디자인 스튜디오를 중심으로 한 소프트웨어 설계 교육

동국대학교 ▌최 은 만*

1. 서 론

소프트웨어의 규모가 커지고 개발 기술이 빠르게 변하면서 소프트웨어 설계 작업의 중요성이 날로 높아지고 있다. 즉 소프트웨어를 구축하려 할 때 단순한 프로그래밍 기법과 컴퓨터 과학의 개념에 대한 이해와 적용만으로 좋은 소프트웨어를 만들어 내기가 어려운 실정이 되었다. 국내 대학에서 2만 여명의 소프트웨어 인력을 매년 배출하고 있지만 기업에서는 현장에 바로 투입할 인력이 부족하고 정부도 고급 소프트웨어 인력이 부족하다는 판단으로 소프트웨어 인력양성을 위하여 맞춤형 석사 과정 개설을 지원하고 있다[1].

국내에 2000년도부터 도입된 공학교육 인증제도 역시 공학 교육에서 체계적인 설계 교육의 시급성을 주장하고 있다[2]. 엔지니어가 갖추어야 할 핵심 능력은 문제 해결능력이며 다양한 지식을 종합하여 수요자가 원하는 제품이나 시스템을 설계할 수 있도록 전공 51학점 중 18학점 이상의 설계 교육을 요구하고 있다.

건축, 토목공학이나 기계공학 등 다른 공학의 경우 설계 과목의 특성과 교육 내용이 잘 드러나고 졸업 후의 진로도 설계와 시공 또는 제조로 분명히 나뉜다. 그러나 컴퓨터 공학에서 다루고 있는 소프트웨어의 경우 설계를 어떻게 그리고 무엇을 가르쳐야 하는지 아직 공통된 인식이 없다.

이 논문에서는 먼저 소프트웨어 설계 작업의 본질을 정리해 보고 소프트웨어 설계를 위하여 무엇을 가르쳐야 하고 어떤 교육 방법이 동원되어야 하는지 해외에서 논의되고 시도된 것을 소개한다. 이를 바탕으로 국내 컴퓨터공학 전공 학부 과정에서 소프트웨어 설계라는 이슈를 어떻게 접목하여 교육할 것인지 그 방향에 대하여 기술한다.

2. 소프트웨어 설계 작업의 본질

소프트웨어 설계 교육이 효과적이 되려면 설계 작업의 본질에 대한 바른 이해가 필요하다. 우선 일반적인 설계 작업의 본질에 대하여 기술하고 소프트웨어 설계 작업이 다른 설계와 어떻게 구별되는지 살펴본다.

설계는 엔지니어링 작업의 핵심이다. 일반적으로 엔지니어링 설계 작업은 목표로 하는 생산물을 만들기 전에 경제적, 기술적, 법적 등 여러 측면을 고려하여 궁리한 해결책을 말한다. 일반적인 설계 작업은 다음과 같은 특징을 가진다.

- 설계 작업은 주어진 문제를 알고 있는 이론과 기술을 동원하여 해결하는 탐구 작업이다. 즉 단순히 이론을 적용하여 제작하는 것만이 아니라 사용 환경과 요구를 만족시키기 위하여 타협점을 찾아내고 여러 대안을 저울질하여 최적의 방법을 찾아내는 과정이 필요하다.
- 설계 작업에는 모델이 중요하다. 설계자는 기술과 사용자의 중간에 서서 사용자의 요구를 엔지니어에게 전달할 수 있는 프로토타입 또는 그래픽 등으로 모델링 한다.
- 설계 작업은 품질을 중요시하는 작업이며 프로덕트에 대한 전반적인 관점이 필요하다. 프로덕트의 세부 구현보다는 전체적인 구조를 설계하며 높은 품질을 목표로 하는 작업이다. 품질은 제조 후에 추가할 수 없는 총체적인 것이기 때문이다.
- 설계는 프로덕트를 완성하기 위한 팀 작업의 일부이다. 따라서 설계 관련자와 협력하기 위한 커뮤니케이션 능력, 역할의 이해, 관점들을 잘 알아야 한다.

