소프트웨어 개발방법론의 패러다임 전환

양 단 희*

◈ 목 차 ◈

1. 서 론

소프트웨어공학은 소프트웨어의 품질을 향상시키고 소프트웨어 생산성과 작업 만족도를 증대시키는 것이 목적이다. 그리고 그 궁극적 목표는 최소의 비용으로 계획된 일정보다 가능한 빠른 시일 내에 소프트웨어를 개발하는 것이다[11]. 특히 개발 과정에서 개발자들이 혹사되어서는 절대 안 되며, 경력이 쌓일수록 개발자로서의 자부심과 긍지를 지닐 수 있게 해주어야 한다. 즉 프로젝트도 성공하고 개발자에게도 도움이 돼야 진정한 방법론인 것이다. 그렇지 않고 단순히 절차를 지키기 위한 방법론, 개발자를 희생시키면서 프로젝트만 어떻게든 성공시키기 위한 방법론은 근시안적인 것으로 궁극적으로 실패로 이어질 수밖에 없다[9].

그런데 개발 현장에서는 개발방법론이 무겁고 형식적이어서 소프트웨어를 개발하는 데 실질적인 도움이 되지 않으나, 고객이 요구하기 때문에 '울며 겨자 먹기' 식으로 어쩔 수 없이 적용하고, 이로 인해 개발에 필요도 없는 문서를 억지로 생산해야 하고, 개발은 더디어지는 헛삽질이 허다하다고 한다. 유명한 스타 개발자 조엘은 화성인 아키텍트(architect)가 화성에서나 가능한 계획을 지구로 가져오지 않는 것이 개발자를 진정으로 돕는 길이라고 냉소적으로 얘기했다. 상황이 이러하니 개발자들 사이에 개발방법론은 조롱거리가 된지 오래이며, 심지어 개발방법론 무용론을 주장하는 사람까지도 있다. 특히나 SI 시장의 대부분을 차지하는 웹 개발은 그 특수성 때문에 설계 자체가 필요 없다는 주장도 설득력을 얻고 있다[5].

지금까지 소프트웨어공학은 유구한 역사를 지닌 건축공학이나 전자공학과 같은 인접 공학으로부터 그 이론적 배경을 벤치마킹해 왔다. 그러나 소프트웨어 개발은 다른 공학 분야와는 분명히 차별되는 고유의 특성을 지니고 있다. 그런데 이러한 특성을 무시하고 일반 공학의 틀 속에 소프트웨어공학을 구겨 넣고자 하는 시도는 개발자들에게 엄청난 고통을 안겨다 주었다. 이제 우리는 공학적 마인드로부터 벗어나 인문학으로부터 벤치마킹을 해야 하는 패러다임의 전환기에 이른 것 같다.

모든 조직은 관료화된다. '소프트웨어를 가장 빠르고 효율적으로 개발하자'는 원래의 목적은 사라지고, 규칙 자체가 목적으로 변질된다. 그리고 그러한 규칙을 준수시키기 위해 관리자는 강압하고, 개발자는 어쩔 수 없이 시늉이라도 내야 한다. 관리자와 개발자 모두 엉뚱한 문제로 엄청난 스트레스 하에 놓이게 된다. 그래서 본고에서는 개발방법론의 2가지 대세를 살펴보고, 애자일 소프트웨어 개발 방법론(Agile Software Development)으로 대전환이 필요하다는 점을 역설하고자 한다.

* 평택대학교 컴퓨터학과 부교수

2. 개발방법론의 2가지 대세

소프트웨어 개발방법론은 70년대까지는 구조적 프로그래밍, 80년대는 구조적 개발방법론, 90년대는 객체지향 개발방법론, 2000년대는 애자일 개발방법론이 주목을 받아 왔다. 이러한 방법론의 핵심은 <그림 1>에서처럼 프로그램 '소스(source)'와 '설계'의 관계를 어떻게 설정할 것인지에 있다. 구조적 프로그래밍은 '소스'에 극단적으로 치우친, 구조적 개발방법론은 '설계'에 극단적으로 치우친, 객체지향 개발방법론은 '설계'에 빙점을 둔 방식이다. 그런데 애자일 개발방법론은 '소스'에 빙점을 두고, 인문학적인 실용주의 노선을 취하는 파격을 단행하였다.

