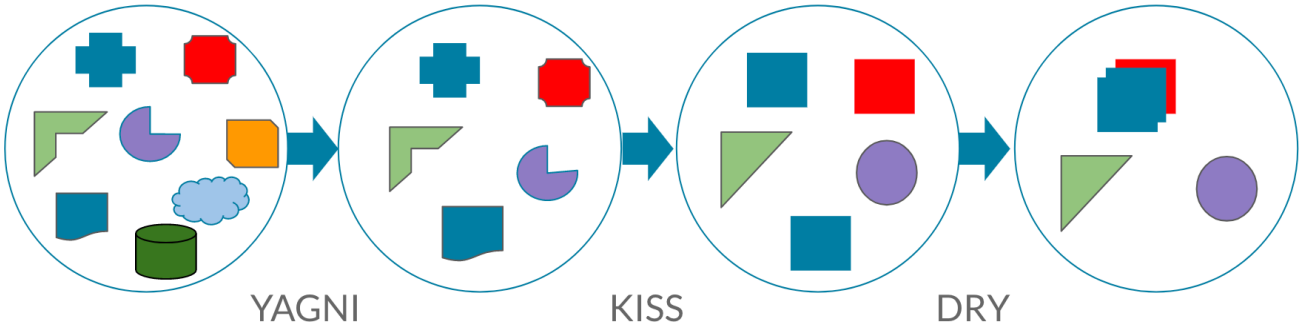
Принципы для разработки: KISS, DRY, YAGNI, BDUF, SOLID, APO и бритва Оккама



Хорошему программисту необходимо уметь совмещать свои навыки со здравым смыслом. Все дело в прагматизме и навыке выбора лучшего решения для вашей проблемы. Когда вы сталкиваетесь с проблемой при разработке ПО, вы можете воспользоваться базовыми принципами, которые помогут в выборе самого правильного подхода.

В этом тексте приводится набор принципов, которые должен знать любой разработчик, и которые следует периодически освежать в памяти. Считайте их своим секретным оружием при программировании.

Последовательное применение этих принципов упростит ваш переход от миддла к сеньору. Вы можете обнаружить, что некоторые (вероятно) вы применяете интуитивно.

Принципов много. Мы остановимся на семи самых важных. Их использование поможет вам в развитии и позволит стать лучшим программистом.

1. YAGNI

You Aren't Gonna Need It / Вам это не понадобится

Этот принцип прост и очевиден, но ему далеко не все следуют. Если пишете код, то будьте уверены, что он вам понадобится. Не пишите код, если думаете, что он пригодится позже.

Этот принцип применим при рефакторинге. Если вы занимаетесь рефакторингом метода, класса или файла, не бойтесь удалять лишние методы. Даже если раньше они были полезны – теперь они не нужны.

Может наступить день, когда они снова понадобятся – тогда вы сможете воспользоваться git-репозиторием, чтобы воскресить их из мертвых.

2. DRY

Don't Repeat Yourself / Не повторяйтесь

Эта концепция была впервые сформулирована в книге Энди Ханта и Дэйва Томаса «Программист-прагматик: путь от подмастерья к мастеру».

Идея вращается вокруг единого источника правды (single source of truth — SSOT). Что это вообще

такое?

> В проектировании и теории информационных систем единый источник истины (SSOT) – это практика структурирования информационных моделей и схемы данных, которая подразумевает, что все фрагменты данных обрабатываются (или редактируются) только в одном месте… SSOT предоставляют достоверные, актуальные и пригодные к использованию данные.
>
> – Википедия

Использование SSOT позволит создать более прочную и понятную кодовую базу.

Дублирование кода – пустая трата времени и ресурсов. Вам придется поддерживать одну и ту же логику и тестировать код сразу в двух местах, причем если вы измените код в одном месте, его нужно будет изменить и в другом.

В большинстве случаев дублирование кода происходит из-за незнания системы. Прежде чем что-либо писать, проявите прагматизм: осмотритесь. Возможно, эта функция где-то реализована. Возможно, эта бизнес-логика существует в другом месте. Повторное использование кода – всегда разумное решение.

3. KISS

Keep It Simple, Stupid / Будь проще

Этот принцип был разработан ВМС США в 1960 году. Этот принцип гласит, что простые системы будут работать лучше и надежнее.

У этого принципа много общего с переизобретением колеса, которым занимались в 1970-х. Тогда он звучал как деловая и рекламная метафора.

Применительно к разработке ПО он значит следующее – не придумывайте к задаче более сложного решения, чем ей требуется.