소프트웨어 분야의 설계는 여러 면에서 다른 엔지니어링 설계와는 다르다. 추상적 모델이 더욱 중요하며 무형의 정보나 프로그램에 대한 구조를 표현한다. 또한, 계속적으로 변경되고 발전을 관리하여야 한다는 측면에서 설계 작업이 매우 중요하다.

* 종신회원

컴퓨터 공학과 관련된 설계는 하드웨어 설계와 소프트웨어 설계, Co-design으로 나눌 수 있다. 하드웨어 설계와 Co-design은 컴포넌트가 정확히 정의되어 있고 설계를 위한 모델링 작업이 표준화 되어있으며 설계와 제조 작업이 분명히 구분되어 별도로 이루어진다. 이에 비하여 소프트웨어 분야는 설계와 제조(또는 구현) 작업의 분명한 선을 긋기가 어렵고 분리되어 작업하기가 어렵다. 그만큼 설계에 표현된 내용과 구현된 소프트웨어의 갭이 크기 때문이며 이를 엔지니어의 경험에 의존하여 극복하게 된다. 즉 소프트웨어 설계 작업은 구현 작업과 매우 밀접하며 구현 기술에 대한 자세한 경험이 없다면 교육하기 어려운 면이 있다.

소프트웨어 설계 작업의 또 다른 특징은 구현 기술에 따라 설계 모델을 표현하는 방법이 크게 달라진다는 점이다. C와 같은 프로시저 중심의 프로그래밍을 위한 설계 모델은 프로시저 구조와 프로시저에 의한 자료의 변화에 중점을 두고 있는 반면 C++, Java 등의 객체지향 프로그래밍을 위한 설계 모델은 클래스들의 정적인 관계와 동적인 인터랙션 위주로 표현하며 패턴이 중요하다. 데이터베이스 설계에는 엔티티들의 관계 표현이 핵심이며 인공지능 프로그래밍은 지식의 표현 방법, 에이전트의 행위 기술 방법이 설계 모델을 좌우한다. 또한 EJB와 같은 컴포넌트 개념을 기초로 하는 재사용 중심 설계에는 컴포넌트 자체의 설계를 표현하고 이를 이용하여 새 시스템을 표현하여야 한다.

결국 소프트웨어 설계 작업도 일반적인 설계와 같이 프로덕트를 제작하는 여러 원리들을 잘 이해하고 이를 적용하여 비전을 잘 표현할 수 있어야 하며 지속적인 설계 대안을 만들어보고 이를 평가하여 최적의 설계를 찾아내는 과정이라 할 수 있다.

3. 소프트웨어 설계 교육의 내용

고층 빌딩을 지을 때 수많은 설계도를 그리는 것처럼 대규모 소프트웨어를 개발하는 과정에도 많은 추상적 표현을 만든다. 따라서 소프트웨어를 설계할 수 있는 능력을 교육하려면 소프트웨어를 이루고 있는 각 단위들에 대한 추상적 계층과 이를 표현하는 방법을 잘 알아야 한다. 즉 프로시저 단위에서는 자료와 알고리즘의 설계, 프로시저의 추상화로부터 시스템 단위의 소프트웨어 구조 설계, 데이터베이스 개념적 스키마 설계와 논리 및 물리적 설계, 소프트웨어의 구조 설계에 이르기까지 각 층에 필요한 다양한 기술을 습득하여야 한다. 설계 교육의 내용을 컴퓨터 공학에서 다루는 학문 분야의 틀 안에서 정리하면 표 1과 같다.

