**ScreenScraper Processing App (ADS)**

1. **Project Outcomes**
   1. **DDD - Domain Layer (business logic)**
      * **DB layer not relevant**
      * **identify entities, aggregates**
   2. **Class diagrams**
   3. TDD
   4. **States, Interactions and outcomes (given, when, then)**
   5. Requirement Communication Techniques
   6. BDD
   7. SOLID
   8. Analysis & Design
   9. Dependency Injection
   10. UML
   11. Prototype

DDD

* **Context**: The setting in which a word or statement appears that determines its meaning. Statements about a model can only be understood in a context.
* **Model**: A system of abstractions that describes selected aspects of a domainand can be used to solve problems related to that domain.
* **Ubiquitous Language**: A language structured around the domain model and used by all team members to connect all the activities of the team with the software.
* **Bounded Context**: A description of a boundary (typically a subsystem, or the work of a specific team) within which a particular model is defined and applicable.

Building blocks

* **Entity**: An object that is identified by its consistent thread of continuity, as opposed to traditional objects, which are defined by their attributes.
* **Value Object**: An immutable (unchangeable) object that has attributes, but no distinct identity.
* **Domain Event**: An object that is used to record a discrete event related to model activity within the system. While *all* events within the system could be tracked, a domain event is only created for event types which the domain experts care about.
* **Aggregate**: A cluster of entities and value objects with defined boundaries around the group. Rather than allowing every single entity or value objectto perform all actions on its own, the collective aggregate of items is assigned a singular aggregate root item. Now, external objects no longer have direct access to every individual entity or value object within the aggregate, but instead only have access to the single aggregate root item, and use that to pass along instructions to the group as a whole. This practice correlates with many of the actual coding practices we’re covering in our [design patterns](https://airbrake.io/blog/category/design-patterns)series.
* **Service**: Essentially, a service is an operation or form of business logic that doesn’t naturally fit within the realm of objects. In other words, if some functionality must exist, but it cannot be related to an entity or value object, it’s probably a service.
* **Repositories**: Not be confused with common version control repositories, the DDD meaning of a repository is a service that uses a global interface to provide access to all entities and value objects that are within a particular aggregate collection. Methods should be defined to allow for creation, modification, and deletion of objects within the aggregate. However, by using this repository service to make data queries, the goal is to remove such data query capabilities from within the business logic of object models.
* **Factories**: As we’ve discussed through a number of [design patterns](https://airbrake.io/blog/category/design-patterns) articles already, DDD suggests the use of a factory, which encapsulates the logic of creating complex objects and aggregates, ensuring that the client has no knowledge of the inner-workings of object manipulation.