[Software Modeling and Analysis]

[최종 보고서]

SSAD vs. OOAD

3조 (Sweet heart)

200412301 권용휘

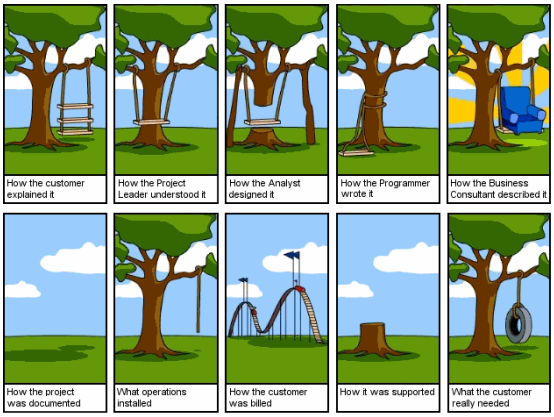
200412359 최원석

200511337 양지승

200611517 정훈섭

1. Software Development

개발 방법론은, 기본적으로 여러 다른 사람들 사이의 의사 소통을 명확하게 하기 위하여 만들어졌다. 아래의 [그림 1]은, www.projectcartoon.com의 의사 소통의 오류로 인한 결과를 묘사한 만화이다.



[그림 1] How Projects Really Work (http://www.projectcartoon.com/cartoon/2)

또한, 이러한 소프트웨어 개발 방법론에 대한 이해를 위해 반드시 필요한 것 중 하나는 "소프트웨어 위기"라는 용어이다. 이는 1968년 1968년 독일의 Garmisch에서 열렸던 NATO Software Engineering Conference에서 F. L. Bauer가 했던 용어로, 복잡성(complexity), 기대치(expectations), 변화 가능성(change) 등의 문제로 인하여, 정상적으로 동작하고(Working correctly), 이해 가능하며(Understandable), 검증 가능한(Verifiable) 소프트웨어를 작성하는 것이 어려워 질 것이라는 것을 말한다.

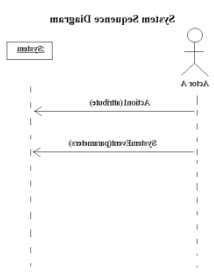
이러한, 여러 문제들을 해결하기 위하여 연구된 방법들도 여러 가지가 있는데, 본 수업에서는 Object Oriented Analysis and Design(OOAD)와 Structured Analysis and Structured Design(SASD)를 다루고 실습하였다. OOAD와 SASD는 소프트웨어 개발을 위한 의사 소통을 쉽게 하고, 의미를 명확하게 전달하기 위한 동일한 목적을 가지고 있지만, 두 방법론이 추구하는 가치와 결론이 다르기 때문에, 실제 방법론을 사용하고, 이해하는 것에는 다소 차이가 있었다.

2. OOAD

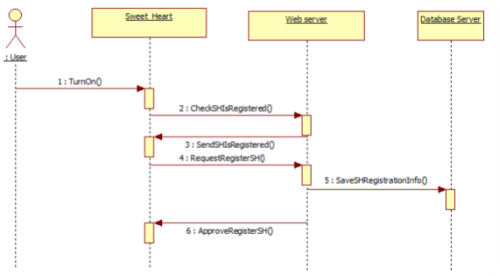
본 수업에서 실습했던 부분은 OOAD방법 중 하나인 RUP(Rational Unified Process)방법의 일종인 OSP(Object Space Process)를 통하여 학습 및 실습하였다. OSP는 여러 단계로 나누어져 있으며, 각 단계마다 처리해야 할 내용들이 지정되어 있다. 이러한 지침 사항들을 따라서 개발하기 위해서는 요구사항과 만들고자 하는 소프트웨어에 대한 팀원들간의 전체적인 이해가 높아야 하는데, 본 수업에서 가장 큰 문제점 중 하나였던 것이 바로 이 팀원들간의 이해가 다소 부족하다는 점이었다. 이는 다소 아이러니한 부분으로, OOAD라는 방법 자체가 서로 다른 사람들간의 이해를 돕기 위한 것이지만, 이를 이해하기 위해서는 각종 표기법들에 대한 이해가 이루어져야 한다는 문제가 있기 때문에, 이러한 표기법에 익숙하지 않은 사람에게는 이러한 방법이 더욱 어렵게 느껴질 수 있다는 문제가 있었다. 다행히, 프로젝트가 끝나갈 무렵에는 프로젝트 자체에 대한 이해와 더불어 표기법에 대한 이해도 함께 높아져 본 기법을 사용하여 좀 더 원활한 의사소통을 할 수

있었다.

