**DDD 作业**

DDD 作为一种软件开发方法论，可以帮助开发人员更好地理解和解决复杂业务领域中的问题。他不像之前一些比如 MVC 的模型一样从数据出发进行设计，而是从业务出发，自顶向下，站在解决领域问题的角度去分析和设计系统，同时还可以产生一些业务文档。可以优化从需求拆解，项目架构，代码编写，甚至对最后的测试交付出现的一些问题。

以下是我的一些想法：

对 Tetris 项目来说，随着项目不断迭代，每个人对一些业务概念或需求的定位和理解会存在差异，导致在进行交互设计或者开发过程中常常会遇到一些问题，又需要反复开会讨论甚至改版；对于新同事或是负责不同业务的同事（包括前端，后端，测试），很多概念都是隔离的或者模糊的，对项目中的一些信息并不是很清楚，需要额外花时间询问沟通，影响迭代效率。

对此有以下建议：

* 依据 DDD 的开发模式，组织所有相关人员共同讨论建立领域模型，再进行开发。使包括产品，交互，开发，测试在内的每个人都能理解业务需求和概念，并达成一致，提高迭代效率；
* 完善项目有关文档，包括项目说明文档、技术实现文档、业务规则、接口契约规则等，同时每个人也可以将自己负责的业务或实现方案整理成文档进行分享。这样的话，可以确保团队成员对系统的理解一致，提高团队成员沟通效率和默契，也有助于新成员的快速融入和项目的可持续发展。

但经过一段时间的学习，发现 DDD 课程基本都是在后端环境进行讨论的。据了解 iQualtiy 和 Tetris 的后端项目都使用了 DDD 的架构。作为前端开发人员，在学习过程中难免会有想：前端可以 DDD 么？该怎么设计 DDD。

前端作为需求方、后端、交互、用户的桥梁，在与不同角色进行沟通时，常常需要切换思维和兼容不同情况。同时前端是最靠近用户的一层，在代码中难免会出现一些业务逻辑，甚至有的逻辑只能由前端来完成，随着业务不断增加，有时根本无法区分哪些是业务逻辑，哪些是交互逻辑。在某些方面来说，前端锁关注的内容甚至比后端还大，因此对前端而言，无论是业务方面还是代码实现方面，实施 DDD 的好处可能更多。

依据 DDD 的四层分层模型，同时结合前端的特点，前端在架构上也可以分为：UI 层、控制层、领域层、基础层。同时将代码分层组织，根据领域进行划分管理，从架构思想和代码组织方面，也可以实现前端的 DDD。同时对于多端复用的业务来说，只需要重写 UI 层即可，这个是前端使用 DDD 最大的优势。但由于前端关注的内容的不同，同时各种前端框架和技术的不断更新，包括如何在当前项目和团队内实施，可能还需要做一些准备和讨论。