**+ public, - private, # protected, ~package visible**

**RDD Stereotypes**

Controller: Object implementing this role makes decisions and closely directs the action of other objects. Coordinator: This role reacts to events by delegating tasks to others. Information Holder: Information holder knows and provides information. Information Provider: A slight variation of an information holder is the information provider, which takes a more active role in managing and maintaining information. This distinction can be used if a designer needs to get more specific. Interfacer: This role transforms information and requests between distinct parts of an application. It is further divided into more specific roles. External Interfacer: External interfacer communicates with other applications rather than its own. It is mainly used for encapsulating non-object-oriented APIs and does not collaborate a lot. Internal Interfacer: Also called intersystem interfacer.[[14]](https://en.wikipedia.org/wiki/Responsibility-driven_design#cite_note-ObjectDesign-page93-14) It act as a bridge between object neighborhoods. User Interfacer: User interfacer communicates with users by responding to events generated in the UI and then passing them on to more appropriate objects. Service Provider: This role performs work and offers computing services. Structurer: This role maintains relationships between objects and information about those relationships.

Single Responsibility Principle (SRP) – Each class should have a single responsibility or a single reason to change.

SCRUM

Focuses mainly on management of software product development

Timeboxed iterations called sprints

Project planned through use of 2 spreadsheets (Product and Sprint)

Daily Meeting (Pigs and Chickens)

Each sprint ends with “ready” deliverable

**Iterative delivery**

Project divided in pieces called iterations

At end of iteration something is delivered

No schedule slip (timeboxing as opposed to scopeboxing)

Each iteration nominally 4 weeks (but can be as short as 1 week in XP, or 6 weeks in RUP)

Each iteration does varying amounts of analysis, design, implementation and testing

**INVEST**

* Independent == Stories should be able to be developed independent of each other.
* Negotiable == Do not put implementation language in story. Keep it general enough so you have design flexibility. Don’t try to put all the details in the title.
* Valuable == Has to provide business value to the customer
* Estimatable == Can’t be too big, too complex, or too unfamiliar to developers
* Small == Cannot span an iteration. If too big break into smaller stories and use the bigger story as an epic
* Testable == Have to be able to specify acceptance tests for the story

Primary Actors – directly use system

Supporting Actors – don’t know anything about the system, but support it

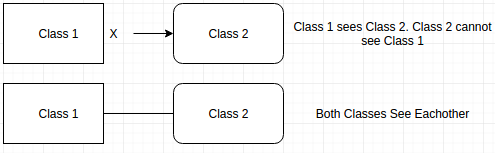
Offstage Actors – don’t use system, but care about things that happen

FALSE: The best way to design any software application is through the use of Object-Oriented techniques.

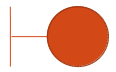
Objects: know things, do things, collaborate with other objects, make decisions.

A software conceptual or logical architecture is most often presented in UML as: A package diagram.

Trust boundaries are important because: They show where extra care must be taken in validating and protecting data, They show where defensive programming techniques might NOT be required.

****

**Robustness Diagrams**

**Boundary Class -** UI or API class/External System

**Entity Class -** Class from Domain Model

**Controller Class -** Class for business logic

**Rules**

-External Actors only talk to Boundary Classes

-Entities only talk to Controllers

-Controllers can talk to Boundary Classes and Entity **Classes**

**Software Architectures**

Client - Server

P2P (Peer to Peer)

Blackboard

Pipe & Filter

MVC (Model View Controller)

N-tier

Layered

**Control Styles**

-Centralized

-Delegated

-Dispersed

**Trust Boundaries**

            Pass copies of data instead of sharing

           Check on timliness, relavance , and correctly formed data

           Make objects read only to prevent modification

**https://lh4.googleusercontent.com/KiHeTwSqW9LhdiDd5lPa6GIZGeqG4PRX-hlgPf7vk-08iB78VDgRDGzPyLQA8_qgdwb-r4RpJQBtU1P4lnJhkE35SmT7gAJ5as2uFypTq1vvSFTVj5a6KYdieRHh1BYfEkZEb-Pb**

**Aggregation - Wheel is part of Car, but wheel can outlive car**

**https://lh5.googleusercontent.com/DZt0IvE3vS1qiyyKZ-mvCA6zCdFTkZ49PLukaKmxbBDAKwVXAso9ol2mynUh5aYCWWcthuRkDOB8-6nPr1EO9-0xyIZaMOvr0ujoDPOwXEodd5_BHMaoHHqlPDEjKQFWZ_CPBqny**

**Composition - If the company object dies, the department object dies**

Class implements interface (points towards interface)

https://lh5.googleusercontent.com/Sb9q7XdmQzknjf9CBm0L5_q6MM4IuNIvr72FysTuSj2WEzRTshTFjWDrdbMMMZS_nRiFqMST04EBa-QdSUGj6AFZR0iIlu57CRcFoIRCudG6cCT-L5KS98rYGlPZFuVSG67QcGXQDependency (used for dependencies on outside packages)