구체적인 개발방법론으로 RUP(Rational Unified Process)와 XP(eXtreme Programing)는 요즘 가장 널리 알려진 개발방법론으로 그 외의 방법론들은 이 둘의 변형이라고 할 수 있다. 여기서 객체지향 개발방법론의 대표 주자는 RUP, 애자일 개발방법론의 대표 주자는 XP라고 할 수 있어 이에 대해 간단히 살펴보겠다.

2.1 RUP

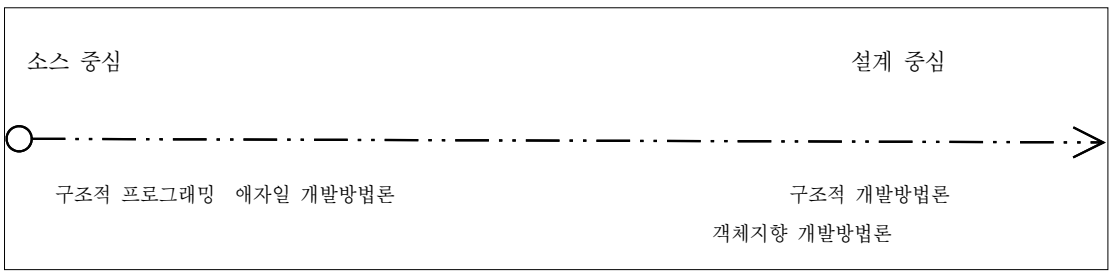
RUP의 핵심 키워드는 '설계'이다. 이것은 반복적이며 관점적(perspective)이고 아키텍처 중심적인 프로세스 모델로서, 이 모델에서 소스코드는 프로젝트의 개선적 반복(iteration)을 통해 생성되는 산출물의 하나에 불과하다. 그래서 설계 기간을 길게 잡은 후 개발에 들어간다. 모토는 뼈대를 잡은 후 살을 붙이자는 것이다. 개념화->구체화->구축->전이 등의 개선적 반복으로 설계가 완료되면 신속한 개발이 가능하나 추후 수정 사항 발생 시 대처가 곤란하다는 단점이 있다[5].

RUP를 기반으로 한 산출물(보통 UML 다이어그램들)만 있다면 어떠한 개발 언어로든지 변경이 가능하다. 즉 설계자가 설계만 잘하면 단순 코더들이 이 설계를 토대로 소스를 잘 뽑아낼 수 있다는 것이다[5]. 이 모델은 건축공학, 기계공학, 전자공학에서 사용하는 개발 프로세스의 틀 안에 있다.

2.2 XP

XP란 eXtream Programing의 약자로 '가장 단순한 것이 가장 좋은 것이다'를 신조로 한다. 핵심 키워드는 '코드진화'이다. XP는 최초의 요구사항이 후반에 바뀌는 것을 당연한 것으로 여긴다. 자체 문서화(소스 자체로 문서화), 프로그래밍 패턴을 통해 빠른 설계가 가능하다. 설계는 필요 이상으로 하지 않는다. 이후 지속적인 리팩토링을 거치기 때문에 개발 후반에 수정사항이 생기더라도 개발 초반과 동일한 수정 기간만이 필요하다. 소스코드 차제가 문서화이자 산출물이며 개발자의 개발 의도를 잘 표현해준다[2,4,5].