이제까지 설명한 컴퓨터 공학의 틀을 벗어나 설계 교육, 특히 소프트웨어 분야의 엔지니어링 교육을 강조하려는 시도가 있다. 예를 들면 현재 국내 ICU에도 도입되어 있는 카네기멜론 대학의 소프트웨어 엔지니어링 석사 과정이 그것이다. 소프트웨어 개발은 단순히 설계 방법을 습득하여 적용함으로 이루어지는 것이 아니라 엔지니어링 전 단계에 관한 이론과 실무적 연습이 집중적으로 교육되어야 한다는 주장이다.

표 1 소프트웨어 설계 교육의 내용

설계 기술	교육 내용	특징
자료구조 설계	- 자료와 오퍼레이션의 추상적 표현 방법 - 자료구조 유형 및 그 적용 설계	모듈 내의 자료와 내부 및 외부 파일 설계
알고리즘 설계	- 알고리즘의 표현 방법 - 알고리즘의 유형과 설계 적용	모듈 내의 알고리즘 설계
소프트웨어 아키텍처 설계	- 소프트웨어 구조의 표현 방법 - 아키텍처 스타일과 그 적용	모듈 사이의 관계 및 인터페이스 설계
데이터베이스 설계	- 개념적 스키마 정의 - 논리적 및 물리적 모델의 표현 방법	효율적인 데이터베이스 설계
인공지능 시스템 설계	- 논리 프로그램의 설계 - 지식 표현 및 설계 - 지능 에이전트의 설계	논리와 지식의 다양한 표현 방법
네트워크 설계	- 네트워크 토폴로지 설계 및 시뮬레이션	네트워크 시스템의 구축
컴포넌트 설계	- 컴포넌트 설계 표현 방법 - 엔터프라이즈 애플리케이션 설계	재사용 중심의 설계
설계 패턴	- 객체지향 설계의 소규모 패턴이 이해 - 패턴을 적용한 설계	좋은 설계의 고찰
캡스톤 설계	- 설계 기법을 종합하여 적용하고 통합	종합적인 시스템 설계 및 통합 능력

표 2 SEEK(2004)의 설계 관련 교육 내용

지식 영역	지식 단위	교육 내용
모델링과 분석	모델링 기초	모델링 원리(분할, 추상화, 일반화, 프로젝션/뷰 등), 모델링 언어, 정형적 모델
	모델의 종류	정보 모델링(ER 다이어그램, 클래스 다이어그램) 행위 모델링(구조적 분석, 상태 다이어그램, 유스 케이스 분석, 인터렉션 다이어그램 등) 임베디드 시스템 모델링, 엔터프라이즈 모델링
	요구 기초	요구의 정의, 추출, 변경 및 관리, 명세와 검증
소프트웨어 설계	설계 개념	설계 원리(정보은닉, 응집력, 결합력), 설계 목표가 되는 품질(신뢰성, 사용용이성, 성능, 보안 등) 설계 Trade-Off, 설계 전략(함수 중심, 객체지향)
	아키텍처 설계	아키텍처 스타일(파이프-필터, 계층, 트랜젝션 중심, Peer-to-peer, 클라이언트-서버 등) 도메인 중심 아키텍처, 프로덕트 라인
	HCI 설계	HCI 설계 원리, 모드와 네비게이션, 코딩과 비주얼 디자인, 반응시간과 피드백, Internationalization
	상세 설계	설계 패턴, 컴포넌트 설계, 설계 표현 방법
	설계 지원도구와 평가	설계 지원도구, 설계 측정 메트릭(응집, 결합, 정보은닉)
직무 능력	그룹 다이나믹스/심리학	팀 작업 다이나믹스, 개발 관련자와의 접촉, 갈등 해결, 공동 작업 방법
	커뮤니케이션 능력	독해(코딩과 문서), 쓰기, 팀 커뮤니케이션(전화, 이메일, 회의) 프레젠테이션 능력
	직업의식	자격증, 직업윤리, 전문 학회, 표준

이런 취지에 따라 2002년까지 북미와 유럽, 호주에 32개의 소프트웨어 엔지니어링 학부 과정이 개설되어 있는데 ACM과 IEEE 공동 커리큘럼 위원회가 2004년 Software Engineering Education Knowledge라는 커리큘럼 표준 가이드를 만들었다[3].

