Contents

1) 프로그램 정보

이번 프로젝트를 통해 개발한 프로그램은 DSM 입니다. 프로젝트의 목적 자체가 새로운 프로그램의 개발이 아닌 기존에 있던 프로그램인 Titan을 본 떠 만드는 것이었기 때문에 기본적인 기능은 Titan과 동일하다고 할 수 있습니다. 프로그램의 기본 요구조건인 dsm과 clsx파일의 입출력 및 이에 대한 matrix로의 표현, 프로그램 상에서의 dsm및 clsx파일의 수정 및 저장 등이 이에 해당됩니다.

+ 더 추가된 것 있으면 내용 추가

프로그램을 기능적인 측면이 아닌 구조적인 측면에서 본다면 다음과 같은 특징을 갖습니다. 기본적으로 입력되는 모듈들을 제어하는 자료구조는 자식 노드 개수에 제한이 없는 트리 입니다. 트리를 통해 상, 하위 폴더 및 하위 모듈들의 상관관계를 처리하며 모듈의 추가, 삭제, 수정 등 일괄작업이 이루어집니다.

다음으로 프로젝트의 기본 요구조건이기도 한 MVC architectural pattern의 적용으로 Model, View 그리고 Controller를 분리시켰고 이를 통해 개발상의 편리함 및 장점, 프로그램 유지보수 상의 유리함 등을 이끌어 낼 수 있었습니다.

또한 컴포넌트 사이의 데이터 공유 시 Singleton pattern 방법을 사용하여 instance를 한번만 생성해도 모든 컴포넌트의 데이터사용이 가능하도록 구현했습니다. 이를 통해 시간적인 오버헤드와 메모리 낭비 차원의 개선을 만들어 낼 수 있었습니다.

+ 쓰여진 자료구조 등 구조적 특징 추가

2) 개발 프로세스

저희 조가 이번 프로젝트에 적용하기 위해 선택한 개발 프로세스는 Waterfall Process입니다. 이 프로세스를 선택한 자세한 이유는 다음과 같습니다.

먼저, 저희는 이번 프로젝트를 프로그램 개발 자체가 아닌 지금까지 배운 지식들을 실제로 적용해 보기 좋은 기회라고 생각했습니다. 이러한 소규모 프로젝트에는 Implemental Development가 적합하다는 것을 알고 있었지만 저희는 지금까지의 프로그래밍 프로젝트처럼 세세한 계획이나 자세한 명세 없이 개발하려는 프로그램의 주제와 대략적인 기능적 명세만으로 프로그램 개발을 시작하는 것은 이번 학기 수업에서 배운 것들을 제대로 사용하지 못할 것 같다고 판단했습니다. 이에 따라 실제 대형 프로젝트에서의 절차대로 Use-Case Diagram이나 Sequence Diagram, Class Diagram같은 자세하고 확실한 명세를 바탕으로 한 개발 프로세스를 선택하게 되었습니다.

다음으로, Waterfall Process만이 갖는 장점을 직접 체험해보고 싶었습니다. 위에 설명했듯이 소프트웨어공학에서 배우는 지식들, 예를 들면 Waterfall Process나 Implemental Development가 갖는 각각의 장점 및 단점 등을 배우기 전에는 자세한 명세 작성 후 설계보다는 구

현이 더 우선시되는 개발 방법을 택했던 것이 보통입니다. 이에 따라 저희는 지금까지의 방식이 아닌 Waterfall Process를 통해 이 방식의 장점을 느껴보려고 했습니다.

물론, Waterfall Process를 처음 사용하면서 겪은 시행착오도 많습니다. Diagram 작성에 대한 숙련도가 부족한 탓에 처음 작성한 Diagram대로 개발을 해나가는 것이 어려웠고 도중에 수정이 필요한 곳이 많았습니다. 하지만 이를 통해 Diagram 자체의 설계뿐 아니라 프로그램 전체적인 설계에 대한 충분한 연습이 되었고 경험이 되었다고 생각합니다.

3) 디자인 다이어그램

+ 다이어그램 완성되면 스크린샷으로 추가

4) MVC 아키텍처 패턴의 적용

저희 조에서는 MVC architectural pattern의 적용의 완성도와 실제 구현 과정에서의 자연스러운 패턴 적용을 위해 개발 시 역할 분담 자체를 Model, View, Controller의 독립적 개발을 할 수 있도록 나누었고 각각의 통합을 통해 프로그램을 완성하는 방향으로 프로젝트를 진행했습니다. 그 결과 저희가 의도했던 대로 개발 과정에서의 MVC pattern의 자연스러운 적용이 가능했습니다.

또한 개발 도중 MVC architectural pattern의 장점을 직접 느낄 수 있는 기회가 있었습니다. 개발 도중 Model과 Controller쪽의 완성의 거의 되가는 상황에서 View 컴포넌트 쪽에서 기능적 문제 때문에 Class Diagram과는 달리 구현해야 하는 상황이 있었습니다. View 컴포넌트를 수정하면서 Model, Controller, View를 분리해놓은 것으로 인해 View에서의 수정이 나머지 두 컴포넌트에 영향이 거의 없다는 것을 실제로 알 수 있었습니다. 이를 통해 준수한 아키텍처 설계는 프로젝트의 개발, 그 이후의 유지 보수에 긍정적인 영향을 크게 미친다는 것을 다시 한번 느낄 수 있었습니다.

5) 테스트과정 및 방법

+ 제출물 중에 test한 data가 있어서 test case 만들어야 할 듯...

그리고 테스트과정이 중요해서 JUnit 간단하게라도 사용한 것 처럼 스크린샷 찍어야 할 듯

6) 객체지향적 개발방법의 적용

TreeNode 클래스를 따로 만들어 DsmModel이나 Cluster-Model 에서 데이터 관리를 위해 트리를 사용할 때 Tree-Node의 내부 구현을 세세히 알 필요 없이 해당 객체의 인터페이스만을 통해 데이터를 관리 할 수 있게 캡슐화 하였습니다. TreeNode뿐 아니라 DsmModel이나 ClusterModel, Tree-Action 클래스도 각각이 캡슐화 되어있어 ModuleController 입장에서 Main_View를 포함해 인터페이스에 따라 사용하며 프로그램의 동작을 제어 할 수 있도록 설계되었습니다.

또한 각각의 모듈에 대한 액션은 모두 TreeAction 클래스에 따로 구분해 놓음으로써 응집도를 높일 수 있는 방법을 생각했습니다. 하지만 결합도를 낮추기 위해 DSM과 Cluster를 각각 별도의 클래스로 나누어 설계하였습니다.

이 방법으로 SRP(Single Responsibility Principle)또한 만족시켰습니다.

7) 프로그램 예제

+ 스샷 찍어서 추가

8) 덧붙일 것 (팀 원 각자 느낀 점 같은)