개발 문서화보다는 소스코드를, 조직적인 개발의 움직임보다는 개개인의 책임과 용기에 중점을 둔다. XP의 목적은 '고객이 원하는 양질의 소프트웨어를 빠른 시간 안에 전달하는 것'이다. 수시로 발생하는 고



〈그림 1〉 개발방법론의 이념표

객의 요구변경에 대처하고, 고객이 원하는 소프트웨어를 고객이 원하는 시간에 인도하기 위해 고객과 팀원 간의 대화를 중시한다[2,4,5].

3. 패러다임의 변화

<표 1>은 자본주의 변천사를 그린 것이다. 국가 경제가 자유방임주의에서는 시장 방임형으로, 수정자본주의에서는 정부 중심으로, 신자유주의에서는 시장 중심으로 자본주의가 운영되었다. 그러다가 신자유주의 정책의 병폐를 뼈아프게 체험하고, 정부와 시장이 역동적으로 상호작용하여 지속 가능한 사회를 만들자는 '따뜻한 자본주의'로 변화하였다[6]. 여기서 우리는 '시장'을 '소스'로, '정부'를 '설계'로, 그리고 '경제'를 '소프트웨어'로 대치해 보면 앞으로 소프트웨어공학이 나아갈 방향을 자본주의 변천사로부터 배울 수 있다.

자본주의 4.0의 개념은 영국 타임즈의 칼럼니스트 아나톨 칼레츠키가 처음 사용한 것으로, 자본주의가 고정된 제도들의 집합이 아니라, 위기를 통해 재탄생되고 재건되며 진화하는 시스템이라는 전제하에, 리먼브라더스의 파산으로 촉발된 2008년 금융위기 이후를 소프트웨어의 버전 방식으로 자본주의 4.0으로 명명했다.

칼레츠키는 글로벌 금융위기 이후 변화된 경제환경을 불확실성의 원리가 지배하는 '적응성 혼합경제' 라고 표현하였다. 미래가 인간의 행위와 기대, 그리고 현실 간의 상호작용에 의존하는 불확실한 세상에서는 시장의 결정과 정부의 결정 모두 시행착오를 거치며 경제 시스템이 변화하는 여건에 적응하면서 계속 진화해 가야 한다고 주장했다.

정부와 관련한 그의 생각은 "미래에 발생할 수 있는 위기를 막으려면 우리에게 더 강력한 정부나 더 상세한 규칙이 필요한 것이 아니라, 시장을 존중하지만 시장의 한계와 결점도 이해하는 사람들이 운영하는 더 뛰어난 정부가 필요하다. 시장은 일반적으로 옳지만 때때로 위험할 정도로 틀릴 수 있기 때문이다" 라고 이야기 했다[6].

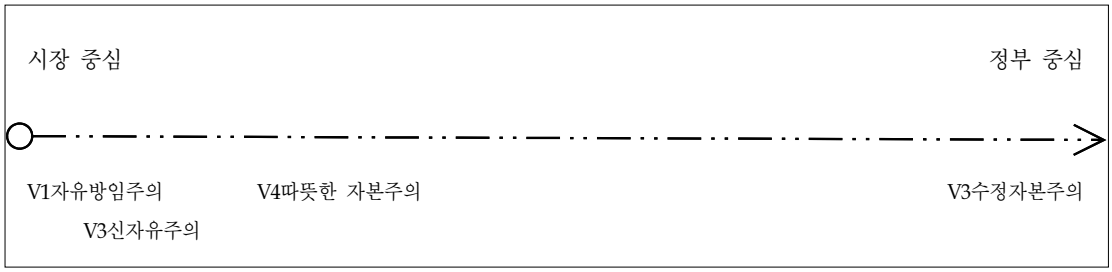
<그림 2>는 <그림 1>에 대한 대응표이다. 애자일 개발방법론이 자본주의 4.0 따뜻한 자본주의에 대응되고, 구조적 개발방법론이나 객체지향 개발방법론은 자본주의 3.0 수정자본주의에 대응된다. 즉 설계의 완전성에 만전을 기하면 나중의 문제점들이 자연스럽게 해결될 것이라고 생각하는 것은 착각일 뿐이라는 확신이다. 설계의 완전성은 개발 프로젝트가 끝났을 때조차 기대할 수 없는 목표이다. 단지 설계와 소스코드의 협력적인 끊임없는 '진화'만이 있을 뿐이다. 여기서 애자일 개발방법론이 설계 무용론에 해당하는 자본주의 3.0 신자유주의로 오인되면 안 된다.