10개의 영역, 즉 컴퓨팅 기초, 수학 및 엔지니어링 기초, 전문가 실무, 소프트웨어 모델링 및 분석, 소프트웨어 설계, 소프트웨어 V&V, 소프트웨어 진화, 소프트웨어 프로세스, 소프트웨어 품질, 소프트웨어 관리로 나뉘어 있는데 대부분 컴퓨터 공학에서 다루는 부분을 기초로 하고 있다. 특히 주목할만한 설계와 관련된 핵심 세 분야의 자세한 교육 내용은 표 2와 같다.

이제까지 소프트웨어 설계 교육의 내용 살펴보았지만 설계 교육은 컴퓨터 공학의 여러 학문 분야에 고루 퍼져 있다. 이러한 설계 교육 내용을 담을 때 어떤 커리큘럼의 그릇에 담을 것인지는 교육 현장의 상황과 환경에 따라 다르다.

4. 소프트웨어 설계 교육의 방법

앞서 언급한 설계 교육의 내용을 잘 전달하려면 어떤 교육 방법이 필요할까? 단순히 이론을 전달하고 과제를 부과하는 강의식 교육으로는 설계 기법을 충분히 교육할 수 없다. 이 장에서는 효과적인 설계 교육의 방법에 대하여 기술한다.

첫째, 설계 교육은 프로젝트 중심의 스튜디오 교육이 되어야 한다. 스튜디오 교육은 프랑스의 Ecole des Beaux-Arts에서 유래된 것으로 주로 건축 설계 교육에 사용되어 왔다. 설계 작업은 여러 가지 가능성 있는 방법들을 선택하는 의사 결정의 연속이다[4]. 따라서 어떤 것이 좋은 설계 결정인지 판단해주고 자세히 튜터링 하는 방식이 필요하다. 건축 설계나 시각 디자인, 패션 디자인에서 사용하는 스튜디오 교육은 아주 적은 수의 학생을 대상으로 실제 설계 작업과 이를 평가하고 가이드 하여 좋은 설계로 개선하는 방법을 사용하고 있다. 즉 소프트웨어 설계도 프로젝트 위주의 과제를 중심으로 피드백이 있는 교육이 되어야 한다. 이러한 의미에서 현재 NEXT 사업 안에서 진행되는 대학과 기업 간의 협업 사이트인 한이음(http://hanium.or.kr)을 통한 산학 연결 프로젝트는 설계 교육에 많은 도움이 되고 있다.

둘째, 설계 교육은 무에서 시작하는 과제를 피해야 한다. 최근의 소프트웨어 시스템은 규모가 상당히 크며 복잡한 계층으로 형성되어 있다. 따라서 기초 단위가 되는 빌딩 블록, 예를 들면 표준 라이브러리, 패턴, 빈 컴포넌트와 같은 것이 잘 준비되어 있지 않다면 교육하기가 어렵다. 따라서 본격적인 설계 교육이 이루어지기 전에 기초 과목에서 이를 잘 다루어 주어야 하고 과제 문제 해결을 위하여 자세한 부품이나 프레임워크가 제공되어야 한다.

셋째, 설계 교육에는 좋은 샘플을 보여주는 것이 좋다. 아키텍처나 패션 분야에서 보듯이 설계 작업은 이미 만들어 놓은 좋은 것에서 새롭게 창조하는 작업

이기 때문이다. 즉 좋은 설계의 모범을 보여주는 샘플이 많이 필요하다. 소프트웨어 분야에서는 설계 패턴이 이와 유사한 개념으로 객체지향의 상세 설계에 사용하는 패턴[5]부터 아키텍처 패턴[6]에 이르기까지 다양하게 소개 되어 있다. 기술력의 보호와 저작권 문제로 시스템의 전체 설계 샘플을 모으는 일은 쉽지 않은 일이지만 NEXT 사업의 산학 연결 프로젝트 중 잘된 것을 정리하여 소개한다면 좋은 샘플이 될 수 있다. 또한 좋은 사례들을 교류할 워크샵이나 설계 교육 포럼 같은 것을 운영한다면 설계 교육의 질 향상에 크기 기여할 수 있을 것으로 예상된다.

