

**Emerging Trends in IT**

Year 3 (2022), Semester 2

***SCHOOL OF INFOCOMM TECHNOLOGY***

Diploma in Information Technology

**ASSIGNMENT 1: Domain-Driven Design**

|  |  |
| --- | --- |
| **Name** | **Nur Hakimi Bin Mohd Yasman** |
| **Teacher** | **Mr. Teo Wen Qiang, Wesley** |

# What is Domain-Driven Design?

Domain-Driven Design, commonly abbreviated as DDD, is a software development method that’s purpose is to simplify the process of creating complex applications. This is done through bridging related pieces of software (microservices) into an ever-evolving model. This concept was introduced and popularised by Eric Evans in 2004, when he released his book *Domain-Driven Design: Tackling Complexity in the Heart of Software*.

To get the full picture, there is a need to dive deeper into the initial definition. Domain, in a development context, is the subject area around which the application logic revolves. In other words, it is the sphere of knowledge and activity around what the application is intended to apply. Subsequently, when discussing the domain layer/logic, it refers to discussing the rules and logic governing the elements of an application. This concept builds upon another concept found in Object-Oriented Analysis and Design; The overarching rules that institutes how business objects interact with one another to create and modify data is the domain logic. There are usually multiple domains in an application, with their respective domain logic. On another note, microservices are an architectural approach to software development where software is composed of small independent services, communicating through APIs. Each of these microservices have their own specific bounded contexts, configurations and dependencies. Microservices should not have more than one bounded context, so as to show its independence and ability to stand on its own.

There are three core principles that Domain-Driven Design is centred around, highlighted by Eric Evans in his book.

1. Focus on the core domain and domain logic

Developers should strive to protect the domain knowledge from any outside influence by other domains. The code and model for a domain should align and not stray from their given subdomains. Boundaries should be made explicit, so as to prevent two things – subdomains from hinging on each other’s structures needlessly and blurring the meaning of domain terms. Therefore, a laser focus should be placed onto not crossing the lines.

1. Base complex designs on models of the domain

Reflect the domain model in the code explicitly, by embedding the domain terminology in the code. This is done simply by using naming conventions followed by the respective domain experts, for code pieces like methods, classes, commands and most importantly, domain events.

1. Collaboration with domain experts to improve the application model and resolve any emerging domain-related issues

Everyone working on the project should strive to capture the domain model, using domain terminology, by communicating with domain experts. Get on the same wavelength as the experts in those domains and areas, in order to understand the problem from their point of view. This is how ubiquitous language is established and the foundation for a harmonious model is set.

Evan further lists out a couple of common terms that are useful when describing and discussing DDD practices.