4. 애자일 소프트웨어 개발

90년대 후반까지의 소프트웨어 개발방법론은 장기간에 걸쳐 많은 사람들을 투입하고 충분한 비용을 투입하여 진행하는 여타 공학의 프로세스와 비슷한 맥락에 바탕을 두었었다. 그러나 애자일 개발 프로세스는 소프트웨어 개발 자체가 여타 공학의 프로세스와는 근본적으로 큰 차이가 있음을 인정하는 데서부터

〈표 1〉 자본주의 변천사[6]

분류	신조	시기	특징
자본주의 1.0	자유방임주의	애덤 스미스~ 1929년 세계 대공황	정부는 시장에 개입하지 않는다.
자본주의 2.0	수정자본주의	1930년대 뉴딜정책 시기 ~ 1970년대 석유파동	'정부가 경제를 살렸다. 정부는 언제나 옳다.'
자본주의 3.0	신자유주의	1980년대 신자유주의 ~ 2008년 금융위기	'시장은 언제나 옳다'
자본주의 4.0	따뜻한 자본주의	2008년 세계 금융위기 이후	'정부는 시장과 유기적인 상호작용을 이뤄가야 한다.'



〈그림 2〉 자본주의 이념표

시작하였다.

애자일 개발 프로세스는 제한된 시간과 비용 안에서 정보는 불완전하고 예측은 불가능하다는 전제를 가진다. 이 전제가 자본주의 4.0에서 얘기하는 '경제 환경은 불확실성의 원리가 지배한다'는 현실에 대한 인식이 같다. 그리고 그 전제 하에서 합리적인 답을 내도록 하는 것이 애자일 개발 프로세스이다.

전통적인 개발 프로세스들은 공업에서 사용하는 정형적 프로세스 제어 모델을 따른다. 정형적 프로세스 제어 모델은 동일한 입력에 대해 동일한 결과가 기대될 경우에 적합하다. 그러나 소프트웨어 개발은 경험적 프로세스 제어 모델로 접근해야 한다. 경험적 프로세스 제어 모델은 항상 불확실성을 수반하고 포용한다. 애자일 개발 프로세스가 바로 그러하다.

> 우리는 스스로 행하고 다른 이들도 이를 행할 수 있도록 도움을 줌으로써 소프트웨어 개발의 더 나은 방법을 전파한다. 이러한 작업을 통해 우리는 아래와 같은 가치에 도달하게 되었다.
> - 절차와 도구를 넘어선 개성과 화합
> - 종합적인 문서화를 넘어선 동작하는 소프트웨어
> - 계약과 협상을 넘어선 고객과의 협력
> - 계획 준수를 넘어서 변화에의 대응
>
> 이들의 앞선 가치들을 인정하면서도 뒤에 오는 가치들에 보다 큰 무게를 둔다.

애자일 방법론은 무계획적인 개발 방법과 지나치게 계획하는 개발 방법 사이에서 타협점을 찾고자 하는 방법론이다. 무계획적인 방법론은 앞으로의 일을 예측하기 힘들고 효율적이지 못하다는 점에서 취약점을 가지며, 계획에 너무 의존하는 경우는 그 형식적인 절차를 따르는데 필요한 시간과 비용을 무시할 수 없으며, 전체적인 개발의 흐름 자체를 더디게 한다. 애자일 연합에서 추구하는 사상은 다음 선언문에 잘 나타난다[7].