Иногда самое разумное решение оказывается и самым простым. Написание производительного, эффективного и простого кода – это прекрасно.

Одна из самых распространенных ошибок нашего времени – использование новых инструментов исключительно из-за того, что они блестят. Разработчиков следует мотивировать использовать новейшие технологии не потому, что они новые, а потому что они подходят для работы.

4. Big Design Up Front

Глобальное проектирование прежде всего

Этот подход к разработке программного обеспечения очень важен, и его часто игнорируют. Прежде чем переходить к реализации, убедитесь, что все хорошо продумано.

> Зачастую продумывание решений избавляло нас от проблем при разработке… Внесение изменений в спецификации занимало час или два. Если бы мы вносили эти изменения в код, на это уходили бы недели. Я даже не могу выразить, насколько сильно я верю в важность проектирования перед реализацией, хотя адепты экстремального программирования предали эту практику анафеме. Я экономил время и делал свои

> продукты лучше, используя BDUF, и я горжусь этим фактом, чтобы там ни говорили фанатики экстремального программирования. Они просто ошибаются, иначе сказать не могу.
>
> — Джоел Спольски

Многие разработчики считают, что если они не пишут код, то они не добиваются прогресса. Это неверный подход. Составив план, вы избавите себя от необходимости раз за разом начинать с нуля.

Иногда в недостатках и процессах разработки архитектуры должны быть замешаны и другие люди. Чем раньше вы все это обсудите, тем лучше будет для всех.

Очень распространенный контраргумент заключается в том, что стоимость решения проблем зачастую ниже стоимости времени планирования. Чем с меньшим количеством ошибок столкнется пользователь, тем лучше будет его опыт. У вас может не быть другого шанса справиться с этими ошибками.

5. SOLID

Это наиболее известный принцип разработки ПО. Solid — это аббревиатура от:

S) Single-responsibility principle /Принцип единственной ответственности

Его важность невозможно переоценить. Каждый объект, класс и метод должны отвечать только за что-то одно. Если ваш объект/класс/метод делает слишком много, вы получите спагетти-код. Вот пример:

```
const saveTodo = async () => {
    try {
        response = await saveTodoApi();
        showSuccessPop('Success');
        window.location.href = '/successPage';
    } catch (error) {
        showErrorPopup(`Error: ${error} `);
    }
}
```

Этот метод кажется безобидным, но на самом деле он делает слишком много:

1. Сохраняет объект
2. Обрабатывает уведомление в UI
3. Выполняет навигацию

Еще один побочный эффект такого кода – проблемы с тестированием. Запутанный функционал тестировать сложно.

O) Open–closed principle / Принцип открытости-закрытости

Программные объекты должны быть открыты для расширения, но закрыты для модификации. Речь о том, что нельзя переопределять методы или классы, просто добавляя дополнительные функции по мере необходимости.

Хороший способ решения этой проблемы – использование наследования. В JavaScript эта проблема

решается с помощью композиции.

Простое правило: если вы изменяете сущность, чтобы сделать ее расширяемой, вы впервые нарушили этот принцип.

L) Liskov substitution principle / Принцип подстановки Лисков

Этот принцип гласит, что объекты старших классов должны быть заменимы объектами подклассов, и приложение при такой замене должно работать так, как ожидается.

I) Interface segregation principle / Принцип разделения интерфейсов

Этот принцип был сформулирован Робертом Мартином, когда он консультировал Xerox, и он очевиден.

> Объекты не должны зависеть от интерфейсов, которые они не используют

ПО должно разделяться на независимые части. Побочные эффекты необходимо сводить к минимуму, чтобы обеспечивать независимость.

Убедитесь, что вы не заставляете объекты реализовывать методы, которые им никогда не понадобятся. Вот пример:

```
interface Animal {
  eat: () => void;
  walk: () => void;
  fly: () => void;
  swim: () => void;
}
```

Не все животные могут fly, walk или swim, поэтому эти методы не должны быть частью интерфейса или должны быть необязательными.

D) Dependency inversion principle / Принцип инверсии зависимостей

Этот принцип невозможно переоценить. Мы должны полагаться на абстракции, а не на конкретные реализации. Компоненты ПО должны иметь низкую связность и высокую согласованность.

Заботиться нужно не о том, как что-то устроено, а о том, как оно работает. Простой пример – использование дат в JavaScript. Вы можете написать для них свой слой абстракции. Тогда если у вас сменится источник получения дат, вам нужно будет внести изменения в одном месте, а не в тысяче.

Иногда добавление этого уровня абстракции требует усилий, но в конечном итоге они окупаются.