본 팀은 수업의 실습을 위하여 Sweet heart라는 커피 머신을 제어하는 각종 소프트웨어 컨트롤러를 디자인하였다. 이러한 작업을 진행 하면서 나타난 문제점 중 하나는, 초기 디자인 시의 System Boundary에 대한 다소 잘못된 이해였다. 실질적으로, 팀원 모두 소프트웨어 중심의 교육을 받아왔고, 관심도 그러한 부분에 더 많았기 때문에, 기계를 제어한다는 입장에서 생각해보는 것이 다소 어려웠다. 그래서, 어떤 부분들이 어디에 포함되어야 하는지, 그리고 누가 제어해야 하는지를 정확하게 판단하기가 다소 어려웠다. 그래서, 매번 중간 발표를 할 때마다 많은 부분이 변경되어야 했으며, 두 번째 사이클에서는 처음부터 끝까지 모든 내용을 다시 작성하기도 하였다. 두 번째 사이클에서 모든 부분을 재 작성하면서 많이 힘들기는 했지만, OOAD에서 제시하는 여러 조건에 부합하는 형태로 시스템이 디자인 되는 것을 보았을 때, OOAD를 진행하면서 계속 느껴오던 뭔가 부족한 느낌을 완전히 해소할 수 있어서 좋은 경험이 되었다고 생각한다. 하지만, 이러한 부분들을 좀 더 일찍 알았었다면, 두 번째 사이클에서 더 견고한 시스템을 설계하면서 얻을 수 있는 좋은 경험을 할 수 있었을 것이라는 아쉬움도 남는다. 총 4회의 발표에서 본 조는 매번 시스템 변경이 있었다. 매번 이루어지는 시스템 변경을 통해, 본 조는 처음에는 왜 시스템 변경을 해야 하는가를 알지도 못하다가 그 이후에는 Object Oriented 중심이 중요하다는 교수님의 설명에서 지금까지 가장 중요한 객체 지향적인 사고가 미비했었다는 점이 본 조에서 개선해야 할 점임을 깨달았다. 두 번째 사이클에서 전체 시스템 모두를 변경하며 OOAD에 어울리는 시스템을 디자인하였는데, 이에 대한 자세한 코멘트가 없어서 다소 아쉽다. 다른 조들은 시스템 변경이 없었으나 본 조는 매우 역동적인 시스템 변화가 있었기에 다른 조보다 조금 힘들었지만 총체적인 경험을 할 수 있었던 뜻 깊은 시간이었다. 또 하나 느꼈던 점은 Sequence Diagram에서 여러 개의 시스템이 서로 통신하는 부분을 각각 Main이라는 개체를 사용하는 구조로 작성하였는데, 이 부분을 우리가 잘 이해한 것인지 의문이 들었지만 교수님의 강의를 듣고 우리가 성공적으로 구현했다고 느껴서 성취감이 들었다. SASD와는 다르게, OOAD 수업에서 본 조는 고군분투하였는데, 이는 OOAD에 대한 기본적인 이해 및 시스템에 대한 이해가 두루 부실하였던 것에 대한 총체적인 결과였다. 매 수업시간마다 교수님은 합리적인 피드백을 해 주셨지만 본 조의 팀원들은 객체 지향적인 사고의 중요성을 상기하지 못하여, 교수님의 피드백을 잘 반영하지 못했던 점이 아쉬움이 남는다. 마치 무지한 우리 조원들이 교수님의 소중한 피드백을 "소 귀에 경읽기" 식으로 반영하여, 교수님께 송구스러운 마음이 들었다. 관련 자료 검색을 통해 피할 수 있었던 이슈들도 많았으나, 이러한 느낀 점을 실제 구현에 제대로 반영하지 못했던 점도 아쉽다. 특히, System Sequence Diagram의 경우에는 조금 더 관심을 가지고 찾아 보았다면 이와 관련된 다른 부분까지도 총괄적으로 알 수 있었을 텐데, 처음 발표했을 때 이를 고려하지 못하여 아쉽다.



