

Domain Driven Design

Over the last decade or two, a philosophy has developed as an undercurrent in the object community. The premise of domain-driven design is two-fold:

- * For most software projects, the primary focus should be on the domain and domain logic; and
- * Complex domain designs should be based on a model.

Domain-driven design is not a technology or a methodology. It is a way of thinking and a set of priorities, aimed at accelerating software projects that have to deal with complicated domains.

To accomplish that goal, teams need an extensive set of design practices, techniques and principles. Refining and applying these techniques will be the subject of discussion for this site, generally starting from the language of patterns laid out in Domain-Driven Design, by Eric Evans.

of course many things can put a project off course, bureaucracy, unclear objectives, lack of resources, to name a few, but it is the approach to design that largely determines how complex software can become. When complexity gets out of hand, the software can no longer be understood well enough to be easily changed or extended. By contrast, a good design can make opportunities out of those complex features.

Some of these design factors are technological, and a great deal of effort has gone into the design of networks, databases, and other technical dimension of software. Books have been written about how to solve these problems. Developers have cultivated their skills.

Yet the most significant complexity of many applications is not technical. It is in the domain itself, the activity or business of the user. When this domain complexity is not dealt with in the design, it won't matter that the infrastructural technology is well-conceived. A successful design must systematically deal with this central aspect of the software.