5. 소프트웨어 설계 교육이 강조된 사례

소프트웨어 분야에서 설계 이슈에 관하여 관심을 가지고 논의한 것은 그리 오래되지 않았다. 1994년에 설계 패턴 관련 책이 출간되었고 그 해에 비로소 ACM SIGCHI와 ASM(Association of Software Design)이 Interactions라는 출판물과 여러 연구 워크샵을 통하여 논의되고 있다. 대학의 교육에서 소프트웨어 설계를 체계적으로 고려하고 커리큘럼에 설치한 것도 불과 15년 전의 일이다. 이렇게 설계 교육의 역사가 짧지만 이 장에서는 3, 4장에서 설명한 좋은 설계 교육과 방법을 담고 있는 커리큘럼과 강좌를 소개한다.

설계 교육이 강조된 첫 번째 스타일은 컴퓨터 과학과 엔지니어링 두 분야의 트랙을 운영하면서 학생들이 엔지니어링이나 설계 과목에 더 많이 노출될 수 있는 기회를 주는 방법이다. 미국의 Auburn 대학이 이런 방법을 취하고 있는데 SE 트랙에서 설계를 집중적으로 교육하기 위하여 표 3과 같은 강좌를 개설하고 있다.

두 번째 스타일은 대학원 과정에서 엔지니어링 및 설계 실무를 강조하는 방법이다. 카네기멜론 대학과 같이 컴퓨터 과학의 학부 과정에서 다루는 다양한 기술 강좌를 기초로 대학원 과정에 엔지니어링 과정과 설계가 강조된 스튜디오를 운영하는 방법이다. 국내에도 ICU에 도입되어 잘 알려져 있지만 표 4와 같이 1년 3학기의 짧은 일정으로 스튜디오 위주의 교육하고 있다.

설계를 강조한 특별 트랙이나 석사 과정의 신설이 어려운 현실이라면 현재 컴퓨터 공학의 커리큘럼 안에 개설된 일부 과목에서 설계를 집중적으로 교육할 수 밖에 없다. 표 5와 같은 과목에서 설계와 관련된 이론과 프로젝트 위주의 교육을 집중적으로 실시하는 것이 바람직하다.

표 3 두개의 트랙을 운영하는 사례[7]

단위	과목	교육 내용
200	Software Construction	프로그래밍, 테스팅, 디버깅, 형상 관리 등 소프트웨어 구축과 관련된 작업의 집중 경험
300	Software Modeling and Design	소프트웨어 시스템의 모델링과 설계에 관련된 프로세스, 방법, 도구
	Embedded Systems Development	임베디드 시스템을 위한 설계와 분석, 리얼타임 이슈, 자원 관리, 스케줄링, 예외 처리, 장치 관리자 개발 등
	Wireless Software Engineering	무선 응용 소프트웨어의 개발과 관련된 명세, 설계, 테스트, 성능 평가 등
400	Senior Design Project	소프트웨어 개발의 시작과 끝을 경험하는 프로젝트 위주의 강좌. Extreme Programming 방법론
	Interface Design for Wireless Application	Wireless UI 설계, Low bandwidth 한정된 자원에서의 설계, WAP을 이용한 설계 프로젝트
	Computer Ethics	컴퓨터의 윤리 관련 토픽; 프라이버시, 자원 보호 등

표 4 대학원 과정에 설계를 강조한 사례[8]