[그림 2] 올바른 그림



[그림 3] 잘못된 그림

위와 같이, [그림 2]와 같이 그려야 했던 그림을 [그림 3]과 같이 그렸었던 문제가 있었다. 기본적으로 시스템 시퀀스 다이어그램은 시스템이 사용되는 가장 외부의 접근에 대한 표기만 해야 하나, 본 조가 그린 그림은 시스템 시퀀스 다이어그램이 아닌 시퀀스 다이어그램으로 과도한 정보를 담았다는 문제가 있었다. 이는 본 조의 OOAD 개발 단계에 대한 이해가 미흡한 데서 온 오류인 것이다. 마지막으로, OOAD를 진행하며 가장 아쉬웠던 점의 하나는 좋은 툴의 부재였다. 물론 상용 도구 중 많은 도구(e.g. rational rose)들이 많이 있었지만, 본 팀이 이용한 툴은 무료 툴인 StarUML이어서, 불편한 점이 있었다. 특히, 각 다이어그램의 정확한 줄을 맞추어 주지 않는 불편함은 가장 불편한 점 중 하나였고, 각 개체 복사 시 완전히 개체가 복사되는 것이 아니라 복사된 개체와 원본 개체가 속성을 공유하는 형태의 복사는 매번 새로운 다이어그램을 그릴 때 동일한 작업을 반복해야 하는 시간과 노력의 비효율성이 있었다. 수업을 진행하는 동안 좀 더 좋은 UML 툴을 찾았으나 기존 작업 제작물을 옮기는 것에 대한 문제점으로 인해 이를 십분 활용할 수 없었던 점이 아쉬움으로 남는다. 또한 가장 많이 알려지고 보편적으로 이용되는 객체 지향 방법론이었으나, 예상보다 인터넷 자료 및 참고 문헌들이 아주 기초적인 내용만을 다루고 있어 실전 프로젝트에 대한 내용이 미비한 점이 매우 놀라웠다. 또, 수업 중간에 교수님께서 언급하였던 것처럼 이러한 개발방법론에서 사용되는 일반적인 표기법(e.g. UML)들이 정확하게 옳고 그름이 있다기보다는 표현의 정도의 차이가 있다는 것을 잘 이해하지 못했고, 더불어 이를 잘 표현되지 못함이 아쉬웠다.

3. SASD

SASD의 경우, OOAD와는 다르게 구조적 분석과 구조적 개발 방법을 지향하는 방법론이다. 본 개발 방법론은 하드웨어 개발을 원활하게 하기 위하여 고안된 것으로서, 과거에 많이 사용되었던 방법이다. 본 강의를 듣기 전에는 개발방법론이라는 이야기를 들었을 때 OOAD와 같은 객체지향 방법론만 있는 것으로 잘못 생각하였는데, 본 수업을 통해서 이 방법론을 처음으로 접하게 되었다. 기존 컴퓨터 공학 실무 업계에서는 최신의 것만 중요하게 생각하는 경향이 있으나 오히려 최근 삼성전자와 같이 하드웨어에 동작하는 소프트웨어를 집중적으로 다루는 업체에서 이 방법론을 주로 사용하고 있다는 교수님의 설명을 듣고, 항상 최신 Trend만 듣던 사고를 개선해야겠다는 생각이 들었다. '온고지신'의 정신이 다시 한 번 느껴지는 살아있는 업계에 대한 강의였던 것이다. SASD 수업에 들어가기 전, 교수님께서 본 조원들이 엔지니어적인 성향이 강하기 때문에 SASD 방법론에 더 적합할 것이라고 격려해 주셨지만 사실, 컴퓨터 공학도로서 하드웨어 관련 내용들은 거의 알지 못하여 Tick과 다른 개념들을 이해하고 적용하는데 많은 어려움이 있었다. 구체적으로, 특히 enable과 disable, tick, data input output command를 어떻게 사용해야 하는지 정확하게 이해하기 어려웠으나, 여러 가지 자료를 찾아본 결과 State Machine에서 기본적인 개념을 가져왔다는 것이 이해의 큰 단서가 되었다. OOAD는 여러 가지 전용 툴이 있어서 이를 이용하여 구현하여서 간편하고 관리가 편리한 장점이 있었으나 SASD는 이런 툴들을 사용하지 않고 파워포인트만을 이용하여 만들었던 작업이 힘들었다. 파워포인트만을 이용했던 작업은 새로운 항목을 추가하기 위하여 다른 전체를 재배열해야 했기 때문에 육체적으로는 고된 작업이었지만, 처음부터 설계를 잘하여 시간과 노력을 효율적으로 이용해야겠다는 다짐을 할 수 있었다. 또한 이 방법론에서는 최종적인 구현을 State Diagram과 Structured Chart를 동시에 보고 작성해야 하는데, 이 점이 다소

