

































**254**

Robustness diagrams are written after use cases and before class diagrams. They help to identify the roles of use case steps. You can use them to **ensure your use cases are sufficiently robust** to represent usage requirements for the system you're building.

They involve:

1. Actors
2. Use Cases
3. ***Entities***
4. ***Boundaries***
5. ***Controls***

Whereas the [Model-View-Controller](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller) pattern is used for user interfaces, the Entity-Control-Boundary Pattern (ECB) is used for systems. The following aspects of ECB can be likened to an abstract version of MVC, if that's helpful:

**Entities** *(model)*  
Objects representing system data, often from the domain model.

**Boundaries** *(view/service collaborator)*  
Objects that interface with system actors (e.g. a **user** or **external service**). Windows, screens and menus are examples of boundaries that interface with users.

**Controls** *(controller)*  
Objects that mediate between boundaries and entities. These serve as the glue between boundary elements and entity elements, implementing the logic required to manage the various elements and their interactions. It is important to understand that you may decide to implement controllers within your design as something other than objects – many controllers are simple enough to be implemented as a method of an entity or boundary class for example.

**Four rules apply to their communication:**

1. Actors can only talk to boundary objects.
2. Boundary objects can only talk to controllers and actors.
3. Entity objects can only talk to controllers.
4. Controllers can talk to boundary objects and entity objects, and to other controllers, but not to actors

**Communication allowed:**

Entity Boundary Control

Entity X X

Boundary X

Control X X X