애자일 방법론이 기존 방법론과 구별되는 가장 큰 차이점은 덜 문서화 지향적이고(less document-oriented), 더 코딩 지향적(more code-oriented)이라는 것이다. 그래서 애자일 방법론은 계획을 통해서 주도해 나갔던 과거의 방법론과는 다르게, 앞을 예측하며 개발하지 않는다. 대신 일정한 주기를 가지고 끊임없이 프로토타입(prototype)을 만들어 가며, 그때그때 필요한 요구를 더하고 수정하여 최종적인 소프트웨어를 개발해 나가는 적응적 스타일(adaptive style)이다. 이 점이 칼레츠키가 글로벌 금융위기 이후 변화된 경제환경을 불확실성의 원리가 지배하는 '적응성 혼합경제' 라고 인식한 것과 관점이 유사하다.

4.1 설계

4.1.1설계 관점: 대본 작성

소프트웨어 설계란 설계자가 고객 또는 개발자와 커뮤니케이션하기 위한 수단이다. 그런데 소프트웨어 설계는 여타 공학에서의 설계와는 다른 특성을 인정해야 한다. 그 핵심은 소프트웨어 설계를 일종의 대본 작성으로 보아야 한다는 것이다. 건축공학에서 건축

설계의 최종 산출물은 건축물에 대한 스틸 사진이다. 그러나 소프트웨어 설계의 최종 산출물은 시간의 흐름에 따라 혹은 외부의 이벤트에 대해 어떤 기능을 수행해야 하는지를 계획하는 것이다. 이 점이 바로 연극의 대본에 가깝다[3].

대본에 연극을 위한 모든 정보를 담을 수 없듯이, 소프트웨어 설계도 개발을 위한 모든 정보를 설계 단계에서 담을 수 없다. 개발자가 개발을 진행하면서 많은 부분들을 처리할 수밖에 없다. 만약 설계 단계에서 개발에 필요한 모든 정보를 담을 수 있다면 설계 문서 자체만으로도 컴파일이 가능할 것이다. 연극이 배우의 역량에 많은 부분을 의존하듯이 소프트웨어 개발도 개발자의 역량에 많은 부분을 의존해야 한다[3].

연극의 3요소가 대본, 배우, 관객이라면, 소프트웨어의 3요소는 설계, 개발자, 사용자라고 할 수 있다. 그러니 개발자가 알아서 할 부분까지 상세히 설계할 필요가 없다. 완벽한 요구사항의 명세로 후반의 수정 거리를 없애겠다는 많은 시도들은 실패로 끝났다. 지속되는 무수한 수정으로 문서의 존재 이유가 상실된다. 문서의 갱신에 너무 많은 시간이 소요된다. 문서의 정확도 역시 떨어진다. 이로 인해 문서를 보지 않고 소스를 보게 된다. 설계는 상황에 따라 필요한 만큼 자유롭게 적절히 하는 것이 관건이다.

4.2.2 설계 수준

시스템 개발의 위험요소 중의 하나는 과도하게 명세화(over-specification)하는 것이다. 이것은 미래에 필요한 것들을 예견하고 그것들을 완수하기 위해 시스템을 너무 복잡하게 만드는 것이다. 여러 상황에서 그러한 미래의 요구사항들은 실제로 일어나지 않을 가능성이 높다.

그러면 스펙&설계 단계에서 어느 정도까지 설계가 이루어져야 할까? 개발자가 설계대로 구현을 하면서 약 5% 정도의 내용은 설계자나 관련된 컴포넌트 개발자와 서로 의논하면서 개발할 수 있을 정도가 적당하다. 문서만 보고 전혀 대화하지 않고도 구현할 수 있다면 너무 자세히 적은 것이다. 이렇게 자세히 적는 것은 시간 낭비이고, 개발자의 자유를 너무 없앤 것이다[9].

설계는 스펙을 작성할 때부터 시작되고, 설계 단계에서는 컴포넌트가 모두 구분되고, 인터페이스가 정의되어 소스 상에 모두 적히고 컴파일이 가능해야 한다. 그리고 그 설명을 별도로 문서화 하는 것은 중복이기 때문에 Doxygen이나 Javadoc을 이용해서 소스코드에 주석으로 처리하는 것이 설계 정보를 효율적으로 관리할 수 있는 방법이다[9].

