YAGNI - означает «Вам это не понадобится» и выступает против того, чтобы тратить время на функциональность, которая сейчас не нужна.

Чем меньше кода в проекте, тем лучше.

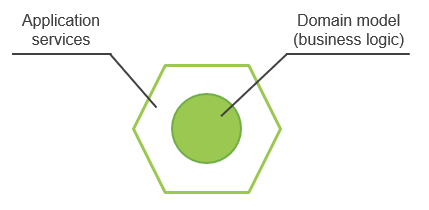
KISS - расшифровывается как «Делайте это коротким и простым» .

YAGNI обрезает ненужную функциональность; KISS - сделать оставшиеся функциональные возможности максимально простыми.

*Гексагональная архитектура*

Лучший способ поддерживать надлежащие границы модели предметной области - придерживаться шестиугольной архитектуры.

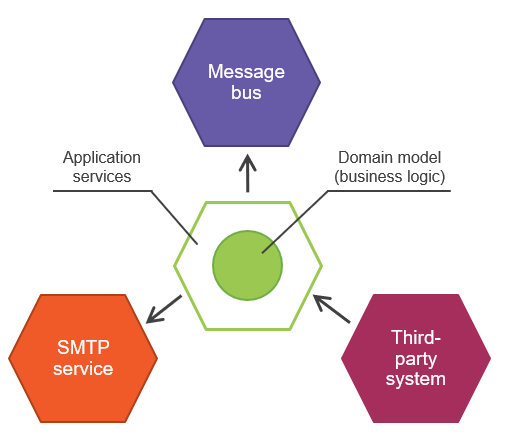
С его помощью вы представляете свое приложение, используя два уровня: модель домена и сервисы приложения.



*Типичное приложение состоит из уровня домена и уровня служб приложений. Уровень домена содержит бизнес-логику приложения; сервисы приложений связывают эту логику со случаями использования в бизнесе.*

Уровень прикладных служб расположен поверх уровня домена и управляет связью между этим уровнем и внешним миром. Например, если ваше приложение представляет собой API RESTful, все запросы к этому API сначала попадают на уровень служб приложений. Затем этот слой координирует работу между классами домена и внепроцессными зависимостями.

Комбинация уровня служб приложений и уровня домена образует шестиугольник , который сам представляет ваше приложение. Он может взаимодействовать с другими приложениями, которые представлены своими собственными шестиугольниками. Этими другими приложениями могут быть служба SMTP, сторонняя система, шина сообщений и т. Д. Набор взаимодействующих шестиугольников составляет шестиугольную архитектуру :



*Гексагональная архитектура - это набор взаимодействующих приложений - шестиугольников.*

Термин *гексагональной архитектуры* был введен Алистером Кокберном. Его цель - подчеркнуть три важных принципа:

* **Разделение проблем между уровнями домена и служб приложений.**  Поскольку бизнес-логика является наиболее важной частью приложения, уровень домена должен отвечать только за эту бизнес-логику и освобождаться от всех других обязанностей. Эти обязанности, такие как связь с внешними приложениями и извлечение данных из базы данных, должны быть отнесены к службам приложений. И наоборот, сервисы приложений не должны содержать никакой бизнес-логики. Их обязанность состоит в том, чтобы адаптировать уровень домена путем преобразования входящих запросов в операции над классами домена, а затем сохранения результатов или возврата их вызывающей стороне. Вы можете просматривать уровень домена как набор знаний домена приложения ( *как*) и уровень сервисов приложений как набор бизнес-вариантов использования ( *что нужно* ).
* **Связь внутри вашего приложения.**  Шестиугольная архитектура предписывает односторонний поток зависимостей: от уровня сервисов приложения до уровня домена. Классы внутри уровня домена должны зависеть только друг от друга; они не должны зависеть от классов на уровне служб приложений. Это руководство вытекает из предыдущего. Разделение проблем между уровнем сервисов приложений и уровнем домена означает, что первый знает о последнем, но обратное неверно. Доменный слой должен быть полностью изолирован от внешнего мира.
* **Связь между приложениями.**  Внешние приложения подключаются к вашему приложению через общий интерфейс, поддерживаемый уровнем служб приложений. Никто не имеет прямого доступа к уровню домена. Каждая сторона в шестиугольнике представляет соединение в или из приложения.

Опять же, шестиугольная архитектура - лучший способ сохранить фокус на основной области. Это не совпадение, что модель предметной области находится в центре шестиугольника.

DDD

Entity vs Value Object:

* Value Object имеют собственную внутреннюю идентичность, а Entity - нет.
* Понятие равенства идентичности относится к Entity; понятие структурного равенства относится к Value Object; понятие ссылочного равенства относится к обоим.
* У Entity есть история; Value Object имеют нулевую продолжительность жизни.
* Value Object всегда должен принадлежать одному или нескольким объектам, он не может жить самостоятельно.
* Value Object должны быть неизменными; сущности почти всегда изменчивы.
* Чтобы распознать Value Object в вашей доменной модели, мысленно замените его целым числом.
* Value Object не должны иметь своих собственных таблиц в базе данных.
* Всегда предпочитайте Value Object над Entity в вашей доменной модели.