불편하였다. 이 두 가지 모델을 합친다면 좀 더 좋은 방법론이 될 수 있지 않을까 생각해 보았다. 또한, 삼성전자와 같은 업체에서도 최근에는 Embedded Software에서도 Tick, Interrupt와 같은 Low Level의 기법들을 사용하여 작성하기 보다는 C언어나 자바와 같은 기반의 High Level기법들을 사용하여 프로그램을 작성하는 추세로 바뀔 것이라고 예상되는데, SASD 방법론도 이러한 추세에 맞추어 보완되어야 할 것으로 보인다. 예를 들면, Tick과 Enable/Disable의 표기를 좀 더 명확하게 바꾼다면 디자이너와 개발자가 어떤 방법론을 사용하여 구현해야 하는가에 대한 모호한 점이 줄어들 것이다. 본 수업에서는 모델링을 하는 사람과 구현하는 사람이 동일하였기에 이러한 의미론적인 부분의 차이에도 불구하고, 큰 문제 없이 구현이 가능하였으나, 만약 서로 다른 사람이 이 방법론을 이용하여 구현한다면 이러한 세밀한 부분이 어떻게 구현되어야 하는가에 대한 이해는 쉽지 않을 것 같다. 최종적으로, State Machine을 구현하여 동작하는 코드를 작성하였을 때 SASD 방법론을 정확하게 이해할 수 있었다. 이것이 교수님께서 처음 말씀해주셨던 엔지니어적 성향이 강해 쉽게 이해할 수 있다는 것의 의미라고 생각되었다. 다만, 본 수업에서 목표로 잡았던 커피 머신 제작은 완전한 하드웨어 컨트롤러 소프트웨어가 아니라 다소 상위레벨적인 소프트웨어 특성이 있어 SASD 방법으로 구현하기에는 까다로웠는데, 특히 대부분의 작업이 Tick과 같이 짧은 시간 내에 작동할 수 없어 각 작업들을 정확하게 정의하고 이 작업이 얼마만큼의 시간을 소요하는지 예측하는 부분은 정교함을 요구하여 어려웠다. 물론 이 Tick이라는 개념을 하드웨어에서 가지고 있는 Tick이 아닌 특정 어떤 시간의 주기로 가정하는 경우에는 좀 더 잘 이해될 것으로 생각된다. 덧붙여, 데이터를 내보내는 부분들에 대한 정확한 표현에 대한 어려움은 본 과정에서 가장 어려웠던 점의 하나였으나, 이는 SASD방법론 자체의 문제점으로 판단된다. 또, Asynchronous한 부분들에 대한 처리는 SASD에서 표현이 다소 어렵다고 수업시간에 들었으나, 본 조에서는 이러한 부분들이 포함되지 않도록 시스템을 디자인 하였기 때문에, 이에 대한 부분들에 대한 문제점은 없었으나, 주요한 문제점 중 하나인 것을 고려하여 볼 때, 이러한 부분들에 대한 경험 또한 좋은 경험이었을 것이라는 생각이 든다.

4. OOAD와 SASD의 비교

4.1. 문제를 바라보는 시점

OOAD와 SASD의 가장 큰 차이는, 문제를 바라보는 시점에 있다. 먼저, OOAD는 모든 문제를 Object Oriented관점에서 보는 것이며, SASD는 각 문제들을 구조적인 관점에서 보는 것이다. SASD의 가장 큰 장점은 Process Oriented의 방식으로 문제를 보는 것이며, 이것이 일반적인 사람들의 기본적인 생각 방식과 일치한다는 점이다. 하지만, 오히려 최근에는 OOAD적인 사고가 더욱 일반화 되어 있어서, 이러한 SASD의 장점이 퇴색된 경향이 있는 것 같다. 이는 앞에서 언급했던 것과 같이, SASD라는 방법론 자체를 처음 들어본 최근의 학생들의 경향과 SASD가 처음 대두되었을 때의 환경이 많이 달라진 탓도 있기 때문일 것이라 생각된다.