4.2 문서화

4.2.1 문서화의 목적

문서화는 '어떻게 하면 작성된 스펙을 가지고 개발자들이 구현을 할 수 있을까?'만 생각하여 가장 효율적이고, 시간을 최대한 절약시킬 수 있는 방법으로 해야 한다. 소프트웨어를 개발하면서 문서를 만드는 이유가 무엇인가?

① 고객의 요청
② 유지보수의 용이성
③ 개발 시간과 비용 단축

문서화의 목적을 ①이나 ②로 생각하는 사람은 문서를 형식적으로 작성하거나, 문서 작성을 거의 하지 않고 개발하면서 문서는 거추장스러운 것이라고 생각할 수밖에 없다. 문서화의 목적을 ③번이라고 체득하여 인식하고 있을 때 적시에 적절한 내용으로 문서를 작성할 수 있다. 개발방법론에서 요구하는 수많은 문서들은 문서 자체가 목적이 아니다. 모든 문서들은 다음 작업을 진행하기 위해 필요하기 때문에 만드는 것이다. 따라서 다른 개발자들이 내가 만들어 놓은 문서들을 보고 작업을 진행할 수 있도록 문서화 해야 한다[9].

4.2.2 UML의 용도

다이어그램을 무엇 때문에 그리는지 명심해야 한

다. 우리가 UML을 사용한다면 효율적인 커뮤니케이션을 위한 것이다. 효율적인 의사소통은 중요한 것은 선택하고 덜 중요한 것은 무시한다는 것을 의미한다. 이러한 선택 기준이 UML을 잘 사용하는 핵심이다. 모든 클래스를 그리지 말고 필요한 것만 그려라. 각각의 클래스에 대해 모든 속성과 동작을 보이지 말고 중요한 것만 보여라. 모든 이용 사례(Use Case)와 시나리오에 대해 시퀀스 다이어그램을 그리지 말고 필요한 것만을 그려라.

설계 원리와 진의를 모른 채 생성된 많은 다이어그램과 상세한 설계서는 실제 개발을 할 때 도움이 되기는커녕 방해만 될 뿐이다. UML을 이용해 완벽한 설계서를 만들려는 시도가 성공적이었는지 생각해 볼 필요가 있다. 코드는 자세한 정보의 저장소이고, 다이어그램은 가장 중요한 이슈를 요약하기 위한 수단일 뿐이다[1,10].

5. 결 론

우리는 이제 인접 공학으로부터 벗어나 인문학으로부터 벤치마킹을 해야 하는 패러다임의 전환 시점에 이른 것 같다. 이런 맥락에서 애자일 프로세스는 소프트웨어 개발 자체가 다른 공학적인 프로세스와는 근원적으로 큰 차이가 있다는 것을 인정한 것이다. 애자일 연합의 선언문에 나타난 사상은 기존 소프트웨어 개발방법론에서 중점을 둔 가치들의 중심추를 정반대로 옮긴 것이다.

본고에서 애자일 개발방법론이 자본주의 4.0 따뜻한 자본주의에 대응된다고 본 것은 미래에 발생할 수 있는 위기를 막으려면 우리에게 더 강력한 정부나 더 상세한 규칙이 필요한 것이 아니라, 불확실한 세상에서는 시장의 결정과 정부의 결정 모두 시행착오를 거치며 경제 시스템이 변화하는 여건에 적응하면서 계속 진화해 가야 한다는 주장에 근거한다. 설계의 완전성에 만전을 기하면 나중의 문제점들이 자연스럽게 해결될 것이라고 생각하는 것은 착각일 뿐이라는 확신이다. 설계의 완전성은 개발 프로젝트가 끝났을 때조차 기대할 수 없는 목표이다. 단지 설계와 소스코드의 협력적인 끊임없는 '진화'만이 있을 뿐이다.