학기	과목
Core(30%)	Methods of Software System, Methods: Deciding What to Design, Managing Software Development, Analysis of Software Artifacts, Architecture of Software System
Electives(30%)	Computer Science; OS, Computer Graphics, Distributed System, Parallel Algorithm, Advanced AI Software Engineering; Software Process, COTS-based System, Realtime Software, Software Measurement, Dependable Software, Wireless Networking, Security, Risk Management, PSP Electrical & Computer Engineering; Robotics, Telecommunications, Packet Switching
Studio(40%)	Boot camp, PSP, Requirement Documents, Statement of Work, Project Plan, Architecture Design, Detail Design, Coding, Reviews

표 5 학부 컴퓨터 공학 커리큘럼 안에서의 설계 집중 과목

단위	과목	교육 내용
200	문제 해결 중심 프로그래밍	프로그래밍, 테스팅, 디버깅 등 소프트웨어 구축과 관련된 작업을 새로운 문제 해결을 위주로 교육
300	설계 이론	모델링, 아키텍처, 설계 패턴, 설계 도구
400	프로젝트	캡스톤 설계 및 디자인 스튜디오 성격의 종합 프로젝트

6. 디자인 스튜디오의 운영 방안

디자인 스튜디오는 현장에서 일어나는 복잡한 문제와 다수의 클라이언트, 모호한 요구 등을 다루기 위한 교육 방법이다. 건축 설계나 그래픽 설계, 패션 디자인 등에서도 스튜디오 스타일의 강좌를 운영하여 예비 전문가들의 창의성과 기술 통합력을 교육하고 있다. 일반적인 스튜디오 강좌는 프로젝트 문제에 대하여 아이디어를 내고, 논의하고 직접 해보고, 이를 비평하는 작업들로 이루어져 있다. 그야말로 예비 전문가들이 훈련받는 공동으로 일하는 작업장이 되어야 한다.

소프트웨어 분야의 디자인 스튜디오도 다른 분야와 마찬가지로 요구 분석, 소프트웨어 설계, 구현까지 작업할 프로젝트 문제를 다루어야 한다. 프로젝트 문제는 수강자의 수준에 맞추어 준비하거나 문제를 던져줄 클라이언트로부터 나온다. 특히 소프트웨어 디자인 스튜디오의 문제는 최신 기술이 적용되는 실용적인 문제이어야 한다. 따라서 산학의 연결이 필수적이며 현재 대학과 기업 간의 협업 사이트인 한이음에 제안된 실용적인 문제들을 이용하는 것도 하나의 방법이 될 수 있다. 소프트웨어를 개발하는 작업은 무수한 의사교환과 의사결정, 작업 내용의 정리나 프로그래밍, 테스트의 연속이다. 따라서 서로 호흡을 맞추는 것이 중요하다. 미팅 방법, 문서 작성 방법, 코딩 스타일, 사용하는 도구와 프로그래밍 언어, 설계 표현 방법 등 서로 맞추어야 하는데 이런 것들을 오리엔테이션 세션에서 다루어야 한다. 카네기멜론 대학의 MCSE 과정에서는 이를 Boot camp라 하여 신병 교육이라 부르고 이런 것들을 소개한다.

디자인 스튜디오 과목의 더욱 구체적인 운영 방안과 이런 교육을 위한 지원하기 위하여 필요한 사항을 표 6에 기술하였다. 이 중에서 프로젝트 팀을 조직하고 협동하여 참여를 유도하기 위한 방안이 매우 중요하다. 기업의 현장에서도 동료 간의 협력이 잘 안되어 일이 잘 진행되지 않는 경우가 많지만 팀 작업을 처음 하는 학생들은 대부분 소극적으로 참여하며 서로의 이해가 부족하다. 따라서 팀원의 성향을 잘 이해하고 서로 협동, 보완할 수 있는 워크샵이 필요하다.

