 도래하면서 SW가 지능형, 맞춤형으로 진화되고 있다. 지능 정보와 정보 통신 기술 융합 환경에서는 창의성, 사고력, 정보 수집, 처리, 활용 능력, 문제 해결력의 중요성이 필요하다. 창의적 인재양성은 문제 발견과 확장된 사고를 해야 하고, 논리적으로 분석하고 의미적으로 표현하기 위해 분석적 사고를 해야 한다. 창의적으로 무언가를 이루어내려면 다양한 분야의 지식과 경험을 통해 문제를 바라봐야한다. 또한 해결책을 찾기 위해 융복합하는 사고를 발휘해야 한다. 이처럼 생각이 한쪽으로 치우치지 않고 융합적으로 접근하는 사고가 디자인 씽킹이다[1].

비전공자/전공자들을 위한 창의적 문제 해결 방법으로써 디자인 씽킹 및 컴퓨팅 씽킹을 통해 문제 해결 하려는 시도가 매우 많다. 문제는 비전공자에게 코딩은 너무 어려운 개념으로 받아들여져 코딩을 어려워한다는 것이다. 창조적 생각은 실현가능할지 모르나, 코딩까지의 연결 고리를 찾지 못하고 있다는 것이다. 이 문제를 해결하고자, SW공학적 디자인 씽킹 기반 코딩 개발 체계(방안)를 제안한다. 이 방법은 SW공학 기반 소프트웨어 개발과 디자인 씽킹 메카니즘을 접목한다. 디자인 씽킹 관점의 창의적 사고 기법을 통해 비전공자/기초 전공자들이 정확한 문제인식 및 해결 능력을 배양하고 창의 융합적 문제해결이 가능한 코딩 능력 향상을 목적으로 한다.

본 논문의 2장에서는 관련연구로서 디자인 씽킹과 소프트웨어 개발을 소개한다. 3장에서는 디자인 씽킹과 소프트웨어 개발을 접목을 제안하고 소개한다. 4장에서는 결론과 함께 향후 연구 방향에 대해 서술한다.

2. 관련연구

2.1 디자인 씽킹

최근 디자인 씽킹은 현장 중심의 디자인과 인간 중심의 창의적 문제 해결을 위한 사고 방법론으로 주목받고 있다. 창의적으로 문제를 해결하기 위해 확산적 사고와 수렴적 사고를 결합하여 다양한 사고 능력을 키우는 것이 디자인 씽킹의 목표이다[1]. 디자인 씽킹은 다음의 절차를 따른다.

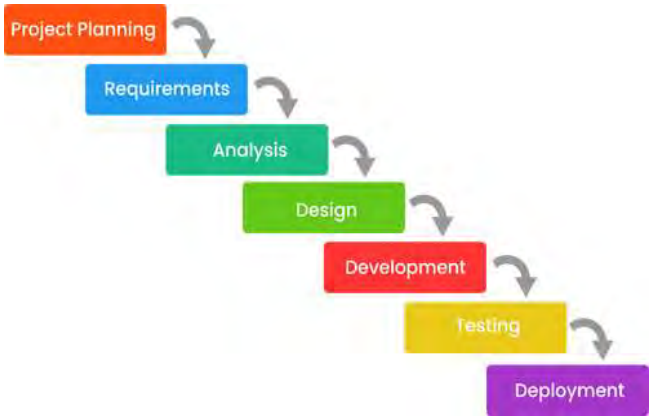
(그림 1) 디자인 씽킹 프로세스

- 공감-인터뷰, 관찰
- 문제정의-문제해석, 문제정의
- 아이디어 찾기-확산적사고/수렴적 사고,

자유로운 아이디어 발상
- 프로토타입-제품구현
- 테스트-프로토타입 테스트, 피드백, 개선

(그림 1)은 디자인 씽킹 프로세스이다. 디자인 씽킹은 공감과 정의를 기반으로 문제를 발견하고 이해하는 단계를 거쳐 창의적 아이디어 발상, 창의적 아이디어 표현을 한다. 아이디어를 제품으로 구현하고, 시제품을 테스팅 한 후, 피드백을 받고 개선시켜 나간다[1].

