**Правила проектирования программы вер. 0.1**

|  |  |
| --- | --- |
| **№** | **Этап проектирования** |
| **1** | **Формулирование бизнес-правил и создание схемы компонентов системы.** |
| 1.1 | Формулирование бизнес-правил. |
| 1.2 | Составление набора функций, которые должна выполнять система, и определение требований к системе на основе бизнес-правил. |
| 1.3 | Разбиение программы на компоненты, объединенные линиями без стрелок (схема компонентов). Линии указывают на то, какие компоненты связаны. Установление зависимостей компонентов на данном этапе не производится [1]. |
| **2** | **Создание базового функционала и схемы зависимостей компонентов.** |
| 2.1 | Правила проектирования.  При проектировании системы (написании кода) необходимо:   1. Руководствоваться принципами гибкой разработки SOLID. 2. Руководствоваться правилом деления системы на уровни или глобальные компоненты: бизнес-правила (BR), графический интерфейс (UI), база данных (DB) [2]. 3. Определять принадлежность каждой сущности (структуры, класса) к одному из трех глобальных компонентов: бизнес-правила (BR), графический интерфейс (UI), база данных (DB). Если возникают затруднения при определении к какому уровню принадлежит класс (структура), значит нужно либо изменить его содержание, назначение (что-то убрать, переделать, добавить), либо требуется создать класс(ы) или компонент(ы), которые будут дополнять проблемный класс, либо разделить класс на несколько классов, не нарушая при этом принципы SOLID. В крайнем случае временно оставить его, пока не будет найдено подходящее решение. Каждая сущность должна принадлежать только одному из трех уровней. 4. Использовать шаблоны проектирования при необходимости, когда шаблон уместен в данной ситуации. 5. Не создавать избыточное количество зависимостей (факт. 4-й принцип SOLID). Например, если сущность (структура) используется во многих классах (к ней идет много зависимостей), но при этом сама сущность содержится в определении класса и используется за пределами класса – там где этот класс не используется, то необходимо вынести ее в отдельный файл(ы); т.е. следует избегать включения кода в файлы, где он не используется. Это не только минимизирует время перекомпиляции, но и упростит проектирование. 6. Руководствоваться зависимостями уровней: BR не должны зависеть от типа DB. Конкретная DB должна быть скрыта за интерфейсом. BR не должны зависеть от UI. UI и DB должны зависеть от BR, однако BR не должны явно содержаться в UI и DB, и должны быть изолированы, чтобы изменения в UI и DВ не повлияли на BR. 7. Руководствоваться правилом проектирования снизу вверх [3]. |
| 2.2 | Создание схемы зависимостей компонентов (ориентированного графа) на основе уже имеющегося кода. На схеме компонентов должны быть указаны зависимости (линия со стрелкой) преимущественно между классами. Также можно построить схему зависимостей компонентов, в которой указаны зависимости между компонентами. |
| **3** | **Качественная и количественная оценка архитектуры системы.** |
| 3.1 | Качественная оценка.  Проверить выполнение требований к графу компонентов:   1. Направление стрелок в графе должно соответствовать принципам SOLID. 2. Проверить выполнение принципа ацикличности зависимостей. Граф должен быть ациклическим. Цикл компонентов ведет себя как один компонент и изменение любого компонента в цикле приведет к изменению остальных. |
| 3.2 | Количественная оценка (при необходимости, т.к. ресурсная):   1. Оценить меру устойчивости каждого компонента по формуле для метрики устойчивости. 2. Оценить меру абстрактности каждого компонента по формуле для метрики абстрактности. 3. Проверить выполнение принципа устойчивых зависимостей: «Зависимости должны быть направлены в сторону устойчивости». 4. Проверить выполнение принципа устойчивости абстракций: «Устойчивость компонента пропорциональна его абстрактности». |
| **4.** | **Повторное прохождение через этапы 1-3.** |
| 4.1 | Следующая итерация разработки, связанная с доработкой системы, ее развитием и пр. |
|  | **Определения.**  Компонент – это набор классов.  BR – бизнес правила.  UI – графический интерфейс.  DB – база данных (данные в СУБД, файлы, данные в ОЗУ).  Зависимость классов – Стрелка, направленная от класса A к классу B, указывает, что A зависит от B, т.е. изменения в коде класса A приведут к перекомпиляции класса B. |
|  | **Цитаты из книги «Чистая архитектура» Роберт С. Мартин:**  [1] «Диаграммы зависимостей компонентов слабо отражают функции приложения. В большей степени они являются отражением *удобства сборки и сопровождения* приложения. В этом главная причина, почему они не проектируются в начале разработки проекта. В этот период нет программного обеспечения, которое требуется собирать и сопровождать, поэтому нет необходимости составлять карту сборки и сопровождения.»  [2] «Графический интерфейс не имеет значения для бизнес-правил, поэтому между ними нужно провести границу. База данных не имеет значения для графического интерфейса, поэтому между ними нужно провести границу. База данных не имеет значения для бизнес-правил, поэтому между ними нужно провести границу.»  [3] «Структура компонентов не может проектироваться сверху вниз … Попытка спроектировать структуру зависимостей компонентов раньше любых классов, скорее всего, потерпит неудачу. На этом этапе мы почти ничего не знаем о согласовании изменений, не представляем, какие элементы можно использовать многократно и почти наверняка создадим компоненты, образующие циклические зависимости. Поэтому структура зависимостей компонентов должна расти и развиваться вместе с логическим дизайном системы.» |