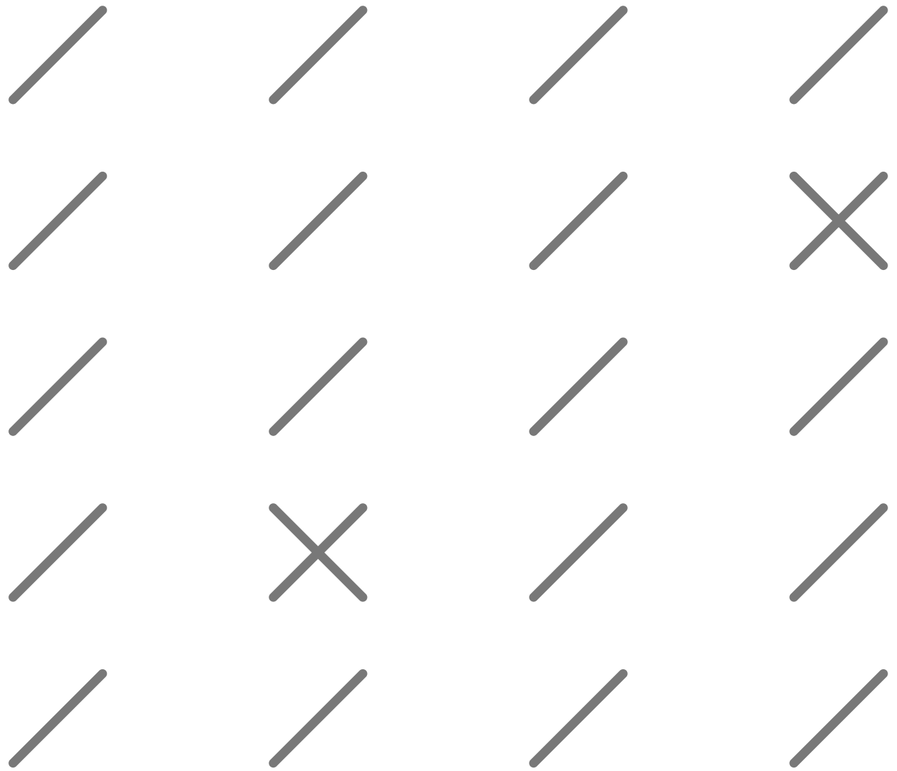
Паттерны и практики написания кода

Принцип DRY (*Don’t Repeat Yourself*)

определение

Каждый фрагмент знания должен иметь единственное, однозначное, надёжное представление в системе →

Повторяющиеся куски кода указывают на упущенную возможность для абстракции



Как правило, дублирующий код выделяют в отдельный метод или класс. *Впервые DRY был описан в книге* **«ПРОГРАММИСТ-ПРАГМАТИК»**, *которую написали Эндрю Хант и Дэвид Томас*

плюсы метода

появляется единая точка для изменений. При смене реализации можно исправить код в одном месте – в ней не возникнет дублирований

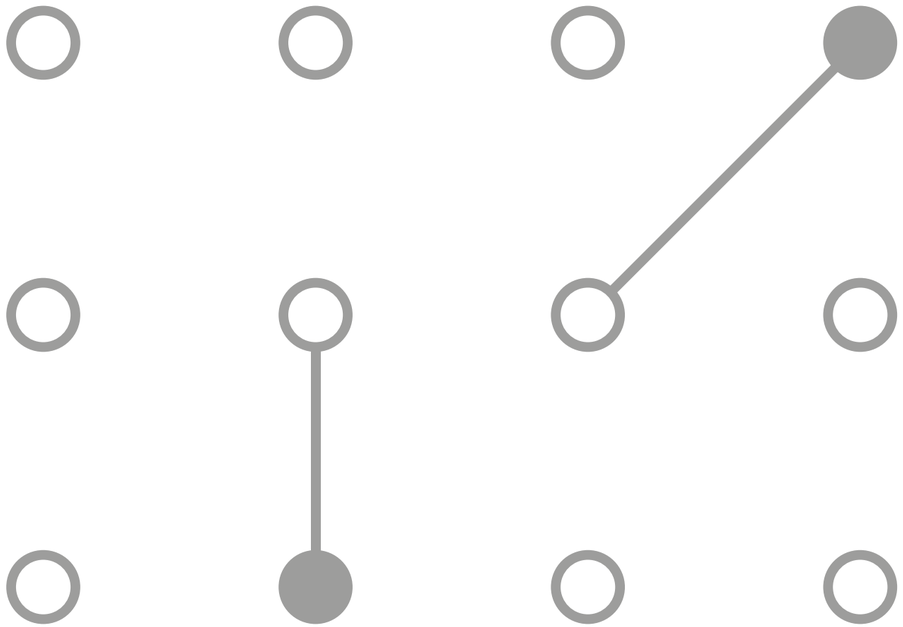
улучшается переиспользование, если раньше в методах было много повторяющихся конструкций и проверок. Теперь, когда они стали небольшими, они сами делают минимально атомарную операцию