4.2. 유연성

OOAD와 SASD중 다소 유연성이 있는 방법론은 SASD로 판단된다. OOAD의 경우, 작업 차이점이라도 다른 모든 부분들에 영향이 가기 때문에, 다소 수정할 부분들이 많지만, SASD의 경우에는 이러한 각 프로세스끼리의 연관성(Dependency)가 적어, 다소 부담 없이 디자인 내용을 도중에 변경할 수 있었다.

4.3. 간편성과 이해도

SASD의 경우 OOAD보다는 다소 간편하고, 좀 더 이해가 쉬웠으나, 이는 시스템이 커지게 되면, SASD에서 표현하여야 할 내용이 기하 급수적으로 많아질 수 있는 문제가 될 수 도 있다고 판단된다. SASD의 간편성과 이해의 편리성은 기본적으로 좀 더 세세한 부분까지 표현하는 부분에 있다고 보기 때문이다. OOAD의 경우에는 객체에 대한 내용과 객체들간의 관계를 기준으로 작성하기 때문에, 시스템이 커지더라도, 객체들끼리의 관계만 잘 정의해 놓는다면 물리적으로 시스템이 커지는 것 보다는 명료하게 작성할 수 있다고 판단되지만, SASD의 경우에는 OOAD에 비하여 시스템이 커지는 만큼 디자인 단계에서 추가되어야 할 부분이 더 많다고 판단된다. 물론, OOAD의 경우 기본적으로 SASD보다 더 많은 표기법과 복잡한 단계를 거치기 때문에, 이를 평면적으로 비교하기는 힘들지만, 프로젝트의 크기가 선형적으로 커진다고 가정했을 때는 이러한 차이가 날 수 있다고 생각된다.

4.4. Non-functional Requirement

SASD는 Non-functional Requirement(e.g. 시스템의 반응 속도, 완성된 시스템이 제공해야 할 QoS(Quality of Service))에 대한 정의가 포함되어 있지 않다. 물론, SASD에서는 Tick과 같은 저 수준 기법들을 사용하므로, 시스템 자체의 동작을 정의하는 것이 이러한 Non-functional Requirement의 일부분을 포함한다고 할 수 도 있으나, OOAD에서는 Non-functional Requirement를 정의함으로 인하여, 시스템에 대한 완전한 이해를 하지 못한 관리자나, 사업가와 같은 사람들에게도 시스템의 일부분을 이해시킬 수 있는데, SASD에서는 이러한 부분에서의 고려가 다소 결여되어 있다.

4.5. 개발 및 설계 방식의 차이

SASD는 반복적이지 않고, 큰 부분에서 작은 부분으로 내려오는 Divide and Conquer을 사용한 방법론이다. 이러한 방법론은 이미 시스템에 대한 이해가 선행되어야 하는 경우에서 효율적으로 작성할 수 있는데, 이러한 제약 사항으로 인하여 본 수업이 OOAD를 진행하고 SASD를 진행하였던 것으로 생각된다. 기본적으로 두 방법론을 모두 적용하는데 큰 문제점은 없었으나, OOAD의 경우에는 각 단계마다 기존의 단계를 다소 반복적으로 적용할 수 있고, 수정할 수 있는 단계들이 존재하여, 새로운 시스템을 만드는 경우에 좀 더 점진적으로 설계할 수 있었으나, SASD의 경우에는 이러한 부분이 다소 미흡한 것으로 보인다.