В качестве примера взгляните на date-io, в этой библиотеке создан тот уровень абстракции, который позволяет вам использовать её с разными источниками дат.

6. Avoid Premature Optimization

Избегайте преждевременной оптимизации

Эта практика побуждает разработчиков оптимизировать код до того, как необходимость этой оптимизации будет доказана. Думаю, что если вы следуете KISS или YAGNI, вы не попадетесь на этот

крючок.

Поймите правильно, предвидеть, что произойдет что-то плохое – это хорошо. Но прежде чем вы погрузитесь в детали реализации, убедитесь, что эти оптимизации действительно полезны.

Очень простой пример – масштабирование. Вы не станете покупать 40 серверов из предположения, что ваше новое приложение станет очень популярным. Вы будете добавлять серверы по мере необходимости.

Преждевременная оптимизация может привести к задержкам в коде и, следовательно, увеличит затраты времени на вывод функций на рынок.

Многие считают преждевременную оптимизацию корнем всех зол.

7. Бритва Оккама

> Бри́тва О́ккама (иногда ле́звие О́ккама) — методологический принцип, в кратком виде гласящий: «Не следует множить сущее без необходимости» (либо «Не следует привлекать новые сущности без крайней на то необходимости»).
>
> — Википедия

Что это значит в мире программирования? Не создавайте ненужных сущностей без необходимости. Будьте прагматичны — подумайте, нужны ли они, поскольку они могут в конечном итоге усложнить вашу кодовую базу.

Заключение

Эти принципы не очень сложны. На самом деле, именно простота делает их красивыми. Если вы сбиты с толку, не пытайтесь применять все их сразу. Просто постарайтесь работать осознанно и пробуйте постепенно включать эти принципы в свой рабочий процесс.

Использование базовых, но действенных принципов позволит вам стать лучшим программистом и иметь более четкое представление о том, почему вы что-то делаете.

Если вы применяете большую часть принципов интуитивно, стоит задумываться и осознавать почему вы делаете что-то определенным образом.

Всего доброго.

Вакансии
НПП ИТЭЛМА всегда рада молодым специалистам, выпускникам автомобильных, технических вузов, а также физико-математических факультетов любых других высших учебных заведений.

У вас будет возможность разрабатывать софт разного уровня, тестировать, запускать в производство и видеть в действии готовые автомобильные изделия, к созданию которых вы приложили руку.

В компании организован специальный испытательный центр, дающий возможность проводить исследования в области управления ДВС, в том числе и в составе автомобиля. Испытательная лаборатория включает моторные боксы, барабанные стенды, температурную и климатическую установки, вибрационный стенд, камеру соляного тумана, рентгеновскую установку и другое специализированное оборудование.

Если вам интересно попробовать свои силы в решении тех задач, которые у нас есть, пишите в личку.

- Старший инженер программист
- Системный аналитик
- Руководитель группы калибровки
- Ведущий инженер-испытатель
- Инженер по требованиям
- Инженер по электромагнитной совместимости
- Системный аналитик
- Старший инженер-программист ДВС

О компании ИТЭЛМА
Мы большая компания-разработчик automotive компонентов. В компании трудится около 2500 сотрудников, в том числе 650 инженеров.

Мы, пожалуй, самый сильный в России центр компетенций по разработке автомобильной электроники. Сейчас активно растем и открыли много вакансий (порядка 30, в том числе в регионах), таких как инженер-программист, инженер-конструктор, ведущий инженер-разработчик (DSP-программист) и др.

У нас много интересных задач от автопроизводителей и концернов, двигающих индустрию. Если хотите расти, как специалист, и учиться у лучших, будем рады видеть вас в нашей команде. Также мы готовы делиться экспертизой, самым важным что происходит в automotive. Задавайте нам любые вопросы, ответим, пообсуждаем.

Список полезных публикаций на Хабре

- Бесплатные онлайн-курсы по Automotive, Aerospace, робототехнике и инженерии (50+)
- [Прогноз] Транспорт будущего (краткосрочный, среднесрочный, долгосрочный горизонты)
- Лучшие материалы по взлому автомобилей с DEF CON 2018-2019 года

- [Прогноз] Motornet — сеть обмена данными для роботизированного транспорта
- Компании потратили 16 миллиардов долларов на беспилотные автомобили, чтобы захватить рынок в 8 триллионов
- Камеры или лазеры
- Автономные автомобили на open source
- McKinsey: переосмысляем софт и архитектуру электроники в automotive
- Очередная война операционок уже идет под капотом автомобилей
- Программный код в автомобиле
- В современном автомобиле строк кода больше чем…