건축 디자인이나 패션 디자인 계통의 디자인 스튜디오 강의실은 그야말로 학생이 작업하는 작업장이다. 건축 디자인 스튜디오 과목의 강의실에는 드로잉 데스크가 있는 것처럼 소프트웨어 설계 과목의 강의도 UML 등 각종 표현 방법으로 설계할 수 있는 도구가 있는 컴퓨터가 있어야 한다. 또한 공동의 작업을 할 수 있도록 작은 회의 공간은 필수적이다.

디자인 스튜디오 과목은 프로젝트의 규모에 따라서 한 학기 이상으로 확대할 수도 있으나 일반적인 스케줄은 계획, 설계, 구현, 발표와 정리 단계로 나누어 진행하는 것이 바람직하다. 표 7에 있는 것이 예로 든 디자인 스튜디오 강의의 상세한 스케줄과 작업 결과물이다. 예시한 스케줄은 구현보다는 설계 작업을 강조한 케이스이다. 하지만 한 학기로 운영하는 경우 구현에 할애할 시간이 많지 않으므로 불필요한 문서 작업은 최소화 하는 것이 좋다. 구현에 집중하기 위해서는 설계 작업을 줄이고 반복되는 작업 과정을 거쳐 결과를 완성하는 Extreme Programming[13] 같은 방식을 채택하는 것도 고려해볼만 하다. Extreme Pro-

표 6 디자인 스튜디오 과목의 운영 방안과 필요한 지원과 프로그램

운영 방안	지원	프로그램
작업 중심 교육	팀별 작업장과 미팅룸	Extreme Programming
결과의 발표와 비평	수강자가 모일 강의실 정기적인 멘토와의 만남	Studio critic
협동과 참여	Team Building을 위한 프로그램	MBTI 성격 테스트 미팅 방법 워크샵
오리엔테이션	작업 과정, 문서 작성법, 코딩 스타일 등이 소개된 프로젝트 핸드북	Boot camp
프로젝트 위주의 교육	실용적 문제를 던져주고 요구를 제시할 클라이언트	클라이언트 미팅
집중 교육	강의 총괄, 멘토 등의 팀티칭 다량의 학점 인정 합숙교육	시간 계획과 측정 (Personal Software Process)

표 7 디자인 스튜디오 과목의 진행 일정(예시)

단계	작업	결과물	스튜디오 프로그램	첫 달				둘째 달				셋째 달				넷째 달			
				1주	2주	3주	4주	5주	6주	7주	8주	9주	10주	11주	12주	13주	14주	15주	16주
계획	팀 구성		MBTI 테스트																
	Boot Camp	프로젝트 핸드북	협동 워크샵																
	클라이언트 배정		클라이언트 미팅																
	Propose	Proposal																	
	계획	계획서	계획 발표																
설계	요구의 도출																		
	요구분석	분석서																	
	작업 범위 규정																		
	구조 설계																		
	디테일 설계	설계서	설계 발표																
구현과 시험	구현 분담																		
	코딩과 단위 테스트																		
	통합과 빌드																		
	시스템 테스트	코드와 테스트 보고서																	
정리	최종 보고 및 정리	프로젝트 결과 보고서																	
	데모		데모 및 결과 발표																

gramming 방식은 요구 사항을 짧은 스토리 위주로 정리하여 2주 단위의 설계, 구현, 테스트 과정을 반복하는 작업 프로세스를 따른다. 따라서 매우 구현 중심적이며 조기에 눈에 보이는 결과를 만들어 낼 수 있다. 작업 프로세스의 유형은 모두 장단점이 있으므로 수강생들로 하여금 작업 프로세스를 비교하여 선택하도록 하는 방법도 좋다. 스튜디오 프로젝트를 진행하면서 진도 관리와 멘토링은 매우 중요하다. 따라서 미팅을 정례화 하고 기록을 잘 남겨 팀 전체의 이슈를 해결해 나가야 한다. 팀 전체의 참여를 유도하기 위하여 팀 사기 진작을 위한 프로그램이 필요하며 문제가 있을 때 튜터가 적극 개입하도록 계속 모니터링 할 필요가 있다. 따라서 한 명의 튜터가 여러 팀을 맡는 것은 충분한 교육성과를 거두기 어렵다.