4.6. 분석 방법

OOAD는 앞에서 언급한 것과 같이, 실무자부터 경영자까지 많은 사람들의 입장에서 시스템을 바라보는 반면, SASD는 다소 실무자의 중심에서 시스템을 바라보는 성향이 강한 것으로 판단된다. 특히, 사용자가 시스템을 어떻게 사용할 것인지에 대한 분석이 OOAD에 비하여 SASD가 약한 부분으로 판단된다. 또한, Divide and Conquer방법을 사용하여 시스템을 분석하게 되는데, 가장 중요한 부분 중 하나인, 어느 정도까지 시스템을 잘게 분해할 것인지에 대한 의문이 남게 된다. 물론, 이러한 부분은 OOAD에서도 각 Use-case를 어느 정도의 수준까지 분해할 것인가와 유사한 고민의 대상이지만, SASD의 경우 OOAD보다 훨씬 더 작은 단계까지 나누게 되기 때문에, 잘못하면

실제 코드를 작성하는 것과 다름 없는 너무 상세한 디자인이 나올 수 있다.

4.7. 각 세부 항목들의 연관성

아래에 OOAD와 SASD의 각 항목들의 연관성과 이의 장단점 및 본 수업을 진행하면서 실제로 느꼈던 내용들을 복합적으로 정리하여 보았다.

OOAD	SASD	의견
Business Use-case, Essential Use-case	System Context Diagram	OOAD는 System Context Diagram과 같은 전체적인 시스템을 다양한 관점(경영자, 관리자, 실무자)에서 조명하는 부분들이 많다. 즉, 좀 더 다양한 사람들이 시스템을 전체적으로 이해하는 것에 좀 더 비중을 두고 있다는 생각이 들었다. 또한, OOAD의 System Boundary 비하여 SASD의 System Context Diagram은 정확한 System의 범위(Scope)가 정의되지 않는 취약점이 있다.
Real Use-case, Interaction Diagram	Data Flow Diagram, State Transition Diagrams	실질적으로 자세한 시스템의 동작을 기술한 부분들이다. SASD가 이러한 부분에서는 좀 더 자세한 내용을 정의하기 위한 단계가 많다고 할 수 있다. 하지만, 기존에 OOAD에 대한 영향을 많이 받았던 것인지, 아니면 팀원 전체가 OOAD에 더 익숙했기 때문이었는지, 표현력 자체는 OOAD가 SASD보다 더 좋았던 것 같다. 비록, OOAD에 대한 완전한 이해가 선행된 것은 아니었으나, OOAD로 표현하였을 때, SASD보다 좀 더 논리적으로 알맞게 표현되었다는 생각을 들었던 것이 사실이다. 또한, State Machine은 최근 더욱 중요성이 커지고 있는 다른 기기와의 통신을 위한 네트워크에 대한 표현이 다소 어려웠다.

5. 최종적인 의견

SASD와 OOAD는 모두 서로 다른 사람들간의 이해를 돕기 위하여 만들어진 방법론이지만, SASD의 경우에는 그 서로 다른 사람들의 정의가 다소 실무에 능한 실무자에 한정되어 있다는 느낌을 많이 받았다. 반면, OOAD는 좀 더 다양한 사람들에 대한 이해를 돕기 위한 부분이 시스템적으로 구축되어 있었기 때문에, 최근의 상황에 더 알맞은 방법론이라고 생각된다.

또한, SASD의 경우에는 기존에 잘 알고 있는 시스템을 설계하는데 주로 사용되며, 상대적으로 시간이 더 오래 걸리는 작업이라고 한다. 이러한 부분들은 본 조의 의견으로는 다소 납득하기 어려운 부분 중 하나인데, 어떠한 방법론이든 간에, 그 방법론을 어떻게 적용하는지에 따라서 시간이

