ПОСТРОЕНИЕ ПРОГРАММНЫХ ПРИЛОЖЕНИЙ С ТОЧКИ ЗРЕНИЯ СТАНДАРТИЗАЦИИ И ОПТИМИЗАЦИИ КОДА

Н. А. Атрощенко
Кафедра экономической информатики, Белорусский государственный университет информатики и радиоэлектороники
Минск, Республика Беларусь
E-mail: natasha@atroshenko.by

Несмотря на огромное количество программных продуктов, производимых на сегодняшний день, нельзя снижать планку требований к их качеству, и основой должны служить общая стандартизация, а также концепции оптимального кода и архитектуры приложений.

Введение

Подходы к архитектуре построения программных проектов имеют некоторое сходство с архитектурными решениями при строительстве зданий: принимаются решения, каков будет фундамент (фреймворки, паттерны проектирования, платформы, ОС), внутренняя инфраструктура (распределение нагрузки, пользовательские сервисы), подключение к источникам снабжения (СУБД, хранилища данных, облачные хранилища), коммуникации (системы передач данных, организация многопоточности, сокетных соединений), комфортабельные переходы из одних помещений в другие (юзабилити), стильный внешний вид (удобство восприятия и хорошо отработанная визуальная часть) и т.д. И так же, как и при строительстве, нужно многое предусмотреть заранее. По аналогии ориентироваться в программировании следует не на обрывочные сведения неизвестного происхождения, а, скорее, на требования заказчика, стандарты кода, чёткую документацию по описанию возможностей предполагаемого «строительного материала», знания его особенностей для исключения досадных недоразумений вроде несовместимости, нехватки ресурсов, невозможности подключить нужные библиотеки или плагины, наличие устаревших систем безопасности, не отслеживаемых утечек памяти и т.д.

I. Концепции требований к оптимальности кода

Оптимальный код – достаточно относительное понятие, так как, во-первых, с изменением требований к конечному продукту и появлением новых подходов и технологий происходят изменения в развитии практически всех направлений в IT- области. Во-вторых, сама концепция оптимального кода не может быть устойчива и неизменна. В-третьих, как определить критерий, что важнее: краткость кода, его понятность, ресурсоёмкость, нагрузка на ОС или успешность решения поставленной задачи? Иногда один фактор нужно ущемить в ущерб другому, и здесь подход может быть только индивидуальным, в зависимости от требований проекта. Но при этом достаточно чётко со временем вырисовываются отрицательные моменты программного кода. Интуитивно понятный код – важнейшая составляющая в программировании. Вопрос, насколько быстро смогут другие программисты выяснить цель и логику вашего кода определяет концепции функционального и логического программирования. Допустим, на этапе разработки выясняется, что код «плохой».[1] Как объяснить программисту, что от него ожидалось изначально и что нужно сделать, чтобы его оптимизировать? Основные требования, такие как:

II. Концепции требований к оптимальности кода

- уменьшение количества повторяемого кода,
- присваивание функциям логических имён,
- разбиение кода на логические части,
- правильная типизация данных,
- первичная инициализация (особенно для C++)
- правильная комплектация проекта по папкам (строгая иерархия визуальной, логической, ресурсной и других составляющих проекта),
- чёткие комментарии,

- это лишь первичные требования, которые должен знать любой новичок, равно как и частные составляющие: сокращение записи за счёт замены тернарной операцией стандартных конструкций if...else, использование операторов инкрементирования, префиксных и постфиксных записей и т.д. Дальнейшие требования касаются классической реализации классов и ООП, пространства имён, полная обработка исключительных ситуаций во избежание аварийного завершения работы приложения, физического разделения кода на модули, использование паттернов проектирования, подключение механизма абстракций данных, систему управления модулями и ресурсами. И, наконец, всё, что относится к высокоуровневой концепции оптимальной организации кода проекта: борьба с утечкой памяти, распределение готовых модулей по собствен-

ным библиотекам, уменьшение ресурсоёмкости и памяти при запуске и работе проекта, организация кроссплатформенности, быстрого развёртывания, локализации, интернационализации и т.д. [2]

III. Практика хорошего кода

Каждый новый взгляд на то, как надо писать код, вызывает бурю эмоций – от восторга до полного неприятия. На какие критерии опираться прежде всего, чтобы написанный код хоть и не вызывал бы определённых восторгов, но имел наименьшее количество нареканий? Не ставя себе цель добиться перфекционизма, из множества требований можно выделить основные требования, к которые трудно сбросить со счетов, так как их необходимость продиктована самой жизнью.