2.2. SW공학 개발 프로세스



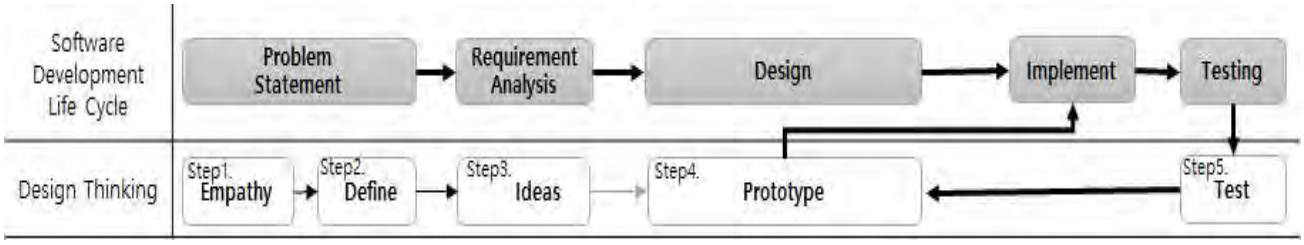
(그림 2) 소프트웨어공학 개발 프로세스

소프트웨어가 발전함에 따라 소프트웨어 개발의 규모가 점점 커지면서 보다 체계적이고 시스템적인 개발이 요구되어왔다. (그림 2)는 소프트웨어공학 개발 프로세스이다[4]. 계획단계에서는 비용, 기간 등 프로젝트 수행에 필요한 요소들을 계획하는 일을 한다. 또한 프로젝트 수행 시에 발생할 수 있는 위험을 파악한다. 요구분석 단계는 시스템 파악 및 문제점을 분석하고 사용자 인터뷰를 통해 새로운 요구사항들을 도출하여 수집한다. 설계 단계에서는 분석 단계에서 표현한 도구들을 바탕으로 시스템 및 구조를 설계하고 구현에 필요한 설계를 한다. 테스트 단계에서는 다양한 테스트 기법을 사용하여 작은 단위에서부터 큰 단위에 이르는 테스트들을 통해 소프트웨어의 품질을 높이고 오류들을 찾고 수정한다. 유지보수 단계에서는 개발이 완료된 시점으로부터 고객이 사용하면서 생기는 문제점 등을 파악하고 수정, 보완, 보강 등의 작업을 통해서 소프트웨어가 지속적으로 잘 작동되도록 돕는다.

3. 디자인 씽킹과 소프트웨어공학 개발 접목

비전공자/기초 전공자의 코딩 수업은 컴퓨팅 사고력 기반의 창의적 문제 발견 및 해결 능력을 위해 디자인 씽킹을 이용한 새로운 어플리케이션 개발 과정을 교육하고 있다. 이런 다양한 이론들은 창의적 사고를 통해 문제점을 도출하고, 아이디어를 설정하는 측면에서는 매우 유용한 방법이다. 하지만 실질적인 코딩으로 넘어가는 단계가 명확하지 않아 학생들이 코딩으로 작성하기에 많은 어려움이 있다. 또한 비전공자/기초 전공자들은 디자인에서 코드를 유추하지 못한다. 프로그래밍을 만들거나 새로운 어플리케이션을 만들기 위해서는 소프트웨어공학 기반 코딩 개발 프로세스가 필수적이라고 생각한다. 따라서 디자인 씽킹과 소프트웨어 개발 프로세스(즉, 문제 정의부터 코딩까지의 공정 단계들)를 접목한 "***소프트웨어 공학적 디자인 씽킹 기반 코딩 개발***"이 필요하다.

제안하는 프로세스는 비전공자와 기초전공자를 위한 코딩 교육 프로세스이다. (그림 3)은 디자인 씽킹 기반 소프트웨어 공학적 개발 접목이다. 이 프로세스는 소프트웨어 개발 라이플 사이클에 있는 문제정의단계와 디자인 씽킹의 공감(Step 1)과 문제정의 단계(Step 2)를 접목시켜 체계적이고 정확한 문제인식을 통해 요구사항을 분석한다. 디자인 씽킹의 '공감'을 통해 고객과의 인터뷰를 하는 것이 문제를 보다 정확하게 인식하는데 도움이 된다. 인터뷰 등의 공감기법을 통해 문제를 인식하게 되면 자연어로 구성된 Problem Statement 고객이 원하는 문제를 보다 정확하게 정의 내릴 수 있다. 다양한 문제정의 방법에 의해 문제를 더 직관적으로 이해하고 이러한 방법들을 통해서 비전공자 및 기초전공자들은 문제를 보다 정확하게 인식 할 수 있다. 그렇게 정리된