개발 프로세스는 현실적이며 자율적이며 개발 문화와 균형을 이뤄야 한다. 개발자가 즐겁게 일할 수 있는 환경이 가장 생산성이 높은 환경이다. 개발 프로세스를 엄격하게 강화하여 소프트웨어 개발 문제를 해결할 수 있다는 생각부터 버려야 한다. 적절한 개발 문화가 뒷받침되지 않는 프로세스는 오히려 프로젝트 수행에 짐만 될 뿐이다[8]. 개발 문화가 바뀌려면 사고의 패러다임이 먼저 바뀌어야 한다.

특히 실제 개발 현장에서는 벌어지지 않는 교과서에나 나오는 것을 실제 개발 현장에서 적용하려는 우를 범하지 말아야 한다. 개발 단계별로 너무 많은 문서를 요구하거나 승인을 요구하면 안 된다. 개발방법론을 하나 정해 놓고 모든 프로젝트에서 그 개발방법론을 무조건 따르게 하는 것도 문제다. 우리는 개발방법론이 관료화되는 것을 항상 조심해야 한다. 소프트웨어 개발의 생산성을 향상시키기 위한 최선의 방법은 좋은 개발문화를 형성하여 개발자간 커뮤니케이션 능력을 배양하고, 협업을 통한 시너지가 창출될 수 있도록 유도하는 것이다. 개발자들이 보람과 긍지를 가지고 일할 수 있는 개발 환경을 조성하자[8].

참 고 문 헌

[1] Martin Fowler, "Is Design Dead? - UML and XP", http://martinfowler.com/articles/designDead.html#N40013B

[2] XP 대문, "한국 eXtreme Programming 사용자 모임", http://xper.org/wiki/xp/

[3] 글쓰는 프로그래머, "소프트웨어 설계가 완벽할 수 없는 다섯 가지 이유", http://swarchi.tistory.com/12

[4] 삽질, "XP 개발방법론", http://yes.imhappyo.com/238.

[5] 삽질, "한국에서의 개발방법론이란",

http://yes.imhappyo.com/239
[6] 아나톨 칼레츠키, 자본주의 4.0, 컬처앤스토리, 2011
[7] 위키백과, 애자일 소프트웨어 개발, http://ko.wikipedia.org/wiki/%EC%95%A0%EC%9E%90%EC%9D%BC_%EC%86%8C%ED%94%84%ED%8A%B8%EC%9B%A8%EC%96%B4_%EA%B0%9C%EB%B0%9C.
[8] 전규현, "개발 프로세스 관료화의 함정", http://allofsoftware.net/entry/개발-프로세스-관료화의-함정, 2014.1.
[9] 전규현, "소프트웨어 개발방법론의 함정", http://www.zdnet.co.kr/news/news_view.asp?artice_id=20091129180815&type=det, 2009.11.
[10] 전규현, "실리콘밸리 개발자 눈에 비친 한국 SW 회사", http://www.zdnet.co.kr/column/column_view.asp?artice_id=20140114132013&type=det, 2014.1.
[11] 최희준 외, 정보처리기사 필기 기본서 4권 소프트웨어 공학, 영진닷컴.

◐ 저 자 소 개 ◑

양 단 희

1989년 연세대학교 전산과학과(이학사)
1991년 연세대학교 대학원 전산과학과(이학석사)
1999년 연세대학교 대학원 컴퓨터과학과(공학박사)
1991년~1995년 현대전자 S/W 연구소
2001년~현재 정보과학회/정보처리학회/인터넷정보학회 논문지 심사위원, 인터넷정보학회 학회지 편집위원
2013년 Visiting Scholar at Texas A&M University
2001년 3월~현재 평택대학교 컴퓨터학과 부교수
관심분야: 멀티미디어, 컴퓨터보안, 자연어처리, 소프트웨어공학, 정보/의미 분석