- Во-первых, он должен выполнять поставленные задачи. Для стабильно работающего проекта обычно снимаются критические замечания разного толка; ни для кого не секрет, что ошибки в коде, безусловно, будут, но ни одна из них не должна приводить к критической ситуации, а оптимизировать и улучшать код можно до бесконечности.
- Во-вторых, важна простота написания кода. Огромное количество логических зависимостей разного вида: между блоками, классами, пространствами имён, методами, функциями и множеством других «структурных объединений», которых достаточно в каждом языке программирования могут создать такой «винегрет», что в нём не в состоянии будет разобраться никто. Если программист не собирается готовить сознательно диверсию против своей компании, такие случаи нужно исключить.
- В-третьих, это одинаковый стиль написания кода в проекте. К примеру, если в C++ для упрощения кода используется перегрузка операторов для пользовательских типов/классов, то неплохо использовать это повсеместно в проекте, либо постараться не использовать совсем.
- В-четвёртых, нужно исключить повторяемость функционала. Этот принцип уже имеет свою аббревиатуру DRY (Don't repeat yourself – не повторяйся!). Это означает, что все функциональные единицы проекта, будь функция, класс или логический блок кода имеют право быть представленными в коде только один раз. Повторяющийся код выносится в функции, блоки библиотеки, пространства имён и т.д.[3]
- В-пятых, не стоит ради одной узкой задачи писать много ненужного кода ради использования одного фреймворка, не нужно «стрелять из пушки по воробьям» и использовать сложные паттерны, если проблему можно решить обычными методами ООП, не стоит использовать сложные технологии там, где их возможности используются менее, чем на 20

IV. Стандартизация программного обеспечения

В любом случае при написании кода и построении проекта необходимо опираться на существующие стандарты: внутрикорпоративные, внутрифирменные, федеральные, национальные, международные, межгосударственные. К ним относятся, например, международная организация по стандартизации (ИСО)[1], Объединенный технический комитет JTC11, имеющий 17 подкомиссий, и регламентирующий множество вопросов от техники программного обеспечения до языков программирования, компьютерной графики и обработки изображения, соединения оборудования, методов защиты и т. д., Total Quality Management (TQM), базирующаяся на управлении качеством, ANSI/NCITS, наши ГОСТ 9126 (Информационная технология. Оценка программного продукта. Характеристики качества и руководящие указания по их применению), ГОСТ 19781-90, ГОСТ Р ИСО/МЭК 9294-93 (Информационная технология. Руководство по управлению документированием программного обеспечения. Стандарт полностью соответствует международному стандарту ИСО/МЭК ТО 9294:1990 и устанавливает рекомендации по эффективному управлению документированием ПС для руководителей, отвечающих за их создание) и т.д. Следование стандартам и хорошо зарекомендовавшим себя концепциям построения проектов и хранилищ данных повышает качество и надёжность программного кода, достигаются поставленные цели, сокращаются сроки развёртывания и внедрения приложения, улучшается обслуживание, обновление, поставка и поддержка.

1. International Organization for Standardization, ISO = Международная организация по стандартизации: Официальный сайт. [Электронный ресурс]. - Режим доступа: http://WWW.ISO.ORG
2. Консорциум Всемирной паутины [англ. World Wide Web Consortium, W3C]: Официальный сайт. [Электронный ресурс]. - Режим доступа: http://WWW.W3.ORG
3. Нильсен, Я. Необходимость разработки стандартов веб-дизайна [Оригинал документа на англ. языке] / Я. Нильсен. [Электронный ресурс]. - Режим доступа: http://www.useit.com/alertbox/20040913.html.