(그림 3) 디자인씽킹 기반 소프트웨어 개발 접목

문제들을 정의하면 간결화된 문제들을 해결하는데 초점을 맞추고 문제의 범주가 좁아지기 때문에 설계에 더 용이하다. 문제가 정의 되면 문제를 해결하기 위한 다양한 아이디어(Step 3)를 제시할 수 있다. 이 단계는 소프트웨어공학의 요구사항 분석단계와 접목시켜 요구사항 분석단계에서 나올 수 있는 코딩에 필요한 입출력 정보와 기능/비기능을 도출할 수 있다. 아이디어 제시 방법으로는 브레인스토밍, 마인드맵 등의 방법들이 있다. 그 중 마인드맵 기법은 소프트웨어공학기법 중 유스케이스 모델로 표현이 가능하다. 유스케이스 모델로 맵핑이 되면 유스케이스 모델로부터 클래스 다이어그램과 시퀀스 다이어그램을 그려낼 수 있다. 시퀀스 다이어그램은 문제 해결을 위해 객체를 정의하고 객체간의 상호작용 메시지 시퀀스를 시간의 흐름에 따라 나타내는 다이어그램을 말한다. 이를 통해 문제로부터 소프트웨어에서 작동하는 메서드의 흐름을 알 수 있다. 또한 클래스 다이어그램은 UML 다이어그램의 한 종류로써 시간에 따라 변하지 않는 시스템의 정적인 면을 보여주는 대표적인 UML구조 다이어그램이다. 이 방법을 통해 소프트웨어를 구성하는 클래스들 사이의 관계를 알 수 있다. 이렇게 설계된 다이어그램들을 통해서 우리는 디자인 씽킹의 프로토타입(Step 4) 기법을 이용해 완성될 소프트웨어 모습을 추상화할 수 있다. 프로토타입이 완성되면 지속적인 고객과의 협력을 바탕으로 완성체에 가까운 프로토타입을 만들 수 있다. 프로토타입이 완성되면 코딩(개발)을 통해 소프트웨어를 구현한다. 구현된 소프트웨어는 테스트를 통해 고객의 요구사항 즉, 정의된 문제들과의 비교를 통해 구현된 소프트웨어의 품질 상태를 알 수 있다. 이때, 다시 디자인 씽킹 방법의 프로토타입 테스팅(Step 5)을 통해 배포된 소프트웨어를 바로 수정하지 않고 프로토타입을 보완, 개선한다. 이렇게 개선된 프로토타입을 다시 구현함으로써 개선된 소프트웨어로 배포가 가능하다.

4. 결론

비전공자 및 기초 전공자들은 소프트웨어 공학적 디자인 씽킹 기반 코딩 개발프로세스를 통해서 창의적 문제 발견 및 해결 능력을 위한 코딩이 가능하다. 또한 창의적인 디자인과 논리적 사고 기반으로 훈련하고 체계적으로 시스템 내에서 코딩함으로써 표준화된 코딩 교육 및 코딩 개발자에게 좋은 습관 내재화가 가능하다. 코딩을 배우는 학생들에게 기초 프로그래밍의 코딩 절차 표준에 대한 성숙도를 향상시킬 수 있다. 또한 프로그래밍 교육을 체계적으로 공부하지 못했던 비전공자들에게 표준 코드 절차를 내재화시킴으로써, 코딩에 대한 두려움을 이겨낼 수 있다. 향후 연구는 소프트웨어 공학적 디자인 씽킹 기반 코딩 개발프로세스 기반의 코딩 교육 시스템을 개발할 것이다.

참고문헌

[1] 송태란, 이정현, 문제해결력을 키우는 디자인 씽킹(대한민국), 한빛미디어, 2019

[2] 소프트웨어 정책 연구소, 지능정보사회를 대비하는 SW교육 관련 정책

[3] Chae Yun Seo, So Young Moon, R. Young Chul Kim, “Open Source of Integrated Service Tools for Venture Small Business Maintenance Solutions,” The Journal of the Korea Information Processing Society (KIPS), Vol. 23, No. 6, pp. 22-32, November 2016.

[4] 김치수, 쉽게 배우는 소프트웨어 공학, 한빛아카데미, 2020.

[5] https://www.interaction-design.org/literature/article/design-thinking-getting-started-with-empathy