повышается скорость разработки. За счёт использования единожды реализованных компактных методов процесс программирования становится конструктором. Объединение базовых деталей даёт разные формы без необходимости постоянно добавлять новый код

уменьшается число ошибок. Если следовать DRY, то код сам собой становится чище, а количество потенциальных багов сильно уменьшается

когда стоит объединять код

ПРАВИЛО ТРЁХ УДАРОВ
из книги *«РЕФАКТОРИНГ»*
Мартина Фаулера



→ **удар 1** добавьте написанный код в тело метода. Пока не предпринимаем никаких действий, потому что неизвестно, будут ли дубликаты в дальнейшем

→ **удар 2** делаете что-то аналогичное ещё раз. Возможно, повторения здесь – это совпадение и переиспользований в коде не будет

→ **удар 3** начинаете рефакторинг. Вынесете весь дублирующий код в отдельный метод или класс

когда мы сталкиваемся с нарушением принципа

- **при копипасте кода.** Почти всегда в скопированный код нужно вносить изменения. Если этого не сделать – мы получим баг
- **при появлении комментариев в коде.** Делайте его самочитаемым, а комментарии оставляйте только для пояснений тонкостей реализации
- **при написании документации.** Без должного надзора документация устаревает намного быстрее, чем обновляется сам код

Этот список лишь часть примеров, а их может быть много. Поэтому постоянно оглядывайтесь по сторонам и находите любые формы проявления нарушения принципа

когда можно пожертвовать принципом DRY

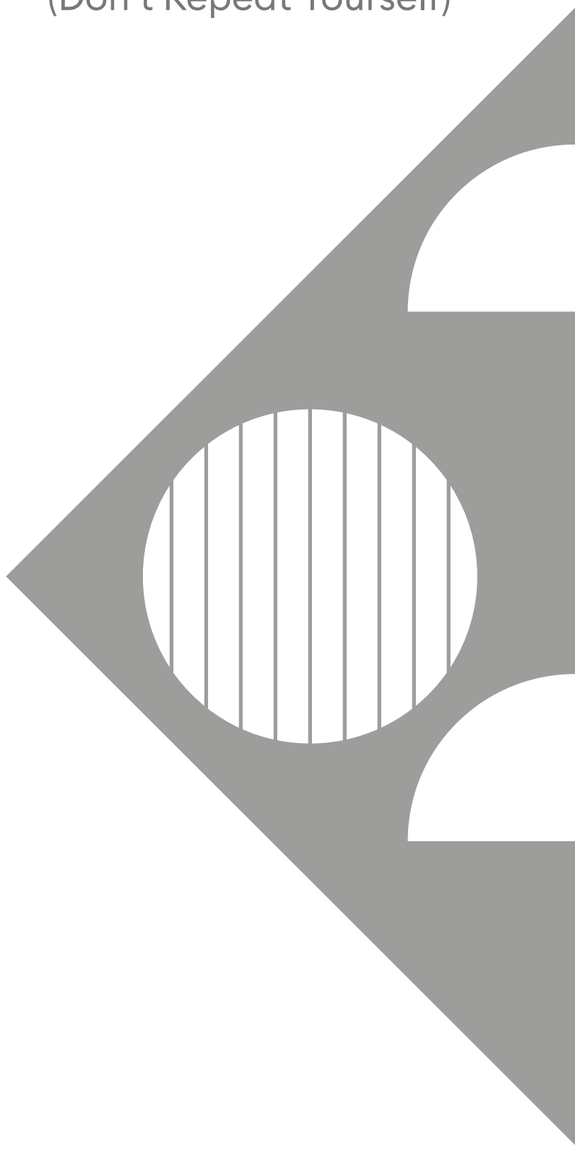
Чётко понимайте что вы делаете, если необходимо пожертвовать DRY, а также хорошо обосновывайте такое решение.
Вот несколько примеров, когда отход от принципа оправдан:

- **при использовании клиент-серверных приложений.**
  В микросервисной архитектуре практически всегда используются повторяющиеся классы в разных сервисах. Различаться они будут своими контекстами, поскольку их доменные ответственности будут отличаться. В этом случае, не нужно делать мега сервис, который занимается конкретной сущностью, такое дублирования является естественным

  Точно такие же проблемы происходят в коде, который находится на backend и frontend. И там и там будут жить одни и те же сущности и это нормально

когда можно пожертвовать принципом DRY

- **при использовании ORM-библиотек, которые дублируют правила базы данных.** В них архитектурно заложено проецирование в коде сущностей и связей таблиц базы. Это полностью нарушает принцип DRY, но нужно для уменьшения разницы подходов реляционной модели баз данных и объектно-ориентированных приложений. Так разработчики могут работать с базой в привычных правилах ООП-приложений



когда использование принципа приносит вред

- **объединение плохо связанных сущностей.** Схожесть атрибутов ещё не означает, что продуктовые сценарии сущностей будут одинаковыми

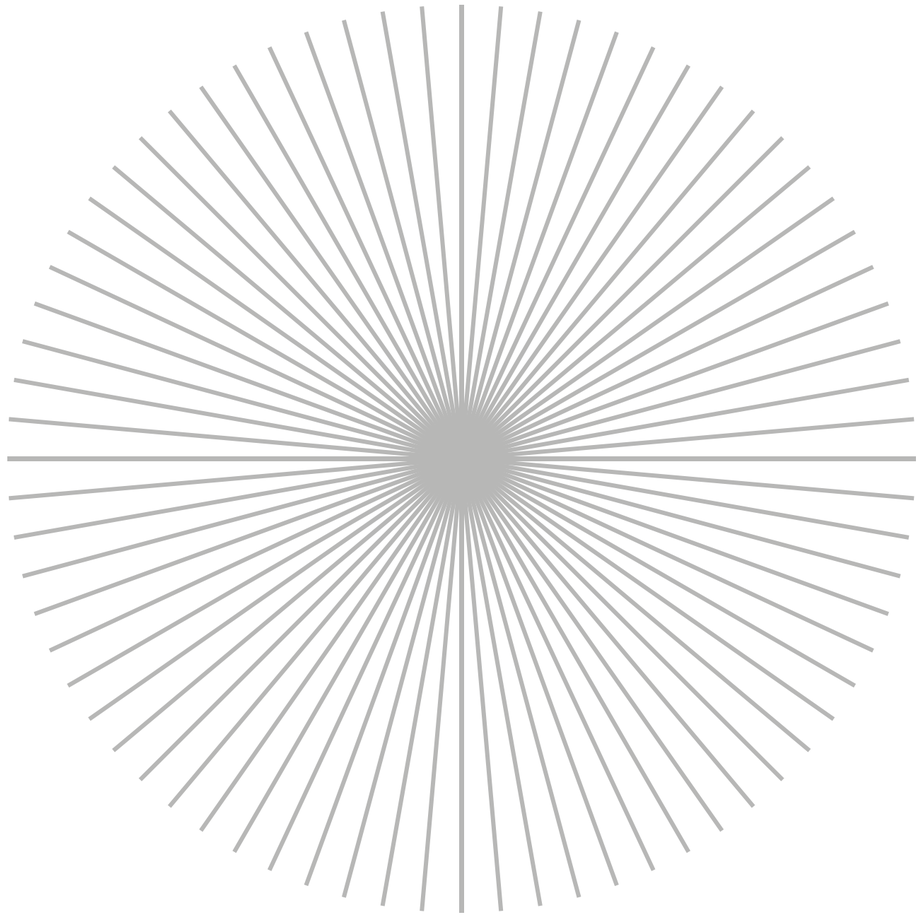
- **объединение мажорных версий API.** При написании новой мажорной версии API стоит стоит писать с нуля, иначе можно случайно сломать старую функциональность. Именно поэтому без дублирования кода здесь не обойтись

- **highload-проекты.** Здесь часто используются денормализованные таблицы. Данные дублируются для отказа от join – получается экономия ресурсов баз данных

рекомендации

- **объединить код намного проще, чем разделить его в будущем.** Если у вас дилемма – сделать один класс или два, то начните с двух классов и только если такой вариант окажется неудобным, объедините код

- **выносите в абстракции небольшие куски кода.** Пишите код так, чтобы методы были небольшими. Иначе потом вы узнаете, что дубликаты попали в уже вынесенные реализации




avito.tech