7. 결 론

소프트웨어 시스템의 규모가 커지고 사용자 인터페이스가 강조될수록 소프트웨어 분야에서 설계가 차지하는 비중은 커질 것이다. 이제까지 살펴본 것과 같이 설계 능력은 되풀이되는 연습과 멘토에 의한 피드백으로 습득될 수 있다. 따라서 설계 교육이 잘 이루어지려면 질을 추구하여야 하며 소규모 클래스가 필수적이다. 컴퓨터 공학이 다루는 전반적인 분야에 설계의 의미를 새롭게 정의하여 각 과목에 접목시킬 수 있고 필요하다면 설계를 강조하여 교육할 수 있는 여러 가지 방법을 소개하였다.

소프트웨어 분야에서의 설계는 아직 건축 설계사처럼 구축과 설계가 별도의 작업으로 분리되지 않고 있다. 그만큼 구현을 위한 복잡한 구조를 독립적으로 다루기가 쉽지 않기 때문이다. 그러나 품질 좋고 사용자가 만족하는 소프트웨어가 되어 산업의 큰 성장 동력이 되려면 설계가 좋아야 하고 이를 위한 교육이 잘 이루어져야 한다.

참고문헌

[1] 소프트웨어 학과 개설지원, 전자신문, 2007년 10월 2일자.

[2] 한송엽, "체계적인 설계교육이 시급하다", 공학교육, 제 13권, 1호, pp.4-5, 2005.

[3] SEEK 2004, Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, The Joint Task Force on Computing Curricula, IEEE Computer Society and Association for Computing Machinery, 2004.

[4] T. Winogard, Bringing Design to Software, Addison Wesley, 1996.

[5] E. Gamma, et al., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994.

[6] F. Buschmann, et al., Pattern-Oriented Software Architecture Volume 1: A System of Patterns,

Wiley, 1996.

[7] Computer Science and Software Engineering Department Homepage, Auburn University, http://eng.auburn.edu/programs/csse/index.html.

[8] Master of Software Engineering, Carnegie Mellon University Homepage, http://www.mse.cs.cmu.edu/Courses-Curriculum.html.

[9] R. Rasala, "Design Issued in Computer Science Education", ACM SIGCSE Bulletin, Vol. 29, No. 4, 1997.

[10] M. Shaw, "Software Engineering Education: A Roadmap", The Future of Software Engineering, New York, NY: ACM Press, pp. 371-380, 2000.

[11] M. Skubic and J. Laffey, "Event-Driven Computing Projects for Software Engineering Education", Proceedings of the 2002 American Society for Engineering Education Annual Conference & Exposition, 2002.

[12] M. Jazayeri, "The Education of a Software Engineer", Proceedings of Automated Software Engineering Conference, Linz, Austria, pp. 22-24, 2004.

[13] K, Bexk, C. Andres, Extreme Programming Explained: Embrace Change, 2nd edition, Addison-Wesley, 2004.

최은만

1982 동국대학교 전산학과(학사)
1985 한국과학기술원 전산학과(공학석사)
1993 일리노이 공대 전산학과(공학박사)
1985~1988 한국표준연구소 연구원
1988~1989 데이콤 주임연구원
1998~2004 한국정보과학회 소프트웨어공학연구회 운영위원
2000, 2007 콜로라도 주립대 전산학과 방문교수
2002 카네기멜론대학 소프트웨어공학 과정 연수
1993~현재 동국대학교 컴퓨터공학과 교수
관심분야 : 객체지향 설계, 소프트웨어 테스트, 프로세스와 메트릭, Program Comprehension
E-mail : emchoi@dgu.ac.kr