결정될 것이기 때문이다. OOAD라고 할 지라도, 매우 세세한 수준까지 작성하고자 한다면, 이는 시간이 오래 걸릴 수 있을 것이며, SASD라도 아주 추상적인 단계에서 설계를 마친다면, 그리 오래 시간이 걸리지는 않을 것이다. 다만, 최근의 관점에서 보았을 때, 이전의 것과 유사하지 않은 새로운 개념과 시스템들이 각광받는 시장에서는 SASD와 같은 기존 시스템을 효율적으로 작성하는 방법론 보다는, 논리와 개념들을 추상화하여 설계하는 SASD가 좀 더 효율적인 방법론이 될 것이라고 생각한다.
또한, 최근의 Real-time Embedded System에서 많이 SASD가 사용되고 있지만, 앞으로는 이러한 하드웨어 기반의 솔루션 보다는 차별성 있는 Contents의 중요성이 더욱 대두될 것이라고 생각된다. 가장 기본적인 좋은 하드웨어와 잘 동작하는 소프트웨어를 작성하기 위하여 SASD방법론은 지속적으로 사용될 것임을 부인할 수 는 없지만, OOAD개발 방법론은 시스템이 복잡해지고 새로워지면서 더 각광받을 것이라고 생각된다.
기본적으로 두 방법론 모두 효과적으로 시스템을 설계하고 표현할 수 있었다. 다만, 이러한 효과적인 설계 및 표현을 위하여 각 방법론에서 제기한 여러 표현 방법들에 대한 이해가 선행되어야 했고, 이러한 것이 직접 구현을 하는 것 보다는 용이하였지만, 그렇다고 해서 매우 쉽지는 않았다. 다만, OOAD의 경우 SASD보다 다양한 관점에서의 설명이 존재하여 좀 더 이해하기 쉽다고 할 수 있었다. SASD는 오래 전부터 발전되어온 기법이라, OOAD에 비하여, 기법을 적용하는 것이 매우 부드럽고 잘 만들어져 있다는 생각이 많이 들었다. 물론, 이러한 느낌은 개인별로 차이가 날 수도 있겠지만, 본 조에서는 OOAD보다 SASD로 작성하였던 기간이 훨씬 짧고, 과정 중간에 발생하였던 변경 사항도 적었다. 하지만, 앞에서 이야기 한 것과 같이, 최근에는 소프트웨어 개발의 주기가 점점 짧아지고 있고, 더욱 추상적이고 새로운 개념에 맞는 소프트웨어가 각광받고 있기 때문에, 최근에는 SASD보다는 OOAD가 더 선호되고 있다고 판단된다.
개인적으로도 오랜 시간 동안 개발을 하면서, 개발 과정을 좀 더 추상적으로 설명하고, 다른 사람들에게 이해시키기 위한 여러 방법론들에 대하여 관심이 많았었는데, 본 수업을 통하여 이러한 부분들을 한번에 정리하고 실습도 해 볼 수 있어 매우 좋은 기회였다고 생각한다. 다만, 여러 서적들마다 약간씩은 다른 이야기를 하고 있는 것이 심한 분야 중 하나이기 때문에, 설계와 디자인에는 왕도가 없다는 사실을 다시 한번 실감하게 되었다. 개인적으로는 다소 실무적인 부분에 도움이 될 수 있는 UX부분이 본 수업과 함께 다루어 진다면, 더 좋은 시너지 효과를 낼 수 있을 것 같다는 생각이 들었다.

6. 끝 맺으며

한 학기 동안, 많은 가르침을 주어 정말 감사합니다. 수업 초반부터 잘 따라가지 못했던 저희를 챙겨주셔서 감사 드립니다. 이번 학기 여러 과목이 있었지만, 이 과목이 가장 신경이 많이 쓰였고, 배운 것이 많은 과목 중 하나라고 생각됩니다. 이렇게 힘들 것이라고는 생각하지 않고 수강한 수업이었기 때문에, 여러 가지로 적응하는데 더 힘들었던 것 같습니다만, 아픈 만큼 성숙해진다는 말도 있듯이, 알차고 꼼꼼한 수업에서 정말 많은 것을 얻어갑니다. 많은 수업들이 단순히 강의하는 내용을 암기하는 중심의 강의이기 때문에, 오히려 교수님의 수업 방식이 더 특이하고 이상하게 느껴졌던 것이 사실이었습니다만, 손바닥으로 하늘을 가릴 수 는 없듯이, 언젠가는 이러한 수업들이 더 많은 대학이 되어야 할 것이라 생각합니다. 다만, 저희와 같이 이러한 방식에 익숙하지 않은 학생들을 위한 약간은 더 상세한 가이드라인이 존재한다면 금상첨화일 것 같습니다. 중간

발표 시에 다소 격양된 반응을 보여 드렸던 점 사과 드리며, 지나고 보니 잘못된 부분들을 지적해 주셨었는데, 너무 저희 생각만 했던 것 같아서 송구스럽습니다. 지식적인 면이 아니라, 여러 방면으로 주셨던 가르침들은 앞으로 살아가는데 큰 도움이 될 것이라 생각됩니다. 감사합니다.